Discrete Applied Mathematics 236 (2018) 42–56

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

A faster parameterized algorithm for Pseudoforest Deletion☆

Hans L. Bodlaender [a,b,*], Hirotaka Ono [c], Yota Otachi [d]

[a] Department of Information and Computing Sciences, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, Netherlands
[b] Department of Mathematics and Computer Science, University of Technology Eindhoven, P.O. Box 513, 5600 MB Eindhoven, Netherlands
[c] Graduate School of Informatics, Nagoya University, Furo-cho, Chikusa-ku, Nagoya, 464-8601, Japan
[d] Faculty of Advanced Science and Technology, Kumamoto University, 2-39-1 Kurokami, Chuo-ku, Kumamoto, 860-8555, Japan

ARTICLE INFO

Article history:
Received 17 January 2017
Received in revised form 14 August 2017
Accepted 12 October 2017
Available online 6 December 2017

Keywords:
Graph algorithms
Fixed parameter tractability
Pseudoforest
Feedback vertex set
Treewidth

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. (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.

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 derives its interest by its relation to the well studied Feedback vertex set problem, where we want to delete at most k vertices from a graph so that the graph becomes a forest. Allowing one more edge per connected component of the graph after removing the deletion set changes the problem significantly, and, somewhat surprisingly, seems to make it simpler in the parameterized setting, as the running time of our algorithm is smaller than that of the best known parameterized algorithms for Feedback Vertex Set.

The Pseudoforest Deletion problem was first studied by Philip et al. [14], 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 $O(k^2)$ vertices, with the constant factor growing with ℓ. For the Pseudoforest Deletion problem, i.e., the case that $\ell = 1$, they give a deterministic algorithm with running time $7.56^k n^{O(1)}$.[1]

☆ 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. An extended abstract of this paper appeared in the proceedings of IPEC 2016 (Bodlaender, 2016) [7].

* Corresponding author at: Department of Information and Computing Sciences, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, Netherlands.
E-mail addresses: h.l.bodlaender@uu.nl (H.L. Bodlaender), ono@nagoya-u.jp (H. Ono), otachi@cs.kumamoto-u.ac.jp (Y. Otachi).

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

https://doi.org/10.1016/j.dam.2017.10.018
0166-218X/

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. [10], 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. [14]), 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. [2] for the Feedback Vertex Set problem and a related problem called the Loop Cutset problem. Fomin et al. [10] consider a large class of problems that includes Pseudoforest Deletion; they show that each problem in this class has a deterministic algorithm that runs in time $O(2^{O(k)}n\log^2 n)$, and a constant-factor approximation algorithm that runs in time $O(nm)$. If one looks closely at the randomized algorithm by Becker et al. [2] and the generalization by Fomin et al. [10], it follows that one can solve the Pseudoforest deletion problem with a randomized algorithm in $O(4^k nk^{O(1)})$ 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 [9] or the rank based approach [6]. With help of *convolutions* [17] (see also [4]), the running time of the dynamic programming algorithm is reduced to $O(3^t nt^{O(1)})$ 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 $O(3^k nk^{O(1)})$ algorithm, while Feedback Vertex Set can be solved in $O(3^k n^{O(1)})$ time with a randomized algorithm [9] and $O(3.63^k n^{O(1)})$ time with a deterministic algorithm [12]; 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 few graph theoretic observations. The main algorithm is given in Section 4. It uses as a subroutine a dynamic algorithm that solves the Pseudoforest Deletion problem when a tree decomposition of bounded width is available. The details of this subroutine are given in the Appendix. In Section 5, we discuss a corollary of our result and an ILP formulation of the problem. Some conclusions are given in Section 6.

2. Preliminaries

When not specified otherwise, a graph $G = (V, E)$ is considered to be undirected, but possibly with self-loops and parallel edges. Allowing self-loops 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 $G[W] = (W, \{e \in E \mid \text{ both endpoints of } e \text{ belong to } W\})$. The subgraph of $G = (V, E)$ induced by all vertices except those in W is denoted by $G \setminus W$, i.e., $G \setminus W = G[V \setminus W]$.

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 self-loops, we can just ignore them, i.e., a tree decomposition of a graph with parallel edges and self-loops is a tree decomposition of the associated simple graph (obtained by keeping only one of each set of parallel edges and removing all self-loops).

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.

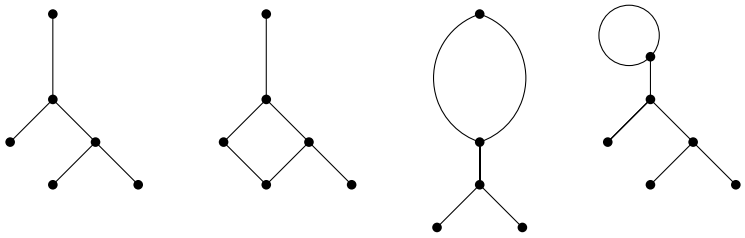

Fig. 1. A pseudoforest.

- **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 exactly one introduce edge node for each parallel edge. E.g., if there are two edges between v and w, we have exactly two edge introduce nodes for the pair v, w; typically, one of these can be the parent of the other in the tree. A self-loop 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 self-loop to G_j.
- **Forget** nodes i. i has one child, say j. There is a vertex v with $X_i = X_j \setminus \{v\}$. 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 $G_r = G$.

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 self-loops and parallel edges, this edge may be a self-loop or a parallel edge. A graph is a *pseudoforest*, if each connected component is a pseudotree. An example is given in Fig. 1.

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

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.

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 obtained by changing f by mapping v to i.

3. Graph theoretic observations

In this section, we give some easy graph theoretic results.

Lemma 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 one.*
3. *Each connected component of G has equally many vertices and edges, or its number of vertices is one larger than its number of edges.*

Proof. (1) $\Rightarrow$ (2): Consider a connected component of G. In case this connected component is a tree, then choose a root, and direct all edges towards the root. If we have a connected component obtained by adding an edge $\{v.w\}$ to a tree, then this edge closes a cycle. Orient all edges in this cycle in one direction, and orient all other edges towards the cycle. See Fig. 2.

(2) $\Rightarrow$ (3): Consider a connected component of G. Suppose it has r vertices and s edges. By connectivity, $s \geq r - 1$. As each vertex has outdegree at most one, $s \leq r$.

(3) $\Leftrightarrow$ (1): This follows directly from the well-known fact that a connected graph with r vertices and s edges is a tree, if and only if $s = r - 1$. □

While Lemma 1 is an easy observation, it is a key point to our result: being a pseudoforest seems to be a global property, but 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.

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 self-loops by building a tree decomposition for the underlying simple graph, and then adding the self-loops and parallel edges in the obvious way.

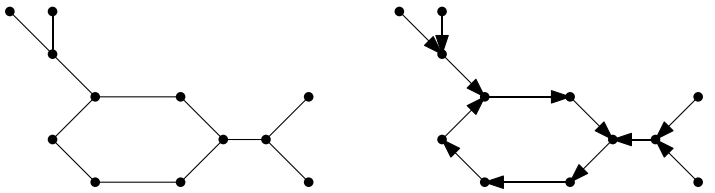

Fig. 2. Orienting the edges in a pseudoforest such that each vertex has outdegree at most one.

Lemma 2. *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 $O(k^2 r + |E|k)$ 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. *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 $G \setminus W$ 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 at most 2.

(2) Take a tree decomposition of width at most 2 of $G \setminus W$, and add W to all bags. □

4. A parameterized algorithm for PSEUDOFOREST DELETION

In this section, we give the main algorithm and prove that it attains the $O(3^k nk^{O(1)})$ time bound. We use a number of subroutines, discussed in a number of subsections. The main algorithm is given in Section 4.3. Section 4.6 discusses some implementation details and the running time. Other subsections discuss necessary subroutines. More precisely, we have the following subroutines:

- PFD. The main procedure. Arguments are a graph $G = (V, E)$, and an integer k. The procedure returns 'No', if G has no pseudoforest deletion set of size at most k, and otherwise it returns a minimum size pseudoforest deletion set of G. See Section 4.3.
- MAKESPARSE. Its argument is a graph $G = (V, E)$, and it outputs a subgraph G' of G, which has $O(n)$ edges and the same collection as pseudoforest deletion sets. See Section 4.2.
- PFD-DP. This procedure solves the problem when a tree decomposition of G is given. Arguments are a graph $G = (V, E)$, an integer k, and a tree decomposition TD of G. The output is a minimum size pseudoforest deletion set of G.
- PFD-LARGEMATCHING. This procedure is called from the main procedure, with arguments a graph $G = (V, E)$, an integer k, and a matching M in G. The procedure returns 'No', if G has no pseudoforest deletion set of size at most k, and otherwise it returns a minimum size pseudoforest deletion set of G. See Section 4.4.
- PFD-MANYSIMPLICIAL. This procedure is called from the main procedure, with arguments a graph $G' = (V, E)$, an integer k, and a set of simplicial vertices in G. The procedure returns 'No', if G has no pseudoforest deletion set of size at most k, and otherwise it returns a minimum size pseudoforest deletion set of G. See Section 4.5.

4.1. Dynamic programming on tree decompositions

In this section, we briefly discuss a subroutine of our algorithm

Theorem 4. *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.*

In the pseudo-code in the following sections, we call the procedure of Theorem 4 PFD-DP. It gets as input a graph G, an integer k, and a tree decomposition of G, and either outputs 'No', if G has no pseudoforest deletion set of size at most k, and otherwise it outputs a minimum size pseudoforest deletion set of G.

The algorithm is a more or less standard dynamic programming algorithm on tree decompositions. The key insight is the alternative characterization of Lemma 1, i.e., we look for a set of vertices to be deleted and an orientation of the edges, such

that each vertex that is not deleted has outdegree at most one. This allows us to work with 'three states' per vertex: a vertex is deleted, has outdegree zero, or has outdegree one.

To obtain the stated running time, attention has to be paid to the computation in join nodes. Here, we need the use of convolutions, i.e., transformations between a standard dynamic programming table and an alternative representation of it (and back), where the alternative representation enables to execute a computation at a join node in $O(3^t t^{O(1)})$ time. For the further discussion and the further (lengthy and somewhat tedious) details, we refer the reader to the Appendix.

4.2. Keeping the graph sparse

We have a simple procedure to ensure that we always work with graphs with $O(n)$ edges, based upon the following lemma.

Lemma 5. *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 self-loop 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 self-loops with v as endpoint, then any pseudoforest deletion set contains v. □

Algorithm 1 Procedure to ensure that we have $O(n)$ edges

```
1: procedure MakeSparse(G = (V, E))
2:     while there is a pair v, w with four or more parallel edges {v, w} do
3:         Remove one of the edges {v, w} from G.
4:     end while
5:     while there is a vertex v with three or more self-loops do
6:         Remove one of the self-loops from v
7:     end while
8:     return G
9: end procedure
```

Lemma 6. *If $G = (V, E)$ has a pseudoforest deletion set S of size at most k, then G has after running* MakeSparse*(G) less than $4kn$ edges.*

Proof. There are at most $n - k$ edges with both endpoints in $V \setminus S$, as $G \setminus S$ is a pseudoforest. In addition we have at most $3 \cdot k(k-1)/2$ edges between vertices in S, $3 \cdot k \cdot (n-k)$ edges between vertices in S and $V \setminus S$, and $2k$ self-loops with the endpoint in S. This totals to less than $4kn$. □

4.3. Outline of the main algorithms

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 7. Perković and Reed [13] showed that the result can be improved with respect to factors polynomial in k; for our purposes, the form below suffices.

Theorem 7 (*Bodlaender [5]*)**.** *There are constants c_1 and c_2, such that for each graph $G = (V, E)$ and integer t, at least one of the following three statements is true.*

- *Any maximal matching of G has at least $\frac{1}{c_1 \cdot t^8} n$ edges.*
- *The t-improved graph of G has at least $\frac{1}{c_2 \cdot t^2} n$ simplicial vertices of degree at most t.*
- *The treewidth of G is at least $t + 1$.*

The overall outline of the algorithm follows the cases of Theorem 7, with an additional simple base case. See Algorithm 2. The function PFD either returns a pseudoforest deletion set of G of size at most k, or 'No', if G has no pseudoforest deletion set of size at least k. There are five cases for the algorithm:

1. If G has at least $4kn$ edges after a call to MakeSparse, then we can answer 'No' (Lemma 6).
2. If $|V| \leq k$, then V is a pseudoforest deletion set of size k, and we are done.
3. If the maximal matching M in G is 'sufficiently large', i.e., of size at least $\frac{1}{c_1((k+2)^8)}n$, then the algorithm proceeds with the subroutine discussed in Section 4.4.
4. If the $k+3$-improved graph G' of G has sufficiently many vertices of degree at most $k+3$ that are simplicial, then the algorithm proceeds with the subroutine discussed in Section 4.5.
5. If none of the cases above applies, then Theorem 7, with $t = k+3$ gives that the treewidth of G is at least $k+4$, and hence the minimum size of a pseudoforest deletion set in G is at least $k+2$. (Recall Lemma 3: the treewidth of a graph is at most two larger than the size of a pseudoforest deletion set.) So, we safely can return 'No'.

Algorithm 2 A recursive algorithm for Pseudoforest Deletion

```
1: procedure PFD(G = (V, E), k)
2:     G = MakeSparse(G).
3:     if |E| ≥ 4k · |V| then
4:         return 'No'
5:     end if
6:     if |V| ≤ k then
7:         return |V|
8:     else
9:         Compute a maximal matching M.
10:        if |M| ≥ 1/(c_1·(k+2)^8) n then
11:            return PFD-LargeMatching(G, k, M)
12:        else
13:            Compute the k + 3-improved graph G' of G.
14:            Compute the set S of vertices that have degree at most k + 3 and are simplicial in G'.
15:            if |S| ≥ 1/(c_2·(k+2)^2) n then
16:                return PFD-ManySimplicial(G', k, S)
17:            else
18:                return 'No'.
19:            end if
20:        end if
21:    end if
22: end procedure
```

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.

4.4. Graphs with a large maximal matching

In this section, we suppose that we have a (maximal) matching M in $G = (V, E)$ with size $\frac{1}{O(k^8)}n$, and show how we can solve the Pseudoforest Deletion problem in this case, with help of recursive calls to the main procedure and with calls to a dynamic programming algorithm on tree decompositions. Before we start with explaining the algorithmic ideas, we need a number of additional definitions and lemmas.

A *p-contraction* ('parallel 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 self-loop. Note that the number of edges of a graph drops by exactly one when doing a p-contraction.

Lemma 8. *Let graph G' be obtained from graph G by p-contracting an edge.*

1. *G is a pseudotree, if and only if G' is a pseudotree.*
2. *G is a pseudoforest, if and only if G' is a pseudoforest.*

Proof. (1) A p-contraction affects one of the connected components of G. In this component, the number of vertices and the number of edges both decrease by one, and thus, by the characterization of Lemma 1, this component is a pseudotree after the contraction, if and only if it is a pseudotree before the contraction. □

(2) follows directly from (1).

From Lemma 8, we directly obtain the following lemma.

Lemma 9. *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 $W \setminus \{x\} \cup \{v, w\}$ is a pseudoforest deletion set in G.*

Let M be a set of edges, and $W \subseteq V$ be a set of vertices. The set W_M that *results from p-contracting* M is obtained as follows: if a vertex $v \in W$ is not an endpoint of an edge in W, then $v \in W_M$. If a vertex in $v \in M$ is endpoint of an edge in W, and it is contracted (by one or more p-contractions) to a vertex x, then $x \in W_M$. All vertices in W_M are obtained in this way. I.e., we repeat the steps in Lemma 9 for each edge in W. By applying Lemma 9 to each edge in M, we obtain the following result.

Lemma 10. *Suppose G has a pseudoforest deletion set X. Let M be a set of edges in G. Let G_M be obtained from G by p-contracting the edges in M. 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.*

A subroutine that gives a pseudoforest deletion set of size at most 2k. We are now ready to give the first subroutine in this section. It receives as input a graph G, an integer k, and a matching M, and either returns that G has no pseudoforest deletion set of size at most k, or it outputs a pseudoforest deletion set of G of size at most $2k$. See Algorithm 3 for the pseudo-code. The algorithm contracts the edges in M and recursively calls the main procedure on the smaller graph G' obtained by contracting the edges. If G' has no pseudoforest deletion set of size at most k, then by Lemma 10, G also has no pseudoforest deletion set of size at most k, and we can safely return 'No'. Otherwise, we can assume that our recursive call gave us a pseudoforest deletion set A of G_M of size at most k. We compute the set S of vertices that are contracted to A; i.e., if a vertex $v \in A$ is the result of p-contracting an edge from x to y, then we have $x, y \in S$; if a vertex $v \in A$ is not the result of a p-contraction, then we place v in S.

Claim 11. *Let A, S, G as above. If A is a pseudoforest deletion set in G_M of size at most k, then S is a pseudoforest deletion set in G of size at most $2k$.*

Proof. Each vertex in A can belong to S or have two vertices in S contracted to it, so $|S| \leq 2|A|$.

Consider the graph $G \setminus S$. If we p-contract in this graph all edges in M belonging to this graph, then we obtain $G_M \setminus A$. As A is a pseudoforest deletion set, $G_M \setminus A$ is a pseudoforest. By Lemma 8, $G \setminus S$ is a pseudoforest. So S is a pseudoforest deletion set in G. □

Algorithm 3 Subroutine when a large matching M is given

```
1: procedure PFD-LargeMatching(G = (V, E), k, M)
2:     Compute the graph G' = (V', E') obtained by p-contracting M in G.
3:     A = PFD(G', k)
4:     if A = 'No' then
5:         return 'No'
6:     else
7:                                          ▷ A is now a pseudoforest deletion set of G of size at most k
8:         Let S be the set of vertices contracted to A.
9:         return PFD-IterativeImprove(G, k, S)
10:    end if
11: end procedure
```

Iterative improvement: decreasing the pseudoforest deletion set size from 2k to k. We now give the second subroutine of Section 4.4. In the first subroutine, we obtained a pseudoforest deletion set of size $2k$. With help of dynamic programming on tree decompositions (Theorem 4) and the iterative improvement technique, we obtain an algorithm that either correctly tells that G has no pseudoforest deletion set of size at most k, or outputs such a set. The input of the subroutine is the graph G, the integer k, and a pseudoforest deletion set W of size at most $2k$. See Algorithm 4.

We number the vertices in W: $w_1, \ldots, w_{|W|}$. Let $G_i = G \setminus \{w_{i+1}, \ldots, w_{|W|}\}$, i.e., G_i is the subgraph of G, induced by $(V \setminus W) \cup \{w_1, \ldots, w_i\}$. Note that we obtain G_{i+1} from G_i by adding w_{i+1} and incident edges ($1 < i \leq |W|$).

Claim 12. *At Line 3, S is a pseudoforest deletion set of G_k of size at most k. At Line 6 of Algorithm 4, S is a pseudoforest deletion set of G_i of size at most $k + 1$.*

Proof. Line 3: As W is a pseudoforest deletion set of G, $\{w_1, \ldots, w_k\} = W \setminus \{w_{k+1}, \ldots, w_{|W|}\}$ is a pseudoforest deletion set of $G_k = G \setminus \{w_{k+1}, \ldots, w_{|W|}\}$.

Line 6: This invariant follows by an inductive argument. If $i = k + 1$, then this follows by the first part of this claim, and observing that we add w_i to the graph and the deletion set. Now suppose $i > k + 1$ and the claim holds for smaller i. At

Line 16 we have that S is a pseudoforest deletion set of G_i. Now, we increase i by one, thus add w_i to the graph (Line 4) and w_i to the deletion set (Line 5). □

At Line 8, we have a tree decomposition of G_i of width at most $|S| + 2 \leq k + 3$, see Lemma 3. At Line 11, we call the dynamic programming algorithm of Theorem 4. If this algorithm tells that G_i has no pseudoforest deletion set of size at most k, then also G has no pseudoforest deletion set of size at most k, and we safely can return 'No' (Lines 12 and 13).

We can conclude that PFD-IterativeImprove either correctly determines that G has no pseudoforest deletion set of size at most k, or it outputs a minimum size pseudoforest deletion set of G. Its running time is dominated by running at most k times the algorithm of Theorem 4.

Algorithm 4 Iterative improvement: from a pseudoforest deletion set of size at most $2k$ to one of size at most k

```
1:  procedure PFD-IterativeImprove(G = (V, E), k, W)
2:      Number the vertices in W, say W = {w_1, ..., w_|W|}.
3:      S = {w_1, ..., w_k}.
4:      for i from k + 1 to |W| do
5:          S = S ∪ {w_i}.
6:          Compute the graph G_i = G[(V \ W) ∪ {w_1, ..., w_i}].
7:          Build a tree decomposition of G_i \ S of width at most two.
8:          Add S to all bags of this tree decomposition.
9:          Let TD be the resulting tree decomposition.
10:                                  ▷ TD is a tree decomposition of G_i of width at most k + 3.
11:         A = PFD-DP(G_i, k, TD).
12:         if |A| > k then
13:             return 'No'.
14:         else
15:                                  ▷ A is now a pseudoforest deletion set of G_i of size at most k.
16:             S = A.
17:         end if
18:     end for
19:     return S.
20: end procedure
```

4.5. Improved graphs with many simplicial vertices

In this section, we look at the procedure PFD-ManySimplicial. This procedure is called from PFD (Algorithm 1) when the $k+3$-improved graph G' of G has 'many' simplicial vertices. More precisely, G' has at last $\frac{1}{c_2 \cdot (k+2)^2} n$ vertices that are simplicial and have degree at most $k + 3$.

In Algorithm 5, we either determine that G' has no pseudoforest deletion set of size k or find one of minimum size. Lemma 13 shows that this also gives the answer for the input graph G.

Lemma 13. *Let G be a graph and let k be an integer. Let G' be the $k + 3$-improved graph of G. Let X be a set of vertices. X is a pseudoforest deletion set of G, if and only if X is a pseudoforest deletion set of G'.*

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

Suppose that X is a pseudoforest deletion set of G. 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 three common neighbors that do not belong to X. See Fig. 3. 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 $G \setminus X$ remains a pseudoforest. □

The algorithm starts by looking at the subgraph G'' of G', obtained by removing the set of simplicial vertices from G'. We recursively solve the pseudoforest deletion set problem on G''. If G'' has no pseudoforest deletion set of size at most k, then G' has no pseudoforest deletion set of size at most k, as G'' is a subgraph of G. By Lemma 13, G has no pseudoforest deletion set of size at most k, so we correctly answer 'No' in Line 5.

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 $k + 3$.

Lemma 14 (*Folklore, See [5]*)**.** *Suppose W is a set of simplicial vertices in G with maximum degree k. Then the treewidth of G is at most the maximum of the treewidth of $G \setminus W$ and k.*

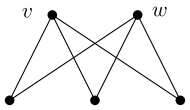

Fig. 3. A subgraph with five vertices and six edges. See the proof of Lemma 13.

Algorithm 5 Subroutine when we have many simplicial vertices in the improved graph

```
1: procedure PFD-ManySimplicial(G' = (V, E'), k, Q)
2:                                          ▷ Q is a set of simplicial vertices in G'.
3:     Compute the graph G'' = G' \ Q.
4:     A = PFD(G'', k)
5:     if A = 'No' then return 'No'
6:     else
7:                                          ▷ A is now a pseudoforest deletion set of G'' of size at most k.
8:         Build a tree decomposition of G'' of width at most k + 2.
9:         Build a tree decomposition TD of G' of width at most k + 3.
10:        A = PFD-DP(G', k, TD)
11:        if |A| > k then
12:            return 'No'
13:        else
14:            return A
15:        end if
16:    end if
17: end procedure
```

Proof. Consider a tree decomposition of $G \setminus W$. It is well-known that for each clique X, there is a bag containing X as a subset. For each vertex $v \in W$, find a bag i containing $N(v)$, and add a new bag with vertex set $v \cup N(v)$, making this bag adjacent to i. For details, see e.g., [5]. □

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 13, 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 $G' \setminus Z$. We can thus build a tree decomposition of width at most $k + 2$ of $G' \setminus Z$, as in Lemma 3 (Line 8). Build a tree decomposition of width at most $k + 3$ of G', following the construction of the proof of Lemma 14. As G is a subgraph of G', we now have a tree decomposition of G of width at most $k + 3$, and thus can run the dynamic programming algorithm from Theorem 4 on this latter tree decomposition, and return the answer of this algorithm.

4.6. Implementation and time analysis

We now discuss some implementation details and show the time bound. We first consider all steps except for the recursive calls:

- Finding a maximal matching, contracting a matching, executing the MakeSparse procedure can be clearly done in time, linear in $O(|V| + |E|)$. As we start with a simple graph, and ensure after contraction or improvement that the graph has a linear number of edges by calling MakeSparse, each of these steps uses $O(n)$ time.
- Finding 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 can be done in $O(nk^{O(1)})$ time [5]. We can use the same procedures as in [5] for these steps.
- PFD-IterativeImprove makes $O(k)$ calls to the dynamic programming algorithm PFD-DP, and hence uses $O(n3^k k^{O(1)})$ time.
- PFD-LargeMatching makes one recursive call to PFD with a graph with $n - \frac{1}{O(k^8)}n$ vertices and one call to PFD-IterativeImprove; its time, not counting the time by the recursive call, is bounded by $O(n3^k k^{O(1)})$.
- PFD-ManySimplicial makes one recursive call to PFD with a graph with $n - \frac{1}{O(k^2)}n$ vertices and one call to PFD-DP; its time, not counting the time by the recursive call, is bounded by $O(n3^k k^{O(1)})$.

The main procedure PFD makes either a call to PFD-LargeMatching or to PFD-ManySimplicial (or answers the problem); these can make one call to PFD but with a graph with at most $n - \frac{1}{O(k^8)}n$ vertices; with the remaining work bounded by

$O(n3^k k^{O(1)})$, 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.

Theorem 15. *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.*

5. Additional results

In this section, we discuss a corollary of our result and give a compact integer linear programming formulation of the Pseudoforest deletion problem.

5.1. Approximation of feedback vertex set

Our main result also implies an approximation algorithm for Feedback Vertex Set.

Corollary 16. *There is an algorithm, that given a graph G and integer k, gives a feedback vertex set of size at most $2k$ or tells that G has no feedback vertex set of size at most k, that runs in $O(3^k k^{O(1)} n)$ time.*

Proof. First run the algorithm of Theorem 15. 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, as each feedback vertex set is a pseudoforest deletion set.

Suppose the algorithm gives a pseudoforest deletion set W with $|W| \leq k$. Consider the graph $G \setminus W$. If $G \setminus W$ has more than k connected components that contain a cycle, then each feedback vertex set in G must contain at least one vertex of each of these cycles, hence G has no feedback vertex set of size at most k.

Otherwise, we choose from each cycle in $G \setminus W$ an arbitrary vertex. Let X be the set containing all vertices in W and the chosen vertices on the cycles in $G \setminus W$. X is a feedback vertex set in G of size at most $2k$. □

For the Feedback Vertex Set problem, polynomial-time 2-approximation algorithms are known [3,1]; these algorithms are not known to use linear time. The algorithm of Corollary 16 uses time linear in n, but at the cost of an additional exponential factor in k. Our 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.

5.2. ILP formulations

It may be interesting to note that the Pseudoforest Deletion problem has compact formulations as Integer Linear Programming problems. Given a graph G, the minimum size of a pseudoforest deletion set equals the solution of the following ILP.

$$\min \quad \sum_{i=1}^{n} x_i \tag{1}$$

$$\sum_j y_{ij} \leq 1 \qquad \text{for all } i \tag{2}$$

$$x_i + x_j + y_{ij} + y_{ji} \geq 1 \qquad \text{for all edges } \{v_i, v_j\} \in E \tag{3}$$

$$x_i \in \{0, 1\} \qquad \text{for all } i \tag{4}$$

$$y_{ij} \in \{0, 1\} \qquad \text{for all } ij \tag{5}$$

This formulation is based on the characterization of Lemma 1, i.e., we look for an orientation of the edges in $G \setminus X$ such that each vertex in $V \setminus X$ has outdegree at most one. We have a variable x_i for every vertex v_i, with $x_i = 1$ denoting that v_i belongs to the deletion set. For every edge $\{v_i, v_j\}$, we have two variables y_{ij} and y_{ji}. If $y_{ij} = 1$ then we orient the edge $\{v_i, v_j\}$ oriented from v_i to v_j; if $y_{ji} = 1$, then we orient this edge from v_j to v_i. However, if either $x_i = 1$ or $x_j = 1$, i.e., an endpoint of the edge belongs to the deletion set, then we do not need to orient this edge. I.e., for each edge $\{v_i, v_j\} \in E$, there are the following possibilities:

- The edge is oriented from v_i to v_j and neither v_i nor v_j is in the deletion set. In this case, we set $y_{ij} = 1$ and $y_{ji} = 0$.
- The edge is oriented from v_j to v_i and neither v_i nor v_j is in the deletion set. In this case, we set $y_{ij} = 0$ and $y_{ji} = 1$.
- v_i or v_j (or both) is in the deletion set. In this case, we set $y_{ij} = y_{ji} = 0$.

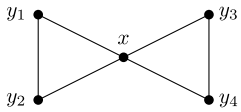

Fig. 4. The Butterfly graph.

One thus easily verifies that the ILP-formulation given above correctly models the Pseudoforest Deletion problem.

The formulation has an integrality gap of at least 4, i.e., there are graphs where the solution of the linear program obtained by replacing the conditions $y_{ij} \in \{0, 1\}$ by $y_{ij} \geq 0$ is at most 1/4th of the solution of the given ILP. An example of such a graph is a butterfly graph (see Fig. 4). The optimal pseudoforest deletion set has size one, namely $\{x\}$. The LP has a solution with value $\frac{1}{4}$: give x weight 1/4, all other vertices weight 0, and give arcs with head x weight 1/4 and other arcs weight 1/2.

The following variation is somewhat stronger. The integrality gap for the butterfly graph is 3. Here, an optimal solution is to assign $\frac{1}{3}$ to x, $\frac{1}{6}$ to the arcs with x as head, and $\frac{1}{2}$ to all other arcs.

$$\min \quad \sum_{i=1}^{n} x_i \tag{6}$$

$$x_i + \sum_j y_{ij} \leq 1 \qquad \text{for all } i \tag{7}$$

$$x_i + x_j + y_{ij} + y_{ji} \geq 1 \quad \text{for all edges } \{v_i, v_j\} \in E \tag{8}$$

$$x_i \in \{0, 1\} \qquad \text{for all } i \tag{9}$$

$$y_{ij} \in \{0, 1\} \qquad \text{for all } ij \tag{10}$$

It is unclear (and worth further study) whether these ILP formulations have algorithmic applications, and what the exact integrality gaps are for the formulations. Still, it is interesting that this problem can be expressed as such a compact ILP.

6. Concluding remarks

The main result of this paper is an $O(3^k \cdot k^O(1)n)$-time parameterized algorithm for the Pseudoforest Deletion problem. The dependency of the running time on k is the currently best known, while the dependency on the number of vertices is linear.

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. [9] 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 [9].

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 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. [15] 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.

Another interesting direction for further study is to explore what other problems give efficient parameterized algorithms that are linear in n and have a good running time in terms of the function of k, with help of the technique based on Theorem 7. See for instance the recent $O(2^{O(k^2)}n)$ algorithm for Pathwidth by Fürer [11].

Appendix. Solving Pseudoforest Deletion on tree decompositions

In this section, we will prove the following result, which was briefly discussed in Section 4.1.

Theorem 8. *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. [17], 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 G_i of G 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 $V_i \setminus Y$ 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* a 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 a 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 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 17. *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 and $X_i = X_j \cup \{v\}$.

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 one.

For a function f with $f(v) = 0$, we have $T_i(f) = T_j(f|_{X_j})$, and for functions f with $f(v) = X$, we have $T_i(f) = T_j(f|_{X_j}) + 1$. Indeed, 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 zero; 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 self-loops; the subroutine below is also correct in case the introduced edge is parallel to an existing edge or is a self-loop (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 $T_i(f) = T_j(f)$. (Only the case where the edge is oriented from v to w needs to be considered here.)
- 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 zero 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 zero 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 18. *$T_i(f)$ 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, which deletes some table entries that will never lead to an optimal solution. The modification shows that Pseudoforest Deletion is *finite integer index* (see [8]), and in fact, has the *de Fluiter property*, as defined by van Rooij [16, Chapter 11.2]. We do not give the formal definition of this property, but state the elements that are needed for our algorithm. Intuitively, Lemma 19 tells us that when we solve the Pseudoforest Deletion problem on graphs of treewidth t, then we can work with dynamic programming tables where for each table, the difference between the smallest and largest number in the table is at most t.

Lemma 19. *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.*

Proof. For (1), take a partial solution (Y, Λ) with characteristic f, and change it by placing every vertex in $X_i \setminus Y$ in Y, i.e., we take the partial solution $(Y \cup X_i, \Lambda)$. This partial solution has characteristic f_X, and thus $T_i(f_X) \leq |Y \cup X_i| \leq T_i(f) + |X_i|$.

For (2), suppose (Y, Λ) is a partial solution for i with characteristic f. Suppose it extends to a solution (Z, Λ'). Let (Q, Λ'') be a partial solution for i with characteristic f_X, and $|X| = T_i(f_X)$. Note that $X_i \subseteq Q$. We can use an exchange argument as follows: take $(Z \setminus Y \cup X, \Lambda''')$, where an edge in Λ''' is oriented as in Λ'' if it belongs to G_i and as in Λ' otherwise. As, whenever a vertex is incident to an edge oriented as in Λ' and an edge oriented as in Λ'', it belongs to X_i and hence to the deletion set $Z \setminus Y \cup Q$, and hence is an orientation with all vertices not in $Z \setminus Y \cup X$ of outdegree at most one. As $|X| = T_i(f_X) < T_i(f) \leq |Y|$, we have that $|Z \setminus Y \cup Q| < |Y|$ and thus (Z, Λ') is not 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 their 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. [17,16]. Our exposition will not be entirely self-contained, in order to avoid repeating lengthy details which can be found in [16, Chapter 12].

Suppose we have a join node i with children j_1 and j_2. For each $Z \subseteq X_i$, we call a subroutine; this subroutine will compute all values $T_i(f)$ for f with $f(v) = X \Leftrightarrow v \in Z$; i.e., we fix which vertices from X_i are mapped to X (are placed in the pseudoforest deletion set) and compute the table entries in T_i of this type.

The computation of these values then is almost identical to the computation for the number of perfect matchings for join nodes, as given by van Rooij [16, Chapter 11.4]. We give some main ideas here; for more details we refer to [16].

We first transform the tables T_{j_1} and T_{j_2} to tables T'_{j_1} and T'_{j_2}. Functions T' have three arguments:

- A function $f : X_i \setminus Z \to \{0, ?\}$. A '?' denotes that the vertex can be mapped to 0 *or* to 1.
- An integer α; using the finite integer index property, the range of the possible value of α has size $t + O(1)$. α will denote the 'target value', i.e., the size of the pseudoforest deletion set for corresponding partial solutions.
- An integer $\beta \in \{0, 1, \ldots, |X_i \setminus Z|\}$. β denotes the number of vertices with state 1.

Now, $T'_{j_1}(f, \alpha, \beta)$ equals the number of functions $f' : X_{j_1} \to \{0, 1, X\}$, such that

- For $v \in X_{j_1} = X_i$, $f'(v) = X \Leftrightarrow v \in Z$.
- For $v \in X_{j_1}$: $f(v) = 0 \Rightarrow f'(v) = 0$. (Note that the implication is only in one direction!)
- $T_{j_1}(f) = \alpha$.
- There are exactly β vertices v in $X_{j_1} = X_i$ with $f'(v) = 1$.

We define T'_{j_2} in exactly the same way.

The tables T'_{j_1} and T'_{j_2} can be computed in $O(2^{|X_i \setminus Z|})$ time. For details how this is done, see [16, Chapter 11.4]. We convert the tables T to T' 'coordinate-wise', i.e., we first build an intermediate table, similar to T' but with states $0, 1, X$, and then change this to a table with states $0, ?, X$ by changing the set of possible states for one vertex at the time. Each number to compute is either a copy of the value (if $f(v) = 0$), or one addition (if $f(v) = ?$, then we add the cases with $f'(v) = 0$ and $f'(v) = 1$).

We then compute a table T'_i. Again T'_i has three arguments: a function $f : X_i \setminus Z \to \{0, ?\}$, and values α, β, where $T'_i(f, \alpha, \beta)$ is the sum of $T'_{j_1}(f, \alpha', \beta') \cdot T'_{j_2}(f, \alpha'', \beta'')$ with

- $\alpha = \alpha' + \alpha'' - |Z|$. (The argument is similar as earlier if we take the union of a pseudoforest deletion set of size α' in G_{j_1} and a pseudoforest deletion set of size α'' in G_{j_2}, the size of this union is $\alpha' + \alpha''$ minus the size of the intersection of these two sets; this latter value is $|Z|$.)
- $\beta = \beta' + \beta''$.

We now change the table back to a table with entries $\{0, 1\}$ for $v \in X_i \setminus Z$. We do this coordinate-wise, again, with a copy of a value when $f(v) = 0$, and one subtraction when we want to have $f(v) = 1$: fix the value for all $w \neq v$, fix α, and subtract the value with $f(v) = 0$ from the value with $f(v) = ?$. Let T''_i be the resulting table. $T''_i(f, \alpha, \beta)$ denotes the number of ways a partial solution corresponding to a value in table T_{j_1} can be combined with a partial solution corresponding to a value in table T_{j_2} to obtain a partial solution for i with f giving the states of vertices in $X_i \setminus Z$, Z still the vertices in X_i in the pseudoforest deletion set, α the size of the pseudoforest deletion set, and β the number of vertices in $X_i \setminus Z$ with status 1 in the first partial solution (for G_{j_1}), plus the number of vertices in $X_i \setminus Z$ with status 1 in the second partial solution (for G_{j_2}).

Now, in these combinations, we possibly combined two partial solutions both with state 1. That would lead to a vertex with outdegree two. However, we can detect this with a technique from [17,16]: precisely in these cases, we do not have that β equals the number of vertices in $X_i \setminus Z$ with $f(v) = 1$. Thus, for f with $f(v) = X \Leftrightarrow v \in Z$, we have that $T_i(f)$ equals the minimum α such that there are $f' : X_i \setminus Z \to \{0, 1\}$, and β, such that

- $T''_i(f, \alpha, \beta) \geq 1$.
- For all $v \in Z$: $f(v) = X$; for all $v \in X_i \setminus Z, f(v) = f'(v)$.
- $\beta = |\{v \in X_i \setminus Z \mid f(v) = 1\}|$.

This allows us to compute the values of $T_i(f)$ for this choice of Z; we repeat the steps above for each $Z \subseteq X_i$.

For one choice of Z, the time for the computations is bounded by $O(2^{|X_i \setminus Z|} t^{O(1)})$; summing this over all $Z \subseteq X_i$ gives time $O(3^{|X_i|} |X_i|^{O(1)}) = O(3^t t^{O(1)})$.

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 '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 18. 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 18 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 [16, 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 self-loops. This ends the proof of Theorem 4.

References

[1] V. Bafna, P. Berman, T. Fujito, A 2-approximation algorithm for the undirected feedback vertex set problem, SIAM J. Discrete Math. 12 (1999) 289–297.
[2] A. Becker, R. Bar-Yehuda, D. Geiger, Randomized algorithms for the loop cutset problem, J. Artificial Intelligence Res. 12 (2000) 219–234.
[3] A. Becker, D. Geiger, Optimization of Pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem, Artificial Intelligence 83 (1996) 167–188.
[4] A. Björklund, T. Husfeldt, P. Kaski, M. Koivisto, Fourier meets Möbius: Fast subset convolution, in: Proceedings of the 39th Annual Symposium on Theory of Computing, STOC 2007, 2007, pp. 67–74.
[5] H.L. Bodlaender, A linear-time algorithm for finding tree-decompositions of small treewidth, SIAM J. Comput. 25 (1996) 1305–1317.
[6] H.L. Bodlaender, M. Cygan, S. Kratsch, J. Nederlof, Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth, Inform. and Comput. 243 (2015) 86–111.

[7] H.L. Bodlaender, H. Ono, Y. Otachi, A faster parameterized algorithm for pseudoforest deletion, in: J. Guo, D. Hermelin (Eds.), 11th International Symposium on Parameterized and Exact Computation, IPEC 2016, in: LIPIcs, vol. 63, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, pp. 7:1–7:12.
[8] H.L. Bodlaender, B. van Antwerpen-de Fluiter, Reduction algorithms for graphs of small treewidth, Inform. and Comput. 167 (2001) 86–119.
[9] M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J.M.M. van Rooij, J.O. 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, 2011, pp. 150–159.
[10] F.V. Fomin, D. Lokshtanov, N. Misra, S. Saurabh, Planar *F*-deletion: Approximation, kernelization and optimal FPT algorithms, in: Proceedings of the 53rd Annual Symposium on Foundations of Computer Science, FOCS 2012, 2012, pp. 470–479.
[11] M. Fürer, Faster computation of path-width, in: Proceedings 27th International Workshop on Combinatorial Algorithms, IWOCA 2016, in: Lecture Notes in Computer Science, vol. 9843, Springer, 2016, pp. 385–396.
[12] T. Kociumaka, M. Pilipczuk, Faster deterministic feedback vertex set, Inform. Process. Lett. 114 (10) (2014) 556–560.
[13] L. Perković, B. Reed, An improved algorithm for finding tree decompositions of small width, Internat. J. Found Comput. Sci. 11 (2000) 365–371.
[14] G. Philip, A. Rai, S. Saurabh, Generalized pseudoforest deletion: Algorithms and uniform kernel, in: 40th International Symposium on Mathematical Foundations of Computer Science 2015, MFCS 2015, in: Lecture Notes in Computer Science, vol. 9235, Springer Verlag, 2015, pp. 517–528.
[15] G. Philip, V. Raman, S. Sikdar, Polynomial kernels for dominating set in graphs of bounded degeneracy and beyond, ACM Trans. Algorithms 9 (1) (2012) 11.
[16] J.M.M. van Rooij, Exact Exponential-Time Algorithms for Domination Problems in Graphs, Utrecht University, 2011 (Ph.D. thesis).
[17] J.M.M. van Rooij, H.L. Bodlaender, P. Rossmanith, Dynamic programming on tree decompositions using generalised fast subset convolution, in: A. Fiat, P. Sanders (Eds.), Proceedings of the 17th Annual European Symposium on Algorithms, ESA 2009, in: Lecture Notes in Computer Science, vol. 5757, Springer Verlag, 2009, pp. 566–577.