A Faster Parameterized Algorithm for Pseudoforest Deletion[*]

Hans L. Bodlaender[1], Hirotaka Ono[2], and Yota Otachi[3]

1 **Department of Information and Computing Sciences, Utrecht University; and Department of Mathematics and Computer Science, University of Technology Eindhoven, The Netherlands**
`h.l.bodlaender@uu.nl`

2 **Department of Economic Engineering, Kyushu University, Fukuoka, Japan**
`hirotaka@econ.kyushu-u.ac.jp`

3 **School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa, Japan**
`otachi@jaist.ac.jp`

Abstract

A pseudoforest is a graph where each connected component contains at most one cycle, or alternatively, a graph that can be turned into a forest by removing at most one edge from each connected component. In this paper, we show that the following problem can be solved in $O(3^k nk^{O(1)})$ time: given a graph G and an integer k, can we delete at most k vertices from G such that we obtain a pseudoforest? The result improves upon an earlier result by Philip et al. [MFCS 2015] who gave a (nonlinear) $7.56^k n^{O(1)}$-time algorithm both in the exponential factor depending on k as well as in the polynomial factor depending on n.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases pseudoforest deletion, graph class, width parameter, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.7

1 Introduction

In this paper, we consider the Pseudoforest Deletion problem. A pseudoforest is an undirected graph that is obtained from a forest by adding at most one edge to each connected component. In the Pseudoforest Deletion problem, we are given a graph $G = (V, E)$ and an integer k, and ask if there is a set of at most k vertices in G, that, when deleted from G, turns G into a pseudoforest. The Pseudoforest Deletion problem is closely related to the well known Feedback vertex set problem, where we want to delete at most k vertices from a graph so that the graph becomes a forest.

The Pseudoforest Deletion problem was first studied by Philip et al. [12], together with the generalization where each connected component is a tree plus at most ℓ edges. They showed that for each ℓ, the problem to delete at most k vertices such that we obtain such an *ℓ-pseudoforest* has a kernel with $f(\ell)k^2$ vertices. For the Pseudoforest Deletion problem,

[*] This research was partially supported by the Networks project, funded by the Dutch Ministry of Education, Culture and Science through NWO and by MEXT/JSPS KAKENHI grant numbers 24106004, 24220003, 25730003, 26540005. The third author was partially supported by FY 2015 Researcher Exchange Program between JSPS and NSERC.

i.e., the case that $\ell = 1$, they give a deterministic algorithm with running time $7.56^k n^{O(1)}$.[1] In this paper, we improve upon the latter result, both with respect to the exponential factor in k, as well as in the polynomial factor in n, which is, in our case, linear.

It is easy to see that the Pseudoforest deletion problem belongs to the class of problems studied by Fomin et al. [9], and thus, by these results, the problem has a constant factor polynomial time approximation algorithm, a polynomial kernel (improved to quadratic by the results of Philip et al. [12]), and a randomized algorithm that runs in time $O(c^k n)$ for some constant c. The randomized algorithm is a generalization of an algorithm by Becker et al. [3] for the Feedback Vertex Set problem and a related problem called the Loop Cutset problem. Fomin et al. [9] also give deterministic algorithms running in time $O(2^{O(k)} n \log^2 n)$ and $O(nm)$ time constant factor approximation algorithms for a large class of problems that includes Pseudoforest deletion. If one looks closely at the randomized algorithm by Becker et al. [3] and the generalization by Fomin et al. [9], it follows that one can solve the Pseudoforest deletion problem with a randomized algorithm in $O(4^k n k^{O(1)})$ time.

Our improvement on these two algorithms is based upon the combination of a few different insights and techniques, in particular:

- Positive instances, i.e., graphs that can be turned into a pseudoforest by deleting at most k vertices have treewidth at most $k + 2$.
- The notion of *pseudoforest* has the following *local characterization*: a graph is a pseudoforest if and only if it has an edge orientation such that each vertex has outdegree at most one.
- The local characterization allows us to solve the problem with dynamic programming on a tree decomposition in time that is linear in the number of vertices and single exponential in the treewidth, without the need to use advanced techniques like the cut and count method [8] or the rank based approach [6].
- With help of *convolutions* [15] (see also [4]), the running time of the dynamic programming algorithm is reduced to $O(3^t n t^{O(1)})$ on tree decompositions of width t.
- What remains is the need to find an initial tree decomposition to run the dynamic programming algorithm on. For this, we use a modification of the $O(f(t)n)$ algorithm for Treewidth by Bodlaender [5]. The modification includes the use of *iterative compression* inside one of the subroutines.

It is interesting to contrast our result with the currently best known parameterized algorithms for Feedback Vertex Set: for the Pseudoforest Deletion problem we have a deterministic $O(3^k n k^{O(1)})$ algorithm, while Feedback Vertex Set can be solved in $O(3^k n^{O(1)})$ time with a randomized algorithm [8] and $O(3.63^k n^{O(1)})$ time with a deterministic algorithm [10]; in both cases, the running time is not linear in n.

This paper is organized as follows. In Section 2, we give some preliminary definitions. Section 3 contains a number of graph theoretic observations; in many cases these are not hard to observe, from existing literature or folklore. Section 4 discusses how the Pseudoforest Deletion problem can be solved when a tree decomposition of bounded width is available. This method is used as a subroutine in the main algorithm, that is given in Section 5. The paper ends with some conclusions in Section 6.

[1] They did not specify the exact dependency in n, which is at least quadratic.

2 Preliminaries

When not specified otherwise, a graph $G = (V, E)$ is considered to be undirected, but possibly with selfloops and parallel edges. Allowing selfloops and parallel edges makes the description of the main algorithm easier. An *orientation* of a graph $G = (V, E)$ is a directed graph obtained by giving each edge in G a direction. For a graph $G = (V, E)$ and vertex set $W \subseteq V$, the *subgraph of G induced by W* is denoted by $G[W] = (W, \{e \in E \mid \text{ both endpoints of } e \text{ belong to } W\})$.

A *tree decomposition* of a graph $G = (V, E)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$ with T a tree, and $\{X_i \mid i \in I\}$ a collection of subsets (called *bags*) of V, such that

1. $\bigcup_{i\in I} X_i = V$;
2. for all $\{v, w\} \in E$, there is an $i \in I$ with $\{v, w\} \subseteq X_i$
3. for all $v \in V$, the set of nodes $\{i \in I \mid v \in X_i\}$ forms a connected subtree of T.

The *width* of a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ is $\max_{i\in I} |X_i| - 1$. The *treewidth* of a graph G is the minimum width over all tree decompositions of G.

For the definition above, if there are parallel edges or selfloops, we can just ignore them, i.e., a tree decomposition of a graph with parallel edges and selfloops is a tree decomposition of the associated simple graph (obtained by keeping only one of each set of parallel edges and removing all selfloops).

In this paper, we also use the related notion of *nice* tree decomposition. In the literature, there are a few variants of this notion that differ in details. In this case, we use the variant with *edge introduce nodes* and leaf bags of size one.

A *nice tree decomposition* is a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ where T is a *rooted* tree, and nodes are of one of the following five different types. With each bag/node in the tree decomposition, we also associate a subgraph of G; the subgraph associated with node i is denoted $G_i = (V_i, E_i)$. We give each type together with how the corresponding subgraph is formed.

- **Leaf** nodes i. i is a leaf of T; $|X_i| = 1$, and $G_i = (\{v_i\}, \emptyset)$ is the graph consisting of the vertex v_i and no edges.
- **Introduce vertex** nodes i. i has one child, say j. There is a vertex v with $X_i = X_j \cup \{v\}$, $v \notin V_j$, and $G_i = (V_j \cup \{v_i\}, E_j)$, i.e., G_i is obtained from G_j by adding v_i as isolated vertex.
- **Introduce edge** nodes i. i has one child, say j. There are two vertices $v, w \in X_i$, $X_i = X_j$, and $G_i = (V_j, E_j \cup \{v, w\})$. I.e., G_i is obtained from G_j by adding an edge between two vertices in $X_i = X_j$. If we have parallel edges, we have one introduce edge node for each parallel edge. E.g., if there are two edges from v to w, we have two edge introduce nodes for the pair v, w; typically, one of these can be the parent of the other in the tree. A selfloop with endpoint v is handled in the same way, i.e., there is an introduce edge node i with $v \in X_i$, and G_i is obtained by adding the selfloop to G_j.
- **Forget** nodes i. i has one child, say j. There is a vertex v with $X_i = X_j - \{v\}$. G_i and G_j are the same graph.
- **Join** nodes i. i has two children, say j_1 and j_2. $X_i = X_{j_1} = X_{j_2}$, $V_{j_1} \cap V_{j_2} = X_i$ and $E_{j_1} \cap E_{j_2} = \emptyset$. $G_i = (V_{j_1} \cup V_{j_2}, E_{j_1} \cup E_{j_2})$. I.e., G_i is obtained by taking the union of G_{j_1} and G_{j_2}, where the vertices in X_i are the intersection of these two graphs.

If r is the root of T, then $G_r = G$.

Restricting a function f to a sub-domain Z is denoted $f|_Z$. With $f + v \to i$ we denote the new function, obtained by adding v to the domain of f, mapping v to i. $f^{v\to i}$ denotes the function, obtaining by changing f by mapping v to i.

A *pseudotree* is a connected graph that is either a tree or obtained by adding one edge to a tree. Note that, as we allow selfloops and parallel edges, this edge may be a selfloop or a parallel edge. A graph is a *pseudoforest*, if each connected component is a pseudotree.

A *pseudoforest deletion set* in a graph $G = (V, E)$ is a set of vertices $W \subseteq V$ such that $G[V - W]$ is a pseudoforest.

A *p-contraction* of an edge $\{v, w\}$ is the operation that identifies v and w, removes the edge $\{v, w\}$, but keeps parallel edges, e.g., if there are edges $\{v, x\}$ and $\{w, x\}$ before the contraction, then x has two parallel edges to the newly formed vertex; if there is an edge parallel to the contracted edge, then this turns into a selfloop. Note that the number of edges of a graph drops by exactly one when doing a p-contraction.

The *c-improved graph* of a graph $G = (V, E)$ is the graph, obtained by adding an edge between each pair of vertices that have at least c common neighbors of degree at most $c + 1$. (We do not take the closure of this operation.)

A vertex v is *simplicial* in a graph $G = (V, E)$ if the neighborhood of v is a clique.

3 Graph theoretic observations

In this section, we give some graph theoretic results that are either folklore or easy to see. The following lemma is a trivial observation.

▶ **Lemma 3.1.** *Let $G = (V, E)$ be a graph. The following statements are equivalent.*

1. *G is a pseudoforest.*
2. *G has an orientation such that each vertex has outdegree at most 1.*

While Lemma 3.1 is an easy observation, it is a key point to our result: being a pseudoforest seems to be a global property, it actually can be expressed by a local property: having an orientation with outdegree at most one allows a dynamic programming algorithm on tree decompositions with three states per vertex, i.e., with tables of size bounded by 3^t, t being the width of the tree decomposition.

▶ **Lemma 3.2.** *Let $G = (V, E)$ be a graph.*

1. *Suppose that there are four or more parallel edges from v to w. Let G' be the graph, obtained from G by removing one parallel edge from v to w. The minimum size of a pseudoforest deletion set in G equals the minimum size of the pseudoforest deletion set of G'.*
2. *Suppose that there are three or more self loops with v as endpoint. Let G'' be the graph, obtained from G by removing one selfloop with v as endpoint. The minimum size of a pseudoforest deletion set in G equals the minimum size of the pseudoforest deletion set of G''.*

Proof. The result follows by observing that if there are three or more parallel edges from v to w then any pseudoforest deletion set must contain v or w, and that if there are two or more selfloops with v as endpoint, then any pseudoforest deletion set contains v. ◀

The following lemma, used in our algorithm is the main reason why we use p-contractions and graphs with parallel edges and self-loops: we do not have such a result when we would use simple graphs and the usual notion of contraction.

▶ **Lemma 3.3.** *Let G' be obtained from G by a p-contraction of the edge $\{v, w\}$. Let x be the vertex resulting from the contraction of $\{v, w\}$. Suppose W is a pseudoforest deletion set in G'.*

1. *If $x \notin W$, then W is a pseudoforest deletion set in G.*
2. *If $x \in W$, then $W - \{x\} \cup \{v, w\}$ is a pseudoforest deletion set in G.*

Proof. The result follows by observing that when we contract an edge from a pseudoforest, we again obtain a pseudoforest. ◀

The following result is folklore. While the folklore result deals with simple graphs, we can build a nice tree decomposition for a graph with parallel edges and selfloops by building a tree decomposition for the underlying simple graph, and then adding the selfloops and parallel edges in the obvious way.

▶ **Lemma 3.4.** *Suppose $G = (V, E)$ is given with a tree decomposition of width k with r bags. Then one can construct a nice tree decomposition of G with $O(kr + |E|)$ bags in $O(k^2r + |E|k)$ time.*

The following result is a trivial consequence of treewidth folklore. As the construction in the proof is used in the algorithm, we give the constructive proof here.

▶ **Lemma 3.5.** *Let $G = (V, E)$ be a graph.*
1. *If G is a pseudoforest, the treewidth of G is at most 2.*
2. *If there is a set $W \subseteq V$ such that $G[V - W]$ is a pseudoforest, the treewidth of G is at most $2 + |W|$. The corresponding tree decomposition can be computed in $O(n \cdot |W|)$ time.*

Proof.
1. The treewidth of a tree is 1; the treewidth of a tree plus one edge is 2: add one endpoint of the new edge to all bags. The treewidth of a graph equals the maximum treewidth of a connected component, hence the treewidth of a pseudoforest is 2.
2. Take a tree decomposition of width 2 of $G[V - W]$, and add W to all bags. ◀

One ingredient of our algorithm is an approach, first used by Bodlaender [5] to obtain an algorithm for TREEWIDTH that uses $O(f(k)n)$ time, see Theorem 3.6 below. Perković and Reed [11] showed that the result can be improved with respect to factors polynomial in k; for our purposes, the form below suffices.

▶ **Theorem 3.6** (Bodlaender [5]). *Let $G = (V, E)$ be a graph and t an integer. At least one of the following three statements is true.*
- *Any maximal matching of G has $\frac{1}{O(t^8)}n$ edges.*
- *The t-improved graph of G has at least $\frac{1}{O(t^2)}n$ simplicial vertices of degree at most t.*
- *The treewidth of G is at least $t + 1$.*

▶ **Lemma 3.7.** *Let G be a graph and let k be an integer. G has a set $X \subseteq V(G)$ of size at most k such that $G - X$ is a pseudoforest if and only if the $k + 3$-improved graph of G has a set X' of size at most k such that $G - X'$ is a pseudoforest.*

Proof. As a subgraph of a pseudoforest is a pseudoforest, the 'if'-direction is trivial.

Suppose G has a set X of size at most k such that $G - X$ is a pseudoforest. Consider two vertices v, w, with at least $k + 3$ common neighbors. We claim that $v \in X$ or $w \in X$. Suppose not. Vertices v and w have at least 3 common neighbors that do not belong to X. We now have five vertices with at least six edges between them, so for any orientation, at least one of these five vertices has outdegree two or more, contradiction. As $v \in X$ or $w \in X$, we can safely add the edge $\{v, w\}$, as $G - X$ remains a pseudoforest. ◀

4 Solving Pseudoforest Deletion on tree decompositions

In this section, we will prove the following result.

▶ **Theorem 4.1.** *Suppose $G = (V, E)$ is given with a tree decomposition of width at most t with $O(n)$ bags. One can find in $O(3^t nt^{O(1)})$ time a minimum size pseudoforest deletion set.*

For easier explanation of the algorithm, we will first derive an algorithm that uses $O(4^t nt^{O(1)})$ time and solves the decision problem, i.e., computes the *size* of the minimum pseudoforest deletion set. Then, with help of the *convolutions* technique for tree decompositions, introduced by van Rooij et al. [15], we obtain a decision problem with $O(3^t nt^{O(1)})$ running time. At the end, we discuss how we can compute within the same time bound also the corresponding minimum size pseudoforest deletion set.

An algorithm that runs in $O(4^t nt^{O(1)})$ time

We first transform the tree decomposition to a nice tree decomposition, which has $O(tn)$ bags.

Recall that we associate a subgraph of G, G_i with each node i in the nice tree decomposition. A *partial solution* for a node $i \in I$ is a pair (Y, Λ), with $Y \subseteq V_i$ a set of vertices and Λ an orientation of E_i such that each vertex in $V_i - Y$ has at most one outgoing arc in Λ. If r is the root of the nice tree decomposition, then a partial solution for r is called a *solution*. We say a solution (Y, Λ) *extends* partial solution (Y', Λ') for i if $Y' = Y \cap V_i$ and Λ' is the restriction of Λ to E_i.

The *characteristic* of a partial solution (Y, Λ) for i is the function $f : X_i \to \{X, 0, 1\}$, such that

- For all $v \in X_i$, $f(v) = X$ if and only if $v \in Y$.
- If $v \in X_i$ and $f(v) = 0$, then v has no outgoing arcs in Λ.
- If $v \in X_i$ and $f(v) = 1$, then v has exactly one outgoing arc in Λ.

The main ingredient of the algorithm is to compute for each node in i a table (function) T_i, in postorder, i.e., we compute the table for a node after the tables for its children are known. A table T_i maps each function $f : X_i \to \{0, 1, X\}$ to an nonnegative integer or to ∞, in the following way.

Suppose i is a bag in a nice tree decomposition, with corresponding set X_i and subgraph G_i. For a function $f : X_i \to \{0, 1, X\}$, $T_i(f)$ equals the minimum of $|Y|$ over all partial solutions (Y, Λ) at i with characteristic f. If no such partial solution exists, then $T_i(f) = \infty$.

The following claim trivially holds by Lemma 3.1, and shows how to obtain the answer to the decision version of the PSEUDOFOREST DELETION problem given T_r for the root r of the tree decomposition.

▶ **Claim 4.2.** *Let r be the root of a nice tree decomposition of $G = (V, E)$. The minimum size of a pseudoforest deletion set in G equals the minimum of $T_r(f)$ over all $f : X_r \to \{0, 1, X\}$.*

We will now discuss for each of the types of nodes in a nice tree decomposition how to compute the table T_i, given the tables of the children of the node.

Leaf nodes. Let i be a leaf node, with $X_i = \{v\}$. Now, if $f(v) = 0$, then $T_i(f) = 0$; if $f(v) = 1$, then $T_i(f) = \infty$, and if $f(v) = X$, then $T_i(f) = 1$.

Introduce vertex nodes. Suppose i is an introduce vertex node i with child j with $X_i = X_j \cup \{v\}$.

As the degree of v in G_i is 0, for each f with $f(v) = 1$, we have $T_i(f) = \infty$, as there are no partial solutions with v having outdegree 1.

For a function f with $f(v) = 0$, we have $T_i(f) = T_i(f|_{X_i})$; and for functions f with $f(v) = X$, we have $T_i(f) = T_i(f|_{X_i}) + 1$ — we can just extend any partial solution for G_j by either not placing v in the pseudoforest deletion set, in which case v has outdegree 0; or placing v in the pseudoforest deletion set, in which case v is mapped to X and the size of the set is increased by one.

Introduce edge nodes. Consider an introduce edge node i with child j, where we introduce an edge with endpoints v and w. Note that we allow parallel edges and selfloops; the subroutine below is also correct in case the introduced edge is parallel to an existing edge or is a selfloop (i.e., $v = w$.)

For each $f : X_i \to \{0, 1, X\}$, we consider the two cases in which $\{v, w\}$ can be oriented. We then obtain the following cases; for brevity, we omit the isomorphic cases with the roles of v and w switched.

- If $f(v) = X$ and $f(w) = X$, then $T_i(f) = T_j(f)$.
- If $f(v) = X$ and $f(w) = 0$, then $T_i(f) = T_j(f)$. (We must orient the edge from v to w.)
- If $f(v) = X$ and $f(w) = 1$, then $T_i(f) = \min\{T_j(f), T_j(f^{w\to 0})\}$.
- If $f(v) = 1$ and $f(w) = 1$, then $T_i(f) = \min\{T_j(f^{v\to 0}), T_j(f^{w\to 0})\}$.
- If $f(v) = 1$ and $f(w) = 0$, then $T_i(f) = T_j(f^{v\to 0})$. (We must orient the edge from v to w, and thus v has outdegree 0 in the corresponding orientation of G_j.)
- If $f(v) = 0$ and $f(w) = 0$, then $T_i(f) = \infty$. (No orientation with both v and w having outdegree 0 is possible.)

Forget nodes. Let i be a forget node with child j with $X_j = X_i \cup \{v\}$. Then $T_i(f) = \min\{T_j(f + v \to 0), T_j(f + v \to 1), T_j(f + v \to X)\}$.

Join nodes. Suppose i is a join node with children j_1 and j_2. The following claim gives that we can compute T_i, given T_{j_1} and T_{j_2} in time $O(4^t t^{O(1)})$. As said, we later will improve the exponential factor to 3^t with help of convolutions.

▶ **Lemma 4.3.** *$T_i(f)$ is the minimum over all f_1 and f_2 of $T_{j_1}(f_1) + T_{j_2}(f_2) - \alpha$, where*

- *For all $v \in X_i$, $f(v) = X \Leftrightarrow f_1(v) = X \Leftrightarrow f_2(v) = X$.*
- *For all $v \in X_i$, $f(v) = 0 \Leftrightarrow f_1(v) = 0 \Leftrightarrow f_2(v) = 0$.*
- *For all $v \in X_i$, if $f(v) = 1$ then either $f_1(v) = 1$ and $f_2(v) = 0$, or $f_1(v) = 0$ and $f_2(v) = 1$.*
- *$\alpha = |\{v \in X_i \mid f(v) = X\}|$.*

Proof. The proof follows standard techniques for dynamic programming on tree decompositions. The number of elements in the vertex deletion set Z in G_i equals the number of elements in Z in G_{j_1} plus the number of elements in Z in G_{j_2}, minus the number of elements in Z in both — the latter number is α; we thus have to subtract α once to prevent counting vertices in $Z \cap X_i$ twice. ◀

The claim above shows that we can compute T_i given T_{j_1} and T_{j_2} in $O(4^t t^{O(1)})$ time: for each $v \in X_i$, there are four combinations to consider: $f_1(v) = f_2(v) = X$; $f_1(v) = f_2(v) = 0$; $f_1(v) = 1$ and $f_2(v) = 0$; $f_1(v) = 0$ and $f_2(v) = 1$. This gives $4^{|X_i|}$ combinations in total;

for each, look up the table entries in T_{j_1} and T_{j_2}, compute the value which arrives when we combine these entries. We initialize each value in T_i to ∞, and for each computed value, we set the value of the corresponding entry in T_i to the minimum of its current value and the just computed value.

Pseudoforest Deletion is finite integer index

We now discuss a small modification, that deletes some table entries which will never lead to an optimal solution. The modification shows that Pseudoforest Deletion is *finite integer index* (see [7]), and in fact, has the *de Fluiter property*, as defined by van Rooij [14, Chapter 11.2]. We do not give the formal definition of this property, but state the elements that are needed for our algorithm.

▶ **Lemma 4.4.** *Let i be a bag, and let f_X be the function, that maps each element of X_i to X.*

1. *For all $f : X_i \to \{0, 1, X\}$, $T_i(f_X) \leq T_i(f) + |X_i|$.*
2. *Let $f : X_i \to \{0, 1, X\}$. If $T_i(f) > T_i(f_X)$, then no partial solution at i with characteristic f will extend to an optimal solution.*

As a result, we have that we can ignore in our computations, all values for T_i that are larger than $T_i(f_X)$ without affecting the correctness of the algorithm. In the implementation, we just delete these entries from the tables or set there values to $T_i(f_X) + 1$. As a result, all values in a table T_i are in the range $T_i(f_X) - |X_i|, \ldots, T_i(f_X)$.

Using convolutions for Join Nodes

In order to speed up the dynamic programming algorithm, we use convolutions. The use of this technique in the setting of dynamic programming on tree decompositions was introduced by van Rooij et al. [15, 14].

Obtaining a constructive algorithm

As for many dynamic programming algorithms, constructing an optimal *solution* is done after computing its value, by traversing the tree top-down. We first select an entry from the root table T_r with minimum value, i.e., a function $f : X_r \to \{0, 1, X\}$ with $T_r(f) = \min_{f' : X_r \to \{0,1,X\}} T_r(f')$. We construct a solution corresponding to f by finding (a) 'corresponding' table entries in the child nodes, constructing partial solutions corresponding to these nodes, and placing the vertices in X_r with $f(v) = X$ in the pseudoforest deletion set. What are 'corresponding' table entries is different for the different types of nodes of a nice tree decompositions; e.g., for a forget node an entry corresponding to f is where the minimum in $\min\{T_j(f + v \to 0), T_j(f + v \to 1), T_j(f + v \to X)\}$ is attained. Obtaining these entries is trivial, except for join nodes.

For a join node i, we must solve the following problem: we are given an $f : X_i \to \{0, 1, X\}$, and must find f_1 and f_2 as in Lemma 4.3. It is easy to see, and for our purposes sufficient to notice that we can try all combinations f_1 and f_2, such that for all $v \in X_i$:

- If $f(v) = X$, then $f_1(v) = f_2(v) = X$.
- If $f(v) = 0$, then $f_1(v) = f_2(v) = 0$.
- If $f(v) = 1$, then ($f_1(v) = 1$ and $f_2(v) = 0$) or ($f_1(v) = 0$ and $f_2(v) = 1$).

These are at most 2^{t+1} different combinations to try; for each, we can see if these combine to f as in Lemma 4.3 in $O(t^{O(1)})$ time. With $O(n)$ nodes in the tree decomposition, the time to construct a solution after all tables T_i have been computed is bounded by $O(n2^t t^{O(1)})$.

(As a side remark, using self reduction (see [14, Chapter 12]) it is possible to avoid the factor exponential in t here and perform this step in $O(nt^{O(1)})$ time, but as the asymptotic running time is not dominated by this step, we prefer to give the simpler argument.)

Note that the algorithm remains correct when we run it on multigraphs with possible parallel edges and selfloops. This ends the proof of Theorem 4.1.

5 Main algorithm

In this section, we give the main algorithm and prove that it attains the $O(3^k nk^{O(1)})$ time bound. We first give the general outline of the algorithm (Section 5.1); then discuss two subroutines for two cases in Sections 5.2 and 5.3. Some implementation details and the time analysis will be discussed in Section 5.4.

5.1 Outline

We now give the overall outline of the algorithm. We have a recursive algorithm, that follows the cases of Theorem 3.6. In addition, we have a base case: if we have a graph $G = (V, E)$ with at most k vertices, we can just return V and are done. So suppose $|V| > k$. Let $t = k + 2$.

The algorithm first computes an arbitrary maximal matching M. If this matching M is large enough, i.e., has size $\frac{1}{O(t^8)}n = \frac{1}{O(k^8)}n$ as in the first case of Theorem 3.6, then we proceed with the subroutine discussed in Section 5.2. If this matching is not of this size, we compute the $k+3$-improved graph, and then find the set of simplicial vertices S of degree at most $k+3$. If set S is large enough, i.e., has size $\frac{1}{O(t^2)}n = \frac{1}{O(k^2)}n$ as in the second case of Theorem 3.6, then we proceed with the subroutine discussed in Section 5.3. If neither M nor S is large enough, we halt and reject: by Theorem 3.6, we know that G has treewidth at least $t + 1 = k + 3$, and hence G has no pseudoforest deletion set of size at most k; see Lemma 3.5.

We will discuss how each of the subroutines solves the problem when the corresponding case holds, and how this leads to an algorithm with the stated time bounds below.

5.2 Graphs with a large maximal matching

In this section, we suppose that we have a (maximal) matching M in $G = (V, E)$ with size $\frac{1}{O(k^8)}n$. We give a subroutine that either gives a pseudoforest deletion set of size at most k, or decides that G has no such set.

Let $G_M = (V_M, E_M)$ be the graph obtained by p-contracting all edges in M, i.e., we contract the edges but keep parallel edges and selfloops.

Now, recursively solve the problem on G_M. From Lemma 3.3, we have:

▶ **Lemma 5.1.** *Suppose G has a pseudoforest deletion set X. Let X_M be the set of vertices in G_M obtained from X by the p-contraction of edges. Then X_M is a pseudoforest deletion set of G.*

From Lemma 5.1, it follows that if our recursive call to G_M tells us that G_M has no pseudoforest deletion set of size at most k, then also G has no pseudoforest deletion set of size at most k, and thus we say 'no' and halt.

So, now assume that our recursive call gives us a pseudoforest deletion set S of G_M of size at most k. Let S' be the set of vertices that are contracted to S; i.e., if a vertex $v \in S$ is the result of contracting an edge from x to y, then we have $x, y \in S'$; if a vertex $v \in S$ is not the result of a contraction, then we place v in S'.

▶ **Claim 5.2.** *S' is a pseudoforest deletion set of size at most $2k$.*

We thus build S', and now we apply *iterative improvement*. Number the vertices in S', i.e., write $S' = \{v_1, v_2, \ldots, v_r\}$; we have $r \leq 2k$. Write $V_i = (V - S') \cup \{v_1, v_2, \ldots, v_i\}$, and $G_i = G[V_i]$. Note that $\{v_1, \ldots, v_k\}$ is a pseudoforest deletion set of G_k. Set $W = \{v_1, \ldots, v_k\}$.

Now, for $i = k + 1$ to r, do iteratively the following steps.

- Set $S = \{v_i\} \cup W$.
- *An invariant of the algorithm is that S is a pseudoforest deletion set of size at most $k + 1$ of G_i.*
- Compute a tree decomposition of G_i of width at most $k + 3$:
  - $G_i - S$ is a pseudoforest, so we can build a tree decomposition of $G_i - S$ of width at most 2 in linear time.
  - Add S to all bags of this tree decomposition.
- Run the algorithm of Theorem 4.1 and solve the Pseudoforest Deletion problem on G_i with parameter k.
- If this algorithm returns that G_i has no pseudoforest deletion set of size at most k, then G has no pseudoforest deletion set of size at most k, and we say 'no' and halt.
- Otherwise, let W be the pseudoforest deletion set of G_i that was obtained.

When we are done, we either have decided that G has no pseudoforest deletion set of size k, or we obtained a pseudoforest deletion set W of $G_r = G$ of size at most k.

5.3 Improved graphs with many simplicial vertices

We now suppose that we have the $k+3$-improved graph G', and a set Z with $\frac{1}{O(t^2)}n = \frac{1}{O(k^2)}n$ simplicial vertices of degree at most $k + 3$.

We recursively run the algorithm on $G' - Z$. If $G' - Z$ has no pseudoforest deletion set of size at most k, then G' has none, and hence, by Lemma 3.7 G has no pseudoforest deletion set of size at most k; we can halt and answer 'no'.

Otherwise, we obtain a pseudoforest deletion set of size at most k of $G' - Z$. We can thus build a tree decomposition of width at most $k + 2$ of $G' - Z$, as in Lemma 3.5. Build a tree decomposition of width at most $k + 2$ of G', by adding a bag with vertex set $N[z]$ for all $z \in Z$; making this bag adjacent to a bag that contains the clique $N(z)$. This is identical to an operation from the algorithm in [5]. As G is a subgraph of G', we now have a tree decomposition of G of width at most $k + 2$, and thus can run the dynamic programming algorithm from Theorem 4.1 on this latter tree decomposition. Return the answer of this algorithm.

5.4 Implementation and time analysis

We discuss here some implementation details. Each of the steps except for the recursive calls and the call to the dynamic programming algorithm of Theorem 4.1 can be done in $O(nk^{O(1)})$ time: finding a maximal matching and contracting a maximal matching is trivially within this time bound; how to find the improved graph, the simplicial vertices of bounded degree, and how to transform a tree decomposition of the graph without these simplicial vertices to one with the simplicial vertices (the main steps from Sections 5.2 and 5.3) in $O(nk^{O(1)})$ time is shown in [5]; we can use the same procedures as in [5] here. We call the

$O(n3^k k^{O(1)})$ dynamic programming algorithm $O(k)$ times, and thus the time per recursive call is bounded by $O(n3^k k^{O(1)})$. One call of the procedure makes one recursive call on a graph where we lost a fraction of $\frac{1}{O(k^8)}$ of the vertices, and thus our running time satisfies the following recurrence:

$$T(n) = T\left(n - \frac{1}{O(k^8)}n\right) + O(n3^k k^{O(1)}).$$

This resolves to $T(n) = O(n3^k k^{O(1)})$, which shows our main result Theorem 4.1.

▶ **Theorem 5.3.** *The problem, given a graph G and integer k, to decide if G has a pseudoforest deletion set of size at most k, and if so, find one, can be solved in $O(n3^k k^{O(1)})$ time.*

6 Concluding remarks

In this paper, we gave a fast parameterized algorithm for the Pseudoforest Deletion problem, with a running time with the currently best known factor depending on the parameter k, and a factor, linear in the number of vertices.

It is an interesting open problem whether this is (up to factors, polynomial in k) optimal, assuming the (Strong) Exponential Time Hypothesis, or whether a result similar to the lower bound proofs by Cygan et al. [8] can show that there is no $O((3-\epsilon)^t n^{O(1)})$ algorithm for Pseudoforest Deletion on graphs given with a tree (or path) decomposition of width t; compare the similar result for Feedback Vertex Set in [8].

A generalization of the Pseudoforest Deletion problem is the ℓ-Pseudoforest Deletion problem; a graph is an ℓ-pseudoforest, if it can be obtained from a forest by adding at most ℓ edges to each tree. It seems that the problem is harder when $\ell > 1$, as there is no apparent 'local formulation', whereas for $\ell = 1$, we have the formulation from Lemma 3.1. Thus, we wonder whether there exist deterministic algorithms for ℓ-Pseudoforest Deletion that run in $O(c_\ell^k n)$ time for constant c_ℓ depending on ℓ. Philip et al. [13] show that for every ℓ, ℓ-Pseudoforest Deletion has a kernel with $O(k^2)$ vertices. Given the local nature of Pseudoforest Deletion, it is interesting to see if there exists a kernel for it with a linear number of vertices.

Our result also implies a 2-approximation algorithm for Feedback Vertex Set, see below. There exist polynomial-time[2] 2-approximation algorithms for this problem [2, 1]; our algorithm uses linear time at the cost of a factor, exponential in k. The result can possibly be used as a first step in an fpt algorithm for Feedback Vertex Set using iterative compression, aiming at an algorithm that is efficient both in the term depending on k as well as in the term depending on n.

▶ **Corollary 6.1.** *There is a 2-approximation algorithm for* Feedback Vertex Set *that runs in $O(n3^k k^{O(1)})$ time.*

Proof. Run the algorithm of Theorem 5.3. If G has no pseudoforest deletion set of size at most k, then G also has no feedback vertex set of size at most k. Otherwise, let X be a pseudoforest deletion set of size at most k. If $G - X$ contains more than k cycles, then $G - X$ has no feedback vertex set of size k; otherwise, choose a set Y with one vertex per cycle in $G - X$; $X \cup Y$ is a feedback vertex set in G of size at most $2k$. ◀

[2] $O(m + n \log n)$-time [2] and $O(\min\{n^2, m \log n\})$-time [1].

References

1 V. Bafna, P. Berman, and T. Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics*, 12:289–297, 1999.

2 A. Becker and D. Geiger. Optimization of Pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem. *Artificial Intelligence*, 83:167–188, 1996.

3 Ann Becker, Reuven Bar-Yehuda, and Dan Geiger. Randomized algorithms for the loop cutset problem. *Journal of Artificial Intelligence Research*, 12:219–234, 2000.

4 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: Fast subset convolution. In *Proceedings of the 39th Annual Symposium on Theory of Computing, STOC 2007*, pages 67–74, 2007.

5 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.

6 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015.

7 Hans L. Bodlaender and Babette van Antwerpen-de Fluiter. Reduction algorithms for graphs of small treewidth. *Information and Computation*, 167:86–119, 2001.

8 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proceedings of the 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011*, pages 150–159, 2011.

9 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar F-deletion: Approximation, kernelization and optimal FPT algorithms. In *Proceedings of the 53rd Annual Symposium on Foundations of Computer Science, FOCS 2012*, pages 470–479, 2012. doi:10.1109/FOCS.2012.62.

10 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Inf. Process. Lett.*, 114(10):556–560, 2014.

11 Ljubomir Perković and Bruce Reed. An improved algorithm for finding tree decompositions of small width. *International Journal of Foundations of Computer Science*, 11:365–371, 2000.

12 Geevarghese Philip, Ashutosh Rai, and Saket Saurabh. Generalized pseudoforest deletion: Algorithms and uniform kernel. In *40th International Symposium on Mathematical Foundations of Computer Science 2015, MFCS 2015*, volume 9235 of *Lecture Notes in Computer Science*, pages 517–528. Springer Verlag, 2015.

13 Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar. Polynomial kernels for dominating set in graphs of bounded degeneracy and beyond. *ACM Transactions on Algorithms*, 9(1):11, 2012.

14 Johan M. M. van Rooij. *Exact Exponential-Time Algorithms for Domination Problems in Graphs*. PhD thesis, Utrecht University, 2011. URL: dspace.library.uu.nl/bitstream/handle/1874/205442/rooij.pdf.

15 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In Amos Fiat and Peter Sanders, editors, *Proceedings of the 17th Annual European Symposium on Algorithms, ESA 2009*, pages 566–577. Springer Verlag, Lecture Notes in Computer Science, vol. 5757, 2009.