Theory Comput Syst (2010) 46: 566–597
DOI 10.1007/s00224-009-9234-2

A Cubic Kernel for Feedback Vertex Set and Loop Cutset

Hans L. Bodlaender · Thomas C. van Dijk

Published online: 13 August 2009

Abstract The FEEDBACK VERTEX SET problem on unweighted, undirected graphs is considered. Improving upon a result by Burrage et al. (Proceedings 2nd International Workshop on Parameterized and Exact Computation, pp. 192–202, 2006), we show that this problem has a kernel with $O(k^3)$ vertices, i.e., there is a polynomial time algorithm, that given a graph G and an integer k, finds a graph G' with $O(k^3)$ vertices and integer $k' \leq k$, such that G has a feedback vertex set of size at most k, if and only if G' has a feedback vertex set of size at most k'. Moreover, the algorithm can be made constructive: if the reduced instance G' has a feedback vertex set of size k', then we can easily transform a minimum size feedback vertex set of G' into a minimum size feedback vertex set of G. This kernelization algorithm can be used as the first step of an FPT algorithm for FEEDBACK VERTEX SET, but also as a preprocessing heuristic for FEEDBACK VERTEX SET.

We also show that the related LOOP CUTSET problem also has a kernel of cubic size. The kernelization algorithms are experimentally evaluated, and we report on these experiments.

Keywords Graphs · Algorithms · Kernelization algorithms · Preprocessing · Data reduction · Feedback vertex set · Loop cutset · Polynomial kernels · Fixed parameter tractability

1 Introduction

The FEEDBACK VERTEX SET problem is a classic and fundamental graph problem, with several applications. See e.g., [18] for an overview paper on this and related

H.L. Bodlaender (✉) · T.C. van Dijk
Department of Information and Computing Sciences, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
e-mail: hansb@cs.uu.nl

T.C. van Dijk
e-mail: tcdijk@cs.uu.nl

problems. It is one of the problems that was proved to be NP-complete in Karp's seminal paper from 1972 [27]. In this paper, we consider the undirected and unweighted case of the problem. I.e., we are given an undirected graph $G = (V, E)$, and an integer k, and ask if there is a set of vertices S with $|S| \leq k$, such that each cycle of G contains at least one vertex from S. To facilitate the description of the algorithms, we allow G to have parallel edges and self loops.

A common approach to solving NP-hard problems in practical settings is to apply *preprocessing* or *data reduction*. Before applying some time consuming algorithm (e.g., based on integer linear programming techniques or branch and bound), first, a procedure is called that tries to transform the input to an equivalent and possibly smaller input. In this paper, we look to such preprocessing methods with techniques from the theory of *fixed parameter tractability*: we show that in polynomial time, an equivalent instance can be obtained whose size is only cubic in the parameter k, i.e., the requested upper bound on the size of the feedback vertex set. Such algorithms are called *kernelization algorithms*.

The theory of fixed parameter tractability was pioneered by Downey and Fellows to study the complexity of combinatorial problems where one part of the input, the *parameter* is assumed to be small. See [17, 31] for more information on this theory. A parameterized problem with input I and parameter k is said to be *fixed parameter tractable* (i.e., in FPT), if there is an algorithm that solves the problem in $p(|I|, k) \cdot f(k)$ time, where p is a polynomial and f an arbitrary function. Another possible manner of characterizing the class FPT is by means of kernelization algorithms. A kernelization algorithm takes an input-parameter pair, and transforms it to an equivalent input-parameter pair (called the *kernel*), such that for the latter, the size of the input is a function of the (possibly new) parameter, and the new parameter is at most the old parameter. The kernelization algorithm is supposed to run in time that is both polynomial in the size of the input and the value of the parameter. If we have a kernelization, then we can run any algorithm on the kernel, and obtain an algorithm that uses $O(p(n, k) + f(k))$ time, p a polynomial and f some function. The reverse also holds: if a parameterized problem belongs to FPT, then it has a kernelization algorithm. An excellent overview on kernelization was made by Guo and Niedermeier [25]. See e.g., [12, 30] for a kernel of size $2k$ for VERTEX COVER, [1, 2] for a linear kernel for DOMINATING SET on planar graphs, and [23] for a linear kernel for CLUSTER EDITING.

In a certain sense, kernelization is the common technique of preprocessing with in addition a mathematical guarantee on the quality of the preprocessing (the size of the input that remains after the preprocessing.) In this sense, kernelization is for preprocessing what approximation algorithms are for heuristics (i.e., an approximation algorithm can be seen as a heuristic with a guarantee for the quality).

As the time to solve the problem after kernelization would usually be exponential in the size of the kernel, we aim for kernels of small size: preferably polynomial in the parameter. Recently, a new technique has been developed that shows that several problems do not have a kernel of polynomial size, unless $coNP \subseteq NP/poly$ [10], e.g., LONGEST CYCLE.

FEEDBACK VERTEX SET is one of the problems known to be fixed parameter tractable. The problem was first shown to be in FPT by Downey and Fellows [16].

In a series of papers, faster FPT algorithms were obtained [4, 6, 15, 17, 24, 26, 33, 34]. The currently best known bounds (concentrating on the function of k), are a probabilistic algorithm that finds with high probability the feedback vertex set of size at most k, if existing, and uses $O(4^k kn)$ time [4], and a deterministic algorithm that uses $O(10.567^k p(n))$ time (p a polynomial) [15] (see also [24].) An exact algorithm for FEEDBACK VERTEX SET with a running time of $O(1.8899^n)$ was recently found by Razgon [35]. This was improved to $O(1.7548^n)$ time by Fomin et al. [19].

It was long open whether there existed a kernel or a kernel of size polynomial in k for the FEEDBACK VERTEX SET problem. This open problem was resolved in 2006 by Burrage et al. [11], who showed that the FEEDBACK VERTEX SET problem has a kernel of polynomial size, namely one with $O(k^{11})$ vertices. In this paper, we improve on the size of this kernel, and show that FEEDBACK VERTEX SET has a kernel with $O(k^3)$ vertices. The kernelization algorithm uses the 2-approximation algorithm for FEEDBACK VERTEX SET of [3] or [5] as a first step, and then uses a set of relatively simple reduction rules. A combinatorial proof shows that if no rule can be applied, then the graph has $O(k^3)$ vertices, where k is the parameter in the reduced instance. It can be expected that these rules can also be of use for practical preprocessing for the undirected Feedback Vertex Set problem, and our experiments confirm this.

Very recently, Thomassé [37] has further improved on the size of the kernel, showing a kernelization to $O(k^2)$ vertices.

Some of the techniques in this paper were taken from, or inspired by techniques from [11].

A problem, related to FEEDBACK VERTEX SET is the LOOP CUTSET problem. In the LOOP CUTSET problem, we are given a directed acyclic graph $G = (V, A)$, and an integer k, and ask if there is a set of vertices $S \subseteq V$ of size at most k, such that if we remove all arcs whose tail is in S, and then drop direction of edges, we obtain a forest. A loop cutset in a directed acyclic graph is always a feedback vertex set in the undirected graph obtained by dropping direction of edges, and many techniques for the FEEDBACK VERTEX SET problem carry over to the LOOP CUTSET problem. The main motivation for the LOOP CUTSET problem is in the algorithm of Pearl [32] for computing inference in probabilistic networks. We will show that the LOOP CUTSET problem also has a cubic kernel.

The technique we employ could be termed *kernelization by reduction*: we obtain the bound on the kernel for LOOP CUTSET by combinatorially reducing it to a variant of FEEDBACK VERTEX SET, namely BLACKOUT FEEDBACK VERTEX SET, then kernelizing this BLACKOUT FEEDBACK VERTEX SET instance, and then translating the obtained reduced instance back to an instance of LOOP CUTSET. Apart from the result itself, the employed technique is interesting, as this is one of the first times that problem transformation techniques are used to transform kernelization algorithms for one problem to kernelization algorithms for another problem.

This paper is organized as follows. Some preliminary definitions and results are given in Sect. 2. In Sect. 3, we give our main result: the kernelization algorithm and its analysis. In addition, we give a constructive version and discuss which of the reduction are necessary to obtain a cubic kernel. Some variants on the algorithm are discussed in Sect. 4. In Sect. 5, we discuss the kernelization algorithm for the LOOP

CUTSET problem. We report on computational experiments with the kernelization algorithm and variants in Sect. 6. Some final remarks are made in Sect. 7.

2 Preliminaries

Throughout this paper, graphs are undirected, and can have parallel edges and self-loops. A graph is a pair (V, E), with V the set of vertices, and E the multiset of edges. Each edge is an unordered pair (of possibly equal) vertices. Here, $\{v, w\}$ represents the same pair as $\{w, v\}$. A self-loop is an edge of the form $\{v, v\}$. A pair of vertices $\{v, w\}$ is called a *double edge* in a graph $G = (V, E)$ if E contains at least two edges of the form $\{v, w\}$. A pair $\{v, w\}$ is a *non-edge*, if there is no edge of the form $\{v, w\}$ in E, and it is a *non-double pair*, if there is at most one edge of the form $\{v, w\}$ in E.

A *cycle* is a sequence of alternating vertices and edges $v_0, e_1, v_1, e_1, v_2, \ldots, v_{r-1}, e_r, v_r$, such that $v_0 = v_r$, for each i, $1 \leq i \leq r$, $e_i = \{v_{i-1}, v_i\}$, and for all i, i', $1 \leq i < i' \leq r$, $v_i \neq v_{i'}$ and $e_i = e_{i'}$. I.e., we assume that cycles are simple. The length of a cycle is the number of edges on it. Note that if there are at least two parallel edges between vertices v and w, then we have a cycle of length two using v and w; otherwise there is no such cycle. Cycles are often also only denoted by sequences of successive vertices.

For a subset of the vertices $W \subseteq V$, the subgraph of graph $G = (V, E)$, *induced by* W is the graph $G[W] = (W, \{e \in E \mid e \subseteq W\})$. A graph G is a *forest*, if it does not contain a cycle.

Paths are defined similarly as cycles, but now we do not require that $v_0 = v_r$. v_0 and v_r are the *endpoints* of a path $v_0, e_1, v_1, e_1, v_2, \ldots, v_{r-1}, e_r, v_r$. $v_1, \ldots, v_{r-1}$ are the *internal vertices* of the path. A graph $G = (V, E)$ is *connected*, if there is a path between each pair of vertices in V. A tree is a forest that is connected. Two paths are said to be *vertex disjoint*, if all their internal vertices are different. (I.e., we allow that vertex disjoint paths share endpoints.)

A set of vertices $W \subseteq V$ is a *feedback vertex set* in $G = (V, E)$, if $G[V - W]$ is a forest, i.e., for each cycle in G, there is at least one vertex on the cycle that belongs to W.

The following trivial observation is regularly used in this paper.

Proposition 1 *Suppose there are at least two parallel edges $\{v, w\}$ in G. Let W be a feedback vertex set in G. Then $v \in W$ or $w \in W$.*

3 A Kernelization Algorithm for FEEDBACK VERTEX SET

In this section, we give the main kernelization algorithm for FEEDBACK VERTEX SET. We assume that we have as input a graph $G = (V, E)$, and an integer k. The algorithm either returns *no*, in which case we are sure that G has no feedback vertex set of size at most k, or a pair (G', k'), such that G has a feedback vertex set of size at most k if and only if G' has a feedback vertex set of size k', where $k' \leq k$. Alternatively, instead of returning *no*, the algorithm could return K_{k+3}, a clique with $k + 3$ vertices, as this graph has no feedback vertex set of size at most k.

The algorithm runs in time polynomial in $|V| + |E|$ and in k. The number of vertices and edges in G' is bounded by $O(k^3)$. It is also possible to give a constructive version, i.e., one where we can turn a minimum size feedback vertex set of G' of size at most k' into a minimum size feedback vertex set of G in polynomial time. This will be discussed in Sect. 3.5.

3.1 Structure of the Algorithm

The algorithm has two phases: an initialization phase, and an improvement and reduction phase. During the algorithm, we maintain a graph G, initially the input graph, and an integer k. During both phases, we may be able to determine that the graph has no feedback vertex set of value k, and return *no*. In the improvement and reduction phase, the value of k may decrease. When this happens, we restart the initialization phase, but now with the smaller number k and the modified graph.

During the algorithm, two sets of vertices play a special role. We call these sets A and B. A will invariantly be a feedback vertex set of G.

The improvement and reduction phase follows methodology that is typical for kernelization algorithms. We have a collection of rules, and we try to apply these rules until no rule is possible anymore. Rules can do one or both of the following two things:

- The rule *reduces* the instance: the instance is transformed to an equivalent, smaller instance. In our case, we remove some vertices or edges, and possibly decrease k.
- The rule *gains structural insight*. In our case, we have one such rule, the **Improvement Rule**. This rule adds some edges between vertices in $A \cup B$, possibly placing a vertex in B. In this case, we obtain a larger instance. The addition of these edges can possibly enable some reduction rules. In particular, an application of an **Improvement** rule brings us closer to a possible application of a **Reduction** rule. As these rules can happen a bounded number of times, and cannot be reversed by reduction rules, the algorithm terminates and still uses polynomial time. (For details, see the later discussion.)

When no rule can be applied, the resulting instance has $O(k^3)$ vertices and $O(k^3)$ edges. A detailed counting argument is needed to show this, see Sect. 3.4.

3.2 Initialization Phase

We assume that we are given a graph $G = (V, E)$, and an integer k.

If $k = 0$, then we return *yes* if G is a forest, and *no* otherwise. So, suppose $k \geq 1$.

The first step of the kernelization algorithm is to run the approximation algorithm of Bafna et al. [3] or that of Becker and Geiger [5]. These algorithms have a performance ratio of 2. Suppose this approximation algorithm returns a feedback vertex set A of G. If $|A| > 2k$, then from the performance ratio it follows that there is no feedback vertex set of size at most k, and we return *no*.

Otherwise, we continue with the next step, and also initialize the set B as $B = \{w \in V - A \mid \exists v \in A\colon$ there are at least two edges $\{v, w\} \in E\}$. I.e., if there is a double edge between a vertex $v \in A$ and a vertex $w \notin A$, then w is added to B.

Invariantly, B will be the set of vertices in $V - A$ that have a double edge to a vertex in A.

Initializing the set B can be done easily in $O(|V| + |E|)$ time using bucket sort.

3.3 Improvement and Reduction Rules

In this section, we give a number of improvement and reduction rules. Improvement rules add double edges to G; reduction rules remove edges and or vertices from G.

Each of the rules transforms the pair (G, k). We say that a rule is *safe*, if, whenever it transforms (G, k) to (G', k'), we have that G has a feedback vertex set of size k, if and only if G' has a feedback vertex set of size k'. In addition, we require that A is invariantly a feedback vertex set in the graph. We will show that each of the given rules is safe. For several rules, their safeness uses the following simple principle, earlier also used in [11].

Proposition 2 *Let $G = (V, E)$, and $v \in V$, such that there is at least one feedback vertex set W in G of minimum size with $v \in W$. Then a rule that removes v and its incident edges from G, and decreases k by one is safe.*

Proof Safeness follows from the fact that for each $W' \subseteq V - \{v\}$, W' is a minimum feedback vertex set in $G[V - \{v\}]$, if and only if $W' \cup \{v\}$ is a minimum feedback vertex set in G. □

In our description below, we assume always that we restart the initialization phase, whenever k is decreased by one. This just simplifies some counting arguments (in particular, it ensures that $|A| \leq 2k$); it is also possible to give a variant without such restarts.

3.3.1 Simple Rules

Each of the following rules makes a simple change to the graph. Many are taken from [11]. Safeness is easy to see, or follows directly from Proposition 2.

Rule 1 Islet Rule
If v is an isolated vertex, i.e., there is no edge incident to v, then remove v from G.

Rule 2 Twig Rule
If v has degree one, then remove v and its incident edge from G.

Rule 3 Triple Edge Rule
If there are three or more parallel edges of the form $\{v, w\}$, then remove all but two of these edges.

As a result of the **Triple Edge** rule, we have either 0, 1, or 2 edges between each pair of vertices when this rule cannot be applied.

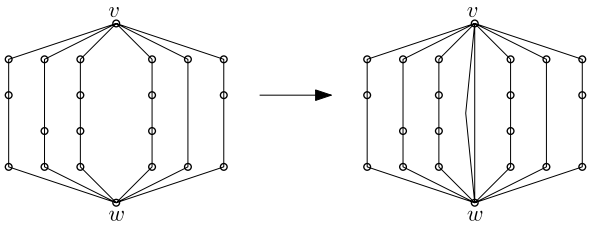

Fig. 1 Example of the **Improvement** rule

Rule 4 Degree Two Rule
Suppose v has degree two. Let the edges, incident to v be $\{v, w\}$ and $\{v, x\}$. Remove v, its incident edges, and add an edge $\{w, x\}$ to G. If $\{w, x\}$ becomes a double edge, then if $w \in A$, $x \notin A \cup B$, then add x to B, and if $x \in A$, $w \notin A \cup B$, then add w to B.

Note that the **Degree Two** rule can create a parallel edge or a self-loop.

Rule 5 Self-loop Rule
Suppose there is a self-loop $\{v, v\}$. Then remove v, all edges incident with v, and decrease k by one. Restart the initialization phase with the new graph and new value of k.

3.3.2 Improvement

An important rule is the **Improvement** rule. It is inspired by the improvement rule used in [7, 8, 14] in the context of algorithms to compute the treewidth of graphs. An example is given in Fig. 1.

Rule 6 Improvement Rule
Suppose $v \in A$, $w \in V$, $v \neq w$. Suppose there is no double edge between v and w, and that there are at least $k + 2$ vertex disjoint paths from v to w in G. Then add two edges $\{v, w\}$ to G. If $w \notin A \cup B$, then put w in B.

For a given pair of vertices, $v, w \in V$, one can compute in polynomial time the maximum number of vertex disjoint paths from v to w, using standard flow techniques. Nagamochi and Ibaraki [29] gave an algorithm that uses $O(k^2 n)$ time for checking if there are k vertex disjoint paths between a given pair of vertices v, w. See also [36, Chapt. 9].

Lemma 3 *Suppose there are at least $k + 2$ vertex disjoint paths from v to w in G. Then in each feedback vertex set of size S at most k, we have that $v \in S$ or $w \in S$.*

Proof Suppose S is a feedback vertex set of size at most k with $v, w \notin S$. There are at least two paths from v to w that do not contain a vertex in S. These form, with v and w a cycle that does not intersect S, so S is not a feedback vertex set, contradiction. □

Corollary 4 *The **Improvement** rule is safe.*

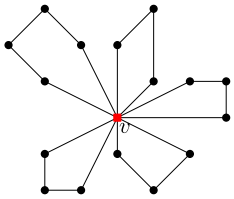

Fig. 2 Example of the **Flower** rule. All vertices may have neighbors outside the flower

3.3.3 Flower

Another important rule is the following. We first give the rule, and then comment on its particular form. An example is given in Fig. 2.

Rule 7 Flower Rule
Let $v \in A$. Suppose there is a collection of at least $k+1$ cycles in G, such that

- *Each cycle in the collection uses v.*
- *No pair of cycles in the collection shares a vertex, different from v.*

Then remove v and all its incident edges from G, and decrease k by one. Restart the initialization phase with the new graph and new value of k.

This rule looks for a set of cycles that are vertex disjoint, except that they intersect in the vertex v.

Suppose we want to verify whether the **Flower** rule can be applied for vertex $v \in A$. We can do this using generalized perfect matchings. Let $b : V \to N$ be a function. A *generalized perfect matching with respect to b* in $G = (V, E)$ is a set of edges $F \subseteq E$, such that each $v \in V$ is incident to exactly $b(v)$ edges in F. Let G_v^* be the graph, obtained by adding a triangle to each vertex $\neq v$, i.e., for each $w \neq v$, we add two new vertices w^1 and w^2, and three edges $\{w, w^1\}$, $\{w^1, w^2\}$, and $\{w^2, w\}$.

As the existence of generalized perfect matchings in a graph can be checked in polynomial time, see e.g., [20, Chapt. 7.1], the following lemma shows that we can determine in polynomial time if the **Flower** rule can be applied.

Lemma 5 *Let $G = (V, E)$ be a graph, $k \in \mathbf{N}$ and $v \in A \subseteq V$. The following three statements are equivalent.*

1. *Vertex $v \in V$ can be removed by an application of the **Flower** rule.*
2. *There is a function $b : V \to \mathbf{N}$, with for all $w \neq v$, $b(w) \in \{0, 2\}$, and $b(v) = 2k+2$, such that G has a generalized perfect matching with respect to b.*
3. *There is a generalized perfect matching with respect to b' in G_v^*, with for all $w \neq v$, $w \in V$: $b'(w) = 2$, $b'(w^1) = b'(w^2) = 1$, and $b'(v) = 2k+2$.*

Proof (1) $\to$ (2) Take a flower with exactly $k+1$ cycles that intersect precisely in v. Let F be the set of edges that belong to the cycles in the flower. Each vertex $w \neq v$ either belongs to one cycle and thus is incident to two edges in F, or does not belong to a cycle, and hence is incident to 0 edges in F. On each cycle, v is incident to two vertices, and this gives $2k+2$ different edges incident to v in F.

(2) → (1) Suppose we have such a generalized perfect matching $F \subseteq E$. Each of the $2k+2$ edges in F incident to v is the start of a path with edges in F that must visit v (each other vertex on this path is incident to exactly 2 edges in F). Each of these paths uses two edges in F, incident to v, and thus we have $k+1$ cycles through v that do not share vertices different from v.

(2) → (3) Suppose we have a generalized perfect matching $F \subseteq E$ with respect to b, with for all $w \neq v$, $b(w) \in \{0, 2\}$, and $b(v) = 2k+2$. Now, let F' be the set of edges in G^*, obtained by taking the edges in F, and then, for each $w \neq v$, if $b(w) = 0$, adding the edges $\{w, w^1\}$ and $\{w, w^2\}$, and if $b(w) = 2$, adding the edge $\{w^1, w^2\}$ to F'. F' now is the required generalized perfect matching with respect to b'.

(3) → (2) Suppose we have such a generalized perfect matching F' with respect to b' in G^*. Let $F = F' \cap E$. F is the required generalized perfect matching with respect to b: consider a $w \in V$, $w \neq v$. If $\{w^1, w^2\} \in F'$, then $\{w, w^1\} \notin F'$, and $\{w, w^2\} \notin F'$, hence two edges in $E \cap F'$ are incident to w. If $\{w^1, w^2\} \notin F'$, then $\{w, w^1\} \in F'$ and $\{w, w^2\} \in F'$, and no edges in $E \cap F'$ are incident to w. As v is incident to the same edges in G as in G^*, v is incident to $2k+2$ edges in F. □

A more precise estimate of the running time can be made as follows. The algorithm to find a generalized perfect matching first transforms the instance of GENERALIZED PERFECT MATCHING to an instance of PERFECT MATCHING. Each vertex $v \in V$ is replaced by $b(v)$ vertices, and each edge $\{v, w\} \in E$ is replaced by $b(v) \cdot b(w)$ edges. See e.g., [20, Chap. 7.1]; the transformation is due to Tutte [39]. Now, we can observe that if we apply this transformation to G_v^*, then the resulting graph still has $O(n)$ vertices and $O(m)$ edges. Thus, determining whether the flower rule can be applied for one vertex $v \in A$ costs as much as the fastest perfect matching algorithm on a simple undirected graph with $O(n)$ vertices and $O(m)$ edges. See e.g., [36, Chap. 24]. We can e.g., use the algorithm by Goldberg and Karzanov, that uses $O(n^{0.5} m \log_n \frac{n^2}{m})$ time [22].

We now argue safeness of the rule.

Lemma 6 *Let $v \in A$. Suppose there is a collection of at least $k+1$ cycles in G no two different cycles in the collection share another vertex except v. Then v belongs to each feedback vertex set S in G of size at most k.*

Proof Consider a feedback vertex set S in G with $v \notin S$. Then, each cycle in the collection contains a vertex in S, and these are all different vertices, so $|S| \geq k+1$. □

Corollary 7 *The **Flower** rule is safe.*

Proof This follows directly from Lemma 6 and Proposition 2. □

A simple special case of the **Flower** rule is the following.

Rule 8 Large Double Degree Rule
Suppose $v \in V$, such that there are at least $k+1$ vertices w with $\{v, w\}$ a double edge.

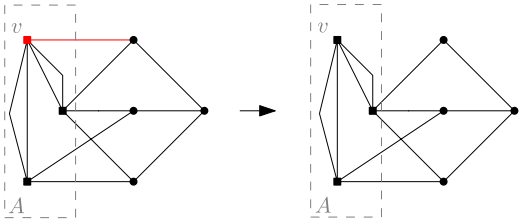

Fig. 3 Example of the **First Abdication** rule: v governs the piece to which it has one edge

Then remove v and its incident edges, and decrease k by one. Restart the initialization phase with the new graph and new value of k.

Safeness of the **Large Double Degree** rule follows directly from the safeness of the **Flower** rule.

3.3.4 Abdication

To describe the two abdication rules, we introduce some additional terminology. This terminology will also be used in the counting phase.

A *piece* is a connected component of $G[V - A - B]$. Let X be the set of vertices of a piece. The *border* of this piece is the set of vertices in $A \cup B$ that is adjacent to a vertex in X. A vertex v in the border of a piece *governs* the piece if it has a double edge to each other vertex $w \neq v$ in the border of the piece.

Rule 9 First Abdication Rule
Suppose $v \in A \cup B$ governs a piece with vertex set X. If there is exactly one edge with one endpoint v and one endpoint in X, (i.e., one edge of the form $\{v, w\}$ with $w \in X$), then remove this edge $\{v, w\}$ with $w \in X$ from G.

See Fig. 3 for an example. As a result of the **First Abdication** rule, v will no longer belong to the border of the piece.

Lemma 8 *The **First Abdication** rule is safe.*

Proof Let v, w, X be as in the **First Abdication** rule. Let G' be the graph obtained by removing the edge $\{v, w\}$.

We claim that for each set $S \subseteq V$, S is a feedback vertex set in G if and only if S is a feedback vertex set in G'. If S is a feedback vertex set in G, then, as G' is a subgraph of G, S is also a feedback vertex set in G'. Conversely, suppose S is a feedback vertex set in G'. Each cycle in G that is not a cycle in G' uses the edge $\{v, w\}$. Hence, if $v \in S$, S is also a feedback vertex set in G. Suppose $v \notin S$. As v governs the piece X, all vertices in the border of the piece except v must belong to S. Each cycle that uses the edge $\{v, w\}$ uses besides v one other border vertex of the piece (as v has only one edge to the piece, and the vertex set of a piece induces a tree), and thus contains a vertex in S. So, again S is a feedback vertex set.

As removing the edge $\{v, w\}$ does not change the collection of feedback vertex sets, the rule is safe. □

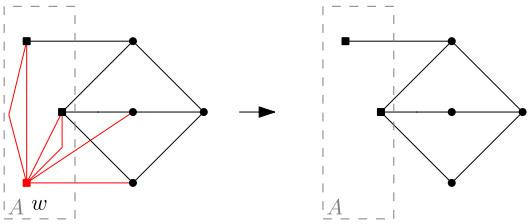

Fig. 4 Example of the **Second Abdication** rule: w governs the piece to which it has at least two edges

Rule 10 Second Abdication Rule
Suppose $v \in A \cup B$ governs a piece with vertex set X. If there are at least two edges with one endpoint v and one endpoint in X, then remove v and all its incident vertices from G, and decrease k by one. Restart the initialization phase with the new graph and new value of k.

An example is given in Fig. 4.

Lemma 9 *Suppose $v \in A \cup B$ governs a piece with vertex set X. Suppose there are at least two edges with one endpoint v and one endpoint in X. Then there is a minimum size feedback vertex set in G that contains v.*

Proof Consider a minimum size feedback vertex set S with $v \notin S$. As v governs the piece, all other border vertices of the piece must belong to S. Let w_1, w_2 be two neighbors of v in the piece. The path in X from w_1 to w_2 together with v forms a cycle, so there is at least one vertex in X that belongs to S, say x. Consider the set $S - \{x\} \cup \{v\}$. This is again a feedback vertex set: any cycle that contains x must contain a vertex of the border of X and hence a vertex in $S - \{x\} \cup \{v\}$. As $|S| = |S - \{x\} \cup \{v\}|$, there is a feedback vertex set of minimum size that contains v. □

From Proposition 2 and Lemma 9, we directly conclude the following.

Corollary 10 *The **Second Abdication** rule is safe.*

Consider a piece with vertex set X and border set Y, such that for each pair of disjoint vertices in Y, there is a double edge. Consider what happens when we apply the abdication rules to this piece. Each vertex in Y governs the piece. If $v \in Y$ has one edge to the piece, this edge will be removed, and v no longer is in the border of X. If $v \in Y$ has two or more edges to the piece, then v itself is removed. Thus, after all the vertices in Y have been handled, the border of X will be empty. A piece with an empty border is a connected component of G that is a tree: it is a subgraph of $G[V - A]$ and hence does not have a cycle. Now, repeated application of the **Twig** rule, and then an application of the **Islet** rule will remove all vertices in the piece.

Above, we have seen that the given rules remove each piece for which there is a double edge between each pair of disjoint vertices in its border. A direct consequence of this is the following lemma.

Lemma 11 *Suppose none of the Rules* 1–10 *can be applied to* G. *Suppose* $Y \subseteq V$ *is the border of a piece in* G. *Then there are two disjoint vertices* $v, w \in Y$ *such that* $\{v, w\}$ *is not a double edge*.

It is straightforward to see that we can check in polynomial time whether an abdication rule is possible for given G, A, B.

3.4 Size of the Kernel

As we will show below, Rules 1–10 will transform G to a kernel with $O(k^3)$ vertices, in case G has a feedback vertex set of size at most k.

A graph $G = (V, E)$, with sets A, B, and integer k is called *a reduced instance*, if none of the Rules 1–10 is applicable anymore.

We start with a number of lemmas.

Lemma 12 *In a reduced instance, there are at most* $2k$ *vertices in* A *and at most* $2k^2$ *vertices in* B.

Proof We start with a set A of size at most $2k$. During the algorithm, we recompute A whenever k is changed.

Each vertex in B has at least one neighbor in A to which it has a double edge, but no vertex in A has more than k neighbors to which it has a double edge, otherwise the **Large Double Degree** rule can be applied. So the result follows. □

We construct an auxiliary graph, which we call the B-piece graph. In the B-piece graph, there are two types of vertices: each vertex in B is a vertex in the B-piece graph, and for each piece, there is a vertex representing the piece in the B-piece graph. The B-piece graph is bipartite, with an edge between a vertex $v \in B$ and a vertex x representing a piece, if B is in the border of the piece.

Lemma 13 *The* B*-piece graph is a forest*.

Proof Suppose we have a cycle c in the B-piece graph. We transform this to a cycle in G, as follows. Consider a vertex v that represents a piece with vertex set X, and suppose it is incident to w_1 and w_2 in the cycle c. Thus, there is a path, using only vertices in X from w_1 to w_2. Replace w in c by this path. Doing this for each vertex that represents a piece in c gives a cycle c' in G. Note that there is no vertex in A on c'. Thus, A is not a feedback vertex set in G, contradiction. □

Lemma 14 *Let* $v \in B$ *be in the border of a piece with vertex set* X. *Then there is exactly one edge from* v *to a vertex in* X.

Proof By definition of border, there is at least one such edge. Suppose there are at least two edges from v to X, so to vertices w_1 and w_2. Consider the following cycle: start at v, go to w_1, then take the path using vertices in X to w_2, and then close the cycle at v. This is a cycle that does not use a vertex in A. Hence, A is not a feedback vertex set, contradiction. □

Lemma 15 *Suppose we have a reduced instance. There are at most* $8k^3 + 9k^2 + k$ *pieces.*

Proof We associate each piece to a non-double pair in its border. I.e., for each piece in the reduced instance, there are two border vertices $v, w \in A \cup B$, such that $\{v, w\}$ is not a double edge; we select one such pair and associate the piece with this pair. We now count the number of pieces that can thus be associated with the different types of pairs.

Case I: Non-double pairs $\{v, w\}$ *with* $v \in A$, $w \in A$. If $k + 2$ or more pieces are associated with a pair $\{v, w\}$, $v, w \in A$, then by the **Improvement** rule, $\{v, w\}$ would be a double edge. So, at most $k + 1$ pairs are associated with the pair, and hence we have at most $\frac{1}{2}|A|(|A| - 1) \cdot (k + 1) \leq k \cdot (2k - 1) \cdot (k + 1)$ pieces associated with pairs of this type.

Case II: Non-double pairs $\{v, w\}$, *with* $v \in A$, $w \in B$. This case needs the most detailed counting argument. We consider two subcases.

Case IIa: Non-double pairs $\{v, w\}$, *with* $v \in A$, $w \in B$, *to which at most one piece is associated.* Clearly, the number of pieces associated to these edges is at most $|A| \cdot |B| \leq 2k \cdot 2k^2 = 4k^3$.

Case IIb: Non-double pairs $\{v, w\}$, *with* $v \in A$, $w \in B$, *to which at least two pieces are associated.* Consider a vertex $v \in A$. Let $X_v = \{w \in B \mid$ at least two pieces are associated to the non-double pair $\{v, w\}\}$. For each $w \in X_v$, consider the cycle, that starts in v, moves to w through one of the pieces associated to $\{v, w\}$, and then moves back to w through another piece associated to $\{v, w\}$. This is a cycle, and for $w, w' \in X_v$, $w \neq w'$, the cycle of w only intersects the cycle of w' in v. So, if $|X_v| \geq k + 1$, we have a collection of at least $k + 1$ cycles that move through v, and pairwise have $\{v\}$ as intersection. Thus, the **Flower** rule would apply to v. As we have a reduced instance, we can assume that $|X_v| \leq k$.

So, for a fixed $v \in A$, at most k pairs $\{v, w\}$, $w \in B$, have at least two pieces associated to it. Thus, in total at most $|A| \cdot k \leq 2k^2$ such pairs have at least two pieces associated to it. The same argument as in Case I shows that each of these pairs has at most $k + 1$ pieces associated to it, and this gives a maximum of $2k^2 \cdot (k + 1)$ pieces for this case.

Case III: Non-double pairs $\{v, w\}$, *with* $v \in B$, $w \in B$. If we associate two pieces to one such pair $\{v, w\}$, then these pieces and v, w form a cycle in the B-piece graph, which contradicts Lemma 13. So, to each such pair, we associate at most one piece. Moreover, if a piece is associated to such a pair, it has at least two neighbors in the B-piece graph. As the B-piece graph is a forest, and there are at most $2|A| \cdot k \leq 4k^2$ vertices in B, the number of pieces adjacent to two vertices in B can be at most $4k^2$, and hence also at most $4k^2$ pieces can be associated to pairs of this type.

The total number of pieces in the reduced instance is the sum of the number of pieces over the possible cases: $k \cdot (2k - 1) \cdot (k + 1) + 4k^3 + 2k^2 \cdot (k + 1) + 4k^2 = 8k^3 + 7k^2 - k$. □

We now count the number of vertices and edges in the pieces. We partition the vertices that belong to a piece into two sets. C is the set of vertices in a piece (i.e., in $V - (A \cup B)$), that are adjacent to a vertex in $A \cup B$. D is the set of vertices in a piece that are adjacent to vertices in $C \cup D$ only, i.e., $D = V - (A \cup B \cup C)$. We first estimate the size of C; a simple argument then shows a bound on $|D|$.

For each $v \in A \cup B$, let $C_v = \{w \in C \mid \{v, w\} \in E\}$. I.e., C_v are the vertices in pieces, adjacent to v. The sets C_v are partitioned into two sets, $C_{v,1}$, and $C_{v,\geq 2}$. $C_{v,1}$ is the set of the vertices $w \in C_v$, such that w is the only vertex in its piece that is adjacent to v. $C_{v,\geq 2} = C_v - C_{v,1}$ is the set of vertices $w \in C_v$, such that the piece of w has at least one other vertex, also adjacent to v.

We give a number of different lemmas that give bounds on the size of these sets. In each case, we assume we have a reduced instance.

Lemma 16 *Let* $v \in A$. $|C_{v,1}| \leq 3k^2 + 3k - 1$.

Proof Consider a vertex $w \in C_{v,1}$, and the piece containing w. As v does not govern this piece (otherwise the **First Abdication** rule would be applicable), there is a vertex $x \in (A \cup B) - \{v\}$, in the border of the piece for which the pair $\{v, x\}$ is not a double edge. For each $w \in C_{v,1}$, we associate w with such a vertex x in the border of the piece of w with $x \neq v$ and $\{v, x\}$ is not a double edge. (If there is a choice for different vertices x in the border of a piece, one of these is chosen arbitrarily.)

No vertex $x \in A \cup B$ has more than $k + 1$ vertices associated to it in this way. Suppose x has $k + 2$ or more vertices in $C_{v,1}$ associated to it. For each vertex w in $C_{v,1}$, associated to x, we take the path, starting at v, and moving through the piece containing w to x. As each of these paths uses a different piece, we have $k + 2$ vertex disjoint paths from v to x, and hence the **Improvement** rule will add the double edge $\{v, x\}$. However, if $\{v, x\}$ is a double edge, no vertices in $C_{v,1}$ will be associated to x, contradiction.

There are at most k vertices in $A \cup B$ that have two or more vertices associated to it. Suppose we have at least $k + 1$ vertices in $A \cup B$ that have two or more vertices associated with it. For each vertex x in $A \cup B$ that has two or more vertices associated with it, we build a cycle as follows. Suppose X_1, X_2 are pieces containing vertices associated to x. Start at v, move through X_1 to x, move from x through X_2 to v. As all pieces used by these cycles are different, these cycles only intersect at v, and thus the **Flower** rule applies. This contradicts the assumption that we have a reduced instance.

Now we can count the number of vertices in $C_{v,1}$, or, equivalently, the total over all $x \in A \cup B - \{v\}$ of the number of vertices associated to x. There are at most $|A| + |B| - 1$ vertices to which one vertex is associated. There are at most k vertices in $A \cup B - \{v\}$ to which two or more vertices are associated, and to each, at most $k + 1$ vertices are associated. This gives a total of at most $2k + 2k^2 - 1 + k(k + 1) = 3k^2 + 3k - 1$. □

Lemma 17 *Let* $v \in A$. $|C_{v,\geq 2}| \leq k^2 + 3k + 3$.

Proof Suppose $v \in A$, and $|C_{v,\geq 2}| > k^2 + 3k + 3$. We will show that the instance is not reduced, thus arriving at a contradiction. In particular, we will show that the **Flower** rule can be applied. To do so, a collection of cycles $\mathcal{C}$ is built. Initially, $\mathcal{C} = \emptyset$.

A *region* is a maximal connected set of vertices in pieces (i.e., a subset of $V - (A \cup B)$, that does not contain any vertex on a cycle in $\mathcal{C}$. A region is *filled*, if it contains at least two vertices in $C_{v,\geq 2}$.

If we have a filled region Y, we build a cycle through v that is vertex disjoint with any cycle already in $\mathcal{C}$ in the following way. Start at v, then go to an arbitrary neighbor of v in the region. Suppose our path so far p is at vertex w. If w is adjacent to v, and is not the second vertex on the path. Then, we close the cycle by going back from w to v. Suppose now w is not adjacent to v, or is the second vertex on the path. For each neighbor x of w, except a neighbor of w that belongs to the path constructed so far, we define a number n_x. n_x is the number of vertices z in C_v for which there is a path from x to z that use only vertices in the region Y and that uses no vertex on p. (In particular, this means the path belongs to the subtree, obtained by removing the vertices on p from Y.) If there is a neighbor x of w with $n_x = 1$, x not on p, then select such a neighbor x as next vertex on the path p; set $w = x$. Otherwise, select any neighbor x of w with $n_x \geq 2$, x not on p. It is not hard to see, that selecting vertices in this way gives a cycle through the region. As the region does not contain vertices on existing cycles in $\mathcal{C}$, we invariantly have a set of cycles that only intersect at v.

After a cycle has been added, the set of regions and filled regions is changed. We consider the number of vertices in $C_{v,\geq 2}$, that belonged to a filled region, before the addition of a cycle to $\mathcal{C}$, but do no longer belong to a filled region after this addition. We claim that this number is at most $k + 2$. Clearly, the first vertex after v and the last vertex before closing the cycle at v are of this type. Suppose we go from w to x while building the cycle. Suppose y is a neighbor of w that is not used by the path that we are constructing. The new region formed by that path that contains y contains n_y vertices in C_v. Thus, if $n_y \geq 2$, this new region is filled. If $n_y = 0$, then no vertices in C_v are in this new region. So, the interesting case is when $n_y = 1$. By the algorithm we followed, we go to another neighbor x of w with $n_x = 1$. We claim that w has at most $k + 1$ neighbors y with $n_y = 1$. Suppose w has at least $k + 2$ neighbors y with $n_y = 1$. Then we can build a collection of $k + 2$ vertex disjoint paths from w to x, each going from w to a neighbor y of w with $n_y = 1$, and then following the path through the region to v. Thus, by the **Improvement** rule, w would have been added to B, which contradicts the assumption that w belongs to a piece. Thus, at this point, there are at most k vertices that play the role of y. Each amounts to one vertex in C_v that belonged to a filled region before the addition of the cycle, and belongs to a region with only one vertex in C_v after the addition. During the construction of the cycle, there is only one point where this can happen: after we selected an x with $n_x = 1$, all unvisited neighbors of the vertex on the path will have $n_z = 0$. So, by the addition of a cycle, two vertices in $C_{v,\geq 2}$ belong to a cycle, and at most k vertices belong to a region with only one vertex in $C_{v,\geq 2}$.

So, we can prove with induction that if there are at least $2 + \ell \cdot (k + 2)$ vertices in filled regions, we can build ℓ cycles in regions, vertex disjoint from cycles already in $\mathcal{C}$. (The base case $\ell = 0$ is trivial.)

When we start the construction, each piece containing vertices in $C_{v,\geq 2}$ is a filled region, and hence the procedure constructs ℓ cycles, when $|C_{v,\geq 2}| \geq 2 + \ell \cdot (k+2)$. As we assumed that $|C_{v,\geq 2}| \geq k^2 + 3k + 4 = 2 + (k+1)(k+2)$, we can construct a collection of $k+1$ cycles through v that intersect only at v. Thus, the **Flower** rule is applicable, which contradicts the assumption that we have a reduced instance. □

Lemma 18 $\sum_{v \in B} |C_v| \leq 8k^3 + 11k^2 + k - 1$.

Proof By Lemma 13, if $v \in B$, then $C_{v,\geq 2} = \emptyset$. So, for all $v \in B$, $C_v = C_{v,1}$.

Suppose $w \in C_{v,1}$, $v \in B$. There is an edge from the piece, containing w to v in the B-piece graph. As w is the only vertex in this piece with an edge to v, we can associate this edge to w. In this way, each vertex in $\bigcup_{v \in B} C_v$ has at least one edge from the B-piece graph associated to it, and hence $|\bigcup_{v \in B} C_v|$ is at most the number of edges of the B-piece graph. As the B-piece graph is a forest (Lemma 13), the number of edges in this graph is smaller than the number of vertices, which equals $|B|$ plus the number of pieces. By Lemmas 12 and 15, the result follows. □

Lemma 19 $|D| \leq |C| - 1$.

Proof Consider the graph $G[C \cup D]$, the subgraph of G, induced by $C \cup D$. As A is a feedback vertex set, this subgraph is a forest. Each vertex in D has the same degree in $G[C \cup D]$ as it has in G, by definition of D. So, in a reduced instance, each vertex in D has degree at least three in G and in $G[C \cup D]$. The number of vertices of degree at least three in a forest is less than the number of leaves in the forest, and each leaf in $G[C \cup D]$ must belong to C, hence $|D| < |C|$. □

Theorem 20 *In a reduced instance with $k \geq 1$, there are at most $32k^3 + 48k^2 + 8k - 3$ vertices and at most $44k^3 + 64k^2 + 12k - 4$ edges.*

Proof We use the results, shown above. We have $|A| \leq 2k$, $|B| \leq 2k^2$. For each $v \in A$, $|C_v| = |C_{v,1}| + |C_{v,\geq 2}| \leq 4k^2 + 6k + 1$. Thus, $|C| \leq \sum_{v \in A} |C_v| + \sum_{v \in B} |C_v| \leq 8k^3 + 12k^2 + 2k + 8k^3 + 11k^2 + k - 1 = 16k^2 + 23k^2 + 3k - 1$, and hence $|D| \leq 16k^3 + 23k^2 + 3k - 3$. We conclude that $|V| = |A| + |B| + |C| + |D| \leq 32k^3 + 48k^2 + 8k - 3$.

As $G[V - A]$ is a forest, there are at most $|V - A| - 1 \leq 32k^3 + 48k^2 + 8k - 4$ edges with no endpoint in A. Now, we only need to count the edges with at least one endpoint in A. There are at most $|A| \cdot (|A| + 1) \leq 4k^2 + 2k$ edges with both endpoints in A and at most $2|A| \cdot |B| \leq 4k^3$ edges with one endpoint in A and one endpoint in B. (Note that we must take the possibility of parallel edges into account.) By definition, there are no edges between vertices in A and vertices in D. If there is an edge $\{v, w\}$, $v \in A$, $w \in C$, then $w \in C_v$, and there is no edge parallel to this one (otherwise $w \in B$), so the number of edges in $A \times C$ equals $\sum_{v \in A} |C_v| \leq 8k^3 + 12k^2 + 2k$. The bound now follows. □

The counting here was done in a very rough way, where we aimed at a shorter proof at the cost of a higher constant factor for the number of vertices and number of edges. Improvements to the factors of 32 and 44 probably will not be hard to obtain.

Theorem 21 *A reduced instance can be obtained in polynomial time.*

Proof From the preceding discussion, we can conclude that given G, A, B, we can verify in polynomial time if a rule can be applied. The number of times we apply a rule or a restart is also polynomially bounded. The number of restarts is bounded by the initial value of k, as for each restart, k is decreased by one. We need to check for the **Improvement** rule only at the beginning of the algorithm, or right after a restart: the other rules cannot enable an improvement. As there are $O(k|V|)$ pairs where we check for a possible improvement, the total time for checking and performing all possibilities for the **Improvement** rule is polynomial. All other rules either cause a restart, or delete at least one edge from G, so take overall polynomial time. □

In Sect. 4, we give some variants of the rules. These allow for a polynomial time algorithm with a better asymptotic running time; see 4.6.

For experimental evaluation of our implementation of the algorithm, see Sect. 6.

3.5 A Constructive Version

It is not hard to transform the algorithm to a constructive one. Suppose the kernelization algorithm transforms the pair (G, k) to a pair (G', k'). The constructive algorithm would ensure that when we have a minimum size feedback vertex set of G' of size at most k', then we can obtain (in polynomial time) a minimum size feedback vertex set of G. We need to modify the reduction rules as follows.

In the **Degree Two** rule, if we contract a vertex v of degree two to a neighbor w, we give the new vertex the name of w. We keep a set S, which initially is empty. Each vertex, removed by the **Self-loop** rule, **Flower** rule, **Second Abdication** rule, or **Large Double Degree** rule is placed in S.

Now, for any minimum size feedback vertex set W in G' with $|W| \leq k'$, $S \cup W$ is a minimum size feedback vertex set in G.

3.6 A Reduced Instance with $\Theta(k^3)$ Vertices

We have shown that a reduced instance contains $O(k^3)$ vertices. We will now show that this bound is asymptotically tight.

Consider a graph family G_i defined as follows. For example, see G_3 in Fig. 5. Start with a cycle of i vertices, which we call the *center*. Connected to that are i widgets. A *widget* consists of a double pair (two vertices with a double edge), and i frobs. A *frob* consists of i vertices, each connected to both vertices of the double pair and to one of the vertices of the center: a widget has a frob to every vertex in the center.

The graph G_i has $i^3 + 3i$ vertices and has a feedback vertex set of size $2i$: e.g. all vertices in the center and one vertex from each double pair. If $k > 2i$, then G_i is a reduced instance. Thus, there is a reduced instance with $\Theta(k^3)$ vertices. Notice that, though asymptotically tight, this construction gives a reduced instance with 'only' $\frac{1}{8}k^3 + \frac{3}{4}k^2 + 3k + 4$ vertices, which is still far from the bound shown in Lemma18.

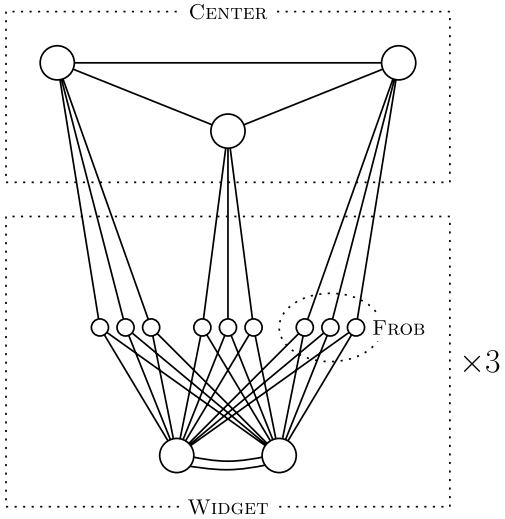

Fig. 5 G_3, a reduced instance if $k > 6$. In general, G_k is a reduced instance with $\Theta(k^3)$ vertices

3.7 Justification of the Rules

We will now show that all rules (except **Self-loop** and **Large Double Degree**) are, in a sense, required: if you leave one rule out, but do not change the remainder of the algorithm, the size of a reduced instance is no longer bounded in k.

The results of this section show that if for an implementation of a data reduction or kernelization algorithm for feedback vertex set, all rules are necessary if we want to guarantee that the reduced instances have size $O(k^3)$. In a practical setting, one will want to try some rules earlier and more often than other rules, giving preference to rules that often can be applied and cost not too much time to test. However, the other rules need to be used also when we want to keep the size guarantee on the resulting instances.

To start with one of the exceptions, the **Self-loop** rule is not actually required. A vertex with a self-loop is part of any feedback vertex set, and therefore the 2-approximation algorithm must select it. This directly gives a bound of $2k$ on the number of nodes with a self-loop in a reduced instance. We include this rule because it is simple and cheap.

Next we show that each of the simple rules is required. Without the **Islet** rule, a reduced instance can contain an unbounded amount of isolated vertices; without the **Twig** rule, a reduced instance can contain an unbounded amount of pendant edges; without the **Degree Two** rule, a reduced instance can contain a path of unbounded length. While the **Triple Edge** rule is not required for the vertex bound, it is of course required for the edge bound.

The **Flower** rule is required: consider an arbitrarily large wheel graph, with an extra vertex connected to three of the outer vertices; see Fig. 6a. Note that the center of the wheel is a flower if the number of spokes is at least $2k + 2$, but that none of the other rules apply if $k \geq 2$. (Without the extra outside vertex, the (second) **Abdication** rule would delete the center vertex, but in this graph there is no abdication.)

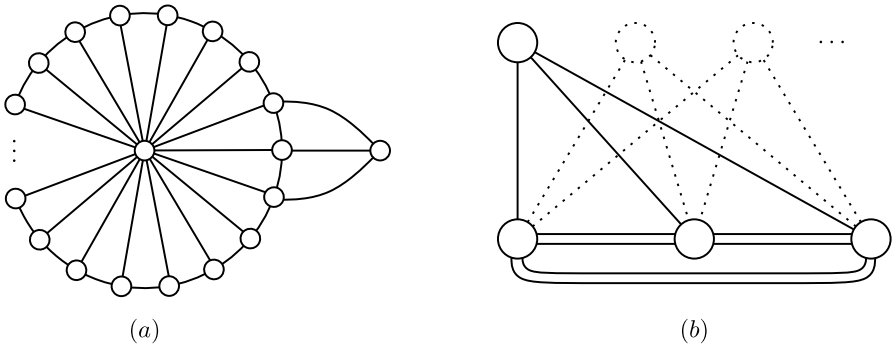

Fig. 6 Arbitrarily large reduced instances if (**a**): the **Flower** rule is not used, or (**b**): the **Abdication** rules are not used

As has been noted, the **Large Double Degree** rule is just a special case of the **Flower** rule and is therefore not required.

The **Improvement** rule is required: consider $K_{3,i}$. For any $i \geq 3$ and $k \geq 1$, only the **Improvement** rule applies.

The **Abdication** rules are required: consider again $K_{3,i}$. With $i \geq k+2$, the **Improvement** rule adds double edges between each of the vertices in the partite set of three (see Fig. 6b). But after that, **Abdication** is the only rule that applies when $k \geq 1$.

If the 2-approximation returns a feedback vertex set of more than $2k$ vertices, we conclude there is none of at most k vertices, and can therefore return *no*. Doing this is actually required: consider an arbitrarily large grid graph. The **Degree Two** rule will bypass the four corners, but with $k \geq 3$, no other rule applies.

4 Variants

In this section, we describe a few alternatives to the kernelization algorithm described in Sect. 3.

4.1 Without Restarts

To speed up, we can refrain from doing restarts after k has been decreased. The size of the kernel may become somewhat larger, and is a function of both the initial as the final value of k. One could also choose to only do a restart when $|A| > ck$, for some $c \geq 2$.

4.2 Without an Initial Value of k

It is also possible to run the algorithm while we do not have an initial value of k as input. In such a way, the algorithm could be used as a preprocessing heuristic. We first start with setting k to the size of the feedback vertex set A, returned by the approximation algorithm of Bafna et al. [3] or the algorithm of Becker and Geiger [5].

4.3 With a Different Algorithm to Find the Initial Feedback Vertex Set

Of course, we could use another approximation algorithm or heuristic for the feedback vertex set problem in the initialization phase. If we use, instead of, or in addition to, the algorithm of Bafna et al. [3] or Becker and Geiger [5] some other algorithms that finds close to optimal feedback vertex sets, and this algorithm finds a feedback vertex set that is smaller, then the guaranteed bound on the size of the kernel is also smaller. For instance, we could try to improve upon a solution found by the algorithms of [3, 5] with a local search algorithm.

4.4 Strongly Forced Vertices

The approximation algorithms of Bafna et al. [3] and of Becker and Geiger [5] also can solve weighted versions of the feedback vertex set problem. This can also be of help for preprocessing and kernelization of the unweighted problem.

Lemma 22 *Let $v \in V$. Suppose an approximation algorithm for* WEIGHTED FEEDBACK VERTEX SET *with performance ratio* 2 *returns a solution of weight at least* $2k+1$ *on the weighted instance of G, obtained by setting the weight of v to* $2k+1$, *and the weight of all other vertices to* 1. *Then v belongs to any feedback vertex set in G with at most k vertices.*

The correctness of this lemma is trivial. It directly implies the correctness of the following rule.

Rule 11 Strongly Forced Vertex Rule
Let $v \in V$. Build the following weighted instance of the problem. Set the weight of v to $2k+1$, *and the weight of all other vertices to* 1. *If the algorithm of* [3] *or* [5] *returns a feedback vertex set of weight at least* $2k+1$, *then remove v and its incident vertices from G, decrease k by one, and restart with the initialization phase.*

The **Strongly Forced Vertex** rule could be used instead of (or, in addition to) the **Flower** rule. We would run this check for each $v \in A$. It still gives an $O(k^3)$ bound on the kernel size, but with a slightly higher constant. Namely, if there are at least $2k+1$ cycles that are vertex disjoint except that they may share v, then any feedback vertex set that does not use v has size at least $2k+1$, so the rule will remove v. I.e., we find a flower with $2k+1$, instead of $k+1$ cycles (as in the **Flower** rule.)

4.5 Strongly Forced Pairs

In a similar way, we have a new rule that is a counter part to the **Improvement** rule.

Lemma 23 *Let $v, w \in V$. Suppose an approximation algorithm for* WEIGHTED FEEDBACK VERTEX SET *with performance ratio* 2 *returns a solution of weight at least* $2k+1$ *on the weighted instance of G, obtained by setting the weights of v and of w to* $2k+1$, *and the weight of all other vertices to* 1. *If W is a feedback vertex set in G of size at most k, then $v \in W$ or $w \in W$.*

This lemma, whose correctness is again trivial, implies the correctness of the following rule.

Rule 12 Strongly Forced Pair Rule
Let $v, w \in V$, $v \neq w$. Build the following weighted instance of the problem. Set the weight of v to $2k+1$, and the weight of all other vertices to 1. If the algorithm of [3] *or* [5] *returns a feedback vertex set of weight at least $2k+1$, then add two edges $\{v, w\}$ to G, and if $v \in A$ and $w \notin A \cup B$, then add w to B; if $w \in A$ and $v \notin A \cup B$, then add v to B.*

We can use this as an additional rule, as a heuristic for stronger preprocessing, or we can replace the **Improvement** rule by this rule. In the latter case, a similar proof as in Sect. 3.4 can be used to show that we obtain a kernel of size $O(k^3)$, although the constant factor is somewhat larger.

4.6 Time Analysis

Using the **Strongly Forced Vertex** rule and the **Strongly Forced Pair** rule instead of the **Flower** rule and the **Improvement** rule still yields a kernel of size $O(k^3)$. The constant hidden in the O is somewhat larger, but the asymptotic running time is smaller. Write $|V| = n$, and $|E| = m$.

If $G = (V, E)$ has a feedback vertex set X of size at most k, then $m \leq (k+1)n$: the forest $G[V - X]$ has at most $n-1$ edges, and each other edge has one endpoint in X. So, we start with checking if $m \leq (k+1)n$, if not, we reject. Thus, we may assume that $m = O(kn)$.

As 2-approximation algorithm for obtaining the initial set A and carrying out the **Strongly Forced Vertex** rule and the **Strongly Forced Pair** rule, we use the algorithm of Becker and Geiger [5], which runs in $O(m + n \log n)$ time. We maintain both the adjacency list and the adjacency matrix data structure for the graph; in the latter, we store for each pair the number of edges between the vertices. By keeping these numbers at most two, we automatically carry out the **Triple Edge** rule.

We perform the rules in the following order. After each of the at most k starts/restarts, we find A, and then carry out for each $v \in A$, and each $w \in V$ the **Strongly Forced Pair** rule. We maintain the number of double edges, incident to a vertex, and as soon as this number is at least $k+1$ for some vertex, we carry out the **Large Double Degree** rule, and restart. If we added double edges to G, then possibly we enabled new strongly forced pairs, so we check all pairs $v \in A$, $w \in V$ again. Possibly, we have to do $O(k^2)$ rounds, each doing a check for a strongly forced pair for $O(kn)$ pairs of vertices. Each such check costs $O(m + n \log n)$ time. The total time for all **Strongly Forced Pair** rule checks thus is bounded by $O(k^4(m + n \log n)) = O(k^4 \cdot n \cdot \max\{k, \log n\})$. After this, we do the checks for the **Strongly Forced Vertex** rule for each $v \in A$, and all the other rules. Note that these cannot enable a **Strongly Forced Pair** rule, unless a restart is caused. The time for the other rules is dominated by the time for the **Strongly Forced Pair** rules, and thus the total time of the kernelization algorithm is bounded by $O(k^4 \cdot n \cdot \max\{k, \log n\})$.

Theorem 24 *A kernel for* FEEDBACK VERTEX SET *with* $O(k^3)$ *vertices and edges can be obtained in* $O(k^4 \cdot n \cdot \max\{k, \log n\})$ *time.*

5 Loop Cutset

In this section we will look at the LOOP CUTSET problem. We are given a directed acyclic graph $G = (V, A)$. A *loop* is a cycle in the underlying undirected graph (i.e., in a loop we are allowed to follow arcs against their direction.) A *loop cutset* is a set of vertices $S \subseteq V$, such that the graph obtained by removing all arcs whose tail is in S has no loops. (In other words, a vertex v on a loop cuts the loop if at least one of its arcs points away from it.) The LOOP CUTSET problem is to find a minimum size loop cutset in a given directed acyclic graph.

This problem is clearly strongly related to FEEDBACK VERTEX SET, as can also be seen from the fact that many techniques carry over from one problem to the other (see e.g. [4, 5].) The LOOP CUTSET problem is well studied, in particular by researchers on probabilistic (or *Bayesian*) networks, as the algorithm of Pearl [32] for the INFERENCE problem on probabilistic networks uses a loop cutset, and uses time exponential in the size of the loop cutset, but linear in the number of vertices/variables.

We will now show that our algorithm can be adapted to work on the following parameterized version of the LOOP CUTSET problem.

Loop Cutset
Instance: A directed acyclic graph $G = (V, A)$.
Parameter: A positive integer k.
Question: Is there a set $U \subseteq V$ of at most k vertices such that each loop of G passes though some vertex of U which is not a sink on that loop?

The algorithm proceeds as follows. First, we transform the LOOP CUTSET instance into an instance for the BLACKOUT FEEDBACK VERTEX SET problem. Next, we transform this instance into an $O(k^3)$ kernel using an adapted version of the algorithm from Sect. 3. The resulting reduced instance is then transformed back into a LOOP CUTSET instance, yielding a kernel for LOOP CUTSET. These three steps are described in the following subsections.

5.1 Transform to Blackout-FVS

The main steps of the kernelization algorithm will be done on an instance of FEEDBACK VERTEX SET where in addition some vertices are *blacked out*. A vertex that is *blacked out* is not allowed to be part of the solution set. Vertices that are not blacked out are called *allowed*. We now formally define the parameterized BLACKOUT FEEDBACK VERTEX SET problem.

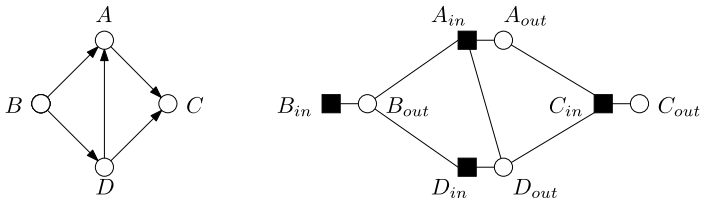

Fig. 7 Example of the transformation from Loop Cutset to Blackout Feedback Vertex Set

Blackout Feedback Vertex Set

Instance: An undirected graph $G = (V, E)$ and a set $Z \subseteq V$ of *blacked out* vertices.
Parameter: A positive integer k.
Question: Is there a set $U \subseteq V \setminus Z$ of at most k vertices such that each cycle of G passes though some vertex of U?

We now give the standard transformation from Loop Cutset to Blackout Feedback Vertex Set, that was used amongst others in [3, 4]. Given a directed acyclic graph $G = (V, A)$, we split every vertex $v \in V$ into two vertices, called v_{in} and v_{out}, and add an edge $\{v_{in}, v_{out}\}$. For each arc $(v, w) \in A$, we take an edge $\{v_{out}, w_{in}\}$. To complete the transformation, we *blackout* all in vertices, i.e., $Z = \{v_{in} \mid v \in V\}$.

See Fig. 7 for an example. Notice in particular that

- the loop $A\ C\ D$ can be cut using A and indeed the corresponding cycle $A_{out}\ C_{in}\ D_{out}\ A_{in}$ can be cut using A_{out}.
- the loop $A\ D\ B$ can *not* be cut using A (it is a sink on that loop) and indeed the corresponding cycle $A_{in}\ D_{out}\ D_{in}\ B_{out}$ cannot be cut using a vertex corresponding to A: A_{in} has been blacked out and A_{out} is not even on the cycle.

So, yes-instances are mapped to yes-instances, and no-instances to no-instances. This transformation doubles the number of vertices and the number of edges becomes n larger than the number of arcs. Conveniently, the transformation does not affect the parameter: a size k loop cutset in the original graph corresponds directly to a size k feedback vertex set.

5.2 Kernelization for Blackout Feedback Vertex Set

We have turned the instance for Loop Cutset into one for Blackout Feedback Vertex Set. We now give a modification of our algorithm from Sect. 3 to obtain a kernel for Blackout Feedback Vertex Set with $O(k^3)$ vertices.

First of all, the 2-approximation algorithms can also work on the weighted feedback vertex set problem, where vertices have weights and we ask for a minimum weight solution instead of a minimum cardinality one. We can use this to 2-approximate the Blackout Feedback Vertex Set problem as well, by giving

the allowed vertices weight 1 and the blacked out vertices weight $2n+1$: as the set of all weight 1 vertices is a feedback vertex set, no optimal solution or 2-approximation will ever select a blacked out vertex of weight $2n+1$.

5.2.1 The Rules

We leave the **Islet**, **Twig** and **Triple edge** rules unchanged: they are still safe when we have blacked out vertices. However, the original **Degree Two** rule is no longer safe in the presence of blacked out vertices. We use the following modified version.

Rule 13 Blackout Degree Two Rule
Suppose v is a vertex with exactly two neighbors x and y. If v is blacked out, or if v is allowed and at least one of x and y is also allowed, then remove v and add the edge $\{x, y\}$.

Lemma 25 *The **Blackout Degree Two** rule is safe.*

Proof Consider two cases for this rule. First, if v is blacked out, v just acts as a conduit for cycles while we cannot use it to break any; we can safely bypass such vertices. Secondly, if v is allowed, any cycle that is broken by v will also be broken if we take an allowed neighbor of v instead of v. This implies that if v *has* an allowed neighbor, there exists an optimal solution that does not use v. In that case, we can bypass it. □

I.e., we can bypass all vertices of degree two, except those that are allowed and have two blacked out neighbors. We leave the remaining degree-two vertices in the graph. In Sect. 5.2.2 we show that there can be only $O(k^3)$ of such vertices in a reduced instance.

If a blacked out vertex has a self-loop, then clearly the instance has no solution at all (this vertex *must* be chosen, but at the same time it *cannot*). The **Self-loop** rule is therefore extended with the condition that if the vertex with the self-loop is blacked out, we conclude *no* immediately. Likewise, we modify the **Flower** rule: if it can be applied to a vertex $v \in Z$, then we directly conclude *no*, instead of removing v.

In similar spirit, we add the following rule.

Rule 14 Blackout Double Edge *(new)*
Suppose v is an allowed vertex that has a double edge to a blacked out vertex. Then remove v, all edges incident to v and decrease k by one. If there is a double edge between two blacked out vertices, then conclude no, otherwise restart the initialization phase.

Lemma 26 *The **Blackout Double Edge** rule is safe.*

Proof A double edge to a blacked out vertex is essentially a self-loop. A double edge between two blacked out vertices is a cycle that cannot be broken. □

We add one more new rule.

Rule 15 Blacked Out Adjacent Vertices *(new)*
Suppose there is a single edge $\{v, w\}$ and suppose that v and w are both blacked out. Then contract $\{v, w\}$: delete v, w and $\{v, w\}$, and insert a new vertex with edges to all of v and w's old neighbors. (If v or w had a double edge, apply the **Blackout Double Edge** *rule first.)*

This has the effect of contracting all blacked out vertices that are adjacent to each other, and means that a reduced instance cannot have neighboring blacked out vertices.

The rest of the rules are carried over unchanged. Regarding their safeness, consider the following lemma.

Lemma 27 *All vertices in $A \cup B$ are allowed.*

Proof Vertices in A are allowed, because A is a feedback vertex set selected by the 2-approximation. Recall that B is the set of vertices with a double edge to a vertex in A. By the **Blackout Double Edge** rule, a blacked out vertex cannot have a double edge. Therefore, all vertices in B are also allowed. □

Since all vertices in A and B are allowed, the original proofs (e.g. of Lemma 9, about the safeness of the **Second Abdication** rule) can be used without changes.

5.2.2 Size of the Kernel

We will now show that this is a kernelization for Blackout Feedback Vertex Set, again with size $O(k^3)$. The proof goes along the same lines as before, but we need to bound the number of degree-two vertices.

The only degree-two vertices remaining in a reduced instance are allowed vertices whose two neighbors are blacked out: those are the only ones that the **Blackout Degree Two** rule does not delete. We make a distinction between those in A and those not in A.

The first kind is bounded by the size of A, which is $O(k)$. This leaves the vertices not in A, which we will bound with the aid of the following lemma.

Lemma 28 *Suppose x and y are blacked out vertices. Let U be the set of degree-two vertices whose two neighbors are x and y. Then there is at most one vertex $v \in U$ with $v \notin A$.*

Proof Assume to the contrary that there are two vertices $u_1, u_2 \in U \setminus A$. Then x, u_1, y, u_2 is an unbroken cycle, which contradicts that A is a feedback vertex set. □

Now consider a reduced instance G. Let G' be the graph obtained by bypassing all degree-two vertices that remain in G. The same proof as in Sect. 3.4 shows that G' has $O(k^3)$ vertices and edges. We have already bounded the degree-two vertices in A, and will now bound the other degree two vertices by associating each degree-two vertex in $G - A$ with an edge in G', namely the edge created to bypass it. By Lemma 28, there

can be at most one vertex associated to an edge in this way, bounding the number of degree two vertices by $O(k^3)$.

Theorem 29 *In a reduced instance, there are $O(k^3)$ vertices and $O(k^3)$ edges.*

5.3 Transforming Back to Loop Cutset

As a final step, to obtain a kernelization algorithm for Loop Cutset, we need to transform the instance of Blackout Feedback Vertex Set back to the original problem Loop Cutset. Conveniently, the kernelization algorithm for Blackout Feedback Vertex Set contracts all connected components of forbidden vertices. In this way, all neighbors of a forbidden vertex are allowed. This makes the transformation particularly easy.

First we consider an edge between allowed vertices. In the Blackout Feedback Vertex Set instance, either vertex breaks all cycles through this edge. If we were to simply turn the edge into an arc in either direction, we would risk making its target a sink on some loop. To prevent this, we introduce a new vertex and replace the original edge with two arcs going toward the new vertex. It is safe to introduce this middle vertex: it is a sink on any loop it is contained in and thus, it will not be included in any minimal solution. This part of the transformation adds at most m vertices and arcs.

Next we consider edges between a blacked out vertex v and an allowed vertex w. Each such edge $\{v, w\}$ is replaced by an arc (w, v) from the allowed vertex to the blacked out vertex. As has already been noted, there are no edges between blacked out vertices. This means that if a vertex v is blacked out, after the transformation all its arcs will be incoming. Therefore, v will be a sink on any cycle that contains v and v will not be included in any minimal solution. Conversely, if a vertex w is allowed, all its arcs will be outgoing and w will not be a sink on any cycle. This part of the transformation does not increase the size of the instance.

This concludes the kernelization for Loop Cutset. Combining the two transformations and the kernelization of Blackout Feedback Vertex Set gives the following result.

Theorem 30 *A kernel for* Loop Cutset *with $O(k^3)$ vertices and $O(k^3)$ edges can be obtained in polynomial time.*

It is not hard to turn the result into a constructive one. A minimum size loop cutset for the kernel can be transformed to a minimum size loop cutset for the original instance, as follows. Suppose we have a minimum size loop cutset S for the kernel. By the optimality of S, each vertex in S has outgoing edges, and thus is allowed in the corresponding kernel of Blackout Feedback Vertex Set. This gives us a minimum size feedback vertex set S' without blacked out vertices in the kernel of Blackout Feedback Vertex Set. As in Sect. 3.5, we obtain from S' a minimum size feedback vertex set S'' without blacked out vertices in the original instance of Blackout Feedback Vertex Set. Now, $S''' = \{v \in V \mid v_{out} \in S''\}$ is a minimum size loop cutset in the original graph G.

6 Experimental Evaluation

In this section we report on the results of our experiments with an implementation of the algorithm. Additional experiments and a more extensive treatment are available in Van Dijk's master thesis [40].

We have implemented our algorithm in C++, using LEDA (the *Library of Efficient Data Types and Algorithms* [28]) for the graph data structure and the computations of maximum flows and matchings. Our experiments were carried out on a 2.8 GHz Pentium 4 running Windows 2000. All times reported are CPU-time, as measured by LEDA's timer functionality.

6.1 Some Implementation Issues

The overall structure of the algorithm is to keep trying all rules until none are applicable. The rules change the graph, so the application of one rule can enable another. Our implementation does not track this explicitly, but instead surrounds the rule checks with a while-loop: while the graph keeps changing, we try the rules. (There are some small exceptions that are easy to do; for example, we explicitly check whether the **Twig** rule enables itself or the **Islet** rule.)

The order in which to check for the different rules makes a significant difference. The improvement rule adds double edges to the graph. If we try **Improvement** on all pairs of vertices but do not simultaneously also check for other rules, then the graph might get very dense: once we start adding more and more edges to the graph, it gets more and more likely that there is improvement between vertices. But checking for improvement will also become more and more expensive, and although such a dense graph is likely to be full of flowers, checking for that will be extra expensive as well.

We therefore interleave checking for improvement and flowers. For each vertex v in the graph we do the following. Start by checking for improvement between v and each other vertex, but if v gets to have $k+1$ double edges, the **Large Double Degree** rule deletes it immediately and we continue with the next vertex. If this does not happen, we check whether v is a flower.

6.2 Runtime of the Kernelization

In this section, we look at the actual runtime of the implementation of our algorithm. We have done several experiments with random graphs, which we present first.

Figure 8 shows the results of running our kernelization on random graphs with twice as many edges as vertices. (Results for other amounts of edges are similar, even when the number of edges grows quadratically instead of linearly.) We have run the kernelization on 10,000 random graphs for every $n \in [20..500]$ with $k = 20$, and measured the amount of time spent in the different rules. This shows some interesting behavior.

First of all, notice that the algorithm is not expensive at all, using approximately 40 seconds for the 10,000 graphs at the most expensive point ($n \approx 150$).

The most striking observation to notice in the graph, however, is that the runtime actually *decreases* significantly in the interval $n = 150$ to 180, and that at $n = 500$, the runtime still has not exceeded that of $n = 150$.

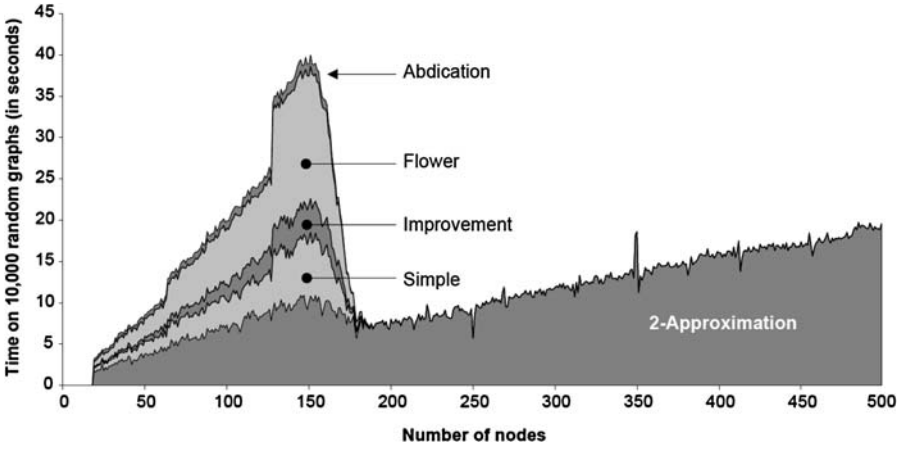

Fig. 8 Breakdown of the time spent in the different rules, on 10,000 random graphs with $m = 2n$ and $k = 20$

The reason for this can actually be seen in the graph. Recall the algorithm starts out by finding a 2-approximate solution. If the size of this solution is larger than $2k$ (40, in this case), the algorithm immediately concludes *no*, forgoing the other rules. This is a quick early-out on graphs that, in a certain sense, 'clearly' do not have a feedback vertex set of at most k vertices.

In this experiment, this early-out did not happen in more than 1% of the graphs if the number of vertices was less than 138 vertices; on the other hand, it happened on at least 99% of the graphs with more than 200 vertices. In between, there is a smooth transition of the fraction of graphs on which the early-out happens. It is this proportion of graphs that is 'easy' that causes the drop in runtime. (Note that the 2-approximation itself also spends more time around $n = 150$: this is because several of the rules can cause a restart, in which case the 2-approximation is run more than once.)

The point at which this cutoff occurs changes with k, of course, which can be seen in Fig. 9. There is not much more difference between the different values of k, however: until their respective cutoffs, all four curves are almost the same, and after the cutoff, they all fall back to the same curve of running the 2-approximation once.[1]

Finally, note the jumps in runtime at power-of-two input sizes: this is due to implementation details (at several points, hashmaps are used extensively).

6.3 Solving the Kernel Versus Solving the Original

After one has run the kernelization routine, one does not have an answer to your original problem yet: there still is the need to solve the problem on the kernel. Hopefully, the time spent on kernelization is (much) less than the time gained as we need

[1] The 2-approximation can be modified to do an even-earlier out because we know any feedback vertex set larger than $2k$ will lead to the same conclusion: *no*. We did not implement this, though.

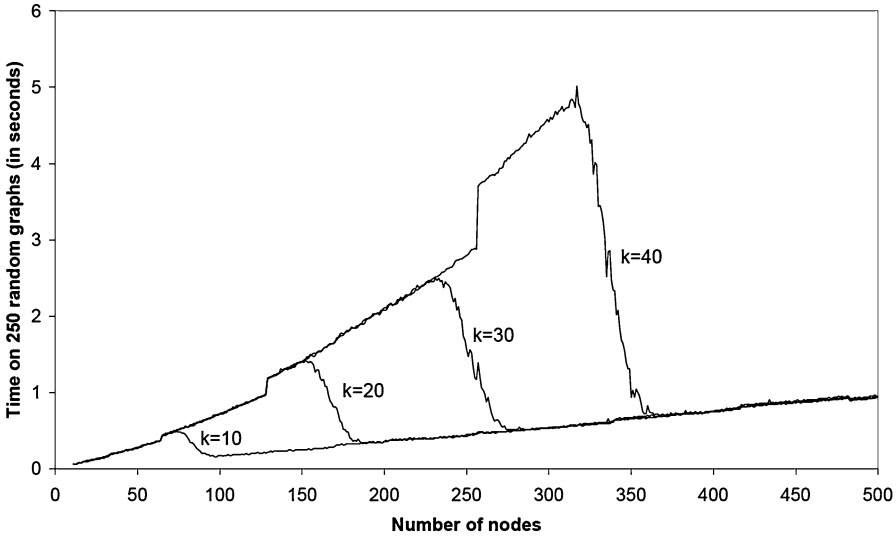

Fig. 9 Runtime on 250 random graphs with $m = 2n$, for $k \in \{10, 20, 30, 40\}$

to solve the problem on a smaller instance. Having analyzed the efficiency of our kernelization, we will now look at its effectiveness.

To this end, we have implemented a randomized algorithm for FEEDBACK VERTEX SET [4]. This algorithm computes the optimum value with high probability; the probability of failure can be made arbitrarily small. We will compare the runtime of the algorithm on the original instance with the runtime of our kernelization added to the runtime of the algorithm of [4] on the resulting kernel. The algorithm by [4] uses time exponential in k, but polynomial in n.

We do not explicitly look at the fixed parameter version of the problem here; instead we use the size of the 2-approximation as k: as has been noted, this is safe and makes our kernelization a preprocessing heuristic.

In Table 1 we show the results of running this comparison on a few graphs that have been used as a benchmark for treewidth algorithms, see e.g. [21, 38]. We indicate the number of nodes n and edges m before and after kernelization, and the (safe) parameter k with which the kernelization was run. The kernelization algorithm never used more than half a second. On some instances, we did not let the algorithm of [4] finish (this would, in one case, have taken years), but instead let it run for 1000 iterations and then extrapolated the time taken on those, to the required amount of iterations. Runtimes that we have not explicitly measured, but extrapolated in this way are indicated with a $*$.

Depending on whether the kernelization does anything to the graph, it can achieve a very significant speedup. On the above graphs, the kernelization was effective. If, on the other hand, the kernelization does not do much to the graph, its runtime is still negligible in comparison to the runtime of the algorithm of [4]: it never really hurts to use it.

Table 1 Results of the FEEDBACK VERTEX SET kernelization algorithm for selected graphs

Name	Solve original			Kernelize + solve			
	n	m	Runtime	k	n	m	Runtime
Oesoca-42	42	72	137 seconds	5	9	19	4.6 seconds
Alarm	37	65	122 minutes	8	12	25	11.5 seconds
Pathfinder	109	211	3 weeks ∗	12	17	53	22.3 minutes
Barley	48	126	28 years ∗	16	41	117	20 years ∗

7 Discussion

In this paper, we showed that the FEEDBACK VERTEX SET problem on undirected graphs and the LOOP CUTSET problem have kernels of size $O(k^3)$; the kernelization algorithm takes time polynomial in n and in k. We expect that the bound given in Sect. 4.6, for the running time of the algorithm can be improved with further analysis. We also expect that a modification of the techniques can lead to a kernel for the case where vertices have integer weights with the parameter k an upper bound on the sum of the weights of the vertices in the vertex set.

Several recent developments on fixed parameter algorithms for FEEDBACK VERTEX SET are the following. Very recently, Thomassé [37] has shown a quadratic kernel. Penninkx and Bodlaender [9] showed that FEEDBACK VERTEX SET has a linear kernel when the input graph is planar. Chen et al. [13] gave an FPT algorithm for DIRECTED FEEDBACK VERTEX SET and FEEDBACK ARC SET, solving a long standing open problem. Here we look for a minimum size set of vertices or arcs in a directed graph G, such that each cycle in G contains a vertex or arc in the set.

We end the paper with mentioning some open problems.

- As has been mentioned, Thomassé [37] has shown a kernel for FEEDBACK VERTEX SET of size $O(k^2)$. Is an even better result possible, that is, is there a kernel of size $o(k^2)$? See e.g. the discussion in [11]. Would the techniques from [9] be of help to obtain provably smaller kernels in the non-planar case?
- Are there polynomial size kernels for the DIRECTED FEEDBACK VERTEX SET and FEEDBACK ARC SET problems? Here, we may hope that the recent results of Chen et al. [13] are of help, but we expect this problem to be very hard, as the membership of DIRECTED FEEDBACK VERTEX SET and FEEDBACK ARC SET has been open for a long time till its resolution in [13].

Acknowledgements We thank the anonymous referees of the paper for useful comments.

References

1. Abu-Khzam, F.N., Collins, R.L., Fellows, M.R., Langston, M.A., Suters, W.H., Symons, C.T.: Kernelization algorithms for the vertex cover problem: theory and experiments. In: Proc. 6th ACM-SIAM ALENEX, 2004, pp. 62–69. ACM-SIAM, New York, Philadelphia (2004)

2. Alber, J., Fellows, M.R., Niedermeier, R.: Polynomial-time data reduction for dominating sets. J. ACM **51**, 363–384 (2004)
3. Bafna, V., Berman, P., Fujito, T.: A 2-approximation algorithm for the undirected feedback vertex set problem. SIAM J. Discrete Math. **12**, 289–297 (1999)
4. Becker, A., Bar-Yehuda, R., Geiger, D.: Randomized algorithms for the loop cutset problem. J. Artif. Intell. Res. **12**, 219–234 (2000)
5. Becker, A., Geiger, D.: Optimization of Pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem. Artif. Intell. **83**, 167–188 (1996)
6. Bodlaender, H.L.: On disjoint cycles. Int. J. Found. Comput. Sci. **5**(1), 59–68 (1994)
7. Bodlaender, H.L.: A linear time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. **25**, 1305–1317 (1996)
8. Bodlaender, H.L.: Necessary edges in k-chordalizations of graphs. J. Comb. Optim. **7**, 283–290 (2003)
9. Bodlaender, H.L., Penninkx, E.: A linear kernel for planar feedback vertex set. In: Grohe, M., Niedermeier, R. (eds.) Proceedings 3rd International Workshop on Parameterized and Exact Computation, IWPEC 2008. Lecture Notes in Computer Science, vol. 5018, pp. 160–171. Springer, Berlin (2008)
10. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels (extended abstract). In: Proceedings 35th International Colloquium on Automata, Languages and Programming, ICALP 2008. Lecture Notes in Computer Science, vol. 5125, pp. 563–574. Springer, Berlin (2008)
11. Burrage, K., Estivill-Castro, V., Fellows, M.R., Langston, M.A., Mac, S., Rosamond, F.A.: The undirected feedback vertex set problem has a poly(k) kernel. In: Bodlaender, H.L., Langston, M.A. (eds.) Proceedings 2nd International Workshop on Parameterized and Exact Computation, IWPEC 2006. Lecture Notes in Computer Science, vol. 4169, pp. 192–202. Springer, Berlin (2006)
12. Chen, J., Kanj, I.A., Jia, W.: Vertex cover: further observations and further improvements. J. Algorithms **41**, 280–301 (2001)
13. Chen, J., Liu, Y., Lu, S., O'Sullivan, B., Razgon, I.: A fixed-parameter algorithm for the directed feedback vertex set problem. In: STOC '08: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, New York, USA, 2008, pp. 177–186. ACM, New York (2008)
14. Clautiaux, F., Carlier, J., Moukrim, A., Négre, S.: New lower and upper bounds for graph treewidth. In: Rolim, J.D.P. (ed.) Proceedings International Workshop on Experimental and Efficient Algorithms, WEA 2003. Lecture Notes in Computer Science, vol. 2647, pp. 70–80. Springer, Berlin (2003)
15. Dehne, F., Fellows, M., Langston, M., Rosamond, F., Stevens, K.: An $O(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. In: Wang, L. (ed.) Proceedings 11th International Computing and Combinatorics Conference COCOON 2005. Lecture Notes in Computer Science, vol. 3595, pp. 859–869. Springer, Berlin (2005)
16. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness. Congr. Numer. **87**, 161–178 (1992)
17. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Berlin (1998)
18. Festa, P., Pardalos, P.M., Resende, M.G.C.: Feedback set problems. In: Handbook of Combinatorial Optimization, vol. A, pp. 209–258. Kluwer, Amsterdam (1999)
19. Fomin, F.V., Gaspers, S., Knauer, C.: Finding a minimum feedback vertex set in time $O(1.7548^n)$. In: Bodlaender, H.L., Langston, M.A. (eds.) Proceedings 2nd International Workshop on Parameterized and Exact Computation, IWPEC 2006. Lecture Notes in Computer Science, vol. 4169, pp. 183–191. Springer, Berlin (2006)
20. Gerards, A.M.H.: Matching. In: Ball, M.O. et al. (ed.) Network Models. Handbooks in Operations Research and Management Sciences, vol. 7, Chap. 3, pp. 135–224. Elsevier Science, Amsterdam (1995)
21. Gogate, V., Dechter, R.: A complete anytime algorithm for treewidth. In: Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence UAI-04. Arlington, Virginia, USA, 2004, pp. 201–208. AUAI Press, Berkeley (2004)
22. Goldberg, A.V., Karzanov, A.V.: Maximum skew-symmetric flows and matchings. Math. Program. **100**, 537–568 (2004)
23. Guo, J.: A more effective linear kernelization for cluster editing. In: Chen, B., Paterson, M., Zhang, G. (eds.) Proceedings First International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies ESCAPE 2007. Lecture Notes in Computer Science, vol. 4614, pp. 36–47. Springer, Berlin (2007)

24. Guo, J., Gramm, J., Hüffner, F., Niedermeier, R., Wernicke, S.: Improved fixed-parameter algorithms for two feedback set problems. In: Proc. 9th Int. Workshop on Algorithms and Data Structures WADS 2004. Lecture Notes in Computer Science, vol. 3608, pp. 158–168. Springer, Berlin (2004)
25. Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. ACM SIGACT News **38**, 31–45 (2007)
26. Kanj, I.A., Pelsmajer, M.J., Schaefer, M.: Parameterized algorithms for feedback vertex set. In: Downey, R.G., Fellows, M.R. (eds.) Proceedings 1st International Workshop on Parameterized and Exact Computation, IWPEC 2004. Lecture Notes in Computer Science, vol. 3162, pp. 235–248. Springer, Berlin (2004)
27. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–104. Plenum Press, New York (1972)
28. Mehlhorn, K., Näher, S.: LEDA: A Platform for Combinatorial and Geometric Computing. Cambridge University Press, Cambridge (1995)
29. Nagamochi, H., Ibaraki, T.: A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. Algorithmica **7**, 583–596 (1992)
30. Nemhauser, G.L., Trotter, L.E.: Vertex packing: structural properties and algorithms. Math. Program. **8**, 232–248 (1975)
31. Niedermeier, R.: Invitation to fixed-parameter algorithms. Universität Tübingen. Habilitation Thesis (2002)
32. Pearl, J.: Probablistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, Palo Alto (1988)
33. Raman, V., Saurabh, S., Subramanian, C.R.: Faster fixed parameter tractable algorithms for undirected feedback vertex set. In: Proceedings 13th International Symposium on Algorithms and Computation, ISAAC 2002. Lecture Notes in Computer Science, vol. 2518, pp. 241–248. Springer, Berlin (2002)
34. Raman, V., Saurabh, S., Subramanian, C.R.: Faster algorithms for feedback vertex set. Electronic Notes in Discrete Mathematics **19**, 273–279 (2005). Proceedings 2nd Brazilian Symposium on Graphs, Algorithms, and Combinatorics, GRACO 2005
35. Razgon, I.: Exact computation of maximum induced forest. In: Arge, L., Freivalds, R. (eds.) Proceedings of the 10th Scandinavian Workshop on Algorithm Theory, SWAT 2006. Lecture Notes in Computer Science, vol. 4059, pp. 160–171. Springer, Berlin (2006)
36. Schrijver, A.: Combinatorial Optimization. Polyhedra and Efficiency. Springer, Berlin (2003)
37. Thomassé, S.: A quadratic kernel for feedback vertex set. In: SODA '09: Proceedings of the Nineteenth Annual ACM–SIAM Symposium on Discrete Algorithms, Philadelphia, PA, USA, 2009. pp. 115–119. Society for Industrial and Applied Mathematics, Philadelphia (2009)
38. Treewidthlib. http://www.cs.uu.nl/people/hansb/treewidthlib (2004)
39. Tutte, W.T.: A short proof of the factor theorem for finite graphs. Can. J. Math. **6**, 347–352 (1954)
40. van Dijk, T.C.: Fixed parameter complexity of feedback problems. Master's thesis, Utrecht University (2007)