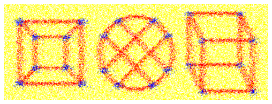
Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 17, no. 6, pp. 615–628 (2013)
DOI: 10.7155/jgaa.00308

An FPT algorithm for Tree Deletion Set

Venkatesh Raman[1] *Saket Saurabh*[1] *Ondřej Suchý*[2]

[1]The Institute of Mathematical Sciences, Chennai, India
[2]Faculty of Information Technology, Czech Technical University
Prague, Czech Republic

Abstract

We give a $5^k n^{O(1)}$ time fixed-parameter algorithm for determining whether a given undirected graph on n vertices has a subset of at most k vertices whose deletion results in a tree. Such a subset is a restricted form of a feedback vertex set. While parameterized complexity of feedback vertex set problem and several of its variations have been well studied, to the best of our knowledge, this is the first fixed-parameter algorithm for this version of feedback vertex set.

Submitted:	Reviewed:	Revised:	Accepted:	Final:
May 2013	September 2013	October 2013	October 2013	October 2013

Published:
November 2013

Article type:
Regular paper

Communicated by:
S. K. Ghosh

A preliminary version of this work appeared in the proceedings of WALCOM 2013 [21].
Part of the work of the third author was done while with the Universität des Saarlandes, Saarbrücken, supported by the DFG Cluster of Excellence on Multimodal Computing and Interaction (MMCI) and DFG project DARE (GU 1023/1-2), and while visiting IMSc Chennai, supported by the Indo-German Max Planck Center for Computer Science (IMPECS).
E-mail addresses: vraman@imsc.res.in (Venkatesh Raman) saket@imsc.res.in (Saket Saurabh) ondrej.suchy@fit.cvut.cz (Ondřej Suchý)

1 Introduction

The goal of parameterized complexity is to find ways of solving NP-hard problems more efficiently than brute force: our aim is to restrict the combinatorial explosion to a parameter that is hopefully much smaller than the input size. Formally, a *parameterization* of a problem is assigning an integer k to each input instance and we say that a parameterized problem is *fixed-parameter tractable* (FPT) if there is an algorithm that solves the problem in $f(k) \cdot n^{O(1)}$ time, where n is the size of the input and f is an arbitrary computable function depending on the parameter k only. There is a long list of NP-hard problems that are FPT under various parameterizations: finding a vertex cover of size k, finding a cycle of length k, finding a maximum independent set in a graph of treewidth at most k, etc. For more background, the reader is referred to the monographs [6, 7, 20].

One of the most well studied directions in parameterized complexity is to "delete vertices of the input graph such that the resulting graph satisfies some interesting properties". More precisely, a natural optimization problem associated with a graph class $\mathcal{G}$ is the following: given a graph G, what is the minimum number of vertices to be deleted from G to obtain a graph in $\mathcal{G}$? For example, when $\mathcal{G}$ is the class of empty graphs, forests or bipartite graphs, the corresponding problems are Vertex Cover, Feedback Vertex Set and Odd Cycle Transversal, respectively. In the parameterized setting, a natural parameter for vertex-deletion problems is the solution size, that is, the number of vertices to be deleted so that the resulting graph belongs to the given graph class. This line of research has been at the forefront of research in parameterized complexity and various new results have been obtained in the last few years. For examples, an improved algorithm for Odd Cycle Transversal [14, 19], meta theorems for class of deletion problems [9, 13], Proper Interval Vertex Deletion [23], Directed/Undirected Subset Feedback Vertex Set [4, 5].

In this paper we consider the following variant of the classical Feedback Vertex Set problem in the realm of parameterized complexity:

Weighted Tree Deletion Set (WTDS)	
Input:	An undirected graph $G = (V, E)$, a weight function $w : V \to \mathbb{N}^+$ on vertices, and an integer $k \in \mathbb{N}$.
Parameter:	k
Question:	Is there a set $S \subseteq V$ of total weight $\sum_{v \in S} w(v)$ at most k, such that $G[V \setminus S]$ is a tree?

If $w(v) = 1$ for every $v \in V$, then we speak simply about Tree Deletion Set (TDS). We also refer the subset S as a *tree deletion set*.

TDS is a special case of WTDS, but on the other hand, if $k = n^{O(1)}$, where $n = |V|$, then WTDS is polynomial time reducible to TDS, by adding to each vertex v of the graph $min\{k, w(v) - 1\}$ pendant vertices. Clearly the resulting unweighted graph has a TDS of size at most k if and only if the original graph has a WTDS of weight at most k. This is because if an original vertex

is in the tree deletion set, then all the pendant vertices adjacent to it must also be in that set.

If we simply want to find a subset S of vertices such that $G[V \setminus S]$ is a forest, then S is a feedback vertex set. Finding a (size at most k or minimum) feedback vertex set is a well known NP-complete problem and has been well studied in the paradigms of parameterized complexity [2, 3, 22], approximation [1] and exact algorithms [8]. As a tree deletion set is also a feedback vertex set, it is clear that the size of the minimum tree deletion set is at least the size of the minimum feedback vertex set. However the minimum tree deletion set can be arbitrarily large compared to the size of the minimum feedback vertex set. Consider a graph which simply has a cycle on three vertices, with each vertex attached to a large number of pendant vertices. Any of the vertices of the cycle forms a feedback vertex set (of size 1), but the minimum tree deletion set must contain all the pendant vertices attached to that vertex. Furthermore, standard preprocessing rules like deleting degree one vertices and 'short circuiting' degree two vertices no longer work for tree deletion sets. We would like to point out that Tree Deletion Set has been considered before in the realm of approximation algorithm and has been shown to be hard to approximate within $O(n^{1-\epsilon})$ for any $\epsilon > 0$ unless P=NP [24]. This is in sharp contrast to the fact that Feedback Vertex Set has a factor 2-approximation algorithm [1].

Variations of Feedback Vertex Set, Dominating Set, and Vertex Cover when the solution S is required to induce an independent set or a connected graph have also been well studied [15, 16, 17, 18]. To the best of our knowledge, this is the first paper that studies the variation of a problem where the demand of connectivity is on $G[V \setminus S]$ rather than the solution. That is, in our problem we want $G[V \setminus S]$ to induce a connected graph (i.e., it is a tree).

We first present a simple proof of the NP-completeness of the problem, and then show that it is fixed-parameter tractable when parameterized by k, the solution size by giving an $O^*(5^k)$ time[1] fixed-parameter tractable algorithm. This is in contrast to, and comes reasonably close to the $O^*(3.83^k)$ bound known for the general feedback vertex set problem [2].

The next section provides a simple proof of the problem being NP-complete. Section 3 is the main section that gives the fixed-parameter algorithm for the problem. Finally in Section 4, we conclude with open problems.

2 NP-completeness

It follows from the general results of Yannakakis [24], that Tree Deletion Set is NP-complete. As Yannakakis's result is general, the proof is a bit involved. To make the paper self-contained, we present a short and simple proof of the NP-completeness.

Proposition 1 Tree Deletion Set *is* NP*-complete.*

[1] O^* notation ignores polynomial factors

Proof: The problem is obviously in NP. For the hardness we reduce Vertex Cover (VC), which is well known to be NP-complete [11]. An instance of VC consists of a graph G and a positive integer k and the question is whether there is a set S of at most k vertices (*vertex cover*) such that $G \setminus S$ contains no edges. Given an instance (G, k) of VC we obtain an equivalent instance (G', k) of TDS as follows. G' is obtained from G by introducing a new universal vertex u (i.e. u is adjacent to all vertices of G) and attaching $k+1$ new pendant vertices to it. Now if S is a vertex cover in G, then $G \setminus S$ is a star. On the other hand, if S is a tree deletion set of size at most k in G', then $u \notin S$, as otherwise there would be at least two of the newly added pendant vertices left in $G \setminus S$ and they would become disconnected. But then, as u is adjacent to all vertices of G, there must be no edge in $G \setminus S$ in order for $G' \setminus S$ to be a tree, which implies that S is a vertex cover for G. □

3 FPT Algorithm

The main result of this section is the following:

Theorem 1 Weighted Tree Deletion Set *can be solved in time* $O^*(5^k)$.

The rest of this section is devoted to the proof of this theorem.

3.1 Reduction Rules

We begin with some reduction rules which simplify the input instance. These rules modify the graph G, the weight function w, and the parameter k. For the purpose of the analysis, we denote the original value of the parameter k given on input by k_0. We say that a reduction rule is safe if the instance obtained by application of the rule is a yes-instance if and only if the original instance was.

The following two rules formalize obvious constraints to the solvability of the instance.

Reduction Rule 1 *If* $k < 0$*, then answer NO.*

Reduction Rule 2 *Let* N' *be the set of vertices which have weight more than* k*. If* $G[N']$ *contains a cycle, then answer NO.*

Lemma 1 *Reduction Rule 2 is safe.*

Proof: As no vertex of weight more than k can be included in any set of total weight at most k, no set of total weight at most k forms a tree deletion set. □

While the structure of the vertices of weight more than k is fixed, the following rule helps to simplify the neighborhood of such vertices.

Reduction Rule 3 *Let* N' *be the set of vertices which have weight more than* k*. If there is a vertex* v *in* $V(G) \setminus N'$ *which has two neighbors in the same connected component of* $G[N']$ *then delete* v *and decrease* k *by* $w(v)$*.*

Lemma 2 *Reduction Rule 3 is safe.*

Proof: The vertex v must be included in any tree deletion set of total weight at most k, as otherwise it would form a cycle together with the vertices in a connected component of N'. □

The following rule helps us to deal with isolated vertices and the case when the graph is disconnected.

Reduction Rule 4 *If the input graph is disconnected, then delete all vertices in connected components of weight less than* $\left(\sum_{v\in V} w(v)\right) - k$ *and decrease* k *by the weight of the deleted vertices.*

Lemma 3 *Reduction Rule 4 is safe.*

Proof: If the vertices of some connected component of total weight less than $\left(\sum_{v\in V} w(v)\right) - k$ were not taken into the constructed tree deletion set, then all vertices outside the connected component have to be taken, as the resulting graph must have only one component. But this would mean that the constructed tree deletion set would contain vertices of total weight more than $\left(\sum_{v\in V} w(v)\right) - \left[\left(\sum_{v\in V} w(v)\right) - k\right] = k$ — a contradiction. □

Remark 1 *If* $\left(\sum_{v\in V} w(v)\right) > 2k$ *or there is a vertex of weight more than* k*, then after the application of Reduction Rule 4 the graph has at most one connected component.*

The following rule deals with vertices of degree 1 in the graph.

Reduction Rule 5 *If* v *is of degree 1 and* u *is its only neighbor, then delete* v *and set* $w(u) = w(u) + w(v)$*.*

Lemma 4 *Reduction Rule 5 is safe.*

Proof: Let G, w, k be the instance before the application of the rule and G', w', k the instance after the application of the rule. Let us first assume, that S is a tree deletion set in G with $w(S) \leq k$. If S does not contain u, then $S \setminus \{v\}$ is also a tree deletion set for G of lower total weight and it is also a tree deletion set in G' of the same weight. If S contains u, but not v, then v is the only vertex not in S, $S \setminus \{u\}$ is also a tree deletion set for G of lower total weight and it is also a tree deletion set in G' of the same weight. Finally, if S contains both u and v, then $S \setminus \{v\}$ is a tree deletion set in G' of the same weight.

Assume now that S' is a tree deletion set in G'. If S' does not contain u, then S' is also a tree deletion set in G of the same weight. If S' contains u, then $S' \cup \{v\}$ is a tree deletion set in G of the same weight. □

Now we deal with degree two vertices. In the case of (unweighted) feedback vertex set, a degree two vertex can be removed by making its neighbors adjacent (even if they were adjacent before) without affecting the size of the feedback vertex set. For the weighted case, it is sufficient to keep only the minimum

weight vertex among the degree two vertices in a long path. However, for tree deletion sets, the degree two vertices may help in making the resulting graph a tree, and so we need a slightly different reduction rule.

We observe that if there is a long path with several degree two intermediate vertices, it is sufficient to keep only two of them distributing the total weight among the two with one of them having the minimum weight. This is because, if both end points of the long path are in the tree deletion set, then all the intermediate vertices or none of them will be in the tree deletion set. If only one of the end points is in the tree deletion set, then none of the intermediate vertices can be in the minimum weight tree deletion set. If neither of the end points is in the tree deletion set, then either the minimum weight intermediate vertex or none of them is in the tree deletion set.

We make the reduction rule and the arguments formal in the following discussion. In effect, the following rule reduces the number of degree two vertices by shortening long paths.

Reduction Rule 6 *If $v_0, v_1, \ldots, v_l, v_{l+1}$ is a path in the input graph, such that $l \geq 3$ and $\deg(v_i) = 2$ for every $i \in \{1, \ldots, l\}$, then (a) replace the vertices $v_1, \ldots, v_l$ by two vertices u_1 and u_2 with edges $\{v_0, u_1\}$, $\{u_1, u_2\}$, and $\{u_2, v_{l+1}\}$ and with $w(u_1) = \min\{w(v_i) \mid 1 \leq i \leq l\}$ and $w(u_2) = \left(\sum_{i=1}^{l} w(v_i)\right) - w(u_1)$. Moreover, in case of $l \geq 2$, if $w(v_0) > k$ or $w(v_{l+1}) > k$ holds, then apply (a) and then (b) delete u_2 and connect u_1 directly to v_{l+1}.*

Lemma 5 *Reduction Rule 6 is safe.*

Proof: Let G, w, k be the instance before the application of the rule and G', w', k the instance after the application of the rule. Let us first assume, that S is a tree deletion set in G with $w(S) \leq k$. We show that there is a tree deletion set S' for G' with $w'(S') \leq w(S)$. We distinguish three cases.

- *S contains both v_0 and v_{l+1}.* Note that this can only happen when only the reduction (a) was applied. In this case v_1 is disconnected from $G \setminus \{v_0, \ldots, v_{l+1}\}$ and, therefore, either $\{v_1, \ldots, v_l\} \subseteq S$ or $(V(G) \setminus \{v_1, \ldots, v_l\}) \subseteq S$. In the former case $S' = (S \setminus \{v_1, \ldots, v_l\}) \cup \{u_1, u_2\}$ is a tree deletion set for G' with $w'(S') = w(S)$ while in the latter case, for $S' = S \setminus \{v_1, \ldots, v_l\}$ we have $G \setminus S'$ is a path, $G' \setminus S'$ is a path, and $w'(S') = w(S') \leq w(S)$.

- *S contains exactly one of v_0 and v_{l+1}.* As the situation is symmetric, we can assume that v_{l+1} is in S. As $\{v_1, \ldots, v_l\}$ induces a path in G, $\{v_1, \ldots, v_l\} \setminus S$ induces a path in $G \setminus S$ and $G \setminus (S \cup \{v_1, \ldots, v_l\})$ is also a tree. Attaching at a node of this tree a path, we obtain again a tree. Hence, $S' = S \setminus \{v_1, \ldots, v_l\}$ is also a tree deletion set in G. Since $G \setminus (S \cup \{v_1, \ldots, v_l\}) = G' \setminus (S \cup \{u_1, u_2\})$, it follows that S' is also a tree deletion set in G' with $w'(S') = w(S') \leq w(S)$. This is true both in case (a) and (b).

- *S contains neither of v_0 and v_{l+1}.* If $S \cap \{v_1, \ldots, v_l\} = \emptyset$, then S is also a tree deletion set in G'. Otherwise, $\{v_1, \ldots, v_l\} \setminus S$ induces two

paths each attached to two different nodes in the tree $G \setminus S$. Assume that we have $w(v_r) = \min\{w(v_i) \mid 1 \leq i \leq l\}$. We first show that $S'' = (S \setminus \{v_1, \ldots, v_l\}) \cup \{v_r\}$ is also a tree deletion set for G. This is true, as attaching the two pending paths $v_1, \ldots, v_{r-1}$ and $v_{r+1}, \ldots, v_l$ to the tree $G \setminus (S \cup \{v_1, \ldots, v_l\}) = G' \setminus (S \cup \{u_1, u_2\})$ again creates a tree. By the same reason $S' = (S'' \setminus \{v_r\}) \cup \{u_1\}$ is a tree deletion set in G'. Also $w(S'') \leq w(S)$ as v_r is the vertex of the minimum weight and $w'(S') = w(S'')$.

Now assume that S' is a tree deletion set in G'. We show that there is a tree deletion set S for G with $w(S) \leq w'(S')$. We again distinguish three cases.

- *S' contains both v_0 and v_{l+1}.* Note that this can only happen when only the reduction (a) was applied. In this case u_1 is disconnected from $G' \setminus \{v_0, u_1, u_2, v_{l+1}\}$ and, therefore, either $\{u_1, u_2\} \subseteq S'$ or $(V(G') \setminus \{u_1, u_2\}) \subseteq S'$. In the former case $S = (S' \setminus \{u_1, u_2\}) \cup \{v_1, \ldots, v_l\})$ is a tree deletion set for G with $w(S) = w'(S')$ while in the latter case, for $S = S' \setminus \{u_1, u_2\}$ we have $G' \setminus S$ is a path, $G \setminus S$ is a path, and $w(S) = w'(S) \leq w'(S')$.

- *S' contains exactly one of v_0 and v_{l+1}.* In case only (a) was applied, we know that $\{u_1, u_2\}$ induces a path in G', $\{u_1, u_2\} \setminus S'$ induces a path in $G' \setminus S'$ and $G' \setminus (S' \cup \{u_1, u_2\})$ is also a tree. Attaching at a node of this tree a path, we obtain again a tree. Hence, $S = S' \setminus \{u_1, u_2\}$ is also a tree deletion set in G'. Since $G' \setminus (S' \cup \{u_1, u_2\}) = G \setminus (S \cup \{v_1, \ldots, v_l\})$, it follows that S is also a tree deletion set in G with $w(S) = w'(S) \leq w'(S')$. The case (b) follows along the same lines, just replacing $\{u_1, u_2\}$ with $\{u_1\}$.

- *S' contains none of v_0 and v_{l+1}.* If $S' \cap \{u_1, u_2\} = \emptyset$, then S' is also a tree deletion set in G. Otherwise, $\{u_1, u_2\} \setminus S'$ forms at most a vertex pending to a node in the tree $G' \setminus S'$. We first show that $S'' = (S' \setminus \{u_2\}) \cup \{u_1\}$ is also a tree deletion set for G'. This is true, as in $G' \setminus S''$ the vertex u_2 (if present) is pending to a node v_{l+1} of $G' \setminus (S' \cup \{u_1, u_2\})$ which is a subtree of $G' \setminus S'$. Now assume $w(v_r) = \min\{w(v_i) \mid 1 \leq i \leq l\}$ and let $S = (S'' \setminus \{u_1\}) \cup \{v_r\}$. Then $G \setminus S$ can be obtained from the tree $G' \setminus (S' \cup \{u_1, u_2\}) = G \setminus (S \cup \{v_1, \ldots, v_l\})$ by attaching two pending paths $v_1, \ldots, v_{r-1}$ and $v_{r+1}, \ldots, v_l$. Thus S is a tree deletion set in G. If (b) was applied, then $S'' = S'$, otherwise we have $l \geq 3$, hence $w'(u_2) > w'(u_1)$, and $w'(S'') \leq w'(S')$. Finally, $w(S) = w'(S'') \leq w'(S')$.

□

3.2 Branching Steps

Our FPT algorithm is based on a branching strategy similar to the one applied in [16]. First we use the algorithm of Cao et al. [2] to determine whether G has a feedback vertex set of size at most k in $O^*(3.83^k)$ time. As a tree deletion set of weight at most k is also a feedback vertex set of size at most k, if the

algorithm answers NO, we can also answer NO. Otherwise let F be the feedback vertex set for G found by the algorithm.

Our algorithm now branches into several cases and it returns YES if and only if at least one of the branches answers YES. In a search for a tree deletion set X we first guess its intersection Y with the known feedback vertex set F. This means that we branch into $2^{|F|}$ branches, each corresponding to one subset $Y \subseteq F$ and limit our search to the tree deletion sets X with $X \cap F = Y$.

As the vertices of Y are included in the tree deletion set constructed, we remove them and decrease k by $\sum_{v \in Y} w(v)$. As the tree deletion set we seek does not intersect $N = F \setminus Y$, we assign the vertices of N weight $k+1$. We also know that $G \setminus (Y \cup N)$ is a forest, as $F = (Y \cup N)$.

Now we are ready to describe the branching part of the algorithm. It modifies G, w, k and N. At the beginning and after each branching step, we apply Reduction Rules 1 to 5 and we only apply Reduction Rule 6 if none of $\{v_1, \ldots, v_l\}$ is in N. We always assume that the graph is reduced with respect to these reduction rules.

Let $H = V \setminus N$. Our branching step picks a vertex v from H, and branches by picking v into the tree deletion set or by not picking it (and hence adding it to N) and recursively solving the resulting problem. When we pick v into the solution, k drops by $w(v)$, which is at least one. The key observation in [3] for undirected feedback vertex set was that if v is adjacent to two connected components of N, then when v is added to N, the number of connected components of N decreases by at least one resulting in some progress. However, for the undirected feedback vertex set, it was always possible to choose such a vertex (adjacent to two connected components of N) as the minimum degree of the graph was three, and $V \setminus N$ induces a forest. For tree deletion set though, we are not guaranteed to have vertices with at least two neighbors in N all the time.

Let us call a vertex *useful* if it is in H, have exactly two neighbors in G and both these neighbors are in N. Our strategy is to show that if a vertex, while branching, doesn't decrease k or the number of connected components in N, it increases the count of useful vertices resulting in some progress. Increase in the the number of useful vertices constitute progress as, we argue that, if every vertex in H becomes useful, then we can solve the problem in polynomial time. To bound the depth of the recursion, we use the measure $\mu = k + c - u$ where

- k is the budget - the weight of the vertices we can still add to the tree deletion set being constructed. Initially, we have $k = k_0 - \sum_{v \in Y} w(Y)$.
- c is the number of components in $G[N]$. Initially, we have $N = F \setminus Y$ and therefore $c \leq k - |Y|$.
- u is the number of useful vertices in H.

As $c \leq k$, we have $\mu \leq 2k$. We argue that each (two way) branching rule decreases this measure, and that the reduction rules do not increase the measure. Note also that none of the rules introduces a cycle to $G[H]$ and therefore $G[H]$ is still a forest.

Lemma 6 *No reduction rule increases μ.*

Proof: None of the rules increases k and none of the reduction rules increases c, the number of connected components of N. While Reduction rules 3 and 4 delete some useful vertices, in such cases, k is decreased by the weight of the deleted vertices and therefore by at least the number of deleted useful vertices. □

After applying the reduction rules, we distinguish three cases. If μ is not positive, we return NO, which is justified by the lemma below.

Lemma 7 *If the measure μ becomes non positive, then there is no tree deletion set for the current branch.*

Proof: Suppose that there is a tree deletion set X of total weight at most k for the graph G at a branch with measure $\mu = k + c - u$. Let U' be the set of useful vertices in X and U the set of useful vertices not in X. Contract each connected component of $G[N]$ to a single vertex and denote the resulting graph $\tilde{G}$. Let us denote the set of vertices created by contraction of the components of $G[N]$ by $\tilde{N}$. The contraction does not create parallel edges, because G is reduced with respect to Reduction Rule 3. Moreover, since $G \setminus X$ is a tree, we know that $\tilde{G} \setminus X$ is also a tree, since contracting edges of a tree cannot make it disconnected or create a cycle. Hence, $G' = \tilde{G}[\tilde{N} \cup U]$ is a forest. Therefore, G' has at most $|\tilde{N}|+|U|-1$ edges. On the other hand, each vertex in U has exactly two neighbors in $\tilde{N}$ and therefore G' has $2|U|$ edges. It follows that $|U| \leq |\tilde{N}|-1$.

As the number c of components in N equals $|\tilde{N}|$, $u = |U|+|U'|$, and $|U'| \leq k$ (as U' is a subset of the at most k deleted vertices), we have $\mu = k + c - u \geq k + |\tilde{N}| - (|\tilde{N}| - 1 + k) = 1$ — a contradiction. □

If $\mu \geq 1$, and if there is a vertex v in H which satisfies at least one of the following conditions, then we branch on this vertex v.

(i) it has total degree at least three in G and at least two neighbors in N;

(ii) it has a neighbor in N and a neighbor which is a leaf in $G[H]$; or

(iii) it has at least two neighbors in H, which are both leaves in $G[H]$;

More precisely, for such a vertex v we consider two cases:

- v is a part of the tree deletion set constructed — then we delete v from the graph and decrease k by $w(v)$;
- v is not in the sought tree deletion set— then we set the weight of v to $k + 1$ and add v to N.

In both cases the procedure is called recursively on the modified G, w, k, N and the procedure returns YES if in at least one of the branches the recursive call returns YES. If there are several vertices satisfying the conditions, then we select

a vertex which satisfies condition (i) if such a vertex is available. We only select other vertices if there is no vertex satisfying the condition (i).

In this 'two way branch' we show that in each such recursive call, the value of μ is at least one less than that in the current call.

Lemma 8 *If the vertex we branch on satisfies at least one of the conditions* (i) *to* (iii)*, then the measure decreases by at least one in each branch.*

Proof: Let us first consider the case that we delete the vertex we branch on. Since it is not in N, deleting it cannot increase the number of connected components in $G[N]$. Moreover, since the vertex has degree at least three in case (i) and neighbors in H in cases (ii) and (iii), it is not a useful vertex and therefore the number of useful vertices remains the same. Since k is decreased by the weight of the vertex deleted, the measure drops by at least one.

Consider now the case that we add the vertex v to N and suppose it satisfies condition (i). Then v has at least two neighbors in N and since the graph is reduced with respect to Reduction Rule 3, these neighbors are in different connected components of $G[N]$. Therefore c is decreased, k remains the same and u is not decreased, which means that the measure drops.

If v satisfies condition (ii), then adding v to N does not increase the number of components in $G[N]$ as v already has a neighbor in N. On the other hand, a neighbor u of v in H is a leaf in $G[H]$. Since u does not satisfy condition (i), and the graph is reduced with respect to Reduction Rule 5, u has exactly one other neighbor, which is in N. Hence, u becomes useful and the measure is decreased as k remains the same.

Finally, if v satisfies condition (iii), then adding v to N may increase the number of components in $G[N]$ by one. However, both neighbors of v, which are leaves in $G[H]$ and do not satisfy condition (i), become useful. Therefore the measure decreases also in this case. □

Finally, if $\mu \geq 1$, but there is no vertex in H satisfying the conditions, then either N is empty or every vertex in H is useful as we argue below.

Lemma 9 *If no vertex satisfies any of the conditions* (i) *to* (iii) *and* N *is nonempty, then every vertex in* H *is useful.*

Proof: We show that if three is a vertex $v' \in H$ which is not useful, then there is a vertex in H which satisfies some of the conditions (i) to (iii).

If v' is isolated in $G[H]$, then v' must have at least three neighbors in N, as the graph is reduced with respect to Reduction Rules 4, 5, and 3, v' is not useful, and there are no isolated vertices by Remark 1 as N is nonempty. But then v' satisfies condition (i).

Recall that $G[H]$ is a forest. If v' is non-isolated in $G[H]$, then consider a leaf v in the same connected component of $G[H]$, which has the maximal distance from v' (see Figure 1). We know, that v has degree at least two in G, as the graph is reduced with respect to Reduction Rule 5. If v has degree at least three, then v satisfies the condition (i). Otherwise consider a neighbor u of

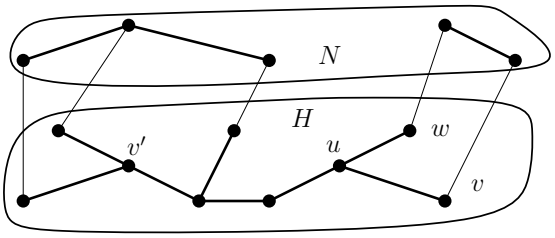

Figure 1: Complicated case in the proof of Lemma 9.

v in H. If u has degree two in G, then the part (b) of Reduction Rule 6 applies on the path formed by v, u and their neighbors. Hence, the vertex u has degree at least three in G. If u has a neighbor in N, then u satisfies the condition (ii).

If u has degree three in $G[H]$, then consider a neighbor w of u which is not on the unique path between v and v' in $G[H]$ (see Figure 1). If w is a leaf in $G[H]$ then u satisfies condition (iii). If w is not a leaf in $G[H]$ then any leaf in the subtree of $G[H]$ rooted in w which does not contain u has greater distance from v' than v, which contradicts the way we selected v. □

If all vertices in H are useful vertices, we proceed as follows. Note that the graph is formed by the vertices in N and useful vertices adjacent to them. We contract each connected component of $G[N]$ to a single vertex. Let us again call the set of vertices created this way $\tilde{N}$. Recall that there is no cycle in $G[N]$ as the graph is reduced with respect to Reduction Rule 2. As we only search for a tree deletion set X among vertices in H, it is easy to verify that $X \subseteq H$ is a tree deletion set in the graph after contraction if and only if it was a tree deletion set in the original graph. Note that the contraction does not create parallel edges, because the graph is reduced with respect to Reduction Rule 3 and, hence, there is no vertex in H with both its neighbors in the same connected component of $G[N]$.

Now if there are two vertices in H with the same neighbors in $\tilde{N}$, then we delete the one with the lower weight and decrease k by its weight. Clearly at least one of them must be in the constructed tree deletion set and if only one of them is in the tree deletion set, then we can assume it is the one with lower weight. Next we construct an auxiliary graph $\overline{G}$ with vertex set $\tilde{N}$ and a weighted edge between a pair of vertices if there is a vertex v in H with this pair of vertices as its neighbors in G. The weight of the edge equals the weight of v. It is easy to see, that a minimum tree deletion set in G corresponds to the edge complement of a maximum spanning tree in $\overline{G}$ and vice versa. More precisely if $T = (\tilde{N}, E')$ is a spanning tree, then the set X of vertices v of H such that the edge corresponding to $N(v)$ in $\overline{G}$ is not in E' is a tree deletion set for G. Similarly, if $X \subseteq H$ is a tree deletion set in G, then $T = (\tilde{N}, S)$, where $S = \{N(v) \mid v \in H \setminus X\}$ is a spanning tree of $\overline{G}$. The weight of S is always $\sum_{v \in H \setminus X} w(v)$

Hence, we use the standard algorithm [10] to find a maximum spanning tree $T = (\tilde{N}, S)$ of $\overline{G}$, and answer YES if and only if $(\sum_{v \in H} w(v)) - w'(S)$ is at most k, where $w'(S)$ denotes the weight of the tree T.

If N is empty, then the graph consists of isolated vertices, since $G[H]$ is a forest and the graph is reduced with respect to Reduction Rule 5. Therefore it is enough to delete all vertices but the one with the largest weight and answer YES if and only if the weight of the deleted vertices is at most k. This finishes the description of the algorithm.

The correctness of the algorithm has been already argued within its description, here we argue about the running time of the algorithm. First note that an application of any of the reduction rules can be recognized as well as applied in linear time. Since the Reduction Rules 1 and 2 only apply once, and the Reduction Rules 3–6 reduce the number of vertices, the rules can be exhaustively applied in $O(nm)$ time. To check the value of the measure and to find a vertex to branch on takes a linear time. Finally, one can contract the connected components in $G[N]$ in $O(mn)$ time and find the maximum spanning tree in $\overline{G}$ in $O(m)$ time. Hence the time spent in each node of the search tree is $O(mn)$.

It remains to count the number of nodes in the search tree. We first branch into at most $2^{|F|}$ branches, each corresponding to one subset Y of the feedback vertex set F. Then we keep branching into two branches, each time reducing the measure μ by at least one. Since at the beginning we have $\mu = k + c - u \leq 2k_0 - 2|Y|$, this part of the search tree has at most $2 \cdot 2^{2k_0 - 2|Y|}$ nodes. Summing this according to $y = |Y|$, we have that the total number of nodes in the search tree is at most $\sum_{y=0}^{k_0} \binom{k_0}{y} 2 \cdot 2^{2k_0 - 2y} = 2 \cdot (1+4)^{k_0} = 2 \cdot 5^{k_0}$. Recall that the first step in our algorithm is to determine whether G has a feedback vertex set of size at most k. This step is done in $O^*(3.83^k)$ time using the algorithm of Cao et al. [2]. Therefore the whole algorithm runs in $O^*(5^k)$ time. This completes the proof.

4 Conclusions and Open Problems

We have shown that the Weighted Tree Deletion Set problem is fixed-parameter tractable. Improving the running time of our algorithm is a natural open problem. Another direction, which has attracted a lot of attention in parameterized complexity recently, is to study the *kernelization complexity* of the problem. Our fixed-parameter algorithm immediately implies an exponential kernel for the problem, but the natural open question is whether the problem has a polynomial size kernel. That is, is there a polynomial time algorithm that reduces the given input (G, k) to an equivalent graph with the number of vertices bounded by a polynomial in k? While the related feedback vertex set problem has an $O(k^2)$ sized kernel [22], we conjecture that the Tree Deletion Set problem does not admit a polynomial sized kernel under standard complexity theoretic assumptions[2].

[2]This conjecture has been recently resolved in the negative, the authors of [12] obtain a polynomial kernel for the problem.

References

[1] V. Bafna, P. Berman, and T. Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM J. Discrete Math.*, 12(3):289–297, 1999. doi:10.1137/S0895480196305124.

[2] Y. Cao, J. Chen, and Y. Liu. On feedback vertex set new measure and new structures. In H. Kaplan, editor, *Algorithm Theory - SWAT 2010*, volume 6139 of *Lecture Notes in Computer Science*, pages 93–104. Springer Berlin / Heidelberg, 2010. doi:10.1007/978-3-642-13731-0_10.

[3] J. Chen, F. V. Fomin, Y. Liu, S. Lu, and Y. Villanger. Improved algorithms for feedback vertex set problems. *Journal of Computer and System Sciences*, 74(7):1188–1198, 2008. doi:10.1016/j.jcss.2008.05.002.

[4] R. H. Chitnis, M. Cygan, M. T. Hajiaghayi, and D. Marx. Directed subset feedback vertex set is fixed-parameter tractable. In A. Czumaj, K. Mehlhorn, A. M. Pitts, and R. Wattenhofer, editors, *ICALP (1)*, volume 7391 of *Lecture Notes in Computer Science*, pages 230–241. Springer, 2012. doi:10.1007/978-3-642-31594-7_20.

[5] M. Cygan, M. Pilipczuk, M. Pilipczuk, and J. Wojtaszczyk. Subset feedback vertex set is fixed-parameter tractable. *SIAM Journal on Discrete Mathematics*, 27(1):290–309, 2013. doi:10.1137/110843071.

[6] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, New York, 1999.

[7] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, Berlin, 2006.

[8] F. V. Fomin, S. Gaspers, A. V. Pyatkin, and I. Razgon. On the minimum feedback vertex set problem: Exact and enumeration algorithms. *Algorithmica*, 52(2):293–307, 2008. doi:10.1007/s00453-007-9152-0.

[9] F. V. Fomin, D. Lokshtanov, N. Misra, and S. Saurabh. Planar f-deletion: Approximation, kernelization and optimal fpt algorithms. In *FOCS*, pages 470–479. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.62.

[10] M. L. Fredman and D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *Journal of Computer and System Sciences*, 48(3):533 – 551, 1994. doi:10.1016/S0022-0000(05)80064-9.

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

[12] A. C. Giannopoulou, D. Lokshtanov, S. Saurabh, and O. Suchý. Tree Deletion Set has a Polynomial Kernel (but no $\mathrm{OPT}^{O(1)}$ approximation). *CoRR*, abs/1309.7891, Sept. 2013.

[13] E. J. Kim, A. Langer, C. Paul, F. Reidl, P. Rossmanith, I. Sau, and S. Sikdar. Linear kernels and single-exponential algorithms via protrusion decompositions. *CoRR*, abs/1207.0835, 2012. URL: `http://arxiv.org/abs/1207.0835`.

[14] D. Lokshtanov, N. S. Narayanaswamy, V. Raman, M. S. Ramanujan, and S. Saurabh. Faster parameterized algorithms using linear programming. *CoRR*, abs/1203.0833, 2012. URL: `http://arxiv.org/abs/1203.0833`.

[15] D. Marx, B. O'Sullivan, and I. Razgon. Treewidth reduction for constrained separation and bipartization problems. In J.-Y. Marion and T. Schwentick, editors, *STACS*, volume 5 of *LIPIcs*, pages 561–572. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010. `doi:10.4230/LIPIcs.STACS.2010.2485`.

[16] N. Misra, G. Philip, V. Raman, and S. Saurabh. On parameterized independent feedback vertex set. *Theoretical Computer Science*, 461:65–75, 2012. `doi:10.1016/j.tcs.2012.02.012`.

[17] N. Misra, G. Philip, V. Raman, S. Saurabh, and S. Sikdar. FPT algorithms for connected feedback vertex set. *J. Comb. Optim.*, 24(2):131–146, 2012. `doi:10.1007/s10878-011-9394-2`.

[18] D. Mölle, S. Richter, and P. Rossmanith. Enumerate and expand: Improved algorithms for connected vertex cover and tree cover. *Theory of Computing Systems*, 43(2):234–253, 2008. `doi:10.1007/s00224-007-9089-3`.

[19] N. S. Narayanaswamy, V. Raman, M. S. Ramanujan, and S. Saurabh. LP can be a cure for parameterized problems. In *STACS*, pages 338–349, 2012. `doi:10.4230/LIPIcs.STACS.2012.338`.

[20] R. Niedermeier. *Invitation to Fixed Parameter Algorithms*. Oxford University Press, USA, March 2006.

[21] V. Raman, S. Saurabh, and O. Suchý. An fpt algorithm for tree deletion set. In S. Ghosh and T. Tokuyama, editors, *WALCOM: Algorithms and Computation*, volume 7748 of *Lecture Notes in Computer Science*, pages 286–297. Springer Berlin Heidelberg, 2013. `doi:10.1007/978-3-642-36065-7_27`.

[22] S. Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Transactions on Algorithms*, 6(2), 2010. `doi:10.1145/1721837.1721848`.

[23] P. van 't Hof and Y. Villanger. Proper interval vertex deletion. *Algorithmica*, 65(4):845–867, 2013. `doi:10.1007/s00453-012-9661-3`.

[24] M. Yannakakis. The effect of a connectivity requirement on the complexity of maximum subgraph problems. *J. ACM*, 26(4):618–630, 1979. `doi:10.1145/322154.322157`.