Linear-Time Kernelization for Feedback Vertex Set*

Yoichi Iwata†

National Institute of Informatics, Tokyo, Japan
yiwata@nii.ac.jp

Abstract

In this paper, we give an algorithm that, given an undirected graph G of m edges and an integer k, computes a graph G' and an integer k' in $O(k^4m)$ time such that (1) the size of the graph G' is $O(k^2)$, (2) $k' \leq k$, and (3) 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'. This is the first linear-time polynomial-size kernel for Feedback Vertex Set. The size of our kernel is $2k^2 + k$ vertices and $4k^2$ edges, which is smaller than the previous best of $4k^2$ vertices and $8k^2$ edges. Thus, we improve the size and the running time simultaneously. We note that under the assumption of NP $\not\subseteq$ coNP/poly, Feedback Vertex Set does not admit an $O(k^{2-\epsilon})$-size kernel for any $\epsilon > 0$.

Our kernel exploits *k-submodular relaxation*, which is a recently developed technique for obtaining efficient FPT algorithms for various problems. The dual of k-submodular relaxation of Feedback Vertex Set can be seen as a half-integral variant of A-path packing, and to obtain the linear-time complexity, we give an efficient augmenting-path algorithm for this problem. We believe that this combinatorial algorithm is of independent interest.

A solver based on the proposed method won first place in the 1st Parameterized Algorithms and Computational Experiments (PACE) challenge.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases FPT Algorithms, Kernelization, Path Packing, Half-integrality

Digital Object Identifier 10.4230/LIPIcs.ICALP.2017.68

1 Introduction

In the theory of parameterized complexity, we introduce parameters to problems and analyze the complexity with respect to both the input length $n = |x|$ and the parameter value k. If an algorithm runs in $f(k)n^{O(1)}$ time for any input of length n and a parameter k, it is called a *fixed-parameter tractable (FPT) algorithm*. If the $n^{O(1)}$ factor is linear, it is called a *linear-time FPT*. The typical goal of parameterized algorithms is to develop FPT algorithms with a small $f(k)$ (e.g., c^k for a small constant c) and a small $n^{O(1)}$ (e.g., linear in n). Although there are many algorithms that have been developed with smaller $f(k)$ *or* $n^{O(1)}$, achieving the smallest $f(k)$ and $n^{O(1)}$ *simultaneously* is a difficult task, and the smallest $f(k)$ factors and the smallest $n^{O(1)}$ factors are often achieved by different algorithms. Moreover, when trying to improve the $f(k)$ factor, the $n^{O(1)}$ factor is often ignored by using the O^* notation, which hides factors polynomial in n, and when trying to improve the $n^{O(1)}$ factor, the $f(k)$ factor is often ignored by assuming k is a constant.

For example, in recent papers, Iwata, Oka, and Yoshida [18], and Ramanujan and Saurabh [27] have independently obtained $O(4^km)$-time algorithms for Almost 2-SAT,

* A full version of the paper is available at https://arxiv.org/abs/1608.01463.
† Supported by JSPS KAKENHI Grant Number JP17K12643.

44th International Colloquium on Automata, Languages, and Programming (ICALP 2017).
Editors: Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl;
Article No. 68; pp. 68:1–68:14


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

which is a parameterized version of Max 2-SAT where a parameter is the number of unsatisfied clauses; on the other hand, when allowing the $n^{O(1)}$ factor to be super-linear, there exists an $O^*(2.32^k)$-time algorithm [23]. These two algorithms are not comparable: the former runs faster when the input is large but the latter runs faster when the parameter is large. Typically, only algorithms with the smallest $f(k)$ factor or the smallest $n^{O(1)}$ factor have been studied. However, if there were three algorithms running in time $O(8^k n)$, $O(4^k n^2)$, and $O(2^k n^3)$, all of them are incomparable: the first is fastest when $4^k < n$, the second is fastest when $2^k < n < 4^k$, and the third is fastest when $n < 2^k$. Do we need to develop algorithms with the smallest possible $f(k)$ factor for each n^d? We observe that *kernelization*, which is another basic research object of the parameterized complexity, is useful for avoiding this incomparability.

A kernelization algorithm (or *kernel*) for a parameterized problem is an algorithm that, given an instance (x, k) in time polynomial in $n = |x|$ and k, returns an equivalent instance (x', k') of the same problem such that $k' \leq k$ and $|x'| \leq g(k)$ for some function g. When the $n^{O(1)}$ factor in the running time is linear in n (i.e., $k^{O(1)}n$), it is called a *linear-time kernel*. If there is a kernel (and the problem is decidable), by solving the reduced instance exhaustively, we can obtain an FPT algorithm. Actually, the converse is also true; if there exists an $f(k)n^{O(1)}$-time FPT algorithm, there also exists a kernel of size $f(k)$. On the other hand, the existence of a polynomial-size (i.e., $|x'| \leq k^{O(1)}$) kernel is non-trivial and, actually, there are known to exist FPT problems which (unconditionally) do not have any sub-exponential-size kernels [4]. As in the case of FPT algorithms, the typical goal is to develop kernelization algorithms with a small size $g(k)$ (e.g., linear in k) and a fast running time (e.g., linear in n).

Compared with linear-time FPT algorithms, the number of studies for linear-time polynomial-size kernels is small. Examples include d-Hitting Set [29], Dominating Set on planar graphs [30, 16], $n-k$ Clique Covering [9], and Max Cut Above Edwards-Erdős Bound [14]. One of the major reasons is that when assuming the parameter k is a constant, which is often done when studying linear-time FPT algorithms, kernels become uninteresting because we cannot distinguish between $f(k)$ and $k^{O(1)}$. Nevertheless, improving the $n^{O(1)}$ factor in the running time of kernels is very important because such kernels can be used as preprocessing for FPT algorithms. Let us assume that we have a $k^{O(1)}n^d$-time polynomial-size kernel and an $f(k)n^{O(1)}$-time FPT algorithm. Then, by applying the FPT algorithm against the instance reduced by the kernelization, we obtain a $k^{O(1)}(f(k) + n^d)$-time FPT algorithm. Thus, the $n^{O(1)}$ factor of any FPT algorithm can be replaced by n^d. Therefore, if we have a linear-time polynomial-size kernel, we obtain a linear-time FPT algorithm that simultaneously achieves the smallest possible $f(k)$ factor (ignoring factors polynomial in k). This also implies that after obtaining a linear-time polynomial-size kernel, we can safely ignore the $n^{O(1)}$ factor and focus on improving the $f(k)$ factor only. Moreover, it can also be combined with another kernel of smaller size. Let us assume that we have a $k^{O(1)}n^d$-time polynomial-size kernel and a $g(k)$-size kernel. Then, by applying the second kernel against the instance reduced by the first kernelization, we obtain a $k^{O(1)}n^d$-time $g(k)$-size kernel. Therefore, in contrast to the case of FPT algorithms, we can always achieve the smallest size and the fastest running time simultaneously.

In this paper, we give a linear-time quadratic-size kernel for Feedback Vertex Set. This is the first linear-time polynomial-size kernel for this problem. Feedback Vertex Set is a problem to decide whether a given undirected graph has a vertex set of size at most a given parameter k whose removal makes the graph a forest. Feedback Vertex Set is one of the most comprehensively studied problems in the field of parameterized complexity and

many different FPT algorithms and kernels have been developed. Moreover, the problem was chosen as a target problem of the 1st Parameterized Algorithms and Computational Experiments (PACE) challenge[1]. Actually, this research is strongly motivated by the PACE challenge. The proposed methods are easy to implement, and a solver[2] based on the proposed methods won first place in the challenge.

The first FPT algorithm for Feedback Vertex Set was given by Downey and Fellows [13]. This algorithm and subsequent improved algorithms [21, 26] use the strategy to branch on short cycles and the $f(k)$ factor of the running time is not a single exponential in k. The first single-exponential FPT algorithms were obtained independently by Dehne et al. [11] and Guo et al. [15], and several improved algorithms have been obtained [8, 7, 22]. The current smallest $f(k)$ factor for deterministic algorithms is 3.62^k given by Kociumaka and Pilipczuk [22]. These single-exponential FPT algorithms use the *iterative compression* technique. For a graph with n vertices and m edges[3], a naive implementation of iterative compression requires n iterations and each iteration takes $f(k)\Omega(m)$ time. Therefore, the total running time is $f(k)\Omega(nm)$. For the case of Feedback Vertex Set, by combining it with 2-approximation algorithms [1, 3], we can solve the problem using only a single iteration; however, this increases the running time for one iteration to $f(2k)\Omega(m)$. Thus, for obtaining a linear-time FPT algorithm, the $f(k)$ factor needs to grow from 3.62^k to 3.62^{2k}. When allowing randomness, a simple $O(4^k km)$-time FPT algorithm using random sampling of edges was given by Becker et al. [2]. The current smallest $f(k)$ factor for randomized FPT algorithms is 3^k given by Cygan et al. [10]. This algorithm uses dynamic programming on tree-decompositions and takes $3^k k^{O(1)} n^2$ time after obtaining a tree-decomposition of width at most k. As discussed above, by using our linear-time polynomial-size kernel, we can obtain a $k^{O(1)}(3.62^k + m)$-time deterministic FPT algorithm and a $k^{O(1)}(3^k + m)$-time randomized FPT algorithm.

The first polynomial-size kernel was given by Burrage et al. [6]. The size of this kernel is $O(k^{11})$, which was improved to $O(k^3)$ by Bodlaender and Van Dijk [5], and to $O(k^2)$ by Thomassé [28]. Finally, Dell and Van Melkebeek [12] showed that there are no kernels of size $O(k^{2-\epsilon})$ for any constant $\epsilon > 0$ unless NP $\subseteq$ coNP/poly. The size of the current smallest kernel by Thomassé is $4k^2$ vertices and $8k^2$ edges. Although the precise running time of each of these kernels was not analyzed, we can easily check that all of them take at least $k^{O(1)}nm$ time. As discussed above, if there is a linear-time polynomial-size kernel, by combining it with the smallest kernel, we can achieve the linear running time and the smallest kernel size simultaneously. However, our linear-time quadratic-size kernel does not rely on such a combination.

Before providing a description of our kernel, we first give a brief description of a key idea behind the existing kernels. All the existing kernels for Feedback Vertex Set exploit *s-flowers*. A set of simple cycles is called an s-flower if each cycle contains the vertex s and no two cycles share any vertex other than s. If the degree of s is large and the graph is well-connected, there exists a large s-flower. Because the size of an s-flower (i.e., the number of cycles) gives a lower bound of the size of the minimum feedback vertex set that does not contain s, if it is larger than the parameter k, we can remove s and decrement k. Otherwise, the degree of s is small, or the graph is not well-connected. In the former case, we know that the graph is small, and in the latter case, we can apply another reduction rule.

[1] `https://pacechallenge.wordpress.com/`
[2] `https://github.com/wata-orz/fvs`
[3] If a graph has a feedback vertex set of size at most k, we have $m = O(kn)$.

In our kernel, instead of s-flowers, we exploit *k-submodular relaxation*, which is a recently developed technique for obtaining efficient FPT algorithms for various problems. The concept of k-submodular relaxation was introduced by Iwata, Wahlström, and Yoshida [19]. The k-submodular relaxation is a technique to obtain *half-integral* and *persistent* relaxations and many existing half-integral LP relaxations (e.g., the LP relaxation of Vertex Cover [24]) can be re-derived by this technique. If a problem admits such a relaxation, the branch-and-bound method gives an FPT algorithm. By applying k-submodular relaxation, they obtained an $O^*(4^k)$-time FPT algorithm for Feedback Vertex Set. A detailed description of the k-submodular for Feedback Vertex Set is given in Section 2. By exploiting k-submodular relaxation, we obtain a very simple kernel for Feedback Vertex Set, which is described in Section 3. The size of our kernel is $2k^2 + k$ vertices and $4k^2$ edges, which is smaller than the previous best of $4k^2$ vertices and $8k^2$ edges [28].

We observe that there is a strong relationship between the k-submodular relaxation of Feedback Vertex Set and s-flowers; the problem of computing a maximum s-flower is the integral dual of the k-submodular relaxation of Feedback Vertex Set. This resembles the situation for Almost 2-SAT. For Almost 2-SAT, Raman et al. [25] obtained an $O^*(9^k)$-time FPT algorithm by a reduction to Vertex Cover above Maximum Matching, and then both the $f(k)$ factor [23] and $n^{O(1)}$ factor [18, 27] were improved by a reduction to Vertex Cover above LP. Here, the maximum matching is the integral dual of the LP relaxation of Vertex Cover. Because the fractional minimum of the primal LP is always at least the integral maximum of the dual LP, by using the half-integral relaxation instead of the integral dual, we can obtain a better lower bound. Moreover, by using the half-integral relaxations, we can directly exploit the persistency of the relaxations.

For obtaining linear-time kernel, we give a max-flow-like augmenting-path algorithm for solving the k-submodular relaxation of Feedback Vertex Set in Section 4. This is the most technical part of the paper. Our algorithm can compute a minimum solution in $O(km)$ time. By combining this algorithm with the simple kernel, we give a linear-time kernel in Section 5. Due to space limitations, some of the proofs are omitted. Lemmas with omitted proofs are marked with ($\star$) and these proofs can be found in the full version [17].

We note that this algorithm can be used not only for the linear-time kernel but also for improving the $n^{O(1)}$ factor of the $O^*(4^k)$-time FPT branch-and-bound algorithm for Feedback Vertex Set. By applying k-submodular relaxations, $O^*(4^k)$-time FPT algorithms for two general versions, Subset Feedback Vertex Set and Group Feedback Vertex Set, have been obtained [19]. Following up our work, Iwata, Yamaguchi, and Yoshida [20] obtained linear-time FPT algorithms for many problems including these general versions of Feedback Vertex Set by developing efficient augmenting-path algorithm for solving k-submodular relaxations in general.

2 Preliminaries

2.1 Definitions

A *multiplicity function* of a multiset S is denoted by $\mathbf{1}_S$; e.g., when $S = \{a, a, b\}$, $\mathbf{1}_S(a) = 2$, $\mathbf{1}_S(b) = 1$, and $\mathbf{1}_S(c) = 0$. Let $f : U \to \mathbb{R}$ be a function. For a multiset S, we write the sum of $f(a)$ over $a \in S$ as $f(S) = \sum_{a \in S} f(a)$; e.g., when $S = \{a, a, b\}$, $f(S) = 2f(a) + f(b)$. We denote the preimage of $i \in \mathbb{R}$ under f by $f^{-1}(i) = \{a \in U \mid f(a) = i\}$.

Let $G = (V, E)$ be an undirected graph. We assume that G may contain a self-loop and multiple edges. We will often denote the number of vertices by n and the number of edges by m. We denote the set of edges incident to a vertex v by $\delta_G(v)$ and define the *degree*

of v as $d_G(v) = |\delta_G(v)|$. Here, we note that multiple edges contribute to the degree by its multiplicity, and we never refer to the degree of a vertex having a self-loop. We omit the subscript G if it is clear from the context. An edge $e \in E$ is called a *bridge* if its removal increases the number of connected components.

For a vertex set S, we denote the graph obtained by removing S and their incident edges by $G - S$. When S is a singleton $\{v\}$, we simply write $G - v$. A vertex set $S \subseteq V$ is called a *feedback vertex set* if $G - S$ is a forest. We denote the size of the minimum feedback vertex set of G by $\mathrm{fvs}(G)$.

A *walk* is an ordered list $(v_0, e_1, v_1, e_2, \ldots, v_{\ell-1}, e_\ell, v_\ell)$ such that $\ell \geq 1$ and each edge e_i connects vertices v_{i-1} and v_i. Note that it may contain a vertex or an edge multiple times. For a walk $W = (v_0, e_1, \ldots, v_\ell)$, we denote the multiset of vertices appearing on W by $V(W) = \{v_0, \ldots, v_\ell\}$ and the multiset of edges appearing on W by $E(W) = \{e_1, \ldots, e_\ell\}$.

2.2 Basic Reductions

We introduce basic reductions that have been commonly used in kernelization algorithms for Feedback Vertex Set [6, 5, 28]. The correctness of these reductions is almost trivial.

1. If there is a vertex v containing a self-loop, remove v and decrease k by one.
2. If there is a vertex of degree at most one, remove it.
3. If there is a vertex of degree two, remove it and connect its two neighbors by an edge.
4. If two vertices are connected by more than two edges, replace these edges with a double edge.

Note that rule 3 can remove a vertex that is only incident to a double edge; in this case, it creates a self-loop on its neighbor. These basic reductions never increase the degree of any vertex and can be fully applied in $O(m)$ time. After the reduction, the obtained graph has no self-loops and has minimum degree at least three.

We will use the following lemma to bound the size of the kernel. This is a general version of the lemma in [28], and a modified proof can be found in the full version [17].

▶ **Lemma 1** (Thomassé [28]). *If a graph without self-loops satisfies both of the following for an integer d, the size of the minimum feedback vertex set is larger than k:*

- *At least one of $n > dk + k$ or $m > 2dk$ holds; and*
- *for any $v \in V$, it holds that $3 \leq d(v) \leq d$.*

In [28], a kernel of $4k^2$ vertices and $8k^2$ edges is obtained by applying the lemma against $d = 4k - 1$. In the next section, we obtain a kernel of $2k^2 + k$ vertices and $4k^2$ edges by applying the lemma against $d = 2k$.

2.3 k-submodular Relaxation of Feedback Vertex Set

A walk $W = (v_0, e_1, \ldots, v_\ell)$ is called an *s-cycle* if $v_0 = v_\ell = s$, $v_i \neq s$ for all $i \in \{1, \ldots, \ell - 1\}$, $e_i \neq e_{i+1}$ for any $i \in \{1, \ldots, \ell - 1\}$ (i.e., there are no U-turns), and each edge is contained in the walk at most twice. For example, walks $(s, e_1, u, e_2, v, e_3, s)$ and $(s, e_1, u, e_2, v, e_3, w, e_4, u, e_1, s)$ are s-cycles but a walk $(s, e_1, u, e_2, v, e_2, u, e_1, s)$ is not. Note that in this definition, we distinguish each of multiple edges; e.g., if there is only a single edge e between s and v, a walk (s, e, v, e, s) is not an s-cycle; however, if there is a double edge $\{e_1, e_2\}$ between s and v, a walk (s, e_1, v, e_2, s) is an s-cycle.

For a graph $G = (V, E)$ without self-loops and a vertex $s \in V$, a function $x : V \to \mathbb{R}_{\geq 0}$ is called an *s-cycle cover* if it satisfies that (1) $x(s) = 0$ and (2) for any s-cycle C, $x(V(C)) \geq 1$.

Note that $V(C)$ is the multiset of vertices on C, and therefore if $x(v) = \frac{1}{2}$ holds for a vertex v contained twice in C, we have $x(V(C)) \geq 1$. The *size* of an s-cycle cover x is defined as $x(V)$, and when the size $x(V)$ is minimum among all the possible s-cycle covers, it is called a *minimum s-cycle cover*.

By introducing the idea of k-submodular relaxation, Iwata, Wahlström, and Yoshida [19] obtained the following lemma.

▶ **Lemma 2** (Iwata, Wahlström, and Yoshida [19]). *For any graph $G = (V, E)$ without self-loops and $s \in V$, the following holds:*

- *The size of any feedback vertex set of G that does not contain s is at least the size of the minimum s-cycle cover.*
- *There exists a minimum s-cycle cover that takes values $\{0, \frac{1}{2}, 1\}$ (half-integrality).*
- *If there exists a minimum feedback vertex set that does not contain s, then for any half-integral minimum s-cycle cover x, there also exists a minimum feedback vertex set S such that $x^{-1}(1) \subseteq S$ and $s \notin S$ (persistency).*

3 Simple Quadratic-size Kernel

In this section, we give a simple polynomial-time quadratic-size kernel for Feedback Vertex Set. By exploiting the persistency of the k-submodular relaxation, we give the following reduction rule called *s-cycle cover reduction*.

For a graph $G = (V, E)$, a vertex $s \in V$, and a half-integral minimum s-cycle cover x, create a graph $G' = (V, E')$ as follows. Let $X = x^{-1}(1)$ and let $B \subseteq \delta(s)$ be the set of bridges of $G - X$ connecting s and tree components of $G - X - s$. Then, G' is obtained from G by inserting a double edge between s and each of $v \in X$ and removing the edges B.

▶ **Lemma 3.** *For a graph $G = (V, E)$, a vertex $s \in V$, and a half-integral minimum s-cycle cover x, let $G' = (V, E')$ be a graph obtained by applying the s-cycle cover reduction. Then, $\mathrm{fvs}(G) = \mathrm{fvs}(G')$ holds.*

Proof. ($\geq$) Let S be a minimum feedback vertex set of G. Observe that all the inserted edges are between s and $X = x^{-1}(1)$. If $s \in S$, S is also a feedback vertex set of G'. Otherwise, from the persistency, we can assume that S contains all the vertices of X. Thus, S is also a feedback vertex set of G'.

($\leq$) Let S be a minimum feedback vertex set of G'. If $s \in S$, S is also a feedback vertex set of G. Otherwise, because all the vertices of X are connected to s by double edges in G', S must contain all of X. Because all the deleted edges are bridges in $G - X$, S is also a feedback vertex set of G. ◀

After applying this reduction, the degree of s can be bounded as the following shows.

▶ **Lemma 4.** *For a graph $G = (V, E)$, a vertex $s \in V$, and a half-integral minimum s-cycle cover x, let $G' = (V, E')$ be a graph obtained by applying the s-cycle cover reduction. Then, $d_{G'}(s) \leq 2x(V)$ holds.*

Proof. First, we show that x is also an s-cycle cover of G'. Let us assume that there is an s-cycle C of G' such that $x(C) < 1$. Because $x(v) = 1$ for $v \in X = x^{-1}(1)$, C contains none of X. Because all the inserted edges are incident to X, C is also an s-cycle of G, which is a contradiction.

For $i \in \{1, 2\}$, let N_i denote the set of vertices that are connected to s by edges of multiplicity i in G'. For each $v_i \in N_1$, we define a vertex w_i as follows.

Algorithm 1 Simple quadratic-size kernelization for Feedback Vertex Set.

```
1:  procedure Kernelize(G, k)
2:      while true do
3:          Apply the basic reductions
4:          if k < 0 then return NO
5:          if n ≤ 2k^2 + k and m ≤ 4k^2 then return (G, k)
6:          if ∀v ∈ V, d(v) ≤ 2k then return NO
7:          Pick a vertex s of degree larger than 2k
8:          Compute a half-integral minimum s-cycle cover x
9:          if x(V) > k then G ← G − s; k ← k − 1
10:         else apply the s-cycle cover reduction
```

If the edge sv_i is a bridge in $G' - X$, let C_i be the connected component of $G' - X - s$ containing v_i. Because the reduction removes all the bridges between s and tree components, C_i is not a tree. Thus, there exists an s-cycle contained in $C_i \cup \{s\}$ and, therefore, there must exist a vertex $w_i \in C_i$ with $x(w_i) = \frac{1}{2}$.

If sv_i is not a bridge in $G' - X$, there exists a path P_i from v_i to $N_1 \setminus \{v_i\}$ in $G' - X - s$. Fix an arbitrary path P_i and let w_i be the first vertex on the path such that $x(w_i) = \frac{1}{2}$. Because x is an s-cycle cover, there always exists such a vertex.

If $w_i = w_j$ holds for some $i \neq j$, there exists an s-cycle C such that $x(C) = \frac{1}{2}$, which is a contradiction. Therefore, all w_i are distinct. Thus, we have $d_{G'}(s) = |N_1| + 2|N_2| \leq |x^{-1}(\frac{1}{2})| + 2|x^{-1}(1)| = 2x(V)$. ◀

Now, we describe our simple quadratic-size kernelization algorithm (see Algorithm 1). First, we apply the basic reductions. If k becomes negative, we return a NO instance. If the graph becomes small enough, we return it. If all the vertices have degree at most $2k$, we return a NO instance. Otherwise, pick an arbitrary vertex s of degree larger than $2k$, and compute a half-integral minimum s-cycle cover x. If the size of the s-cycle cover is larger than k, we remove s and decrement k. Otherwise, we apply the s-cycle cover reduction.

▶ **Lemma 5.** *Algorithm 1 runs in $(k+m)^{O(1)}$ time and correctly computes (G', k') satisfying $k' \leq k$ and $\mathrm{fvs}(G) \leq k \Leftrightarrow \mathrm{fvs}(G') \leq k'$. The size of G' is at most $2k^2 + k$ vertices and $4k^2$ edges.*

Proof. It obviously holds that $k' \leq k$ and the size of G' is at most $2k^2 + k$ vertices and $4k^2$ edges. From Lemma 4, after applying the s-cycle cover reduction, the degree of s changes from more than $2k$ to at most $2k$. Therefore, the number of edges strictly decreases for each iteration. Thus, it stops in at most m iterations. Because each iteration can be done in time polynomial in k and m, the total running time is also polynomial in k and m.

Next, we show the correctness. By applying Lemma 1 against $d = 2k$, when the maximum degree is at most $2k$ and at least one of $n > 2k^2 + k$ and $m > 4k^2$ holds, $\mathrm{fvs}(G) > k$ holds. Thus, we can safely return a NO instance (line 6). From Lemma 2, if $x(V) > k$, there is no feedback vertex set of size at most k that does not contain s. Thus, we can safely remove s (line 9). The correctness of the s-cycle cover reduction follows from Lemma 3. ◀

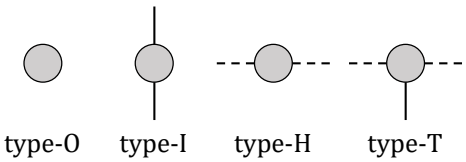

Figure 1 Four types of vertices.

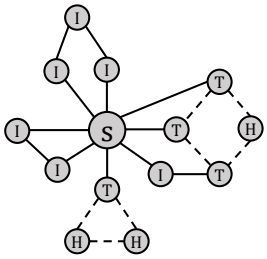

Figure 2 Example of the basic s-cycle packing.

4 Efficient Computation of a Half-integral Minimum s-cycle Cover

In this section, we prove the following theorem.

▶ **Theorem 6.** *Given a graph $G = (V, E)$ without self-loops, a vertex $s \in V$, and an integer k, in $O(km)$ time, we can compute a half-integral minimum s-cycle cover or conclude that there are no s-cycle covers of size at most $\frac{k}{2}$.*

First, we give several definitions. Let $\mathcal{C}_s$ denote the set of all s-cycles. A function $y : \mathcal{C}_s \to \mathbb{R}$ is called an *s-cycle packing* if for any vertex $v \in V \setminus \{s\}$, it holds that $\sum_{C \in \mathcal{C}_s} \mathbf{1}_{V(C)}(v) y(C) \leq 1$. The *size* of an s-cycle packing y is defined as $y(\mathcal{C}_s)$, and when the size $y(\mathcal{C}_s)$ is the maximum among all the possible s-cycle packings, it is called a *maximum s-cycle packing*. Because the problem of finding a maximum s-cycle packing is the LP dual of the problem of finding a minimum s-cycle cover, the size of the minimum s-cycle cover is equal to the size of the maximum s-cycle packing. Thus, if we can find a pair of an s-cycle cover x and an s-cycle packing y of the same size, we can confirm that x is a minimum s-cycle cover and y is a maximum s-cycle packing.

▶ **Definition 7.** A function $f : E \to \{0, \frac{1}{2}, 1\}$ is called a *basic s-cycle packing* if it satisfies the following three conditions.

1. For any $e \in \delta(s)$, $f(e) \in \{0, 1\}$.
2. Each vertex $v \in V \setminus \{s\}$ satisfies exactly one of the following four conditions (see Figure 1):
   a. $f(e) = 0$ for all edges $e \in \delta(v)$ (called type-O);
   b. $f(e) = 1$ for exactly two edges $e \in \delta(v)$ and $f(e) = \frac{1}{2}$ for none of $e \in \delta(v)$ (called type-I);
   c. $f(e) = 1$ for none of $e \in \delta(v)$ and $f(e) = \frac{1}{2}$ for exactly two edges $e \in \delta(v)$ (called type-H);
   d. $f(e) = 1$ for exactly one edge $e \in \delta(v)$ and $f(e) = \frac{1}{2}$ for exactly two edges $e \in \delta(v)$ (called type-T).
3. For each vertex $v \in V \setminus \{s\}$ of type-H or type-T, the cycle obtained by following edges of value $\frac{1}{2}$ from v contains an odd number of type-T vertices.

We call the cycle in the third condition the *half-integral cycle of v*. The size of a basic s-cycle packing f is defined as $\frac{1}{2} f(\delta(s))$. Figure 2 illustrates an example of the basic s-cycle packing, where solid lines denote edges of value 1, and dotted lines denote edges of value $\frac{1}{2}$.

▶ **Lemma 8** ($\star$). *If there exists a basic s-cycle packing of size k, there also exists an s-cycle packing of size k.*

Note that this lemma only says that the size of the maximum basic s-cycle packing is always at most the size of the maximum s-cycle packing and does not imply these two are equal. The equality is shown at the end of this section.

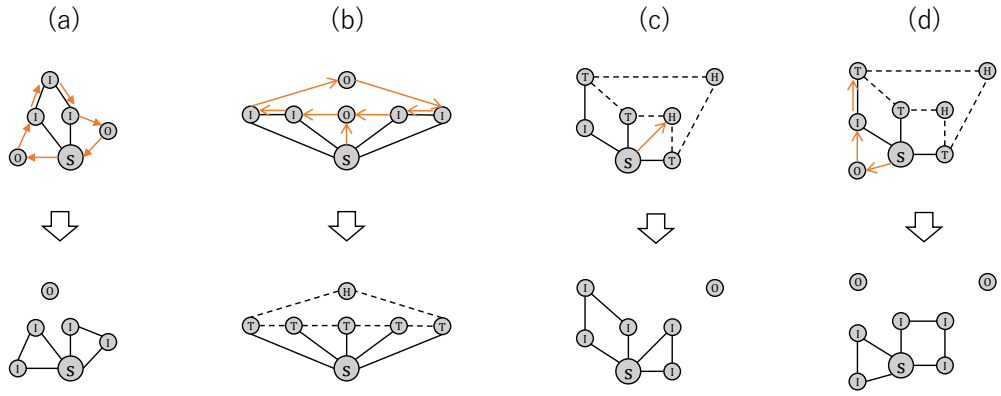

Figure 3 Examples of augmenting walks.

▶ **Definition 9.** For a basic s-cycle packing f, a walk $W = (v_0, e_1, \ldots, v_\ell)$ is called an *f-augmenting walk* if it satisfies all the following conditions.

1. We have $v_0 = s$.
2. We have $f(e_1) = 0$.
3. All the edges $\{e_1, \ldots, e_\ell\}$ are distinct.
4. The vertices $\{v_0, \ldots, v_{\ell-1}\}$ are distinct (the last vertex v_ℓ can be identical to v_i for some $i < \ell$).
5. For each $i \in \{1, \ldots, \ell - 1\}$, exactly one of the following holds:
   a. v_i is type-O;
   b. v_i is type-I and $f(e) = 1$ holds for at least one of $e \in \{e_i, e_{i+1}\}$.
6. If $v_\ell = v_i$ for some $i < \ell$, exactly one of the following holds:
   a. $v_\ell = s$ and $f(e_\ell) = 0$;
   b. v_ℓ is type-O;
   c. v_ℓ is type-I and $f(e) = 1$ holds for at least one of $e \in \{e_i, e_\ell\}$.
7. If $v_\ell \notin \{v_0, \ldots, v_{\ell-1}\}$, v_ℓ is type-H or type-T.

For a basic s-cycle packing f and an f-augmenting walk $W = (v_0, e_1, \ldots, v_\ell)$, let $f_W : E \to \{0, \frac{1}{2}, 1\}$ be a function defined as follows. First, set $f_W(e) = f(e)$ for all edges $e \in E$. If $v_\ell \neq s$ and $v_\ell = v_i$ holds for some $i < \ell$, let $h = i$; otherwise, let $h = \ell$. Then, for each edge $e \in \{e_1, \ldots, e_h\}$, set $f_W(e) = 1 - f(e)$. If $v_\ell = s$, we finish (see Figure 3-(a)). Otherwise, we further modify f_W depending on the type of v_ℓ.

(Case 1) If v_ℓ is type-O or type-I, for each edge $e \in \{e_{h+1}, \ldots, e_\ell\}$, set $f_W(e) = \frac{1}{2}$ (see Figure 3-(b)).

(Case 2) If v_ℓ is type-H, let C be the half-integral cycle of v_ℓ and let $\{t_0 = v_\ell, t_1, \ldots, t_{q-1}\}$ be the vertex set consisting of the vertex v_ℓ and the type-T vertices on C ordered along C (i.e., v_ℓ is located on the path from t_{q-1} to t_1 along the cycle C). We use the notation $t_q = t_0$. Let P_i be the path from t_i to t_{i+1} along the cycle C. For each even i, set $f_W(e) = 0$ for all the edges on P_i, and for each odd i, set $f_W(e) = 1$ for all the edges on P_i (see Figure 3-(c)).

(Case 3) If v_ℓ is type-T, let C be the half-integral cycle of v_ℓ and let $\{t_0 = v_\ell, t_1, \ldots, t_{q-1}\}$ be the type-T vertices on C ordered along C. Then, we proceed in exactly the same way as in case 2 (see Figure 3-(d)). We note that, in case 2, q is even; thus, v_ℓ is connected to t_{q-1} by edges of value one in f_W. On the other hand, in case 3, q is odd; thus, v_ℓ is connected to none of t_1 and t_{q-1}.

We call this operation that creates f_W from f as *augmenting f along W*.

Algorithm 2 Algorithm for computing an f-augmenting walk.

```
1: procedure FindAugmentingWalk(G, s, f)
2:     S ← {s}
3:     prev(v) = ϵ for all v ∈ V
4:     while S ≠ ∅ do
5:         Pick a vertex u ∈ S and remove u from S
6:         for e = uv ∈ δ(u) do
7:             if e = prev(u) then continue
8:             if e ∈ δ(s) and f(e) = 1 then continue
9:             if u is type-I and f(prev(u)) = f(e) = 0 then continue
10:            if v is type-H or type-T then
11:                prev(v) ← e
12:                return the walk from s to v along the search tree
13:            else if prev(v) = ϵ then
14:                prev(v) ← e; S ← S ∪ {v}
15:            else if v is type-O or f(prev(v)) + f(e) ≥ 1 then
16:                return the walk s → u → v → w along the search tree
17:    return NO
```

▶ **Lemma 10 (⋆).** *For a basic s-cycle packing f and an f-augmenting walk $W = (v_0, e_1, \ldots, v_\ell)$, let $f_W : E \to \{0, \frac{1}{2}, 1\}$ be the function obtained by augmenting f along W. Then, f_W is a basic s-cycle packing. Moreover, if $v_\ell = s$, the size of f_W is the size of f plus one; and otherwise, the size of f_W is the size of f plus $\frac{1}{2}$.*

Now, we give an algorithm to compute an f-augmenting walk (see Algorithm 2). First, we initialize a set S and a table $\mathrm{prev} : V \to E \cup \{\epsilon\}$. The set S stores vertices we need to process and is initialized to $\{s\}$. We ensure that only the vertex s and vertices of type-O or type-I are stored in S. The table $\mathrm{prev}(v)$ represents an edge to the parent of v in the search tree and initialized to the dummy edge ϵ, which indicates that the vertex is not visited (or the vertex is the root s). Then, while S is not empty, pick up an arbitrary vertex u from S and process each incident edge $e = uv \in \delta(u)$ as described below. If S becomes empty, the algorithm returns NO.

First, we check whether the edge $e = uv$ is valid by testing the following three conditions. If $e = \mathrm{prev}(u)$, because we have already processed this edge, we skip it. If e is incident to s and $f(e) = 1$, because such an edge cannot be used in an augmenting walk, we skip it. Note that, when $v = s$, at least one of these two conditions are satisfied. Similarly, if u is type-I and both of $f(\mathrm{prev}(u))$ and $f(e)$ are zero, because we cannot use both of $\mathrm{prev}(u)$ and e simultaneously, we skip it.

If v is type-H, or type-T, we return the walk from s to v in the search tree by using the table prev. If $\mathrm{prev}(v) = \epsilon$, we set $\mathrm{prev}(v) = e$ and insert it to S. If v is already visited and v is type-O, let w be the lowest common ancestor of u and v in the search tree. Then, we return the walk obtained by going down from s to u along the search tree, jumping from u to v by the edge e, and then by going up from v to w along the search tree. If v is already visited and v is type-I, we basically do the same; however, we need one additional constraint. If both of $f(\mathrm{prev}(v))$ and $f(e)$ are zero, the walk created as above does not satisfy the condition 6(c) of Definition 9; thus, we skip the edge without returning the walk.

From the construction of our algorithm, we obtain the following lemma.

▶ **Lemma 11.** *A walk returned by Algorithm 2 is an f-augmenting walk.*

Note that this lemma does not say that Algorithm 2 returns an f-augmenting walk whenever there exists an f-augmenting walk; it only says that if the algorithm returns a walk, it is an f-augmenting walk, and the algorithm might return NO even when there exists an f-augmenting walk. We now show that, if the algorithm returns NO, we can construct an s-cycle cover x whose size is equal to the size of f. From Lemma 8 and the LP duality of s-cycle packings and s-cycle covers, the size of a basic s-cycle packing is always at most the size of an s-cycle cover. Therefore, this equality implies that the current basic s-cycle packing f is the maximum and the constructed s-cycle packing x is the minimum. This also implies that when the algorithm returns NO, there are no f-augmenting walks.

To construct such an s-cycle cover x, we first prove a property of the table prev. We call a vertex $v \in V$ *reachable* if $v = s$ or $\mathrm{prev}(v) \neq \epsilon$. For each edge $e \in \delta(s)$ with $f(e) = 1$, by following edges of value 1 from e, we can obtain a simple cycle returning to s or a simple path to a type-T vertex. We denote such a cycle or a path by W_e. Note that when W_e is a cycle, $W_e = W_{e'}$ for another edge $e' \in \delta(s)$.

▶ **Lemma 12 (⋆).** *If Algorithm 2 returns NO, exactly one of the following holds for each edge $e \in \delta(s)$ with $f(e) = 1$:*

1. $\mathrm{prev}(v) = \epsilon$ *for any vertex* $v \in V(W_e)$*;*
2. W_e *is a cycle, all the vertices on* W_e *are reachable, and exactly one vertex* $v \in V(W_e) \setminus \{s\}$ *satisfies* $\mathrm{prev}(v) \notin E(W_e)$*.*

When Algorithm 2 returns NO, by using the obtained table prev, we construct a function $x : V \to \{0, \frac{1}{2}, 1\}$ as follows. For each edge $e = su \in \delta(s)$ with $f(e) = 1$, if W_e is a cycle satisfying the second condition of Lemma 12, we set $x(v) = 1$ for the unique vertex v satisfying $\mathrm{prev}(v) \notin E(W_e)$. Otherwise, we set $x(u) = \frac{1}{2}$. If $x(u)$ is already set to $\frac{1}{2}$, e.g., $W_{e_1} = W_{e_2} = (s, e_1, u, e_2, s)$ for a double edge $\{e_1, e_2\}$, we set $x(u) = 1$.

▶ **Lemma 13 (⋆).** *If Algorithm 2 returns NO, the function x is a minimum s-cycle cover.*

Proof of Theorem 6. Because each augmentation increases the size of f by at least $\frac{1}{2}$, after $k+1$ steps, we can obtain a half-integral minimum s-cycle cover of size at most $\frac{k}{2}$, or conclude that there are no s-cycle covers of size at most $\frac{k}{2}$. Because Algorithm 2 runs in $O(m)$ time, the total running time is $O(km)$. ◀

5 Linear-time Quadratic-size Kernel

In this section, we improve the running time of the quadratic-size kernel presented in Section 3 to $O(k^4 m)$. By using the $O(km)$-time algorithm for computing the minimum s-cycle cover presented in Section 4, each iteration can be done in $O(km)$ time. However, because the number of iterations is only bounded by $O(m)$, the total running time becomes $O(km^2)$. We show that, by a slight modification to Algorithm 1, the number of iteration can be bounded by $O(k^3)$; thus, the total running time becomes $O(k^4 m)$.

We add the following two rules just after line 6 of Algorithm 1. The safeness of these two rules is almost trivial.

- If there is a vertex v incident to more than k double edges, remove v, decrement k, and continue the iteration.
- If there are more than k^2 double edges, return NO.

For bounding the number of iterations, we use the following lemma.

▶ **Lemma 14.** *For a graph $G = (V, E)$ of minimum degree at least three, a vertex $s \in V$, and a half-integral minimum s-cycle cover x, if $2x(V) < d_G(s)$ holds, then $x^{-1}(1) \neq \emptyset$.*

Proof. From Lemma 4, for a graph G' obtained by applying the s-cycle cover reduction, it holds that $d_{G'}(s) \leq 2x(V)$. When $x^{-1}(1) = \emptyset$, the reduction inserts no new edges and only removes the bridges of G connecting s and tree components of $G - s$. Because the graph $G - s$ has minimum degree at least two, it has no tree components. Thus, we have $G' = G$, which is a contradiction. ◀

Now, we can prove the upper bound on the number of iterations.

▶ **Lemma 15.** *The modified Algorithm 1 stops in $O(k^3)$ iterations.*

Proof. Initially, all the double edges are blue, and after applying the s-cycle cover reduction, we color all the double edges incident to s red (not only newly inserted double edges but also blue colored edges are recolored to red). The other double edges, which are created by the deletion of degree two vertices in the basic reductions, are colored blue. Let α denote the number of red double edges and β denote the number of vertices of degree larger than $2k$ and incident to at least one red double edge. Let k_0 be the initial value of k and ϕ be a potential defined as $\phi = (2k_0^2 + 4k_0 + 3)k - 2\alpha + \beta$. Observe that, because red double edges are created only by the s-cycle cover reduction, each vertex can be incident to at most $k_0 + 1$ red double edges, and that the number of red double edges is always at most $k_0^2 + k_0$ (at most k_0^2 edges before applying the s-cycle cover reduction and the reduction can create at most k_0 red double edges).

Initially, there are no red edges; thus, the initial potential is $(2k_0^2 + 4k_0 + 3)k_0 = O(k_0^3)$. If ϕ becomes negative, we have $k < 0$ or $\alpha > k_0^2 k \geq k^2$. Thus, the algorithm returns NO.

When k is decremented, α can decrease by at most $k_0 + 1$. Because there are at most $k_0^2 + k_0$ red double edges, β can increase by at most $2k_0^2 + 2k_0$. Thus, ϕ decreases by at least $(2k_0^2 + 4k_0 + 3) - 2(k_0 + 1) - (2k_0^2 + 2k_0) \geq 1$.

When applying the s-cycle cover reduction, if the reduction creates c (≥ 1) new red double edges, α increases by c and β can increase by at most c. Thus, ϕ decreases by at least $2c - c = c \geq 1$. After applying the s-cycle cover reduction, from Lemma 14, s is incident to at least one double edge. Thus, if the reduction does not create any new red double edges, s must be incident to at least one red double edge before the reduction. From 4, the degree of s becomes at most $2k$ after the reduction. Therefore β decreases by at least one; thus, ϕ decreases by at least one.

Now, we have shown that each iteration decreases the potential ϕ by at least one. Because ϕ is initially $O(k^3)$ and is always non-negative, the number of iterations is $O(k^3)$. ◀

References

1. Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM J. Discrete Math.*, 12(3):289–297, 1999.
2. Ann Becker, Reuven Bar-Yehuda, and Dan Geiger. Randomized algorithms for the loop cutset problem. *J. Artif. Intell. Res.*, 12:219–234, 2000.
3. Ann Becker and Dan Geiger. Optimization of Pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem. *Artif. Intell.*, 83(1):167–188, 1996.
4. Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.

5 Hans L. Bodlaender and Thomas C. van Dijk. A cubic kernel for feedback vertex set and loop cutset. *Theory Comput. Syst.*, 46(3):566–597, 2010.

6 Kevin Burrage, Vladimir Estivill-Castro, Michael R. Fellows, Michael A. Langston, Shev Mac, and Frances A. Rosamond. The undirected feedback vertex set problem has a poly(k) kernel. In *IWPEC 2006*, pages 192–202, 2006.

7 Yixin Cao, Jianer Chen, and Yang Liu. On feedback vertex set: New measure and new structures. *Algorithmica*, 73(1):63–86, 2015.

8 Jianer Chen, Fedor V. Fomin, Yang Liu, Songjian Lu, and Yngve Villanger. Improved algorithms for feedback vertex set problems. *J. Comput. Syst. Sci.*, 74(7):1188–1198, 2008.

9 Benny Chor, Mike Fellows, and David W. Juedes. Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps. In *WG 2004*, pages 257–269, 2004.

10 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *FOCS 2011*, pages 150–159, 2011.

11 Frank K. H. A. Dehne, Michael R. Fellows, Michael A. Langston, Frances A. Rosamond, and Kim Stevens. An $O(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. *Theory Comput. Syst.*, 41(3):479–492, 2007.

12 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *J. ACM*, 61(4):23:1–23:27, 2014.

13 Rodney G. Downey and Michael R. Fellows. Fixed parameter tractability and completeness. In *Complexity Theory: Current Research*, pages 191–225, 1992.

14 Michael Etscheid and Matthias Mnich. Linear kernels and linear-time algorithms for finding large cuts. In *ISAAC 2016*, pages 31:1–31:13, 2016.

15 Jiong Guo, Jens Gramm, Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *J. Comput. Syst. Sci.*, 72(8):1386–1396, 2006.

16 Torben Hagerup. Simpler linear-time kernelization for planar dominating set. In *IPEC 2011*, pages 181–193, 2011.

17 Yoichi Iwata. Linear-time kernelization for feedback vertex set. *CoRR*, abs/1608.01463, 2016. Full version of this paper. URL: `https://arxiv.org/abs/1608.01463`.

18 Yoichi Iwata, Keigo Oka, and Yuichi Yoshida. Linear-time FPT algorithms via network flow. In *SODA 2014*, pages 1749–1761, 2014.

19 Yoichi Iwata, Magnus Wahlström, and Yuichi Yoshida. Half-integrality, LP-branching and FPT algorithms. *SIAM J. Comput.*, 45(4):1377–1411, 2016.

20 Yoichi Iwata, Yutaro Yamaguchi, and Yuichi Yoshida. Linear-time FPT algorithms via half-integral non-returning A-path packing. *CoRR*, abs/1704.02700, 2017.

21 Iyad A. Kanj, Michael J. Pelsmajer, and Marcus Schaefer. Parameterized algorithms for feedback vertex set. In *IWPEC 2004*, pages 235–247, 2004.

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

23 Daniel Lokshtanov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *ACM Trans. Algorithms*, 11(2):15:1–15:31, 2014.

24 G. Nemhauser and L. Trotter. Vertex Packing: Structural Properties and Algorithms. *Math. Program.*, 8:232–248, 1975.

25 Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Paths, flowers and vertex cover. In *ESA 2011*, pages 382–393, 2011.

26 Venkatesh Raman, Saket Saurabh, and C. R. Subramanian. Faster fixed parameter tractable algorithms for finding feedback vertex sets. *ACM Trans. Algorithms*, 2(3):403–415, 2006.

27 M. S. Ramanujan and Saket Saurabh. Linear time parameterized algorithms via skew-symmetric multicuts. In *SODA 2014*, pages 1739–1748, 2014.

28 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Trans. Algorithms*, 6(2), 2010.

29 René van Bevern. Towards optimal and expressive kernelization for d-hitting set. *Algorithmica*, 70(1):129–147, 2014.

30 René van Bevern, Sepp Hartung, Frank Kammer, Rolf Niedermeier, and Mathias Weller. Linear-time computation of a linear problem kernel for dominating set on planar graphs. In *IPEC 2011*, pages 194–206, 2011.