On Feedback Vertex Set
New Measure and New Structures*

Yixin Cao[1], Jianer Chen[1], and Yang Liu[2]

[1] Department of Computer Science and Engineering
Texas A&M University
{yixin,chen}@cse.tamu.edu
[2] Department of Computer Science
University of Texas - Pan American
yliu@cs.panam.edu

Abstract. We study the parameterized complexity of the FEEDBACK VERTEX SET problem (FVS) on undirected graphs. We approach the problem by considering a variation of it, the DISJOINT FEEDBACK VERTEX SET problem (DISJOINT-FVS), which finds a disjoint feedback vertex set of size k when a feedback vertex set of a graph is given. We show that DISJOINT-FVS admits a small kernel, and can be solved in polynomial time when the graph has a special structure that is closely related to the maximum genus of the graph. We then propose a simple branch-and-search process on DISJOINT-FVS, and introduce a new branch-and-search measure. The branch-and-search process effectively reduces a given graph to a graph with the special structure, and the new measure more precisely evaluates the efficiency of the branch-and-search process. These algorithmic, combinatorial, and topological structural studies enable us to develop an $O(3.83^k kn^2)$ time parameterized algorithm for the general FVS problem, improving the previous best algorithm of time $O(5^k kn^2)$ for the problem.

1 Introduction

All graphs in our discussion are supposed to be undirected. A *feedback vertex set* (FVS) F in G is a set of vertices in G whose removal results in an acyclic graph. The problem of finding a minimum feedback vertex set in a graph is one of the classical NP-complete problems [16]. The history of the problem can be traced back to early '60s. For several decades, many different algorithmic approaches were tried on this problem, including approximation algorithms, linear programming, local search, polyhedral combinatorics, and probabilistic algorithms (see the survey [10]). There are also exact algorithms finding a minimum FVS in a graph of n vertices in time $\mathcal{O}(1.9053^n)$ [21] and in time $\mathcal{O}(1.7548^n)$ [11].

An important application of the FVS problem is *deadlock recovery* in operating systems [23], in which a deadlock is presented by a cycle in a *system resource-allocation graph* G. Thus, to recover from deadlocks, we need to abort a set of

* Supported in part by the US NSF under the Grants CCF-0830455 and CCF-0917288.

H. Kaplan (Ed.): SWAT 2010, LNCS 6139, pp. 93–104, 2010.

processes in the system, i.e., to remove a set of vertices in the graph G, so that all cycles in G are broken. Equivalently, we need to find an FVS in G.

In a practical system resource-allocation graph G, it can be expected that the size k of the minimum FVS in G, i.e., the number of vertices in the FVS, is fairly small. This motivated the study of the parameterized version of the problem, which we will name FVS: given a graph G and a parameter k, either construct an FVS of size bounded by k in G or report no such an FVS exists. Parameterized algorithms for the FVS problem have been extensively investigated that find an FVS of k vertices in a graph of n vertices in time $f(k)n^{\mathcal{O}(1)}$ for a fixed function f (thus, the algorithms become practically efficient when the value k is small). The first group of parameterized algorithms for FVS was given by Bodlaender [2] and by Downey and Fellows [8]. Since then a chain of dramatic improvements was obtained by different researchers (see Figure 1).

Authors	Complexity	Year
Bodlaender[2] Downey and Fellows [8]	$\mathcal{O}(17(k^4)!n^{\mathcal{O}(1)})$	1994
Downey and Fellows [9]	$\mathcal{O}((2k+1)^k n^2)$	1999
Raman et al.[20]	$\mathcal{O}(\max\{12^k, (4\log k)^k\}n^{2.376})$	2002
Kanj et al.[15]	$\mathcal{O}((2\log k + 2\log\log k + 18)^k n^2)$	2004
Raman et al.[19]	$\mathcal{O}((12\log k/\log\log k + 6)^k n^{2.376})$	2006
Guo et al.[14]	$\mathcal{O}((37.7)^k n^2)$	2006
Dehne et al.[7]	$\mathcal{O}((10.6)^k n^3)$	2005
Chen et al.[5]	$\mathcal{O}(5^k kn^2)$	2008
This paper	$\mathcal{O}(3.83^k kn^2)$	2010

Fig. 1. The history of parameterized algorithms for the unweighted FVS problem

Randomized parameterized algorithms have also been studied for the problem. The best randomized parameterized algorithm for the problems is due to Becker et al. [1], which runs in time $\mathcal{O}(4^k kn^2)$.

The main result of the current paper is an algorithm that solves the FVS problem. The running time of our algorithm is $\mathcal{O}(3.83^k kn^2)$. This improves a long chain of results in parameterized algorithms for the problem. We remark that the running time of our (deterministic) algorithm is even faster than that of the previous best randomized algorithm for the problem as given in [1].

Our approach, as some of the previous ones, is to study a variation of the FVS problem, the DISJOINT FEEDBACK VERTEX SET problem (DISJOINT-FVS), which finds a disjoint feedback vertex set of size k in a graph G when a feedback vertex set of G is given. Our significant contribution to this research includes:

1. A new technique that produces a kernel of size $3k$ for the DISJOINT-FVS problem, and improves the previous best kernel of size $4k$ for the problem [7]. The new kernelization technique is based on a branch and search algorithm for the problem, which is, to our best knowledge, the first time used in the literature of kernelization;

2. A polynomial time algorithm that solves the DISJOINT-FVS problem when the input graph has a special structure;
3. A branch and search process that effectively reduces an input instance of DISJOINT-FVS to an instance of the special structure as given in 2;
4. A new measure that more precisely evaluates the efficiency of the branch and search process in 3;
5. A new algorithm for the FVS problem that significantly improves previous algorithms for the problem.

Due to space limitations, we omit some proofs and refer interested readers to the extended version of the current paper [3].

2 DISJOINT-FVS and Its kernel

We start with a precise definition of our problem.

> DISJOINT-FVS. Given a graph $G = (V, E)$, an FVS F in G, and a parameter k, either construct an FVS F' of size k in G such that $F' \subseteq V \setminus F$, or report that no such an FVS exists.

Let $V_1 = V \setminus F$. Since F is an FVS, the subgraph induced by V_1 must be a forest. Moreover, if the subgraph induced by F is not a forest, then it is impossible to have an FVS F' in G such that $F' \subseteq V \setminus F$. Therefore, an instance of DISJOINT-FVS can be written as $(G; V_1, V_2; k)$, and consists of a partition (V_1, V_2) of the vertex set of the graph G and a parameter k such that both V_1 and V_2 induce forests (where $V_2 = F$). We will call an FVS entirely contained in V_1 a *V_1-FVS*. Thus, the instance $(G; V_1, V_2; k)$ of DISJOINT-FVS is looking for a V_1-FVS of size k in the graph G.

Given an instance $(G; V_1, V_2; k)$ of DISJOINT-FVS, we apply the following rules:

> **Rule 1.** Remove all degree-0 vertices; and remove all degree-1 vertices;
> **Rule 2.** For a degree-2 vertex v in V_1,
> - if both neighbors of v are in the same connected component of $G[V_2]$, then include v into the objective V_1-FVS, $G = G \setminus v$, and $k = k - 1$;
> - otherwise, move v from V_1 to V_2: $V_1 = V_1 \setminus \{v\}$, $V_2 = V_2 \cup \{v\}$.

Our kernelization algorithm is based on an algorithm proposed in [5], which can be described as follows: on a given instance $(G; V_1, V_2; k)$ of DISJOINT-FVS, keep all vertices in V_1 of degree at least 3 (whenever a vertices in V_1 becomes degree less than 3, applying Rules 1-2 on the vertex), and repeatedly branch on a leaf in the induced subgraph $G[V_1]$. In particular, if the graph G has a V_1-FVS of size bounded by k, then at least one $\mathcal{P}$ of the computational paths in the branching program will return a V_1-FVS F of size bounded by k. The computational path $\mathcal{P}$ can be described by the algorithm in Figure 2.

Lemma 1. *If none of Rule 1 and Rule 2 is applicable on an instance $(G; V_1, V_2; k)$ of* DISJOINT-FVS*, and $|V_1| > 2k + l - \tau$, then there is no V_1-FVS of size bounded by k in G, where l is the number of connected components in $G[V_2]$ and τ is the number of connected components in $G[V_1]$.*

```
Algorithm FindingFVS(G, V1, V2, k)
INPUT: an instance (G; V1, V2; k) of DISJOINT-FVS.
OUTPUT: a V1-FVS F of size bounded by k in G.
    F = ∅;
    while |V1| > 0 do
        pick a leaf w in G[V1];
        case 1: \\ w is in the objective V1-FVS F.
            add w to F and remove w from V1;  k = k − 1;
            if the neighbor u of w in G[V1] becomes degree-2
            then apply Rule 2 on u;
        case 2: \\ w is not in the objective V1-FVS F.
            move w from V1 to V2.
```

Fig. 2. The computational path $\mathcal{P}$ that finds the V_1-FVS F of size bounded by k

Note that for those DISJOINT-FVS instances we will meet in Section 4, we always have $|V_2| = k + 1$, which is exactly the characteristic of the iterative compression technique. Also by the simple fact that $l \leq |V_2|$ and $\tau > 0$, we have $2k + l - \tau \leq 3k$, so the kernel size is also bounded by $3k$. With more careful analysis, we can further improve the kernel size to $3k - \tau - \rho(V_1)$, where $\rho(V_1)$ is the size of a maximum matching of the subgraph induced by the vertex set V_1' that consists of all vertices in V_1 of degree larger than 3. The detailed analysis for this fact is given in a complete version of the current paper.

3 A Polynomial Time Solvable Case for DISJOINT-FVS

In this section we consider a special class of instances for the DISJOINT-FVS problem. This approach is closely related to the classical study on graph maximum genus embeddings [4,12]. However, the study on graph maximum genus embeddings that is related to our approach is based on general spanning trees of a graph, while our approach must be restricted to only spanning trees that are constrained by the vertex partition (V_1, V_2) of an instance $(G; V_1, V_2; k)$ of DISJOINT-FVS. We start with the following simple lemma.

Lemma 2. *Let G be a connected graph and let S be a subset of vertices in G such that the induced subgraph $G[S]$ is a forest. Then there is a spanning tree in G that contains the entire induced subgraph $G[S]$, and can be constructed in time $O(m\alpha(n))$, where $\alpha(n)$ is the inverse of Ackermann function [6].*

Let $(G; V_1, V_2; k)$ be an instance for the DISJOINT-FVS problem, recall that (V_1, V_2) is a partition of the vertex set of the graph G such that both induced subgraphs $G[V_1]$ and $G[V_2]$ are forests. By Lemma 2, there is a spanning tree T of the graph G that contains the entire induced subgraph $G[V_2]$. Call a spanning tree that contains the induced subgraph $G[V_2]$ a $T_{G[V_2]}$*-tree*.

Let T be a $T_{G[V_2]}$-tree of the graph G. By the construction, every edge in $G-T$ has at least one end in V_1. Two edges in $G-T$ are V_1-*adjacent* if they have a common end in V_1. A V_1-*adjacency matching* in $G-T$ is a partition of the edges in $G-T$ into groups of one or two edges, called *1-groups* and *2-groups*, respectively, such that two edges in the same 2-group are V_1-adjacent. A *maximum* V_1-*adjacency matching* in $G-T$ is a V_1-adjacency matching in $G-T$ that maximizes the number of 2-groups.

Definition 1. *Let $(G; V_1, V_2; k)$ be an instance of* DISJOINT-FVS*. The V_1-adjacency matching number $\mu(G,T)$ of a $T_{G[V_2]}$-tree T in G is the number of 2-groups in a maximum V_1-adjacency matching in $G-T$. The V_1-adjacency matching number $\mu(G)$ of the graph G is the largest $\mu(G,T)$ over all $T_{G[V_2]}$-trees T in G.*

An instance $(G; V_1, V_2; k)$ of DISJOINT-FVS is *3-regular*$_{V_1}$ if every vertex in the vertex set V_1 has degree exactly 3. Let $f_{V_1}(G)$ be the size of a minimum V_1-FVS for G. Let $\beta(G)$ be the *Betti number* of the graph G that is the total number of edges in $G-T$ for any spanning tree T in G (or equivalently, $\beta(G)$ is the number of *fundamental cycles* in G) [12]. The following lemma is a nontrivial generalization of a result in [17] (the result in [17] is a special case for Lemma 3 in which all vertices in the set V_2 have degree 2).

Lemma 3. *For any 3-regular$_{V_1}$ instance $(G; V_1, V_2; k)$ of* DISJOINT-FVS*, $f_{V_1}(G) = \beta(G) - \mu(G)$. Moreover, a minimum V_1-FVS can be constructed in linear time from a $T_{G[V_2]}$-tree whose V_1-adjacency matching number is $\mu(G)$.*

By Lemma 3, in order to construct a minimum V_1-FVS for a 3-regular$_{V_1}$ instance $(G; V_1, V_2, k)$ of DISJOINT-FVS, we only need to construct a $T_{G[V_2]}$-tree in the graph G whose V_1-adjacency matching number is $\mu(G)$. The construction of an unconstrained maximum adjacency matching in terms of general spanning trees has been considered by Furst, Gross and McGeoch in their study of graph maximum genus embeddings [12]. We follow a similar approach, based on cographic matroid parity, to construct a $T_{G[V_2]}$-tree in G whose V_1-adjacency matching number is $\mu(G)$. We start with a quick review on the related concepts in matroid theory. Detailed discussion on matroid theory can be found in [18].

A *matroid* is a pair $(E, \Im)$, where E is a finite set and $\Im$ is a collection of subsets of E that satisfies: (1) If $A \in \Im$ and $B \subseteq A$, then $B \in \Im$; (2) If $A, B \in \Im$ and $|A| > |B|$, then there is an element $a \in A - B$ such that $B \cup \{a\} \in \Im$.

The *matroid parity* problem is stated as follows: given a matroid $(E, \Im)$ and a perfect pairing $\{[a_1, \overline{a}_1], [a_2, \overline{a}_2], \ldots, [a_n, \overline{a}_n]\}$ of the elements in the set E, find a largest subset P in $\Im$ such that for all i, $1 \le i \le n$, either both a_i and $\overline{a}_i$ are in P, or neither of a_i and $\overline{a}_i$ is in P.

Each connected graph G is associated with a *cographic matroid* $(E_G, \Im_G)$, where E_G is the edge set of G, and an edge set S is in $\Im_G$ if and only if $G-S$ is connected. It is well-known that matroid parity problem for cographic matroids can be solved in polynomial time [18]. The fastest known algorithm for cographic matroid parity problem runs in time $\mathcal{O}(mn \log^6 n)$ [13].

In the following, we explain how to reduce our problem to the cographic matroid parity problem. Let $(G; V_1, V_2; k)$ be a 3-regular$_{V_1}$ instance of the DISJOINT-FVS problem. Without loss of generality, we make the following assumptions: (1) the graph G is connected (otherwise, we simply work on each connected component of G); and (2) for each vertex v in V_1, there is at most one edge from v to a connected component in $G[V_2]$ (otherwise, we can directly include v in the objective V_1-FVS).

Recall that two edges are V_1*-adjacent* if they share a common end in V_1. For an edge e in G, denote by $d_{V_1}(e)$ the number of edges in G that are V_1-adjacent to e (note that an edge can be V_1-adjacent to the edge e from either end of e).

We construct a *labeled subdivision* G_2 of the graph G as follows.

1. shrink each connected component of $G[V_2]$ into a single vertex; let the resulting graph be G_1;
2. assign each edge in G_1 a distinguished label;
3. for each edge labeled e_0 in G_1, suppose that the edges V_1-adjacent to e_0 are labeled by $e_1, e_2, \ldots, e_d$ (the order is arbitrary), where $d = d_{V_1}(e_0)$; subdivide e_0 into d *segment edges* by inserting $d-1$ degree-2 vertices in e_0, and label the segment edges by $(e_0e_1), (e_0e_2), \ldots, (e_0e_d)$. Let the resulting graph be G_2. The segment edges $(e_0e_1), (e_0e_2), \ldots, (e_0e_d)$ in G_2 are said to be *from* the edge e_0 in G_1.

There are a number of interesting properties for the graphs constructed above. First, each of the edges in the graph G_1 corresponds uniquely to an edge in G that has at least one end in V_1. Thus, without creating any confusion, we will simply say that the edge is in the graph G or in the graph G_1. Second, because of the assumptions we made on the graph G, the graph G_1 is a simple and connected graph. In consequence, the graph G_2 is also a simple and connected graph. Finally, because each edge in G_1 corresponds to an edge in G that has at least one end in V_1, and because each vertex in V_1 has degree 3, every edge in G_1 is subdivided into at least two segment edges in G_2.

Now in the labeled subdivision graph G_2, pair the segment edge labeled (e_0e_i) with the segment edge labeled (e_ie_0) for all segment edges (note that (e_0e_i) is a segment edge from the edge e_0 in G_1 and that (e_ie_0) is a segment edge from the edge e_i in G_1). By the above remarks, this is a perfect pairing $\mathcal{P}$ of the edges in G_2. Now with this edge pairing $\mathcal{P}$ in G_2, and with the cographic matroid $(E_{G_2}, \Im_{G_2})$ for the graph G_2, we call Gabow and Stallmann's algorithm [13] for the cographic matroid parity problem. The algorithm produces a maximum edge subset P in $\Im_{G_2}$ that, for each segment edge (e_0e_i) in G_2, either contains both (e_0e_i) and (e_ie_0), or contains neither of (e_0e_i) and (e_ie_0).

Lemma 4. *From the edge subset P in $\Im_{G_2}$ constructed above, a $T_{G[V_2]}$-tree for the graph G whose V_1-adjacency matching number is $\mu(G)$ can be constructed in time $O(m\alpha(n))$, where n and m are the number of vertices and the number of edges, respectively, of the graph G.*

Now we can solve the 3-regular$_{V_1}$ instance as follows: first shrinking each connected component of $G[V_2]$ into a single vertex; then constructing the labeled subdivision graph G_2 of G, and apply Gabow and Stallmann's algorithm [13] on it to get the edge subset P in $\Im_{G_2}$; finally, building the V_1-adjacency matching M from P, and the V_1-FVS from M. This gives our main result in this section.

Theorem 1. *There is an $\mathcal{O}(n^2 \log^6 n)$ time algorithm that on a 3-regular$_{V_1}$ instance $(G; V_1, V_2; k)$ of the* DISJOINT-FVS *problem, either constructs a V_1-FVS of size bounded by k, if such a V_1-FVS exists, or reports correctly that no such a V_1-FVS exists.*

Combining Theorem 1 and Rule 2, we have

Corollary 1. *There is an $\mathcal{O}(n^2 \log^6 n)$ time algorithm that on an instance $(G; V_1, V_2; k)$ of* DISJOINT-FVS *where all vertices in V_1 have degree bounded by 3, either constructs a V_1-FVS of size bounded by k, if such an FVS exists, or reports correctly that no such a V_1-FVS exists.*

4 An Improved Algorithm for DISJOINT-FVS

Now we are ready for the general DISJOINT-FVS problem. Let $(G; V_1, V_2; k)$ be an instance of DISJOINT-FVS, for which we are looking for a V_1-FVS of size k. Observe that certain structures in the input graph G can be easily processed and then removed from G. For example,the graph G cannot contain self-loops (i.e., edges whose both ends are on the same vertices) because by definition, both induced subgraphs $G[V_1]$ and $G[V_2]$ are forests. Moreover, if two vertices v and w are connected by multiple edges, then exactly one of v and w is in V_1 and the other is in V_2 (this is again because the induced subgraphs $G[V_1]$ and $G[V_2]$ are forests). Thus, in this case, we can directly include the vertex in V_1 in the objective V_1-FVS. Therefore, for a given input graph G, we always first apply a preprocessing that applies the above operations and remove all self-loops and multiple edges in the graph G. In consequence, we can assume, without loss of generality., that the input graph G contains neither self-loops nor multiple edges.

A vertex $v \in V_1$ is a *nice V_1-vertex* if v is of degree 3 in G and all its neighbours are in V_2. Let p be the number of nice V_1-vertices in G, and let l be the number of connected components in the induced subgraph $G[V_2]$. The measure $m = k + \frac{l}{2} - p$ will be used in the analysis of our algorithm.

Lemma 5. *If the measure m is bounded by 0, then there is no V_1-FVS of size bounded by k in G. If all vertices in V_1 are nice V_1-vertices, then a minimum V_1-FVS in G can be constructed in polynomial time.*

Proof. Suppose that $m = k + \frac{l}{2} - p \leq 0$, and that there is a V_1-FVS F of size of $k' \leq k$. Let S be the set of any $p - k'$ nice V_1-vertices that are not in F. The subgraph G' induced by $V_2 \cup S$ must be a forest because F is an FVS and is disjoint with $V_2 \cup S$. On the other hand, the subgraph G' can be constructed from the induced subgraph $G[V_2]$ and the $p - k'$ discrete vertices in S, by adding the $3(p - k')$ edges that are incident to the vertices in S. Since $k' \leq k$, we have $p - k' \geq p - k \geq \frac{l}{2}$. This gives $3(p - k') = 2(p - k') + (p - k') \geq l + (p - k')$.

This contradicts the fact that G' is a forest – in order to keep G' a forest, we can add at most $l + (p - k') - 1$ edges to the structure that consists of the induced subgraph $G[V_2]$ of l connected components and the $p - k'$ discrete vertices in S. This contradiction proves the first part of the lemma.

To prove the second part of the lemma, observe that when all vertices in V_1 are nice V_1-vertices, $(G; V_1, V_2; k)$ is a 3-regular$_{V_1}$ instance for DISJOINT-FVS. By Theorem 1, there is a polynomial time algorithm that constructs a minimum V_1-FVS in G for 3-regular$_{V_1}$ instances of DISJOINT-FVS. □

The algorithm **Feedback**(G, V_1, V_2, k), for the DISJOINT-FVS problem is given in Figure 3. We first discuss the correctness of the algorithm. The correctness of step 1 and step 2 of the algorithm is obvious. By lemma 5, step 3 is correct. Step 4 is correct by Rule 1 in section 2. After step 4, each vertex in V_1 has degree at least 2 in G.

If the vertex w has two neighbors in V_2 that belong to the same tree T in the induced subgraph $G[V_2]$, then the tree T plus the vertex w contains at least one cycle. Since we are searching for a V_1-FVS, the only way to break the cycles in $T \cup \{w\}$ is to include the vertex w in the objective V_1-FVS. Moreover, the objective V_1-FVS of size at most k exists in G if and only if the remaining graph $G - w$ has a V_1-FVS of size at most $k - 1$ in the subset $V_1 \setminus \{w\}$. Therefore, step 5 correctly handles this case. After this step, all vertices in V_1 has at most one neighbor in a tree in $G[V_2]$.

Because of step 5, a degree-2 vertex at step 6 cannot have both its neighbors in the same tree in $G[V_2]$. By Rule 2, step 6 correctly handles this case. After step 6, all vertices in V_1 have degree at least 3.

A vertex $w \in V_1$ is either in or not in the objective V_1-FVS. If w is in the objective V_1-FVS, then we should be able to find a V_1-FVS F_1 in the graph $G-w$ such that $|F_1| \leq k - 1$ and $F_1 \subseteq V_1 \setminus \{w\}$. On the other hand, if w is not in the objective V_1-FVS, then the objective V_1-FVS for G must be contained in the subset $V_1 \setminus \{w\}$. Also note that in this case, the induced subgraph $G[V_2 \cup \{w\}]$ is still a forest since no two neighbors of w in V_2 belong to the same tree in $G[V_2]$. Therefore, step 7 handles this case correctly. After step 7, every leaf w in $G[V_1]$ that is not a nice V_1-vertex has exactly two neighbors in V_2.

The vertex y in step 8 is either in or not in the objective V_1-FVS . If y is in the objective V_1-FVS, then we should be able to find a V_1-FVS F_1 in the graph $G - y$ such that $|F_1| \leq k - 1$ and $F_1 \subseteq V_1 \setminus \{w\}$. After removing y from the graph G, the vertex w becomes degree-2 and both of its neighbors are in V_2 (note that step 7 is not applicable to w). Therefore, by Rule 2, the vertex w can be moved from V_1 to V_2 (again note that $G[V_2 \cup \{w\}]$ is a forest). On the other hand, if y is not in the objective V_1-FVS, then the objective FVS for G must be contained in the subset $V_1 \setminus \{y\}$. Also note that in this case, the subgraph $G[V_2 \cup \{y\}]$ is a forest since no two neighbors of y in V_2 belong to the same tree in $G[V_2]$. Therefore, step 8 handles this case correctly. Thus, the following conditions hold after step 8:

1. $k > 0$ and G is not a forest (by steps 1 and 2);
2. $p \leq k + \frac{l}{2}$ and not all vertices of V_1 are nice vertices (by step 3);

Algorithm Feedback(G, V_1, V_2, k)
INPUT: an instance $(G; V_1, V_2; k)$ of DISJOINT-FVS.
OUTPUT: a V_1-FVS F of size bounded by k in G if such a V_1-FVS exists.
1 **if** $(k < 0)$ or ($k = 0$ and G is not a forest) **then** return 'No';
2 **if** $k \geq 0$ and G is a forest **then** return $\emptyset$;
let l be the number of connected components in $G[V_2]$,
and let p be the number of nice V_1-vertices;
3 **if** $p > k + \frac{l}{2}$ **then** return 'No';
if $p = |V_1|$ **then** solve the problem in polynomial time;
4 **if** a vertex $w \in V_1$ has degree not larger than 1 **then**
return **Feedback**$(G - w, V_1 \setminus \{w\}, V_2, k)$;
5 **if** a vertex $w \in V_1$ has two neighbors in the same tree in $G[V_2]$ **then**
F_1 = **Feedback**$(G - w, V_1 \setminus \{w\}, V_2, k - 1)$;
if F_1='No' **then** return 'No' **else** return $F_1 \cup \{w\}$
6 **if** a vertex $w \in V_1$ has degree 2 **then**
return **Feedback**$(G, V_1 \setminus \{w\}, V_2 \cup \{w\}, k)$;
7 **if** a leaf w in $G[V_1]$ is not a nice V_1-vertex and has ≥ 3 neighbors in V_2
F_1 = **Feedback**$(G - w, V_1 - \{w\}, V_2, k - 1)$;
7.1 **if** $F_1 \neq$ 'No' **then** return $F_1 \cup \{w\}$
7.2 **else** return **Feedback**$(G, V_1 \setminus \{w\}, V_2 \cup \{w\}, k)$;
8 **if** the neighbor $y \in V_1$ of a leaf w in $G[V_1]$ has at least one neighbor in V_2
F_1 = **Feedback**$(G - y, V_1 \setminus \{w, y\}, V_2 \cup \{w\}, k - 1)$;
8.1 **if** $F_1 \neq$'No' **then** return $F_1 \cup \{y\}$
8.2 **else** return **Feedback**$(G, V_1 \setminus \{y\}, V_2 \cup \{y\}, k)$;
9 pick a lowest leaf w_1 in any tree T in $G[V_1]$;
let $w_1, \cdots, w_t$ be the children of w in T;
F_1 = **Feedback**$(G - w, V_1 \setminus \{w, w_1\}], V_2 \cup \{w_1\}, k - 1)$;
9.1 **if** $F_1 \neq$'No' **then** return $F_1 \cup \{w\}$
9.2 **else** return **Feedback**$(G, V_1 \setminus \{w\}, V_2 \cup \{w\}, k)$.

Fig. 3. Algorithm for DISJOINT-FVS

3. any vertex in V_1 has degree at least 3 in G (by steps 4-6);
4. any leaf in $G[V_1]$ is either a nice V_1-vertex, or has exactly two neighbors in V_2 (by step 7); and
5. for any leaf w in $G[V_1]$, the neighbor $y \in V_1$ of w has no neighbors in V_2 (by step 8).

By condition 4, any tree of single vertex in $G[V_1]$ is a nice V_1-vertex. By condition 5, there is no tree of two vertices in $G[V_1]$. For a tree T with at least three vertices in $G[V_1]$, fix any internal vertex of T as the root. Then we can find a *lowest leaf* w_1 of T in polynomial time. Since the tree T has at least three vertices, the vertex w_1 must have a parent w in T which is in $G[V_1]$.

Vertex w is either in or not in the objective V_1-FVS. If w is in the objective V_1-FVS, then we should find a V_1-FVS F_1 in the graph $G-w$ such that $F_1 \subseteq V_1 \setminus \{w\}$ and $|F_1| \leq k-1$. Note that after removing w, the leaf w_1 becomes degree-2, and by Rule 2, it is valid to move w_1 from V_1 to V_2 since the two neighbors of w_1 in V_2 are not in the same tree in $G[V_2]$. On the other hand, if w is not in the objective V_1-FVS, then the objective V_1-FVS must be in $V_1 \setminus \{w\}$. In summary, step 9 handles this case correctly.

Theorem 2. *The algorithm* **Feedback**(G, V_1, V_2, k) *correctly solves the* DISJOINT-FVS *problem. The running time of the algorithm is* $\mathcal{O}(2^{k+l/2}n^2)$*, where* n *is the number of vertices in* G*, and* l *is the number of connected components in the induced subgraph* $G[V_2]$*.*

Proof. The correctness of the algorithm has been verified by the above discussion. Now we consider the complexity of the algorithm. The recursive execution of the algorithm can be described as a search tree $\mathcal{T}$. We first count the number of leaves in the search tree $\mathcal{T}$. Note that only steps 7, 8 and 9 of the algorithm correspond to branches in the search tree $\mathcal{T}$. Let $T(m)$ be the number of leaves in the search tree $\mathcal{T}$ for the algorithm **Feedback**(G, V_1, V_2, k) when $m = k+l/2-p$, where l is the number of connected components (i.e., trees) in the forest $G[V_2]$, and p is the number of nice V_1-vertices.

The branch of step 7.1 has that $k' = k - 1$, $l' = l$ and $p' \geq p$. Thus we have $m' = k' + l'/2 - p' \leq k - 1 + l/2 - p = m - 1$. The branch of step 7.2 has that $k'' = k$, $l'' \leq l - 2$ and $p'' = p$. Thus we have $m'' = k'' + l''/2 - p'' \leq m - 1$. Thus, for step 7, the recurrence is $T(m) \leq 2T(m - 1)$.

The branch of step 8.1 has that $k' = k - 1$, $l' = l - 1$ and $p' \geq p$. Thus we have $m' = k' + l'/2 - p' \leq k - 1 + (l - 1)/2 - p = m - 1.5$. The branch of step 8.2 has that $k'' = k$, $l'' = l$ and $p'' = p + 1$. Thus we have $m'' = k'' + l''/2 - p'' = k + l/2 - (p + 1) = m - 1$. Thus, for step 8, the recurrence is $T(m) \leq T(m - 1.5) + T(m - 1)$.

The branch of step 9.1 has that $k' = k - 1$, $l' = l - 1$ and $p' \geq p$. Thus we have $m' = k' + l'/2 - p' \leq k - 1 + (l - 1)/2 - p = m - 1.5$. the branch of step 9.2 has that $k'' = k$, $l'' = l+1$ because of w, and $p'' \geq p+2$ because w has at least two children which are leaves. Thus we have $m'' = k'' + l''/2 - p'' \leq k + (l + 1)/2 - (p + 2) = m - 1.5$. Thus, for step 8, the recurrence is $T(m) \leq 2T(m - 1.5)$.

The worst case happens at step 7. From the recurrence of step 7, we have $T(m) \leq 2^m$. Moreover, steps 1-3 just return an answer; step 4 does not increase measure m since vertex w is not a nice vertex; and step 5 also does not increase m since k decreases by 1 and p decreases by at most 1. Step 6 may increase measure m by 0.5 since l may increase by 1. However, we can simply just bypass vertex w in step 6, instead of putting it into V_2. If we bypass w, then measure m does not change. In Rule 2, we did not bypass w because it is easier to analyze the kernel in section 2 by putting w into V_2. Since $m = k + l/2 - p \leq k + l/2$, and it is easy to verify that the computation time along each path in the search tree $\mathcal{T}$ is bounded by $O(n^2)$, we conclude that the algorithm **Feedback**(G, V_1, V_2, k) solves the DISJOINT FVS problem in time $O(2^{k+l/2}n^2)$. □

5 Concluding Result: An Improved Algorithm for FVS

The results presented in previous sections lead to an improved algorithm for the general FVS problem. Following the idea of *iterative compression* proposed by Reed et al. [22], we formulate the following problem:

> FVS REDUCTION: given a graph G and an FVS F of size $k+1$ for G, either construct an FVS of size at most k for G, or report that no such an FVS exists.

Lemma 6. *The* FVS REDUCTION *problem on an n-vertex graph G can be solved in time $\mathcal{O}(3.83^k n^2)$.*

Proof. The proof goes similar to that for Lemma 2 in [3]. Let G be a graph and let F_{k+1} be an FVS of size $k+1$ in G. For each j, $0 \leq j \leq k$, we enumerate each subset F_{k-j} of $k-j$ vertices in F_{k+1}, and assume that F_{k-j} is the intersection of F_{k+1} and the objective FVS F_k. Therefore, constructing the FVS F_k of size k in the graph G is equivalent to constructing the FVS $F_k - F_{k-j}$ of size j in the graph $G - F_{k-j}$, which, by Theorem 2 (note that $l \leq j+1$), can be constructed in time $\mathcal{O}(2^{j+(j+1)/2} n^2) = \mathcal{O}(2.83^j n^2)$. Applying this procedure for every integer j $(0 \leq j \leq k)$ and all subsets of size $k-j$ in F_{k+1} will successfully find an FVS of size k in the graph G, if such an FVS exists. This algorithm solves FVS REDUCTION in time $\sum_{j=0}^{k} \binom{k+1}{k-j} \cdot \mathcal{O}(2.83^j n^2) = \mathcal{O}(3.83^k n^2)$. □

Finally, by combining Lemma 6 with iterative compression [5], we obtain the main result of this paper.

Theorem 3. *The* FVS *problem on an undirected graph of n vertices is solvable in time $\mathcal{O}(3.83^k k n^2)$.*

The proof of Theorem 3 is exactly similar to that of Theorem 3 in [5], with the complexity $\mathcal{O}(5^k n^2)$ for solving the FVS REDUCTION problem being replaced by $\mathcal{O}(3.83^k n^2)$, as given in Lemma 6.

References

1. Becker, A., Bar-Yehuda, R., Geiger, D.: Randomized algorithms for the loop cutset problem. J. Artif. Intell. Res. 12, 219–234 (2000)
2. Bodlaender, H.: On disjoint cycles. Int. J. Found. Comput. Sci. 5(1), 59–68 (1994)
3. Cao, Y., Chen, J., Liu, Y.: On Feedback Vertex Set New Measure and New Structures (manuscript, 2010)
4. Chen, J.: Minimum and maximum imbeddings. In: Gross, J., Yellen, J. (eds.) The Handbook of Graph Theory, pp. 625–641. CRC Press, Boca Raton (2003)
5. Chen, J., Fomin, F.V., Liu, Y., Lu, S., Villanger, Y.: Improved algorithms for the feedback vertex set problems. Journal of Computer and System Sciences 74, 1188–1198 (2008)
6. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, 2nd edn. The MIT Press and McGraw-Hill Book Company (2001)

7. Dehne, F., Fellows, M., Langston, M., Rosamond, F., Stevens, K.: An $O(2^{O(k)}n^3)$ fpt algorithm for the undirected feedback vertex set problem. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 859–869. Springer, Heidelberg (2005)
8. Downey, R., Fellows, M.: Fixed parameter tractability and completeness. In: Complexity Theory: Current Research, pp. 191–225. Cambridge University Press, Cambridge (1992)
9. Downey, R., Fellows, M.: Parameterized Complexity. Springer, New York (1999)
10. Festa, P., Pardalos, P., Resende, M.: Feedback set problems. In: Handbook of Combinatorial Optimization, vol. A(suppl.), pp. 209–258. Kluwer Acad. Publ., Dordrecht (1999)
11. Fomin, F., Gaspers, S., Pyatkin, A.: Finding a minimum feedback vertex set in time $O(1.7548^n)$. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 184–191. Springer, Heidelberg (2006)
12. Furst, M., Gross, J., McGeoch, L.: Finding a maximum-genus graph imbedding. Journal of the ACM 35(3), 523–534 (1988)
13. Gabow, H., Stallmann, M.: Efficient algorithms for graphic matroid intersection and parity. In: Brauer, W. (ed.) ICALP 1985. LNCS, vol. 194, pp. 210–220. Springer, Heidelberg (1985)
14. Guo, J., Gramm, J., Hüffner, F., Niedermeier, R., Wernicke, S.: Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. J. Comput. Syst. Sci. 72(8), 1386–1396 (2006)
15. Kanj, I., Pelsmajer, M., Schaefer, M.: Parameterized algorithms for feedback vertex set. In: Downey, R.G., Fellows, M.R., Dehne, F. (eds.) IWPEC 2004. LNCS, vol. 3162, pp. 235–247. Springer, Heidelberg (2004)
16. Karp, R.: Reducibility among combinatorial problems. In: Complexity of Computer Computations, pp. 85–103. Plenum Press, New York (1972)
17. Li, D., Liu, Y.: A polynomial algorithm for finding the minimul feedback vertex set of a 3-regular simple graph. Acta Mathematica Scientia 19(4), 375–381 (1999)
18. Lovász, L.: The matroid matching problem. In: Algebraic Methods in Graph Theory, Colloquia Mathematica Societatis János Bolyai, Szeged, Hungary (1978)
19. Raman, V., Saurabh, S., Subramanian, C.: Faster fixed parameter tractable algorithms for finding feedback vertex sets. ACM Trans. Algorithms 2(3), 403–415 (2006)
20. Raman, V., Saurabh, S., Subramanian, C.: Faster fixed parameter tractable algorithms for undirected feedback vertex set. In: Bose, P., Morin, P. (eds.) ISAAC 2002. LNCS, vol. 2518, pp. 241–248. Springer, Heidelberg (2002)
21. Razgon, I.: Exact computation of maximum induced forest. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 160–171. Springer, Heidelberg (2006)
22. Reed, B., Smith, K., Vetta, A.: Finding odd cycle transversals. Oper. Res. Lett. 32(4), 299–301 (2004)
23. Silberschatz, A., Galvin, P.: Operating System Concepts, 4th edn. Addison-Wesley, Reading (1994)