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 so as to maximize the number of vertices whose degree in T is the same as in G (these are called vertices of "full" degree). We show that this problem is NP-hard. We also 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 $coR = NP$. For the case of planar graphs, we present a polynomial time approximation scheme. Additionally, we present some experimental results comparing our algorithm to the previous heuristic used for this problem.

1 Introduction

In this paper we study the problem of computing a spanning tree T in a connected graph $G = (V, E)$ so as 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 [10]. Tree optimization problems have been very well studied. For example, see [6] for a survey. Recently, Solis-Oba obtained a 2-approximation for maximizing the number of leaves of a spanning tree [15], improving the bound of 3 obtained by Lu and Ravi [11].

The problem arises as a central problem in the work by Pothof and Schut [14] 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" [14]. 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

[*]Currently at Bell Labs, Lucent Technologies, Murray Hill, NJ 07974 E-mail:`randeep@research.bell-labs.com`.
[†]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`.

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 [14] 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: In Section 2 we show that the decision version of the problem is NP-hard even when restricted to planar graphs by a reduction from the Maximum Independent Set problem, which is known to be NP-complete for planar graphs [7]. We then 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 3.1 we provide an algorithm with an approximation ratio of $O(\sqrt{n})$ for the FDST problem. In Subsection 3.2 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 $P \neq NP$.

In Section 4 we show that there is a polynomial approximation scheme for the FDST problem for planar graphs using a technique due to Baker [2]. In other words, for any fixed $\epsilon > 0$ there is an algorithm that runs in polynomial time, and guarantees an approximation ratio of $(1+\epsilon)$. This is particularly relevant, since fluid networks are often planar [14]. To make the method work, we have to generalize the problem we solve to allow for restricted vertices, which do not contribute to our count of full degree vertices.

In Section 6 we compare and contrast our heuristic with the one given by Pothof and Schut [14]. For large random graphs (we tried various kinds of general graphs and planar graphs generated by LEDA [12] and Stanford Graphbase [9]) 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).

A very recent paper independently uses identical methods to find a linear PTAS for planar graphs. This paper also gives algorithms to find exact solutions to the FDST problem for interval graphs, cocomparability graphs, graphs of bounded asteroidal number, and bounded tree-width graphs [5].

2 NP-hardness

Full-degree Spanning Tree: Given a graph $G = (V, E)$, and an integer K, does G contain a spanning tree T such that there are at least K vertices with *full* degree?

It is easy to see that this problem is in NP. We prove that the problem is NP-complete by showing a polynomial time reduction from the well known "independent set" problem.

Independent Set: Given a graph $G' = (V', E')$, and an integer K', does G' have a subset of vertices of size K' that form an independent set? An independent set is a subset of vertices with no edges between them.

Theorem 1: The Full-degree Spanning Tree problem is NP-complete even when restricted to planar graphs.

It is worth noting that the FDST problem and the Independent Set problem are "similar", except that in the FDST problem the constraints are not just local (between pairs of vertices), since we have to ensure an acyclic solution.

Proof: We do the reduction from the Independent Set problem, which is NP-complete even when restricted to planar graphs [7], and we note that our reduction preserves planarity.

We do the reduction from $IS(G', K')$ to $FDST(G, K)$ as follows. We create two new vertices corresponding to each edge $e = (x, y)$ of G'. We add edges from these two new vertices to x and y and remove the edge (x, y). Formally, let $V = V' \cup V_{E'} \cup U_{E'}$, where $V_{E'} = \{v_e | e \in E'\}$ and $U_{E'} = \{u_e | e \in E'\}$. Let $E = \{(x, v_e), (x, u_e), (y, v_e), (y, u_e) | (x, y) \in E'\}$. We set $K = K' + |V'| - 1$.

We now prove that G' has an independent set of size K' if and only if G has a spanning tree with K full degree vertices. Suppose G' has an independent set S of size K'. Fix a spanning tree T' in G'. We will construct a tree T in G. For each v_e, $e = (x, y) \in T'$, we include (v_e, x) and (v_e, y) in T. For each v_e if $e = (x, y) \notin T'$ with $x \in S$ then add (v_e, x) to T, otherwise add any one of the two edges incident on v_e to T. For each u_e if $e = (x, y)$ with $x \in S$ then add (u_e, x) to T, otherwise add any one of the two edges incident on u_e to T. This is a spanning tree that spans the vertices in V.

The set of full degree vertices are as follows: $\{v | v \in S\} \cup \{v_e | e \in T'\}$ and has size $|S| + |V'| - 1$, which has size K.

To prove the converse, take a spanning tree T with K vertices of full degree. For any $e \in E'$, we cannot have both v_e and u_e have degree two. In fact, without loss of generality we may assume that u_e has degree one, and v_e may have degree one or two. If a vertex $v \in V$ has full degree then no neighbor of v has full degree (otherwise we have a cycle in T), hence the set of vertices in V with full degree forms an independent set in G'. Consider the vertices in $V_{E'}$ with full degree (degree two). If we replace the pair of edges (v_e, x) and (v_e, y) with the direct edge (x, y) and delete the vertices in $U_{E'}$ then we obtain a spanning tree in G'. Since this has at most $|V'| - 1$ edges, we have at most $|V'| - 1$ vertices in $V_{E'}$ with degree two. The remaining full degree vertices are from V and there are at least K' of them, and these form an independent set in G'. $\square$

3 Approximation Results

3.1 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 edges in 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 [16]) to maintain the connected components induced by the current spanning forest F. 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 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 2: 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 [16].

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 3 shows that $|A_d| \leq 2d|I_d|$ and Lemma 5 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

```
**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

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 3: (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 3.

Corollary 4: 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 5: (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 $|B| \leq \frac{n-2}{d-1}$.

□

3.2 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 $coR = NP$ [8].

Theorem 6: 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 [8] 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. □

4 Planar Graphs

The existence of an optimal solution computable in linear time for graphs of bounded tree-width leads immediately to a polynomial time approximation scheme (PTAS) for all planar graphs by a combination of results of Bodlaender and Baker [2, 3].

Baker gives a general framework that constructs a PTAS for any problem which can be solved optimally for k-outerplanar graphs — planar graphs where all nodes have a path of length $\leq k$ to a node on the outermost face [2]. Bodlaender [3] proves that any k-outerplanar graph has tree-width at most $3k-1$. In the next section we describe an optimal solution to the FDST problem for graphs of bounded tree-width, thus implying one for graphs that are k-outerplanar for a fixed constant k. First we show the modifications necessary to Baker's scheme to design a PTAS for the FDST problem.

By choosing $k = \lceil \frac{2(1+\epsilon)}{\epsilon} \rceil$, we will obtain an algorithm that will give at least $\frac{|OPT|}{1+\epsilon}$ vertices of full degree for any $\epsilon > 0$.

Description of Algorithm:
Let $d(v)$ = shortest path length from v to any node on the outer face of G. This scheme creates a collection of decompositions of the planar graph G into a set of k-outerplanar graphs. For each value of $i = 0 \ldots (k-1)$ we generate a decomposition as follows: delete edges that connect nodes with label $d(v)-1$ and $d(v)$, where $d(v)$ is congruent to $i(\bmod k)$. For example, for $i = 0$, we delete edges connecting nodes with $d(v)$ value $k-1$ and k, $2k-1$ and $2k$ etc. After we delete these edges, we are left with a graph G_i, which is a collection of connected components that are each k-outerplanar. We obtain the optimal solution for G_i by running a linear time algorithm for each connected component, since these have bounded tree-width (see Section 4.1).

In the framework defined by Baker [2], the PTAS for independent sets is achieved by obtaining an optimal solution for k-outerplanar subgraphs. These problems are made independent by removing a layer of vertices between the subgraphs. This does not work for the FDST problem. For example, a vertex that is removed may be a neighbor of two full-degree nodes already connected in the spanning tree. Removing such a node allows a solution in the subproblem, which is not feasible when we consider the entire graph. Instead, in our partition, every vertex in G_i is included in exactly one connected component, since we remove edges between the components. The set of vertices incident on these removed edges are *restricted*, and not eligible to be considered as full-degree nodes for our subproblems. (This is necessary, otherwise we may include nodes that have full degree in the k-outerplanar subgraphs, but do not have full degree in the original graph G.) Specifically, we solve optimally the FDST problem with the restriction that a subset of the nodes cannot be chosen as full-degree nodes. The set of restricted nodes are the boundary nodes of a single k-outerplanar subgraph.

A given partition of G into a collection of k-outerplanar graphs G_i will have a solution of size at least $|OPT| - |OPT \bigcap R_i|$, where R_i is the set of restricted nodes in G_i. If for all $i, |OPT \bigcap R_i| > \frac{2}{k}|OPT|$ then $\sum_{i \text{ even}} |OPT \bigcap R_i| > |OPT|$. Since $\cup_{i \text{ even}} OPT \cap R_i \subseteq OPT$, we obtain a contradiction. Therefore for some i, we obtain a solution of size at least $(1 - \frac{2}{k})|OPT| \geq \frac{|OPT|}{(1+\epsilon)}$.

Formally, we have:

Input: $G = (V, E)$ an instance of the FDST problem, and $\epsilon > 0$, the required degree of accuracy.

Output: A spanning tree T of G and a set of nodes S that have full degree in T.

Define: $S(G_i, R_i)$ = the solution found by the linear time algorithm that works for a collection of bounded tree-width components, returning the maximum set of full degree nodes in the set $V \setminus R_i$, so that all edges incident on this set of nodes form a forest in G_i. In the next section we show how to compute $S(G_i, R_i)$. We set $k = \lceil \frac{2(1+\epsilon)}{\epsilon} \rceil$, and break the graph into k-outerplanar graphs, which have tree-width at most $3k-1$.

4.1 Bounded Tree-width Graphs

In this section we give a linear-time algorithm for bounded tree-width graphs (if the graph has tree-width k, then the time required for the algorithm to run will be exponential in k but linear in the size of the graph).

Since our graph G_i consists of a collection of connected components, each of tree-width $3k-1$, we can run this algorithm on each component separately, and take the union of the solutions we obtain. The following definition is standard (see [3] for example).

Definition 1: Let $G=(V,E)$ be a graph. A **tree-decomposition** of G is a pair $(\{X_i \mid i \in I\}, T = (I,F))$, where $\{X_i \mid i \in I\}$ is a family of subsets of V and $T=(I,F)$ is a tree with the following properties:

1. $\bigcup_{i \in I} X_i = V$.
2. For every edge $e=(v,w) \in E$, there is a subset X_i, $i \in I$, with $v \in X_i$ and $w \in X_i$.
3. For all $i,j,k \in I$, if j lies on the path from i to k in T, then $X_i \bigcap X_k \subseteq X_j$.

The **tree-width** of a tree-decomposition $(\{X_i \mid i \in I\}, T)$ is $\max_{i \in I}\{|X_i|-1\}$. The tree-width of a graph is the smallest value k such that the graph has a tree-decomposition with tree-width k.

Many problems are known to have linear time algorithms on graphs with constant tree-width, and there are frameworks for automatically generating a linear time algorithm, given a problem specification in a particular format [1, 4]. The FDST problem can be expressed in the formalism of Borie *et. al.* [4] as: $\max|V_1|\,[Forest(V_1, IncE(V_1)]$, which states that we want to maximize the set of nodes such that the graph induced by this set of nodes and all edges incident upon these nodes is a forest. The algorithm required for the PTAS additionally requires the restriction that some nodes cannot be included in the set of full degree nodes, so the explicit problem we need an algorithm for can be described as follows: $\max|V_1|\,[V_1 \subseteq V \setminus R \wedge Forest(V_1, IncE(V_1))]$.

5 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 $O(n/\Delta^2)$.

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

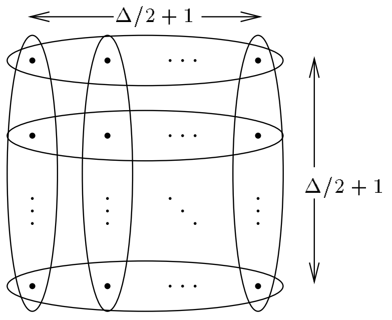


Figure 2: Tight example for the size of the FDST solution.

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.

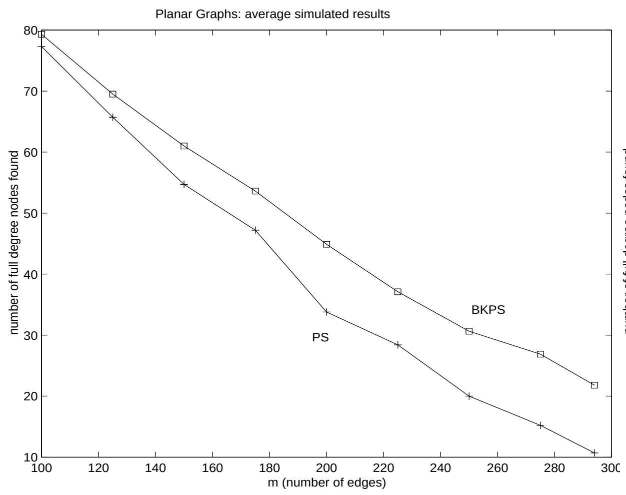
Note: For planar graphs, there are excellent (absolute) bounds of the size of independent sets obtained by the greedy algorithm (see Papadimitriou and Yannakakis [13]). Perhaps such bounds can be obtained for independent sets in the square (G^2) of bounded degree planar graphs? (If the degree is unbounded, 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.)

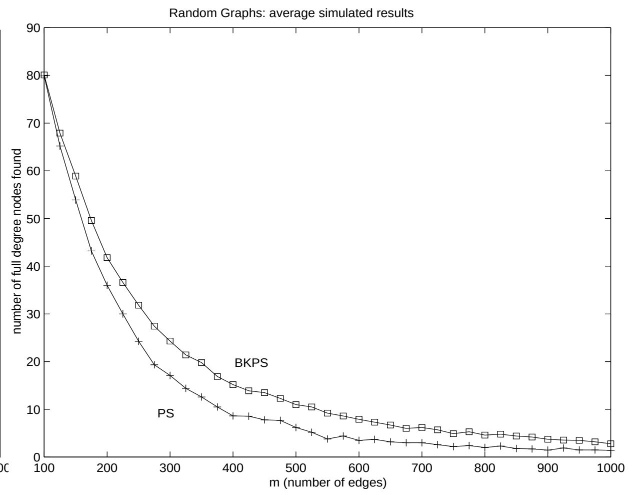
6 Experimental Results

We implemented the Pothof and Schut heuristic [14], 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. Preliminary testing has been done on various kinds of graphs generated in LEDA and Stanford Graphbase [9] (random planar and non-planar graphs) of sizes ranging from 10 vertices to 200 vertices. The charts shown here are 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.

We also computed the optimal solution by writing an integer program capturing the FDST constraints, and ran it through an IP solver. (This method computed the optimal solution only for graphs of at most 30 vertices, and took too long to run for larger graphs.) Another approach based on the following idea was very useful in computing the optimal solution for larger graphs (we were able to obtain optimal solutions for dense graphs with 100 vertices fairly quickly). For each vertex i, we consider both options, namely including or excluding the vertex from the solution. When we decide to include the vertex, we know that many other vertices cannot be included since they would create cycles with the previously included vertices. This enables a pruning step that works extremely well when the optimal solutions are not big (up to 8 vertices), even though the graphs are not small. For dense graphs this does very well, as the optimal solution sizes are pretty small. 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).





Acknowledgments: We thank Jan Schut for sending us a copy of [14] and Sudipto Guha for useful comments.

References

[1] S. Arnborg, J. Lagergren, D. Seese, "Easy Problems for Tree-Decomposable Graphs", *Journal of Algorithms* Vol 12(2), (1991), pp. 308–340.

[2] B. Baker, "Approximation Algorithms for NP-Complete Problems on Planar Graphs", *JACM*, Vol 41 (1), (1994), pp. 153–190.

[3] H. L. Bodlaender, "Some classes of graphs with bounded tree width", *Bulletin of the European Association for Theoretical Computer Science*, (1988), pp. 116–126.

[4] R. B. Borie, R. G. Parker, and C. A. Tovey, "Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families" *Algorithmica*, Vol 7, (1992), pp. 555–581.

[5] H.J. Broersma, A. Huck, T. Kloks, O. Koppius, D. Kratsch, H. Müller, H. Tuinstra, "Degree-preserving forests", *Mathematical Foundations of Computer Science*, (1998) LNCS 1450, pp. 713–721.

[6] 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.

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

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

[9] Donald E. Knuth. "The Stanford Graphbase", ACM/Addison Wesley, (1993).

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

[11] H. Lu and R. Ravi, “Approximating maximum leaf spanning trees in almost linear time”, *Journal of Algorithms*, Vol 29(1), (1998), pp. 132–141.

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

[13] 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), pp. 358–363.

[14] I. W. M. Pothof and J. Schut, “Graph-theoretic approach to identifiability in a water distribution network”, *Memorandum No 1283*, Universiteit Twente (1995). Submitted to *Elsevier Science*.

[15] R. Solis-Oba, “2-Approximation algorithm for finding a spanning tree with maximum number of leaves”, to appear, *6th Annual European Symposium on Algorithms*, (1998).

[16] R. E. Tarjan, “Data Structures and Network Algorithms”, Society for Industrial and Applied Mathematics, (1983).