Improved Algorithms and Combinatorial Bounds for Independent Feedback Vertex Set*

Akanksha Agrawal[1], Sushmita Gupta[2], Saket Saurabh[3], and Roohani Sharma[4]

1 **Department of Informatics, University of Bergen, Norway**
akanksha.agrawal@uib.no
2 **Department of Informatics, University of Bergen, Norway**
sushmita.gupta@uib.no
3 **Institute of Mathematical Sciences, HBNI, Chennai, India**
saket@imsc.res.in
4 **Institute of Mathematical Sciences, HBNI, Chennai, India**
roohani@imsc.res.in

Abstract

In this paper we study the "independent" version of the classic Feedback Vertex Set problem in the realm of parameterized algorithms and moderately exponential time algorithms. More precisely, we study the Independent Feedback Vertex Set problem, where we are given an undirected graph G on n vertices and a positive integer k, and the objective is to check if there is an *independent feedback vertex set* of size at most k. A set $S \subseteq V(G)$ is called an *independent feedback vertex set (ifvs)* if S is an independent set and $G \setminus S$ is a forest. In this paper we design two deterministic exact algorithms for Independent Feedback Vertex Set with running times $\mathcal{O}^\star(4.1481^k)$[1] and $\mathcal{O}^\star(1.5981^n)$. In fact, the algorithm with $\mathcal{O}^\star(1.5981^n)$ running time finds the smallest sized ifvs, if an ifvs exists. Both the algorithms are based on interesting measures and improve the best known algorithms for the problem in their respective domains. In particular, the algorithm with running time $\mathcal{O}^\star(4.1481^k)$ is an improvement over the previous algorithm that ran in time $\mathcal{O}^\star(5^k)$. On the other hand, the algorithm with running time $\mathcal{O}^\star(1.5981^n)$ is the first moderately exponential time algorithm that improves over the naïve algorithm that enumerates all the subsets of $V(G)$. Additionally, we show that the number of minimal ifvses in any graph on n vertices is upper bounded by 1.7485^n.

1998 ACM Subject Classification G.2.2 Graph Algorithms, I.1.2 Analysis of Algorithms

Keywords and phrases independent feedback vertex set, fixed parameter tractable, exact algorithm, enumeration

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.2

1 Introduction

Feedback Vertex Set (FVS) is one of the classic NP-complete problems. In fact, it is one of the problems in the Karp's famous list of twenty one NP-complete problems [21]. FVS together with several of its variants have been extensively studied from both combinatorial as well as algorithmic view points. Indeed, FVS is one of the central problems in any

* The research leading to these results has received funding from the European Research Council (ERC) via grant PARAPPROX, reference 306992.

[1] The $\mathcal{O}^\star()$ notation suppresses polynomial factors in the running-time expression.

algorithmic paradigm that has to cope with NP-hardness, examples being : approximation algorithms, moderately exponential time algorithms, enumeration algorithms and parameterized algorithms [2, 3, 4, 7, 8, 11, 14, 16, 20, 22, 23, 28, 29, 33]. The goal of this article is to study the independent set version of FVS in the realm of parameterized complexity, moderately exponential time algorithms and combinatorial upper bounds.

We begin by formally defining the problem. The formal description of the problem being studied is as follows.

Independent Feedback Vertex Set (IFVS)	**Parameter:** k
Input: An undirected graph G on n vertices and a positive integer k.	
Question: Is there an independent feedback vertex set of size at most k?	

IFVS and Parameterized Complexity. FVS together with Vertex Cover is one of the most well studied problem in the field of parameterized complexity [3, 4, 22, 28]. The other variants of FVS on undirected graphs that have been studied extensively, include, Subset FVS [8, 23, 33], Group FVS [7, 20, 33], Connected FVS [25], Simultaneous FVS [1] and indeed IFVS [24, 30, 31]. The current champion algorithms for FVS are: a randomized algorithm with running time $\mathcal{O}^\star(3^k)$ [6] and a deterministic algorithm running in time $\mathcal{O}^\star(3.619^k)$ [22]. Misra et al. [24] introduced IFVS in 2011 (in the conference version of the cited paper) as a generlization of FVS and gave an algorithm with running time $\mathcal{O}^\star(5^k)$. They also designed a polynomial kernel of size $\mathcal{O}(k^3)$ for the problem. Later, Song claimed a deterministic algorithm with running time $\mathcal{O}^\star(4^k)$ for the problem [30]. However, the algorithm of Song [30] does not seem to be correct.[2] Tamura et al. [31] studied IFVS on special graph classes and showed that the problem remains NP-complete even on planar bipartite graphs of maximum degree four. They also designed linear time algorithms for graphs of bounded treewidth, chordal graphs and cographs. Finally, they gave an algorithm with running time $\mathcal{O}(2^{\mathcal{O}(\sqrt{k}\log k)}n)$ for IFVS on planar graphs. We refer the reader to the recent book on parameterized algorithms for more details regarding the paradigm of parameterized complexity, as well as about the literature on the FVS problem [5]. Our first main result is the following result regarding IFVS.

▶ **Theorem 1.** *There is an algorithm for* IFVS *running in time* $\mathcal{O}^\star(4.1481^k)$.

Our new algorithm is based on iterative compression and the subroutine for iterative compression is based on branching. The branching algorithm itself exploits (a) the fact that once we select a vertex in the independent feedback vertex set then all its neighbors must be in the forest; and (b) an interesting variation of the measure used for analyzing the fastest known deterministic algorithm for FVS [22]. Finally, we also observe that the randomized algorithm designed for FVS, running in time $\mathcal{O}^\star(3^k)$ [6], can be adapted to design a randomized $\mathcal{O}^\star(3^k)$ time algorithm for IFVS.

IFVS and Moderately Exponential Time Algorithms. In moderately exponential time algorithms (or exact algorithms for short), the objective is to design an algorithm for optimization version of a problem that is better than the naïve brute force algorithm. In particular, for FVS the goal will be to design an algorithm that runs in time c^n, $c < 2$ a constant, and finds a minimum sized set S such that $G - S$ is a forest. We refer to the book

[2] We have approached the author with concrete questions but he has not yet responded. Furthermore, we give a family of counter-examples to his algorithm in the Section 3.2.

of Fomin and Kratsch for more details regarding moderately exponential time algorithms [17]. Obtaining a non-trivial exact algorithm for FVS was open for quite some time before Razgon obtained an algorithm with running time $\mathcal{O}(1.8899^n)$ [29]. Later this algorithm was improved to $\mathcal{O}^\star(1.7347^n)$ [18]. Recently, Fomin et al. [13] obtained an interesting result relating parameterized algorithms and exact algorithms. Roughly speaking, they showed that if a problem (satisfying some constraints) has $\mathcal{O}^\star(c^k)$ time algorithm parameterized by the solution size, then there is an exact algorithm running in time $\mathcal{O}^\star((2-\frac{1}{c})^n)$. Both FVS and IFVS satisfies the required constraints and thus we immediately obtain the following exact algorithm for IFVS: (a) a randomized algorithm running in time $\mathcal{O}^\star((2-\frac{1}{3})^n) = \mathcal{O}^\star(1.6667^n)$; and (b) a deterministic algorithm running in time $\mathcal{O}^\star((2-\frac{1}{4.1481})^n) = \mathcal{O}^\star(1.7590^n)$. We give a recursive algorithm based on classical measure and conquer [15, 17] and design faster algorithm than both the mentioned algorithms. In particular, we prove the following theorem.

▶ **Theorem 2.** *There is an algorithm for* IFVS *running in time* $\mathcal{O}^\star(1.5981^n)$.

Combinatorial Upper Bounds. In our final section we address the following question: How many minimal ifvses are there in any graph on n vertices? Proving an upper bound on the number of combinatorial structures is an old and vibrant area. Some important results in this area include an upper bound of

- $3^{n/3}$ on the number of maximal independent sets in a graph [26].
- 1.667^n on the number of minimal feedback vertex sets in a tournament [13].
- 1.8638^n on the number of minimal feedback vertex sets in a graph [14, 16].

One can easily observe that every minimal ifvs is also a minimal feedback vertex set. Thus, an upper bound of 1.8638^n on the number of minimal ifvses in any graph on n vertices follows by [14]. As our final result, we give an improved upper bound on the number of minimal ifvses in any graph on n vertices. We obtain this result by applying reduction rules and branching rules with a carefully choosen measure. At the base case we prove that counting the number of spanning trees is same as counting the number of minimal ifvses. For bounding the number of spanning trees we use the result of Grimmett [19].

▶ **Theorem 3.** *A graph G on n vertices has at most* 1.7485^n *minimal ifvses.*

Let n be divisible by 3 and G be a graph that is union of $n/3$ vertex disjoint triangles. Then any minimal ifvs must contain exactly one vertex from each of $n/3$ triangles and thus G has $3^{n/3}$ minimal ifvses. Closing the gap between $3^{n/3}$ and 1.7485^n remains an interesting open problem. The proofs of Theorem 2 and 3 are omitted due to space constraints.

2 Preliminaries

In this section, we state some basic definitions and introduce terminology from graph theory and algorithms. We also establish some of the notations that will be used throughout.

We denote the set of natural numbers by $\mathbb{N}$. To describe the running times of our algorithms, we will use the $\mathcal{O}^\star$ notation. Given $f : \mathbb{N} \to \mathbb{N}$, we define $\mathcal{O}^\star(f(n))$ to be $\mathcal{O}(f(n) \cdot p(n))$, where $p(\cdot)$ is some polynomial function. That is, the $\mathcal{O}^\star$ notation suppresses polynomial factors in the running-time expression.

Graphs. We use standard terminology from the book of Diestel [9] for those graph-related terms which are not explicitly defined here. We only consider finite graphs possibly having loops and multi-edges. For a graph G, by $V(G)$ and $E(G)$ we denote the vertex and edge sets of the graph G, respectively. For a vertex $v \in V(G)$, we use $d_G(v)$ to denote the degree of v,

i.e the number of edges incident on v, in the (multi) graph G. We also use the convention that a loop at a vertex v contributes two to its degree. For a vertex subset $S \subseteq V(G)$, $G[S]$ and $G \setminus S$ are the graphs induced on S and $V(G) \setminus S$, respectively. For an edge subset $S \subseteq E(G)$, by $G \setminus S$, we denote the graph obtained after removing edges in S from G. For a vertex subset $S \subseteq V(G)$, we let $N_G(S)$ and $N_G[S]$ denote the open and closed neighbourhood of S in G. That is, $N_G(S) = \{v \mid (u,v) \in E(G), u \in S\} \setminus S$ and $N_G[S] = N_G(S) \cup S$. We drop the sub-script G from $d_G(v), N_G(S), N_G[S]$ whenever the context is clear. For a graph G and an edge $e \in E(G)$, G/e denotes the graph obtained after contracting e in G.

A *path* in a graph is a sequence of distinct vertices $v_0, v_1, \ldots, v_\ell$ such that (v_i, v_{i+1}) is an edge for all $0 \leq i < \ell$. A *cycle* in a graph is a sequence of distinct vertices $v_0, v_1, \ldots, v_\ell$ such that $(v_i, v_{(i+1) \mod (\ell+1)})$ is an edge for all $0 \leq i \leq \ell$. We note that both a double edge and a loop are cycles. A tree T rooted at $r \in V(T)$ is called as a *star* if $E(T) = \{(v,r) \mid v \in V(T) \setminus \{r\}\}$.

Let $W \subseteq V(G)$ and $H = G \setminus W$. We define certain useful vertices in $V(H)$. We call a vertex $v \in V(H)$, a *nice vertex* if $d_H(v) = 0$ and $d_G(v) = 2$, i.e. both the neighbours of v belong to the set W. Similarly, we call a vertex $v \in V(H)$, a *tent* if $d_H(v) = 0$ and $d_G(v) = 3$. A *feedback vertex set* is a subset $S \subseteq V(G)$ such that $G \setminus S$ is a forest.

Parameterized Complexity. A parameterized problem Π is a subset of $\Gamma^* \times \mathbb{N}$, where Γ is a finite alphabet. An instance of a parameterized problem is a tuple (x,k), where x is a classical problem instance, and k is called the parameter. A central notion in parameterized complexity is *fixed-parameter tractability (FPT)* which means, for a given instance (x,k), decidability in time $f(k) \cdot p(|x|)$, where f is an arbitrary function of k and p is a polynomial in the input size. For more details on parameterized complexity, we refer the reader to the books of Downey and Fellows [10], Flum and Grohe [12], Niedermeier [27], and the more recent book by Cygan et al. [5].

When we say that we branch on a vertex v, we mean that we recursively generate two instances, one where v belongs to the solution, the other where v does not belong to the solution. This is a standard method of exhaustive branching.

Bounded Search Trees. The running time of an algorithm that uses bounded search trees can be analyzed as follows (see, e.g., [5, 10]). Suppose that the algorithm executes a branching rule which has ℓ branching options (each leading to a recursive call with the corresponding parameter value), such that, in the i^{th} branch option, the current value of the parameter decreases by b_i. Then, $(b_1, b_2, \ldots, b_\ell)$ is called the *branching vector* of this rule. We say that α is the *root* of $(b_1, b_2, \ldots, b_\ell)$ if it is the (unique) positive real root of $x^{b^*} = x^{b^*-b_1} + x^{b^*-b_2} + \cdots + x^{b^*-b_\ell}$, where $b^* = \max\{b_1, b_2, \ldots, b_\ell\}$. If $r > 0$ is the initial value of the parameter, and the algorithm (a) returns a result when (or before) the parameter is negative, and (b) only executes branching rules whose roots are bounded by a constant $c > 0$, then its running time is bounded by $\mathcal{O}^*(c^r)$.

A reduction rule is a polynomial time algorithm that replaces an instance (I,k) of a parameterized language L by a new instance (I',k'). It is said to be *safe* if $(I,k) \in L$ if and only if $(I',k') \in L$.

3 FPT Algorithm for Independent Feedback Vertex Set

In this section we give an FPT algorithm for IFVS running in time $\mathcal{O}^\star(4.1481^k)$. Given an input (G,k), the algorithm starts by computing a feedback vertex set Z in G. A feedback

vertex set in G of size at most k (if it exists) can be computed in time $\mathcal{O}^\star(3.619^k)$ using the algorithm given in [22]. If there is no feedback vertex set of size at most k, then we conclude that (G, k) is a NO instance of ifvs since an ifvs is also a feedback vertex set in G.

We let $H = G \setminus Z$. The algorithm either outputs an ifvs in G of size at most k or correctly conclude that (G, k) is a NO-instance of IFVS. The algorithm guesses a subset $Z' \subseteq Z$, such that for an ifvs X in G, $X \cap Z = Z'$. For each of the guess Z', the algorithm does the following. If $G[Z']$ is not an independent set then it concludes that there is no ifvs X in G such that $Z' \subseteq X$. Otherwise, $G[Z']$ is an independent set. Let $W = Z \setminus Z'$. If $G[W]$ is not a forest, then their is no ifvs X such that, $X \cap Z = Z'$. Therefore, the guess Z' is not correct and the algorithm rejects this guess. Otherwise, it deletes the vertices in Z' and tries to find an ifvs $S \subseteq V(H) \setminus W$ of size at most $k - |Z'|$. Note that any vertex $v \in N_H(Z')$ cannot be part of the solution. Therefore, the algorithm adds the vertices in $N_H(Z')$ to a set $\mathcal{R}$. The set $\mathcal{R}$ consists of those vertices which cannot be included in ifvs in order to maintain the independence of the vertices included in the solution. The algorithm calls the sub-routine Disjoint Independent Feedback Vertex Set (Dis-IFVS) on the instance $(G \setminus Z', W, \mathcal{R}, k - |Z'|)$ to find an ifvs $X \subseteq V(G \setminus Z') \setminus (W \cup \mathcal{R})$. In Section 3.1 we give an algorithm for Dis-IFVS, which given an instance $(G, W, \mathcal{R}, k)$ either finds an ifvs $S \subseteq V(G) \setminus (W \cup \mathcal{R})$ of size at most k or correctly concludes that there does not exits such an ifvs. Moreover, the algorithm for Dis-IFVS runs in time $\mathcal{O}^\star(3.1481^k)$.

▶ **Theorem 1** (restated). *There is an algorithm for* IFVS *running in time* $\mathcal{O}^\star(4.1481^k)$.

Proof. Given an instance (G, k) of IFVS, the algorithm computes a feedback vertex set Z in G of size at most k (if it exists) in time $\mathcal{O}^\star(3.619^k)$. If there is no feedback vertex set of size at most k, it correctly concludes that (G, k) is a NO instance. Otherwise, for each $Z' \subseteq Z$, either it correctly concludes that Z' is a wrong guess (for extending it to an ifvs) or runs the algorithm for Dis-IFVS on the instance $(G \setminus Z', W, \mathcal{R}, k - |Z'|)$. Here, the instance $(G \setminus Z', W, \mathcal{R}, k - |Z'|)$ is created as described above in the description of the algorithm. The correctness of the algorithm follows from the correctness of the algorithm for Dis-IFVS and the fact that all possible intersections of the solution with Z are considered. The running time of the algorithm is given by the following equation: $3.619^k \cdot n^{\mathcal{O}(1)} + \sum_{i=0}^{k} \binom{k}{i} \cdot 3.1481^{k-i} \cdot n^{\mathcal{O}(1)} \leq 4.1481^k \cdot n^{\mathcal{O}(1)}$. This concludes the proof. ◀

3.1 Algorithm for Disjoint Independent Feedback Vertex Set

We give an algorithm for Disjoint Independent Feedback Vertex Set running in time $\mathcal{O}^\star(3.1481^k)$.

Disjoint Independent Feedback Vertex Set (Dis-IFVS) **Parameter:** k
Input: An undirected (multi) graph G, a fvs W in G, $\mathcal{R} \subseteq V(G) \setminus W$ and, an integer k.
Question: Does G have an ifvs $S \subseteq V(G) \setminus (W \cup \mathcal{R})$ such that $\lvert S \rvert \leq k$?

The algorithm for Dis-IFVS either applies some reduction rules or branches on a vertex in $V(G) \setminus W$. The algorithm branches on a vertex in $V(G) \setminus W$ only when (a) none of the reduction rules are applicable; and (b) we are not in the case where we can solve the problem in polynomial time. Let $H = G \setminus W$. We arbitrary root the trees in H at some vertex (preferably a vertex v with $d_H(v) > 2$). We will be using the following measure μ associated with the instance $(G, W, \mathcal{R}, k)$ to bound the number of nodes of the search tree.

$$\mu = \mu(G, W, \mathcal{R}, k) = 2k + \rho(W) - (\eta + 2\tau)\,.$$

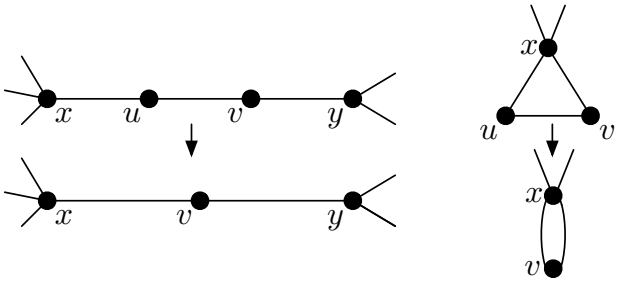

Figure 1 Reduction Rule 2.

Here, $\rho(W)$ is the number of components in W, η denotes the number of nice vertices in $V(H) \setminus \mathcal{R}$ and τ denotes the number of tents in $V(H) \setminus \mathcal{R}$. We note that here nice vertices and tents are defined with respect to the set W. See preliminaries for the definitions of a nice vertex and a tent.

Now we describe all the reduction rules that will be used by the algorithm. The first two reduction rules get rid of vertices of degree at most one and consecutive vertices of degree two in the graph. The safeness of these reduction rules follow from [24].

- Reduction Rule 1. Delete vertices of degree at most one since they do not participate in any cycle.
- Reduction Rule 2. Let u, v be two adjacent degree two vertices in the input graph G which are not nice in H, and x, y be the other neighbors of u, v respectively. Delete the vertex u and add the edge (x, v). Here, if one of u, v belongs to $\mathcal{R}$, say $v \in \mathcal{R}$ then we delete v and add an edge between its neighbors (see Figure 1).
  When applying Reduction Rule 2, if both the degree two vertices belong to $\mathcal{R}$, then the choice of deleting one of them and adding an edge between its neighbors is arbitrary. Observe that the measure μ does not increase after the application of Reduction Rules 1 and 2.
- Reduction Rule 3. If $k < 0$, then return that $(G, W, \mathcal{R}, k)$ is a NO instance.
- Reduction Rule 4. If there is a vertex $v \in \mathcal{R}$ such that v has two neighbors in the same component of W, then return that $(G, W, \mathcal{R}, k)$ is a NO instance.
- Reduction Rule 5. If there is a vertex $v \in \mathcal{R}$ such that v has a neighbor in W, then remove v from $\mathcal{R}$ and add v to W. That is, we solve the instance $(G, W \cup \{v\}, \mathcal{R} \setminus \{v\}, k)$. Observe that by moving v to W we do not increase the number of components of $G[W \cup \{v\}]$.
- Reduction Rule 6. If there is a vertex $v \in V(H) \setminus \mathcal{R}$ such that v has at least two neighbors in the same component of W, then remove v from G and add the vertices in $N_H(v)$ to $\mathcal{R}$. That is, the resulting instance is $(G \setminus \{v\}, W, \mathcal{R} \cup N_H(v), k-1)$. In this case it is easy to observe that v must belong to any ifvs.
- Reduction Rule 7. If there is a vertex $u \in \mathcal{R}$ such that there is a leaf v in H adjacent to u and $d_W(v) \leq 2$. Then remove u from $\mathcal{R}$ and include u in W i.e. the resulting instance is $(G, W \cup \{u\}, \mathcal{R} \setminus \{u\}, k)$. Observe that moving u to W increases the number of components in W, but it also makes v either a nice vertex or a tent.
- Reduction Rule 8. Let T be a tree in H and $u \in V(T) \cap (V(H) \setminus \mathcal{R})$ such that the tree, T_u, rooted at u is a star. That is, all the children of u are leaves of T. Furthermore, each vertex in $V_u = V(T_u) \setminus \{u\}$ (all the children of u) has exactly one neighbor in W and $1 \leq |V_u| \leq 2$. Finally, assume that either $V(T) \setminus V(T_u) = \emptyset$ or the parent x of u is in $\mathcal{R}$. Then include the vertices in $V_u \cup \{x\}$ to W and remove x from $\mathcal{R}$ (if it exists) i.e. the

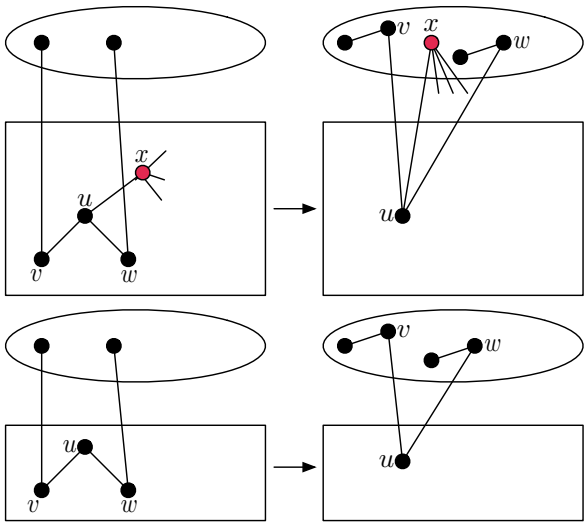

Figure 2 Illustration of Reduction Rule 8.

resulting instance is $(G, W \cup V_u \cup \{x\}, \mathcal{R} \setminus \{x\}, k)$. Here, $\{x\} = \emptyset$, if x does not exists. (see Figure 2)

▶ **Lemma 4.** *Reduction Rule 4 is safe.*

Proof. Let x, y be two neighbors of $v \in \mathcal{R}$ that are present in the same component of W. Since x, y belong to the same component of W, there is a path P in W from x to y. But then, $G[V(P) \cup \{v\}]$ contains a cycle, with v being the only vertex not in W. Therefore, there cannot exist an ifvs, $S \subseteq V(G) \setminus (W \cup \mathcal{R})$ in G. This concludes the proof. ◀

▶ **Lemma 5.** *Reduction Rule 5 is safe. Furthermore, the measure μ does not increase after application of Reduction Rule 5.*

Proof. Let $v \in \mathcal{R}$ be a vertex such that v has a neighbor in W. Note that any ifvs, $S \subseteq V(G) \setminus (W \cup \mathcal{R})$ in G does not contain v. Moreover, since v has a neighbor in W, adding v to W does not increase the number of components in W. This implies that μ does not increase. ◀

▶ **Lemma 6.** *Reduction Rule 6 is safe. Furthermore, the measure μ does not increase after application of Reduction Rule 6.*

Proof. Let $v \in V(H) \setminus \mathcal{R}$ be a vertex such that v has 2 neighbors, say x, y, in the same component of W. Since x, y belong to the same component of W, there is a path P in W from x to y. But then, $G[V(P) \cup \{v\}]$ contains a cycle, with v being the only vertex not in W. Therefore, any ifvs $S \subseteq W$ must include v and hence avoid $N_H(v)$.

When we delete v from G and decrease k by 1, the number of components in W remains the same. If v was either a nice vertex or a tent then $\eta + 2\tau$ can decrease at most by 2. Therefore, the measure μ in the resulting instance can not increase. This concludes the proof. ◀

▶ **Lemma 7.** *Reduction Rule 7 is safe and the measure μ does not increase after its application.*

Proof. Let $u \in \mathcal{R}$ be a vertex such that there is a leaf v in H adjacent to u such that $d_W(v) \leq 2$. Observe that no solution to Dis-IFVS can contain u. Therefore, we only need to show that the measure μ does not increase. When we add u to W, the number of components in W can increase by 1. But then v becomes either a nice vertex or a tent. Therefore, $\eta + 2\tau$ decreases at least by 1. This together with the fact that k remains the same imply that μ cannot increase. ◀

▶ **Lemma 8.** *Reduction Rule 8 is safe and the measure μ does not increase after its application.*

Proof. Let T be a tree in H and $u \in V(T) \cap (V(H) \setminus \mathcal{R})$ such that the tree, T_u, rooted at u is a star. That is, all the children of u are leaves of T. Furthermore, the vertices in $V_u = V(T_u) \setminus \{u\}$ (all the children of u) have exactly one neighbor in W and $1 \leq |V_u| \leq 2$. Also, either $V(T) \setminus V(T_u) = \emptyset$ or the parent x of u is in $\mathcal{R}$. To prove the lemma, we will show that if $(G, W, \mathcal{R}, k)$ is a YES instance of Dis-IFVS then there is an ifvs, $S \subseteq V(H) \setminus (W \cup \mathcal{R})$, of size at most k in G such that $S \cap V_u = \emptyset$. Observe that x (if it exists) cannot belong to S.

Let $S \subseteq V(H) \setminus (W \cup \mathcal{R})$ be an ifvs in G of size at most k. If $S \cap V_u = \emptyset$ then S is the desired solution. Otherwise, let $S' = (S \setminus V_u) \cup \{u\}$. Since $S \cap V_u \neq \emptyset$, we have that u does not belong to S and thus the size of S' is also at most k. We claim that S' is an ifvs of the desired form. Notice that $S' \subseteq V(H) \setminus (W \cup \mathcal{R})$ holds. Also, S' is an independent set since neighbors of u do not belong to S' and $S \setminus V_u$ is an independent set. Therefore, we only need to prove that S' is a feedback vertex set in G. Suppose not, then there is a cycle C in $G \setminus S'$. If C does not contain any vertex from $V_u \cup \{x\}$, then C is also a cycle in $G \setminus S$, contradicting that S in an ifvs in G. If C contains x, but does not contain any other vertex from V_u, then we can conclude that C is a cycle in $G \setminus S$, since $x \notin S$. Otherwise, C contain a vertex say, $v \in V_u$. Note that v is a degree 2 vertex in G. This implies that any cycle containing v must contain both the neighbors of v. But then u belongs to C contradicting that C is a cycle in $G \setminus S'$. This proves the safeness of the reduction rule.

When we add $V_u \cup \{x\}$ to W the number of components can increase at most by 1. Note that none of the vertices in $V_u \cup \{x\}$ is a tent. Therefore, the number of nice vertices or tents does not decrease and u becomes a nice vertex or a tent. This implies that the measure μ does not increase. This concludes the proof. ◀

Algorithm Description. We give an algorithm only for the decision variant of the problem. It is straightforward to modify the algorithm so that it actually finds a solution, provided there exists one.

We will follow a branching strategy with a nontrivial measure function. Let $(G, W, \mathcal{R}, k)$ be the input instance. The algorithm first applies Reduction Rules 1–8, in this order, exhaustively. That is, at any point of time we apply the lowest numbered applicable Reduction Rule. For clarity we denote the reduced instance (the one on which Reduction Rules 1–8 do not apply) by $(G, W, \mathcal{R}, k)$.

We now check whether every vertex in $V(G) \setminus (W \cup \mathcal{R})$ is either a nice vertex or a tent. If this is the case, then in polynomial time we can check whether or not there is an ifvs contained in $V(G) \setminus (W \cup \mathcal{R})$ that is of size at most k; and return accordingly as described by Lemma 9.

▶ **Lemma 9.** *Let (G, X) be an instance of* IFVS *where every vertex in $V(G) \setminus X$ is either a nice vertex or a tent. Then in polynomial time we can find a minimum sized ifvs $S \subseteq V(G) \setminus X$ in G.*

The proof of Lemma 9 follows from Lemma 4.10 in [5], which is based on a polynomial time algorithm for FVS in subcubic graphs by Ueno et at. [32] and the fact that the algorithm described in [5] for finding feedback vertex set on the instances of described type always returns an independent feedback vertex set (if it exists).

Finally, we move to the branching step of the algorithm. We never *branch on a nice vertex or a tent*. We will branch on the vertices in $V(H) \setminus \mathcal{R}$ based on certain criteria. We consider the following three scenarios.

- **Scenario A.** There is a vertex which in not a *tent* and has at least 3 neighbors in W.
- **Scenario B.** There is a leaf which is not a *nice vertex* and has exactly 2 neighbors in W, but no leaf has more than 2 neighbors in W.
- **Scenario C.** All the leaves have exactly one neighbor in W.

Scenario A. If there is a vertex $v \in V(H)$ which is not a *tent* and has at least 3 neighbors in W. Note that $v \notin \mathcal{R}$ as the Reduction Rule 5 is not applicable. In this case we branch on v as follows.

- When v belongs to the solution, then all the vertices in $N_H(v)$ cannot belong to the solution. Therefore, we add all the vertices in $N_H(v)$ to the set $\mathcal{R}$. The resulting instance is $(G \setminus \{v\}, W, \mathcal{R} \cup N(v), k-1)$. In this case k decreases by 1 and $\rho(W), \eta, \tau$ remains the same. Therefore, μ decreases by 2.
- When v does not belong to the solution, then we add v to W. The resulting instance is $(G, W \cup \{v\}, \mathcal{R}, k)$. Note that v cannot have two neighbors in the same component of W, otherwise Reduction Rule 6 would be applicable. Therefore, $G[W \cup \{v\}]$ has at most $\rho(W) - 2$ components. Also, k does not change and η, τ does not decrease. Therefore, μ decreases at least by 2.

The resulting branching vector for this case is $(2, 2)$. When none of the Reduction Rules are applicable and we cannot branch according to Scenario A, then we can assume that there is no vertex $v \in V(H)$, such that v has more than 2 neighbors in W. Of course a tent could have three neighbors in W but as stated before we *never* branch on a nice vertex or a tent. For each tree T (a component) in H, for a vertex $v \in V(T)$ we define the level $\ell(v)$ of v to be the distance of v from the root of T. The root r in a tree has $\ell(r) = 0$. We call a leaf vertex $v \in V(T)$ as a *deep leaf* if $\ell(v) \neq 0$ and for all leaves $v' \in V(T)$, $\ell(v') \leq \ell(v)$.

Scenario B. Let v be a leaf in some tree T in H with the unique neighbor $u \in V(H)$ such that v has exactly two neighbors in W. Observe that $u \notin \mathcal{R}$ since Reduction Rule 7 is not applicable. We branch on u as follows.

- When u belongs to the solution, then all the vertices in $N_H(u)$ cannot belong to the solution. We add all the vertices in $N_H(u) \setminus \{v\}$ to the set $\mathcal{R}$. We add the vertex v to W. The resulting instance is $(G \setminus \{u\}, W \cup \{v\}, \mathcal{R} \cup (N_H(u) \setminus \{v\}), k-1)$. In this case k decreases by 1, and η, τ do not decrease. The number of components in $G[W \cup \{v\}]$ is $\rho(W) - 1$, since v has 2 neighbors in different components of W. Therefore, μ decreases by 3.
- When u does not belong to the solution, then we add u to W. The resulting instance is $(G, W \cup \{u\}, \mathcal{R}, k)$. Note that when we add u to W then v becomes a *tent*. The number of components in $G[W \cup \{u\}]$ is at most $\rho(W) + 1$. Note that k does not increase, η does not decrease and τ increases at least by 1. Therefore, μ decreases by at least 1.

The resulting branching vector for this case is $(3, 1)$.

We now assume that all the leaves in H have exactly one neighbor in W.

Scenario C. Let v be a *deep leaf* in some tree T in H. Let the unique neighbor of v in H be u. We note that the sub-tree T_u rooted at u is a star, i.e. u is the only vertex in T_u which can possibly have degree more than one. This follows from the fact that v is a *deep leaf*. Also, $u \notin \mathcal{R}$ since Reduction Rule 7 is not applicable. We consider the following cases depending on the number of leaves in the sub-tree T_u rooted at u.

Case 1. If T_u has at least two more leaves, say x, y other than v. We branch on the vertex u as follows.

- When u belongs to the solution, then the vertices in $N_H(u)$ does not belong to the solution. We add all the vertices in $N_H(u)$ to the set $\mathcal{R}$. The resulting instance is $(G \setminus \{u\}, W, \mathcal{R} \cup N_H(u), k-1)$. In this case k decreases by 1 and $\eta, \tau, \rho(W)$ does not change. Therefore, μ decreases at least by 2.
- When u does not belong to the solution, we add u to W. The resulting instance is $(G, W \cup \{u\}, \mathcal{R}, k)$. Observe that when we add u to W then, v, x, y becomes nice vertices and the number of components in $G[W \cup \{u\}]$ is at most $\rho(W)+1$. Therefore, μ decreases at least by 2.

The resulting branching vector for this case is $(2, 2)$.

Case 2. If T_u has at most one more leaf other than v. We let x to be the parent of u in T. Note that x exists and $x \notin \mathcal{R}$ because each leaf has exactly one neighbor in W and Reduction Rules 2 and 8 are not applicable. In this case we branch on x.

- When x belongs to the solution, then the vertices in $N_H(x)$ do not belong to the solution. We add all the vertices in $N_H(x) \setminus \{u\}$ to the set $\mathcal{R}$ and add u to the set W. The resulting instance is $(G \setminus \{x\}, W \cup \{u\}, \mathcal{R} \cup (N_H(x) \setminus \{u\}), k-1)$. Observe that Reduction Rule 2 is not applicable. Therefore, at least one of the following holds.
  - u has a neighbor in W.
  - u has one more leaf v' (not in W') adjacent to it in other than v.

  In the former case, when we add u to W, the number of components in $G[W \cup \{u\}]$ is at most $\rho(W)$. Also, v becomes a *nice vertex*. Therefore, η increases at least by 1 and τ does not decrease. Therefore, μ decreases at least by 3. In the latter case when we add u to W, v, v' becomes *nice vertices*. In this case k decreases by 1, η increases by 2, τ does not decrease, and $\rho(W)$ can increase at most by 1. Therefore, μ decreases at least by 3.
- When x does not belong to the solution, we add x to W. But then T_u is a star and u does not have a parent. Therefore, we can apply the Reduction Rule 8. That is, we can add v, v' to W. The resulting instance would be $(G, W \cup \{x, v, v'\}, \mathcal{R}, k)$. Observe that u becomes a *tent*. In this case k, ρ remains the same, while τ increases by 1 and $\rho(W)$ can increase at most by 1. Therefore, μ decreases at least by 1.

The resulting branching vector for this case is $(3, 1)$.

This completes the description of the algorithm.

Analysis and Correctness of the Algorithm. The following Lemma which will be used to prove the correctness of the algorithm.

▶ **Lemma 10.** *For an instance $I = (G, W, \mathcal{R}, k)$ of* Dis-IFVS*, if $\mu < 0$, then I is a NO instance.*

Proof. Let us assume for contradiction that I is a YES instance and $\mu < 0$. Let $S \subseteq V(G) \setminus (W \cup \mathcal{R})$ be an ifvs in G of size at most k. Therefore, $F = G \setminus S$ is a forest. Let $N \subseteq V(G) \setminus (W \cup \mathcal{R})$, $T \subseteq V(G) \setminus (W \cup \mathcal{R})$ be the set of nice vertices and tents in

Table 1 The branch vectors and the corresponding running times.

Scenario	Cases	Branch Vector	c^{μ}
Scenario A		$(2, 2)$	1.4142^{μ}
Scenario B		$(3, 1)$	1.4656^{μ}
Scenario C	Case 1	$(2, 2)$	1.4142^{μ}
	Case 2	$(3, 1)$	1.4656^{μ}

$V(G) \setminus (W \cup \mathcal{R})$, respectively. Since F is a forest we have that $G' = G[(W \cup N \cup T) \setminus S]$ is a forest. In G', we contract each of the components in W to a single vertex to obtain a forest $\tilde{F}$. Observe that $\tilde{F}$ has at most $|V(\tilde{F})| \leq \rho(W) + |N \setminus S| + |T \setminus S|$ vertices and thus can have at most $\rho(W) + |N \setminus S| + |T \setminus S| - 1$ many edges. The vertices in $(N \cup T) \setminus S \subseteq V(G) \setminus (W \cup \mathcal{R})$ forms an independent set in $\tilde{F}$, since they are nice vertices or tents. The vertices in $N \setminus S$ and $T \setminus S$ have degree 2 and degree 3 in $\tilde{F}$, respectively, since their degree cannot drop while contracting the components of $G[W]$. This implies that,

$$2|N \setminus S| + 3|T \setminus S| \quad \leq |E(\tilde{F})| \leq \quad \rho(W) + |N \setminus S| + |T \setminus S| - 1.$$

Therefore, $|N \setminus S| + 2|T \setminus S| < \rho(W)$. But $N \cap T = \emptyset$ and thus

$$|N| + 2|T| < \rho(W) + 2|S| \leq \rho(W) + 2k. \tag{1}$$

However, by our assumption, $\mu(I) = \rho(W) + 2k - (|N| + 2|T|) < 0$ and thus $|N| + 2|T| > \rho(W) + k$. This, contradicts the inequality given in Equation 1 contradicting our assumption that I is a YES instance. ◀

▶ **Lemma 11.** *The algorithm presented for* Dis-IFVS *is correct.*

Proof. Let $I = (G, W, \mathcal{R}, k)$ be an instance of Dis-IFVS. We prove the correctness of the algorithm by induction on $\mu = \mu(I) = 2k + \rho(W) - (\eta + 2\tau)$. The base case occurs in one of the following cases.

- $\mu < 0$. By Lemma 10, when $\mu < 0$, we can correctly conclude that I is a NO instance.
- $k < 0$. By Reduction Rule 3 it follows that when $k < 0$, we can correctly conclude that I is a NO instance.
- When none of the Reduction Rules and Branching Rules are applicable. In this case we are able to solve the instance in polynomial time.

By induction hypothesis we assume that for all $\mu \leq l$, the algorithm is correct. We will now prove that the algorithm is correct when $\mu = l + 1$. The algorithm does one of the following. Either applies one of the Reduction Rules if applicable. By Lemma 4 to Lemma 8 we know that the Reduction Rules correctly concludes that I is a NO instance or produces an equivalent instance I' with $\mu(I') \leq \mu(I)$. If $\mu(I') < \mu(I)$, then by induction hypothesis and safeness of the Reduction Rules the algorithm correctly decides if I is a yes instance or not. Otherwise, $\mu(I') = \mu(I)$. If none of the Reduction Rules are applicable then the algorithm applies one of the Branching Rules. Branching Rules are exhaustive and covers all possible cases. Furthermore, μ decreases in each of the branch by at least one. Therefore, by the induction hypothesis, the algorithm correctly decides whether I is a YES instance or not. ◀

▶ **Theorem 12.** *The algorithm presented solves* Disjoint Independent Feedback Vertex Set *in time* $\mathcal{O}^{\star}(3.1481^{k})$.

Proof. The Reduction Rules 1 to 8 can be applied in time polynomial in the input size. Also, at each of the branch we spend a polynomial amount of time. At each of the recursive calls in a branch, the measure μ decreases at least by 1. When $\mu \leq 0$, then we are able to solve the remaining instance in polynomial time or correctly conclude that the corresponding branch cannot lead to a solution. At the start of the algorithm $\mu \leq 3k$. The worst-case branching vector for the algorithm is $(3, 1)$ (see Table 1). The recurrence for the worst case branching vector is:

$$T(\mu) \leq T(\mu - 3) + T(\mu - 1)\,.$$

The running time corresponding to the above recurrence relation is 3.1481^k. ◀

3.2 A family of counter examples to Song's Algorithm for Independent Feedback Vertex Set

Let $\mathcal{F}$ be the family of even cycles. For any $C \in \mathcal{F}$, let (C_W, C_H) be a bipartition of C. Given a graph G and a feedback vertex set F in G, Lemma 3.1 of [30] claims to output a minimum IFVS in G. But for $G = C$ and $F = C_W$, where $C \in \mathcal{F}$, the algorithm of Lemma 3.1 always returns $\emptyset$.

4 Conclusion

In this paper we studied the Independent Feedback Vertex Set problem in the realm of parameterized algorithms, moderately exponential time algorithms and combinatorial upper bounds. We gave the fastest known deterministic algorithms for the problem running in times $\mathcal{O}^*(4.1481^k)$ and $\mathcal{O}^*(1.5981^n)$, respectively. Finally, we showed that the number of minimal ifvses in any graph on n vertices is upper bounded by 1.7485^n. We also complemented the upper bound result by obtaining a family of graphs where the number of minimal ifvses is at least $3^{n/3}$. Improving running time of all our algorithms is an interesting question. We conclude the paper with few concrete open problems.

- Does Independent Feedback Vertex Set admit a kernel of size $\mathcal{O}(k^2)$?
- Could we close the gap (or even bring closer) between the upper bound and the lower bounds on the number of minimal ifvses in any graph on n vertices?

Acknowledgements. We thank Amer E. Mouawad and Prafullkumar P. Tale for discussions and help in programming.

References

1 Akanksha Agrawal, Daniel Lokshtanov, Amer E. Mouawad, and Saket Saurabh. Simultaneous Feedback Vertex Set: A Parameterized Perspective. In *Proc. of the 33rd Symposium on Theoretical Aspects of Computer Science (STACS'16)*, volume 47 of *LIPIcs*, pages 7:1–7:15. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.STACS.2016.7.

2 Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics*, 12(3):289–297, 1999.

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

4 Jianer Chen, Fedor V. Fomin, Yang Liu, Songjian Lu, and Yngve Villanger. Improved algorithms for feedback vertex set problems. *Journal of Computer and System Sciences*, 74(7):1188–1198, 2008.

5 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

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

7 Marek Cygan, Marcin Pilipczuk, and Michal Pilipczuk. On group feedback vertex set parameterized by the size of the cutset. *Algorithmica*, 74(2):630–642, 2016.

8 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Subset feedback vertex set is fixed-parameter tractable. *SIAM Journal of Discrete Mathematics*, 27(1):290–309, 2013.

9 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

10 Rod G. Downey and Michael R. Fellows. *Parameterized complexity*. Springer-Verlag, 1997.

11 Paola Festa, Panos M. Pardalos, and Mauricio G. C. Resende. Feedback set problems. In *Handbook of combinatorial optimization, Supplement Vol. A*, pages 209–258. Kluwer Acad. Publ., Dordrecht, 1999.

12 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

13 Fedor V. Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact algorithms via monotone local search. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC'16)*, pages 764–775, 2016.

14 Fedor V. Fomin, Serge Gaspers, Artem V. Pyatkin, and Igor Razgon. On the minimum feedback vertex set problem: Exact and enumeration algorithms. *Algorithmica*, 52(2):293–307, 2008.

15 Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A Measure & Conquer approach for the analysis of exact algorithms. *Journal of ACM*, 56(5), 2009.

16 Fedor V. Fomin, Pinar Heggernes, Dieter Kratsch, Charis Papadopoulos, and Yngve Villanger. Enumerating minimal subset feedback vertex sets. *Algorithmica*, 69(1):216–231, 2014.

17 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Springer, 2010. An EATCS Series: Texts in Theoretical Computer Science.

18 Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. Large induced subgraphs via triangulations and CMSO. *SIAM Journal on Computing*, 44(1):54–87, 2015.

19 G. R. Grimmett. An upper bound for the number of spanning trees of a graph. *Discrete Mathematics*, 16(4):323–324, 1976.

20 Sylvain Guillemot. FPT algorithms for path-transversal and cycle-transversal problems. *Discrete Optimization*, 8(1):61–71, 2011.

21 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Plenum Press, New York, 1972.

22 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Information Processing Letters*, 114(10):556–560, 2014.

23 Daniel Lokshtanov, M. S. Ramanujan, and Saket Saurabh. Linear time parameterized algorithms for subset feedback vertex set. In *Automata, Languages, and Programming – 42nd International Colloquium, ICALP*, volume 9134, pages 935–946, 2015.

24 Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, and Saket Saurabh. On parameterized independent feedback vertex set. *Theoretical Computer Science*, 461:65–75, 2012.

25 Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, Saket Saurabh, and Somnath Sikdar. FPT algorithms for connected feedback vertex set. *Journal of Combinatorial Optimization*, 24(2):131–146, 2012.

26 J. W. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3:23–28, 1965.

27 Rolf Niedermeier. Invitation to fixed-parameter algorithms, 2006.

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

29 Igor Razgon. Exact computation of maximum induced forest. In *Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT 2006)*, volume 4059, pages 160–171, 2006.

30 Yinglei Song. An improved parameterized algorithm for the independent feedback vertex set problem. *Theoretical Computer Science*, 535:25–30, 2014.

31 Yuma Tamura, Takehiro Ito, and Xiao Zhou. Algorithms for the independent feedback vertex set problem. *IEICE Transactions*, 98-A(6):1179–1188, 2015.

32 Shuichi Ueno, Yoji Kajitani, and Shin'ya Gotoh. On the nonseparating independent set problem and feedback set problem for graphs with no vertex degree exceeding three. *Discrete Mathematics*, 72(1):355–360, 1988.

33 Magnus Wahlström. Half-integrality, LP-branching and FPT algorithms. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1762–1781, 2014.