A Moderately Exponential Time Algorithm for Full Degree Spanning Tree*

Serge Gaspers† Saket Saurabh‡ Alexey A. Stepanov§

Abstract

We present a moderately exponential time exact algorithm for the well-studied Full Degree Spanning Tree problem, an NP-hard variant of the Spanning Tree problem. Given a graph G, the objective is to find a spanning tree T of G which maximizes the number of vertices that have the same degree in T as in G. The problem is motivated by its application in fluid networks and is basically a graph-theoretic abstraction of the problem of placing flow meters in fluid networks. We give an exact algorithm for Full Degree Spanning Tree running in time $\mathcal{O}(1.9465^n)$. This adds Full Degree Spanning Tree to a very small list of "non-local problems", like Feedback Vertex Set and Connected Dominating Set, for which non-trivial (non brute force enumeration) exact algorithms are known.

1 Introduction

The problem of finding a spanning tree of a connected graph arises at various places in practice and theory, like the analysis of communication or distribution networks, or modeling problems, and can be solved efficiently in polynomial time. On the other hand, if we want to find a spanning tree with certain additional properties, like a maximum number of leaves or minimum maximum degree, the problem becomes NP-hard. This paper deals with one of the NP-hard variants of Spanning Tree, namely Full Degree Spanning Tree, from the viewpoint of moderately exponential time algorithms.

Let T be a spanning tree of a graph G. A vertex has *full degree* in T if it has the same degree in T as in G. We say that T is a *full degree spanning tree*

*Additional support by the Norwegian Research Council. A preliminary version of this paper appeared in the proceedings of the 5th Annual Conference on Theory and Applications of Models of Computation (TAMC 2008) [15]

†Centro de Modelamiento Matemático, Universidad de Chile, 8370459 Santiago de Chile. E-mail: sgaspers@dim.uchile.cl.

‡Institute of Mathematical Sciences, CIT Campus, Taramani, 600 113 Chennai, India. E-mail: saket@imsc.res.in.

§Department of Informatics, University of Bergen, N-5020 Bergen, Norway. E-mail: ljosha@ljosha.org.

if it maximizes the number of full degree vertices. The optimization problem that we consider is the following.

> Full Degree Spanning Tree (FDST): Given an undirected connected graph $G = (V, E)$, find a full degree spanning tree of G.

The FDST problem is motivated by its applications in water distribution and electrical networks [22, 26, 27, 28]. Pothof and Schut [28] studied this problem in the context of water distribution networks where the goal is to determine or control the flows in the network by installing a small number of flow meters. It turned out that to measure flows in all pipes, it is sufficient to find a full degree spanning tree T of the network and install flow meters (or pressure gauges) at each vertex of T that does not have full degree. We refer to [1, 4, 19] for a more detailed description of various applications of FDST.

The FDST problem has attracted a lot of attention recently and has been studied extensively from different algorithmic paradigms developed for coping with NP-hardness. Pothof and Schut [28] studied this problem first and gave a simple heuristic algorithm. The decision version of the problem, asking whether a graph has a spanning tree with at least k full degree vertices, was shown to be NP-complete by Bhatia et al. [1] and by Broersma et al. [4]. Bhatia et al. also gave an approximation algorithm of factor $\mathcal{O}(\sqrt{n})$. On the negative side, they showed that FDST is hard to approximate within a factor of $\mathcal{O}(n^{\frac{1}{2}-\epsilon})$, for any $\epsilon > 0$, assuming $coRP \neq NP$, a well known complexity-theoretic hypothesis. The complexity assumption of this lower bound can be strengthened to $P \neq NP$ if in the proof of [1], the more recent result by Zuckerman [35] on the non-approximability of the Maximum Clique problem is used instead of the result by Håstad [18]. Guo et al. [17] studied FDST in the realm of parameterized complexity and observed that the reduction of Bhatia et al. can be used to show that the decision version of the problem is W[1]-hard. The dual problem has also been studied in the literature, that is the problem of finding a spanning tree that minimizes the number k of vertices not having full degree. For the dual version of the problem, Khuller et al. [19] gave an approximation algorithm of factor $2 + \epsilon$ for any fixed $\epsilon > 0$, and Guo et al. gave a fixed parameter tractable algorithm running in time $4^k n^{\mathcal{O}(1)}$. Lokshtanov et al. [24] showed that the analog to FDST on directed graphs is W[1]-hard and that the dual can be solved in time $5.942^k n^{\mathcal{O}(1)}$ on directed graphs. Further, Broersma et al. [4] have shown that FDST can be solved in polynomial time if the input is restricted to graphs with bounded treewidth and graphs with a bounded asteroidal number. For interval graphs and cocomparability graphs (which have asteroidal number at most 2), Broersma et al. further improved the running time. Polynomial time algorithms were also designed for strongly chordal graphs and directed path graphs [23]. However, it has been shown that the problem remains NP-hard for split graphs [4], bipartite planar graphs of maximum degree 5, and planar graphs of maximum degree 3 [6].

The goal of this paper is to study Full Degree Spanning Tree in the context of moderately exponential time algorithms, another coping strategy to

deal with NP-hardness. We refer to [13, 14, 30, 31, 32, 34] for introductory texts and surveys on exact exponential time algorithms. They have an old history [5, 21] but the last few years have seen a renewed interest in the field. This has led to the advancement of the state of the art on exact algorithms and many new techniques based on Inclusion-Exclusion, Measure & Conquer and various other combinatorial tools have been developed to design and analyze exact algorithms [2, 3, 10, 33]. Branch & Reduce has always been one of the most important tools in the area but its applicability was mostly limited to 'local problems' (where the decision on one element of the input has direct consequences for its neighboring elements) like Maximum Independent Set, Sat and various other problems, until recently. In 2006, Fomin et al. [11] devised an algorithm for Connected Dominating Set (or Maximum Leaf Spanning Tree) and Razgon [29] for Feedback Vertex Set combining sophisticated branching and a clever use of measure (see also [12] and [9]).

In this paper, we give a $\mathcal{O}(1.9465^n)$ time algorithm for Full Degree Spanning Tree, breaking the trivial $2^n n^{\mathcal{O}(1)}$ barrier. Our algorithm adheres to this machinery and adds an important real life problem to a small list of non local problems which can provably be solved exponentially faster than by brute force enumeration of all candidate solutions. The running time analysis of the algorithm uses an involved measure, which is a function of the number of vertices and the number of edges to be added to the spanning tree. In a preliminary version [15], a running time of $\mathcal{O}(1.9172^n)$ was claimed. However, that version presented two correctness issues. First, the reduction rule **R6**, which was supposed to handle degree 2 vertices, was incorrect. The second issue was that a recurrence was missing in the program that computed the upper bound on the running time. We correct the handling of the degree 2 vertices in this paper and provide a simple Python program in the appendix that we used to verify the analysis of the running time.

2 Preliminaries

Let G be a graph. We use $V(G)$ and $E(G)$ to denote the vertices and the edges of G respectively. We simply write V and E if the graph is clear from the context. For $V' \subseteq V$ we define the *induced subgraph* $G[V'] = (V', E')$, where $E' = \{uv \in E : u, v \in V'\}$.

Let $v \in V$. We denote by $N(v)$ the *neighborhood* of v, namely $N(v) = \{u \in V : uv \in E\}$. The *closed neighborhood* $N[v]$ of v is $N(v) \cup \{v\}$. In the same way we define $N[S]$ for $S \subseteq V$ as $N[S] = \bigcup_{v \in S} N[v]$ and $N(S) = N[S] \setminus S$. We define the *degree* of vertex v in G as the number of vertices adjacent to v in G. Hence the degree of v in G is $d_G(v) = |\{u \in V(G) : uv \in E(G)\}|$.

Let G be a graph and G' be a subgraph of G. A vertex $v \in V(G')$ is a *full degree* vertex in G' if $d_G(v) = d_{G'}(v)$. We recall that a *full degree spanning tree* is a spanning tree of a connected graph maximizing the number of full degree vertices. Similarly, we define *full degree spanning forest* by replacing tree with forest in the previous definition (the graph need not be connected in this case).

A set $I \subseteq V$ is an *independent set* of G if no two vertices of I are adjacent in G.

3 Algorithm for Full Degree Spanning Tree

In this section we give an exact algorithm for the FDST problem. Let $G = (V, E)$ denote the input graph on n vertices.

As a basic idea, we use that, given a set of vertices S, we can, in polynomial time, construct a spanning tree T in which all the vertices of S have full degree, or show that no such spanning tree exists. Our first observation towards this is that all the edges incident to the vertices in S, that is

$$E_S = \{uv \in E \text{ such that } u \in S \text{ or } v \in S \} \tag{1}$$

belong to every subgraph of G in which the vertices of S have full degree. If (V, E_S) has a cycle, then G has no spanning tree in which all the vertices of S have full degree. Otherwise, our polynomial time algorithm starts with the forest (V, E_S) and then completes this forest into a spanning tree by adding edges to connect the components of the forest. The last step can be done by using a slightly modified version of the Spanning Tree algorithm of Kruskal [20] that we denote `poly_fdst`. This procedure is described in Subsection 3.2. It uses an idea from Guo et al. [17] that vertices of degree 2 can be removed from S without compromising the optimality of the solution returned by `poly_fdst`. In other words, `poly_fdst` builds a spanning tree in which all vertices of S and a maximum number of vertices in V_2 have full degree, where V_2 is the set of degree-2 vertices of $V \setminus S$. Moreover, we will see that this remains true when V_2 is the set of vertices that have degree 2 in the subgraph of G to which the reduction rules of the next subsection have been applied. Therefore, our algorithm may postpone the decision on degree 2 vertices to the polynomial time procedure.

The exponential part of the algorithm finds a subset of vertices S for which `poly_fdst` returns a spanning tree with the largest number of full degree vertices.

Our algorithm follows a branching strategy and as a partial solution keeps a set of vertices S such that (V, E_S) is acyclic. The algorithm grows one component of the forest $(N[S], E_S)$ at a time. The vertices of this component are denoted by S_a, the active set. We denote $S \setminus S_a$ by S_b, the inactive set. The standard branching step chooses a vertex v that could be included in S_a and then recursively tries to find a solution by including v in S_a and not including v in S. But when v is not included in S, it cannot be removed from further consideration as cycles involving v might be created later on in (V, E_S) by adding neighbors of v to S. Hence we resort to a coloring scheme for the vertices, which can also be thought of as a partition of the vertex set of the input graph. At any point of the execution of the algorithm, the vertices are partitioned as below:

1. *Selected* $S = S_a \cup S_b$: The set of vertices which are decided to be of full degree. The set S_a corresponds to the *active* set of vertices which we use in the current stage of the algorithm. The set S_b is the *inactive* set of

vertices which were decided to be of full degree in an earlier stage of the algorithm.

2. *Discarded* D: The set of vertices which are not required to be of full degree.

3. *Postponed* P: The subset of vertices of degree 2 for which we leave to `poly_fdst` the decision which ones become full degree vertices.

4. *Undecided* U: The set of vertices which are not in S, D or P, that is those vertices for which the algorithm still needs to make a choice. So, $U = V \setminus (S \cup D \cup P)$.

Next we define a generalized form of the FDST problem based on the above partition of the vertex set. But before that we need the following definition.

Definition 1 *Given a vertex set $S \subseteq V$, we define the* partial spanning forest *of G induced by S as $T(S) = (N[S], E_S)$ where E_S is defined as in Equation (1).*

For our generalized problem, we denote by $G = (S_a, S_b, D, P, U, E)$ the graph (V, E) with vertex set $V = S_a \cup S_b \cup D \cup P \cup U$ partitioned as above.

> Generalized Full Degree Spanning Forest (GFDSF): Given an instance $G = (S_a, S_b, D, P, U, E)$ such that $T(S_a \cup S_b)$ is acyclic, $T(S_a)$ is connected, and no vertex of $T(S_b)$ is adjacent to a vertex in U, the objective is to find a spanning forest which maximizes the number of vertices of $U \cup P$ of full degree under the constraint that all the vertices in $S = S_a \cup S_b$ have full degree.

If we start with a graph G, an instance of FDST, with the vertex partition $S = D = P = \emptyset$ and $U = V$ then the problem we will have at every intermediate step of the recursive algorithm is GFDSF. We say that T is a *generalized full degree spanning tree/forest* if it maximizes the number of vertices of $U \cup P$ that have full degree under the constraint that all the vertices in S have full degree. We remark that vertices of D are allowed to have full degree in a valid solution to GFDSF, but the quality of a solution is only measured with respect to how many vertices of $U \cup P$ have full degree in addition to S. Also, note that a full degree spanning forest of a connected graph can easily be extended to a full degree spanning tree by adding some edges to connect the connected components of the forest and that a full degree spanning tree *is* a full degree spanning forest.

As suggested earlier our algorithm is based on branching and will have some reduction rules that can be applied in polynomial time, leading to a refined partitioning of the vertices. Before we come to the detailed description of the algorithm, we introduce a few more important definitions. For given sets S, D, P and U, we say that an edge is

(a) *unexplored* if one of its endpoints is in U and the other one in $U \cup D \cup P$,

(b) *forced* if at least one of its endpoints is in S, and

(c) *superfluous* if both its endpoints are in D.

The basic step of our algorithm chooses an undecided vertex $u \in U$ and considers two subcases that it solves recursively: either u is *selected*, that is u is moved from U to S_a, or u is *discarded*, that is u is moved from U to D. But the main idea is to choose a vertex in a way that the connectivity of $T(S_a)$ is maintained in both recursive calls. To do so we choose u from $U \cap N[N[S_a]]$. This brings us to the following definition.

Definition 2 *The vertices in $U \cap N[N[S_a]]$ are called* candidate *vertices.*

On the other hand, if S_a is not empty and there are no candidate vertices, then set $S_b := S_b \cup S_a$ and $S_a := \emptyset$. If, thereafter, U is not empty, then the algorithm starts to grow a new component of the spanning forest.

Now we are ready to describe the algorithm in details. We start with a procedure for reduction rules in the next subsection and prove their correctness.

3.1 Reduction Rules

Given an instance $G = (S_a, S_b, D, P, U, E)$ of GFDSF, a reduced instance of G is computed by the following procedure.

`Reduce`$(G = (S_a, S_b, D, P, U, E))$

R1 If there is a superfluous edge e, then return `Reduce`$((S_a, S_b, D, P, U, E \setminus \{e\}))$.

R2 If there is a vertex $u \in D \cup U$ such that $d(u) = 1$, then remove the unique edge e incident to u and return `Reduce`$((S_a, S_b, D, P, U, E \setminus \{e\}))$.

R3 If there is an undecided vertex $u \in U$ such that $T(S_a \cup \{u\})$ contains a cycle, then discard u, that is return `Reduce`$((S_a, S_b, D \cup \{u\}, P, U \setminus \{u\}, E))$.

R4 If there is a candidate vertex u that is adjacent to at most one vertex in $U \cup D \cup P$, then select u, and return `Reduce`$((S_a \cup \{u\}, S_b, D, P, U \setminus \{u\}, E))$.

R5 If $S_a = \emptyset$ and there exists a vertex $u \in U$ of degree 2, then select u and return `Reduce`$((S_a \cup \{u\}, S_b, D, P, U \setminus \{u\}, E))$.

R6 If there is a candidate vertex u of degree 2 that has a neighbor v in D, then put u into P, remove the edge uv and return `Reduce`$((S_a, S_b, D, P \cup \{u\}, U \setminus \{u\}, E \setminus \{\{u, v\}\}))$.

R7 If there is no candidate vertex and $U \neq \emptyset$ then return `Reduce`$((\emptyset, S_a \cup S_b, D, P, U, E))$.

Else return G

Now we argue about the correctness of the reduction rules. More precisely, we prove that there exists a spanning forest of G such that a maximum number of vertices preserve their degree and the partitioning of the vertices into the sets S, D, P and U of the graph resulting from a call to `Reduce`$(G = (S_a, S_b, D, P, U, E))$ is respected. Note that the reduction rules are applied in the order of their appearance.

The correctness of **R1** follows from the fact that discarded vertices are not required to have full degree.

For the correctness of reduction rule **R2**, consider a vertex $u \in D \cup U$ of degree 1 with unique neighbor w. Let $G' = (S_a, S_b, D, P, U, E \setminus \{uw\})$ be the graph resulting from the application of the reduction rule. Note that the edge uw is not part of any cycle and that a generalized full degree spanning forest of G can be obtained from a generalized full degree spanning forest of G' by adding the edge uw. As Algorithm `poly_fdst` adds edges to transform the obtained spanning forest into a spanning tree (`poly_fdst` works with the original input graph), the edge uw is added to the final solution.

For the correctness of reduction rule **R3**, it is enough to observe that if for a subset $S \subseteq V$, there exists a spanning forest such that all the vertices of S have full degree, then $T(S)$ is acyclic.

We prove the correctness of **R4** and **R5** by the following lemmata.

Lemma 3 *Let $G = (V, E)$ be a graph and T be a full degree spanning forest for G. If $v \in V$ is a vertex of degree $d_G(v) - 1$ in T, then there exists a full degree spanning forest T' such that v has degree $d_G(v)$ in T'.*

Proof: Let $u \in V$ be the neighbor of v such that uv is not an edge of T. Note that both u and v do not have full degree in T, are not adjacent in T, and belong to the same tree in T. The last assertion follows from the fact that if u and v belong to two different trees of T then one can add uv to T and obtain a forest T' that has a larger number of full degree vertices, contradicting that T is a full degree spanning forest. Now, adding the edge uv to T creates a unique cycle passing through u and v. We obtain the new forest T' by removing the other edge incident to u on the cycle, say uw, $w \neq v$. So, $T' = T \setminus \{uw\} \cup \{uv\}$. The number of full degree vertices in T' is at least as high as in T as v becomes a full degree vertex and at most one vertex, w, could lose the full degree property. ■

We also need a generalized version of Lemma 3.

Lemma 4 *Let $G = (S_a, S_b, D, P, U, E)$ be a graph and T be a generalized full degree spanning forest for G. Let $v \in U$ be a candidate vertex such that its neighbors in $D \cup P \cup U$ are not incident to a forced edge. If v has degree $d_G(v) - 1$ in T, then there exists a generalized full degree spanning forest T' such that v has full degree in T'.*

Proof: The proof is similar to the one of Lemma 3. The only difference is that we need to show that the vertices of $S = S_a \cup S_b$ remain of full degree

and for that we need to show that all the edges of $T(S)$ remain in T'. To this observe that all the edges incident to the neighbors of v in $D \cup P \cup U$ in T do not belong to edges of $T(S)$, that is they are not forced edges. So if uv is the unique edge incident to v missing in T then we can add uv to T and remove the other non-forced edge on u from the unique cycle in $T \cup \{uv\}$ and get the desired T'. ∎

Observe that every vertex of P has degree 0. Therefore, all vertices in U are in different connected components than the vertices in S_b.

Now consider reduction rule **R4**. If u is a candidate vertex with unique neighbor w in $D \cup P \cup U$ then (a) $u \in N(S_a)$ and (b) all the edges incident to w are not forced, otherwise reduction rule **R2** or **R3** would have applied. Now the correctness of the reduction rule follows from Lemma 4.

To prove the correctness of reduction rule **R5**, we need to show that there exists a generalized full degree spanning forest where u has full degree. Suppose not and let T be any generalized full degree spanning forest of G. Without loss of generality, suppose that u has degree 1 in T (if u is an isolated vertex in T, then add one edge incident to u to T; this does not create any cycle in T and does not decrease the number of vertices of full degree in T). Let v be the unique neighbor of u in T. But since $S_a = \emptyset$, there are no forced edges incident to neighbors of u and we can apply Lemma 4 again and conclude.

Reduction rule **R6** postpones the decision for certain degree 2 vertices. The edge uv may be deleted as the decision on this edge is made by the polynomial time procedure `poly_fdst`. Note that u becomes of degree 1, which triggers reduction rule **R2**, removing the other edge incident on u as there exists a generalized full degree spanning forest containing this edge.

Reduction rule **R7** only makes the active set inactive in order to start a new component of the spanning forest as the current component cannot be grown any further and is thus correct.

This finishes the correctness proof of the reduction rules. Before we go into the details of the algorithm we would like to point out that all the reduction rules preserve the connectivity of $T(S_a)$.

3.2 Algorithm

In this section we describe the algorithm in detail. Given an instance $G = (S_a, S_b, D, P, U, E)$ of GDPST, the algorithm recursively solves the problem by choosing a candidate vertex $u \in U$ and including u in S_a or in D and then returning as solution the one with a maximum number of full degree vertices. The algorithm has various cases based on the number of unexplored edges incident to u.

Algorithm `fdst`(G), described below, returns a generalized full degree spanning tree of G. In particular, it searches for a set S^*, such that $S_a \cup S_b = S \subseteq S^* \subseteq S \cup U$, corresponding to (a subset of the) full degree vertices in a generalized full degree spanning forest. Letting G_o denote a copy of the original input graph (the algorithm may remove edges from its working copy of the

graph), which will be known to the algorithm as a global variable, the procedure `poly_fdst` returns then a spanning tree of G_o in which all the vertices of S^* and a maximum number of vertices of P have full degree.

We describe the procedure `poly_fdst` now in more details. Initially, all edges are unweighted. Assign weight 1 to each edge incident to a vertex in S^* and to each edge that has been removed by reduction rule **R2**. To each unweighted edge incident to a vertex in P, assign weight 2. To each remaining unweighted edge, assign weight 3. Then compute a minimum weight spanning tree T of this weighted graph using the algorithm of Kruskal [20]. This algorithm first adds to T all the edges of $T(S)$ ($T(S)$ is acyclic) and all the edges removed by **R2**. After having processed the edges with weight 1, Kruskal's algorithm processes edges of weight 2. Each vertex of P has exactly one incident edge with weight two (see reduction rule **R6**), no incident edge with weight 3, and all its incident edges with weight 1 have been removed by **R2** (as **R4** selects vertices of degree 2 that are in $N(S)$). Edges of weight two are used to connect connected components of the forest consisting of edges of weight one. For each edge of weight two that is added to T, one vertex of P gets full degree. Finally, if T is still not connected after having added as many weight-2 edges as possible, edges of weight 3 (incident to discarded vertices only) are used to connect the remaining connected components of T.

The description of the algorithm consists of the application of the reduction rules and a sequence of cases. A case consists of a condition (first sentence) and a statement to be executed if the condition holds. The first case which applies is used in the algorithm. Thus, inside a given case, the conditions of all previous cases are assumed to be false.

Algorithm `fdst`$(G = (S_a, S_b, D, P, U, E))$

Replace G by `Reduce`(G).

Case 1: U is a set of isolated vertices. Select all vertices in U and return `poly_fdst`$(G_o, (S_a, S_b \cup U, D, P, \emptyset, E))$.

Case 2: $S_a = \emptyset$. Choose a vertex $u \in U$ of degree at least 3. Return the best solution among `fdst`$((S_a \cup \{u\}, S_b, D, P, U \setminus \{u\}, E))$ and `fdst`$((S_a, S_b, D \cup \{u\}, P, U \setminus \{u\}, E))$.

Case 3: There is a candidate vertex u with at least 3 unexplored incident edges. Make two recursive calls: `fdst`$((S_a \cup \{u\}, S_b, D, P, U \setminus \{u\}, E))$ and `fdst`$((S_a, S_b, D \cup \{u\}, P, U \setminus \{u\}, E))$, and return the best solution.

Case 4: There is a candidate vertex $u \in N(S_a)$ with at least one neighbor v in U and exactly two unexplored incident edges. Make two recursive calls: `fdst`$((S_a \cup \{u\}, S_b, D, P, U \setminus \{u\}, E))$ and `fdst`$((S_a, S_b, D \cup \{u, v\}, P, U \setminus \{u, v\}, E))$, and return the best solution.

From now on let v_1 and v_2 denote the discarded neighbors of a candidate vertex u (see Figure 1).

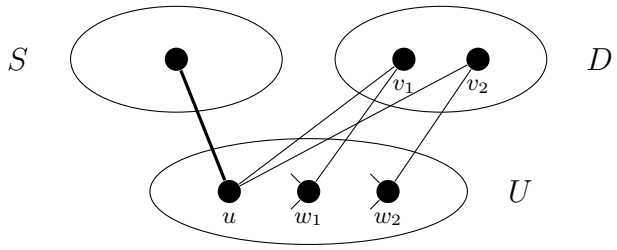

Figure 1: Illustration of Case 6. Cases 5, 7 and 8 are similar.

Case 5: Either v_1 and v_2 have a common neighbor $x \neq u$; or
v_1 (or v_2) has a neighbor $x \neq u$ that is a candidate vertex; or
v_1 (or v_2) has a neighbor x of degree 2.
Return the best solution among $\texttt{fdst}((S_a \cup \{u\}, S_b, D, P, U \setminus \{u\}, E))$ and $\texttt{fdst}((S_a, S_b, D \cup \{u\}, P, U \setminus \{u\}, E))$.

Case 6: Both v_1 and v_2 have degree 2. Let w_1 and w_2 ($w_1 \neq w_2$) be the other (different from u) neighbors of v_1 and v_2 in U, respectively. Make recursive calls as usual, but also explore all the possibilities for w_1 and w_2 if $u \in S$. When u is in S, recurse on all possible ways one can add a subset of $A = \{w_1, w_2\}$ to S. That is make recursive calls $\texttt{fdst}((S, D \cup \{u\}, U \setminus \{u\}, E))$ and $\texttt{fdst}((S \cup \{u\} \cup X, D \cup (A - X), U \setminus (\{u\} \cup A), E))$ for each independent set $X \subseteq A$, and return the best solution.

Case 7: One of $\{v_1, v_2\}$ has degree ≥ 3. Let $\{u, w_1, w_2, w_3\} \subseteq N(\{v_1, v_2\})$ and let $A = \{w_1, w_2, w_3\}$. Make recursive calls $\texttt{fdst}((S_a, S_b, D \cup \{u\}, P, U \setminus \{u\}, E))$ and $\texttt{fdst}((S_a \cup \{u\} \cup X, S_b, D \cup (A - X), P, U \setminus (\{u\} \cup A), E))$ for each independent set $X \subseteq A$, and return the best solution.

Case 8: Both v_1 and v_2 have degree ≥ 3. Let $\{u, w_1, w_2, w_3, w_4\} \subseteq N(\{v_1, v_2\})$ and let $A = \{w_1, w_2, w_3, w_4\}$. Make recursive calls $\texttt{fdst}((S_a, S_b, D \cup \{u\}, P, U \setminus \{u\}, E))$ and $\texttt{fdst}((S_a \cup \{u\} \cup X, S_b, D \cup (A - X), P, U \setminus (\{u\} \cup A), E))$ for each independent set $X \subseteq A$, and return the best solution.

---

4 Correctness and Time Complexity of the Algorithm

We prove the correctness and the time complexity of Algorithm `fdst` in the following theorem.

Theorem 5 *Given an input graph $G = (S_a, S_b, D, P, U, E)$ on n vertices such that $T(S)$ is acyclic, $S = S_a \cup S_b$, and $T(S_a)$ is connected, Algorithm `fdst` solves* GFDSF *in time $\mathcal{O}(1.9465^n)$.*

Proof: The correctness of the reduction rules is described in Section 3.1.

To see that to every instance the algorithm applies some reduction or branching rule, it is sufficient to note that if no reduction rule, nor Cases 1–4 apply, then every candidate vertex is in $N(S_a)$ and has exactly two neighbors in D: candidate vertices with at least 3 unexplored incident edges are handled by Case 3, candidate vertices with at most one unexplored incident edge are taken care of by reduction rule **R4**, and candidate vertices with exactly two unexplored incident edges satisfy the condition of reduction rule **R6** or Case 4, or belong to $N(S_a)$ and have two neighbors in D.

The correctness of **Case 1** follows, as every isolated vertex belonging to U has full degree in any spanning forest. The remaining cases, except **Case 4**, of Algorithm `fdst` are branching steps where the algorithm chooses a vertex $u \in U$ and tries both possibilities: $u \in S_a$ or $u \in D$. Sometimes the algorithm branches further by looking at the local neighborhood of u and trying all possible ways that a subset of the neighbors of u can be added to either S_a or D. Since all possibilities are tried to add vertices of U to D or S_a in **Cases 3** and **5** to **8**, these cases are correct and do not need any further justifications. The correctness of **Case 4** requires special attention. Here we use the fact that there exists a generalized full degree spanning forest, such that either $u \in S$ or u and its neighbor $v \in U$ are in D. We prove the correctness of this assertion by contradiction. Suppose v has full degree and u does not have full degree in all the generalized full degree spanning forests of G. But since $u \in N(S_a)$ and all the neighbors of u in $D \cup U$ do not have any incident forced edges, we can use Lemma 4 to get a generalized full degree spanning forest in which u has full degree.

Now we move on to the time complexity of the algorithm. The measure of an instance is generally chosen as a function of its structure, like the number of vertices, edges or other graph parameters, which change during the recursive steps of the algorithm. In our algorithm, this change is reflected when vertices are moved to either S, D or P from U. The second observation is that any spanning tree on n vertices has at most $n-1$ edges and hence when we select a vertex in S we increase the number of edges in $T(S)$ and decrease the number of edges we can add to $T(S)$. Finally we also gain when the degree of a vertex becomes two because reduction rules apply as soon as the degree 2 vertex is in $N(S_a)$ (**R4**) or in $N(N(S_a))$ and has a neighbor in D. (Candidate vertices in $N(N(S_a))$ of degree 2 that are adjacent only to vertices in U fall into one of these two categories once their neighbor in $N(S_a)$ is put in either S_a or D.) Our measure is precisely a function of these three parameters and is defined as follows:

$$\mu(G) = \eta|U_2| + \beta|U_{\geq 3}| + \alpha m', \tag{2}$$

where U_2 is the subset of undecided vertices of degree 2, $U_{\geq 3}$ is the subset of undecided vertices of degree at least 3, $m' = n - 1 - |E(T(S))|$ is the number of edges that can be added to the spanning tree and $\eta = 0.5, \beta = 0.722$ and $\alpha = 0.23887$ are numerically obtained constants. These constants have been

optimally computed by a convex optimization program [16] (see also [14]). We write μ instead of $\mu(G)$ if G is clear from the context. We prove that the problem can be solved for an instance of size μ in time $\mathcal{O}(2^{\mu})$. As $\mu \leq 0.96087n$, the final running time of the algorithm will be $2^{0.96087n} \cdot n^{\mathcal{O}(1)} = 1.9464..^{n} \cdot n^{\mathcal{O}(1)} = \mathcal{O}(1.9465^{n})$. Denote by $P[\mu]$ the maximum number of times the algorithm is called recursively on a problem of size μ (i. e. the number of leaves in the search tree). Then the running time $T(\mu)$ of the algorithm is bounded by $P[\mu] \cdot n^{\mathcal{O}(1)}$ because in any node of the search tree, the algorithm executes only a polynomial number of steps. We use induction on μ to prove that $P[\mu] \leq 2^{\mu}$. Then $T(\mu) = 2^{\mu} \cdot n^{\mathcal{O}(1)} = \mathcal{O}(1.9465^{n})$. Clearly, $P[0] = 1$. Suppose that $P[k] \leq 2^{k}$ for every $k < \mu$ and consider a problem of size μ.

Case 2: In this case, the number of vertices in $U_{\geq 3}$ decreases by one in both recursive calls and the number of edges in $T(S)$ increases by at least 3 in the first recursive call. Thus,

$$\begin{aligned} P[\mu] &\leq P[\mu - \beta - 3\alpha] + P[\mu - \beta] \\ &\leq 2^{\mu-\beta-3\alpha} + 2^{\mu-\beta} \leq 2^{\mu}. \end{aligned}$$

Case 3: This case has the same recurrence as **Case 2** as the number of vertices in $U_{\geq 3}$ decreases by one in both recursive calls and the number of edges in $T(S)$ increases by at least 3 in the first recursive call.

Case 4: When the algorithm adds u to S, the number of vertices in $U_{\geq 3}$ decreases by one and the number of edges in $T(S)$ increases by 2 while in the other case, $|U_{\geq 3}|$ decreases by two or $|U_2|$ and $|U_{\geq 3}|$ both decrease by one. So we get:

$$\begin{aligned} P[\mu] &\leq P[\mu - \beta - 2\alpha] + P[\mu - 2\beta], \text{ and} \\ P[\mu] &\leq P[\mu - \beta - 2\alpha] + P[\mu - \eta - \beta]. \end{aligned}$$

Case 5: When the algorithm adds u to S, reduction rule **R3** or **R6** applies to x. We obtain the following recurrences, based on the degree of x:

$$\begin{aligned} P[\mu] &\leq P[\mu - \eta - \beta - 2\alpha] + P[\mu - \beta], \text{ and} \\ P[\mu] &\leq P[\mu - 2\beta - 2\alpha] + P[\mu - \beta]. \end{aligned}$$

Case 6: In this case we distinguish two subcases based on the degrees of w_1 and w_2. Our first subcase is when either w_1 or w_2 has degree 3 and the other subcase is when both w_1 and w_2 have degree at least 4. (Note that because of **Case 4**, v_1 and v_2 do not have a common neighbor and do not have a neighbor of degree 2.) Suppose w_1 has degree 3. When the algorithm adds u to D, the edges uv_1 and uv_2 are removed (**R1**), the degree of v_1 is reduced to 1 and then reduction rule **R2** is applied and decreases the degree of w_1 to 2. Thus, μ decreases by $2\beta - \eta$ in this

subcase. The analysis of the remaining branches is standard and we get the following recurrence:

$$P[\mu] \leq P[\mu-3\beta-2\alpha]+2P[\mu-3\beta-5\alpha]+P[\mu-3\beta-8\alpha]+P[\mu-2\beta+\eta].$$

For the other subcase we get the following recurrence:

$$P[\mu] \leq P[\mu-3\beta-2\alpha]+2P[\mu-3\beta-6\alpha]+P[\mu-3\beta-10\alpha]+P[\mu-\beta].$$

Case 7: This case is similar to **Case 6**. Without loss of generality, suppose v_1 has degree 2 and has w_1 as neighbor. If w_1 has degree 3, then it becomes of degree 2 when u is discarded. Thus, we obtain the recurrence

$$\begin{aligned} P[\mu] \quad \leq \quad & P[\mu-4\beta-2\alpha]+3P[\mu-4\beta-5\alpha]+3P[\mu-4\beta-8\alpha]+ \\ & P[\mu-4\beta-11\alpha]+P[\mu-2\beta+\eta]. \end{aligned}$$

On the other hand, if w_1 has degree at least 4, at least one more edge is added to the spanning forest each time w_1 is put into S_a and we obtain the recurrence

$$\begin{aligned} P[\mu] \quad \leq \quad & P[\mu-4\beta-2\alpha]+2P[\mu-4\beta-5\alpha]+P[\mu-4\beta-6\alpha]+ \\ & P[\mu-4\beta-8\alpha]+2P[\mu-4\beta-9\alpha]+P[\mu-4\beta-12\alpha]+ \\ & P[\mu-\beta]. \end{aligned}$$

Case 8: This case easily gives the recurrence

$$\begin{aligned} P[\mu] \quad \leq \quad & P[\mu-5\beta-2\alpha]+4P[\mu-5\beta-5\alpha]+6P[\mu-5\beta-8\alpha]+ \\ & 4P[\mu-5\beta-11\alpha]+P[\mu-5\beta-14\alpha]+P[\mu-\beta]. \end{aligned}$$

In each of these recurrences, $P[\mu] \leq 2^{\mu}$ which completes the proof of the theorem. ∎

The bottleneck of the analysis is the recurrence in **Case 8**. Therefore, an improvement of this case would lead to a faster algorithm.

We provide a Python program in the appendix to verify that the values for η, β and α satisfy all the recurrences.

5 Conclusion

In this paper we have given an exact algorithm for the Full Degree Spanning Tree problem. The most important feature of our algorithm is the way we exploit connectivity arguments to reduce the size of the graph in the recursive steps of the algorithm. We think that this idea of combining connectivity while developing Branch & Reduce algorithms could be useful for various other non-local problems and in particular for other NP-complete variants of the Spanning

TREE problem.[1] Although the theoretical bound we obtained for our algorithm seems to be only slightly better than a brute-force enumeration algorithm, practice shows that Branch & Reduce algorithms perform usually better than the running time proved by a worst case analysis of the algorithm. Therefore we believe that this algorithm, combined with good heuristics, could be useful in practical applications.

In a preliminary version of this paper [15], we mentioned the MINIMUM MAXIMUM DEGREE SPANNING TREE problem, where, given an input graph G, the objective is to find a spanning tree T of G such that the maximum degree of T is minimized. We asked whether there exists a $2^n n^{\mathcal{O}(1)}$ time algorithm for this problem. This question has been answered by Nederlof [25] who designed a polynomial-space $\mathcal{O}(2^n)$ algorithm using Möbius inversion.

Acknowledgments

We thank Henning Fernau, Fedor V. Fomin and Daniel Raible for useful discussions on a preliminary version of the presented algorithm.

References

[1] R. Bhatia, S. Khuller, R. Pless, Y. J. Sussmann, The full degree spanning tree problem, Networks 36(4): 203–209, (2000).

[2] A. Björklund, T. Husfeldt, M. Koivisto, Set partitioning via inclusion-exclusion, SIAM J. Comput. 39(2): 546-563, (2009).

[3] A. Björklund, T. Husfeldt, P. Kaski and M. Koivisto, Fourier meets Möbius: Fast subset convolution, in the proceedings of STOC 2007, 67–74, (2007).

[4] H. Broersma, O. R. Koppius, H. Tuinstra, A. Huck, T. Kloks, D. Kratsch, H. Müller, Degree-preserving trees, Networks 35(1): 26–39, (2000).

[5] N. Christofides, An algorithm for the chromatic number of a graph, Computer Journal 14(1): 38–39, (1971).

[6] P. Damaschke, Degree-preserving spanning trees in small-degree graphs, Discrete Mathematics 222(1–3): 51–60, (2000).

[7] H. Fernau, S. Gaspers, D. Raible, Exact and Parameterized Algorithms for Max Internal Spanning Tree, in the proceedings of WG 2009, LNCS 5911, 100–111, (2009).

[8] F. V. Fomin, S. Gaspers, A. V. Pyatkin, Finding a minimum feedback vertex set in time $O(1.7548^n)$, in the proceedings of IWPEC 2006, LNCS 4169, 184–191, (2006).

[1] Indeed, connectivity arguments used in conjunction with Measure & Conquer have recently been used in [7] to design an algorithm finding a spanning tree with a maximum number of internal vertices in graphs with maximum degree at most 3.

[9] F. V. Fomin, S. Gaspers, A. V. Pyatkin, I. Razgon, On the minimum feedback vertex set problem: Exact and enumeration algorithms, Algorithmica 52(2): 293–307, (2008).

[10] F. V. Fomin, F. Grandoni, D. Kratsch, A measure & conquer approach for the analysis of exact algorithms, Journal of the ACM 56(5), Article 25: 1–32, (2009).

[11] F. V. Fomin, F. Grandoni, D. Kratsch, Solving connected dominating set faster than 2^n, in the proceedings of FSTTCS 2006, LNCS 4337, 152–163, (2006).

[12] F. V. Fomin, F. Grandoni, D. Kratsch, Solving connected dominating set faster than 2^n, Algorithmica 52(2): 153–166, (2008).

[13] F. V. Fomin, F. Grandoni, D. Kratsch, Some new techniques in design and analysis of exact (exponential) algorithms, Bulletin of the EATCS 87: 47–77, (2005).

[14] S. Gaspers, Exponential Time Algorithms: Structures, Measures, and Bounds, PhD thesis, University of Bergen, Norway (2008).

[15] S. Gaspers, S. Saurabh, A. A. Stepanov, A moderately exponential time algorithm for full degree spanning tree, in the proceedings of TAMC 2008, LNCS 4978, 479–489, (2008).

[16] S. Gaspers, G. B. Sorkin, A universally fastest algorithm for Max 2-Sat, Max 2-CSP, and everything in between, in the proceedings of SODA 2009, SIAM, 606–615, (2009).

[17] J. Guo, R. Niedermeier, S. Wernicke, Fixed-parameter tractability results for full-degree spanning tree and its dual, Networks 56(2): 116–130, (2010).

[18] J. Håstad, Clique is Hard to Approximate within n to the power 1-epsilon, Acta Mathematica 182: 105-142, (1999).

[19] S. Khuller, R. Bhatia, R. Pless, On local search and placement of meters in networks, SIAM Journal of Computing 32(2): 470–487, (2003).

[20] J. B. Kruskal, On the shortest spanning subtree and the traveling salesman problem, in the proceedings of the American Mathematical Society 7: 48–50, (1956).

[21] E. L. Lawler, A note on the complexity of the chromatic number problem, Information Processing Letters 5(3): 66–67, (1976).

[22] M. Lewinter, Interpolation theorem for the number of degree-preserving vertices of spanning trees, IEEE Transaction Circ. Syst. 34: 205, (1987).

[23] Ching-Chi Lin, Gerard J. Chang, Gen-Huey Chen, The degree-preserving spanning tree problem in strongly chordal and directed path graphs, Networks, to appear (available online).

[24] D. Lokshtanov, V. Raman, S. Saurabh, S. Sikdar, On the directed degree-preserving spanning tree problem, in the proceedings of IWPEC 2009, LNCS 5917, 276–287, (2009).

[25] J. Nederlof, Fast polynomial-space algorithms using Möbius inversion: Improving on steiner tree and related problems, in the proceedings of ICALP 2009, LNCS 5555, 713–725, (2009).

[26] L. E. Ormsbee, Implicit network calibration, Journal of Water Resources, Planning and Management, 115(2): 243–257, (1989).

[27] L. E. Ormsbee and D. J. Wood, Explicit pipe network calibration, Journal of Water Resources, Planning and Management, 112(2): 166–182, (1986).

[28] I. W. M. Pothof and J. Schut, Graph-theoretic approach to identifiability in a water distribution network, Memorandum 1283, Universiteit Twente, (1995).

[29] I. Razgon, Exact computation of maximum induced forest, in the proceedings of SWAT 2006, LNCS 4059, 160–171, (2006).

[30] S. Saurabh, Exact Algorithms for Optimization and Parameterized Versions of some graph theoretic problems, PhD thesis, Homi Bhabha National Institute, India (2008).

[31] U. Schöning, Algorithmics in exponential time, in the proceedings of STACS 2005, LNCS 3404, 36–43, (2005).

[32] Alexey A. Stepanov, Exact algorithms for hard listing, counting and decision problems, PhD thesis, University of Bergen, Norway (2008).

[33] J. M. M. van Rooij, J. Nederlof, T. C. van Dijk, Inclusion/exclusion meets measure and conquer, in the proceedings of ESA 2009, LNCS 5757, 554–565, (2009).

[34] G. J. Woeginger, Exact Algorithms for NP-Hard Problems: A Survey, in the proceedings of AUSSOIS 2001, 185-208, (2001).

[35] D. Zuckerman, Linear Degree Extractors and the Inapproximability of Max Clique and Chromatic Number, Theory of Computing 3(1): 103–128, (2007).

Appendix: Program to Check the Analysis

```
# weights
alpha = 0.23887
beta = 0.722
eta = 0.5

# case 2
print 2**(-beta-3*alpha) + 2**(-beta) < 1
# case 3
# case 4
print 2**(-beta-2*alpha) + 2**(-2*beta) < 1
print 2**(-beta-2*alpha) + 2**(-eta-beta) < 1
# case 5
print 2**(-eta-beta-2*alpha) + 2**(-beta) < 1
print 2**(-2*beta-2*alpha) + 2**(-beta) < 1
# case 6
print 2**(-3*beta-2*alpha) + 2*2**(-3*beta-5*alpha)
    + 2**(-3*beta-8*alpha) + 2**(-2*beta+eta) < 1
print 2**(-3*beta-2*alpha) + 2*2**(-3*beta-6*alpha)
    + 2**(-3*beta-10*alpha) + 2**(-beta) < 1
# case 7
print 2**(-4*beta-2*alpha) + 3*2**(-4*beta-5*alpha)
    + 3*2**(-4*beta-8*alpha) + 2**(-4*beta-11*alpha)
    + 2**(-2*beta+eta) < 1
print 2**(-4*beta-2*alpha) + 2*2**(-4*beta-5*alpha)
    + 2**(-4*beta-6*alpha) + 2**(-4*beta-8*alpha)
    + 2*2**(-4**beta-9*alpha) + 2**(-4*beta-12*alpha)
    + 2**(-beta) < 1
# case 8
print 2**(-5*beta-2*alpha) + 4*2**(-5*beta-5*alpha)
    + 6*2**(-5*beta-8*alpha) + 4*2**(-5*beta-11*alpha)
    + 2**(-5*beta-14*alpha) + 2**(-beta) < 1
```