On Feedback Vertex Set:  
New Measure and New Structures  

∗Yixin Cao Jianer Chen  
Computer Science and Engineering  
Texas A&M University  
College Station, TX 77843-3112, USA  
\{yixin,chen\}@cse.tamu.edu  

Yang Liu  
Department of Computer Science  
University of Texas - Pan American  
Edinburg, TX 78539-2999, USA  
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 $O(3.83^k n^2)$ time parameterized algorithm for the general fvs problem, improving the previous best algorithm of time $O(5^k n^2)$ 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 [17]. 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 [11]). There are also exact algorithms finding a minimum FVS in a graph of $n$ vertices in time $O(1.9053^n)$ [22] and in time $O(1.7548^n)$ [12].

An important application of the FVS problem is deadlock recovery in operating systems [24], in which a deadlock is presented by a cycle in a system resource-allocation graph $G$. Therefore, in order to recover from deadlocks, we need to abort a set of 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 $f(k)n^{O(1)}$ 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 [3] and by Downey and Fellows [9]. Since then a chain of dramatic improvements was obtained by different researchers (see Figure 1).

∗Supported in part by the US National Science Foundation under the Grants CCF-0830455 and CCF-0917288.
| Authors                        | Complexity                                      | Year  |
|-------------------------------|------------------------------------------------|-------|
| Bodlaender                    | $O(17(k^4)\ln\Theta(k))$                       | 1994  |
| Downey and Fellows            | $O((2k + 1)k^2n^2)$                            | 1999  |
| Raman et al.                  | $O((4\log k)^k n^{2.376})$                     | 2002  |
| Raman et al.                  | $O((12\log k/\log \log k + 6)k^2n^{2.376})$    | 2004  |
| Guo et al.                    | $O((37.7^k n^2)$                               | 2006  |
| Dehne et al.                  | $O((10.6^k n^2)$                               | 2005  |
| Chen et al.                   | $O(5^k n^2)$                                   | 2008  |
| This paper                    | $O(3.83^k k^2 n^2)$                            | 2010  |

Figure 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. [2], which runs in time $O(4^k k^2 n^2)$.

The main result of the current paper is an algorithm that solves the FVS problem. The running time of our algorithm is $O(3.83^k k^2 n^2)$. 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 [2].

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 [8]. 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.

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 two simple rules:
Proof. If none of Rule 1 and Rule 2 is applicable on an instance \( u \), then include \( u \) into the objective \( V_1 \)-FVS, \( G = G \setminus u \), and \( k = k - 1 \); otherwise, move \( u \) from \( V_1 \) to \( V_2 \): \( V_1 = V_1 \setminus \{ u \}, V_2 = V_2 \cup \{ u \} \).

Note that the second case in Rule 2 includes the case where one or both neighbors of \( v \) are not in \( V_2 \). The correctness of Rule 1 is trivial: no degree-0 or degree-1 vertices can be contained in any cycle. On the other hand, although Rule 2 is also easy to verify for the general FVS problem [5] (because any cycle containing a degree-2 vertex \( v \) must also contain the two neighbors of \( v \)), it is much less obvious for the DISJOINT-FVS problem – the two neighbors of a degree-2 vertex \( v \) in \( V_1 \) may not be in \( V_1 \) and cannot be included in the objective \( V_1 \)-FVS. For this, we have the following lemma.

Lemma 2.1 Rule 2 is safe.

Proof. If the two neighbors of the degree-2 vertex \( v \) in \( V_1 \) are contained in the same connected component in \( G[V_2] \), then \( v \) and some vertices in \( V_2 \) form a cycle. Therefore, in order to break this cycle, the vertex \( v \) must be contained in the objective \( V_1 \)-FVS. This justifies the first case for Rule 2.

Now consider the second case for Rule 2. We only need to show that if the graph \( G \) has a \( V_1 \)-FVS of size \( k \), then \( G \) has a \( V_1 \)-FVS of size at most \( k \) that does not contain the degree-2 vertex \( v \). Let \( F \) be a \( V_1 \)-FVS of size \( k \) that contains \( v \). If one \( u_1 \) of the neighbors of \( v \) is in \( V_1 \), then the set \( (F \setminus v) \cup \{ u_1 \} \) will be a \( V_1 \)-FVS of size bounded by \( k \) that does not contain the vertex \( v \). Thus, we can assume that the two neighbors \( u_1 \) and \( u_2 \) are in two different connected components in \( G[V_2] \). Since \( G \setminus F \) is acyclic, there is either no path or a unique path in \( G \setminus F \) between \( u_1 \) and \( u_2 \). If there is no path between \( u_1 \) and \( u_2 \) in \( G \setminus F \), then adding \( v \) to \( G \setminus F \) does not create any cycle. Therefore, in this case, the set \( F' = F \setminus v \) is a \( V_1 \)-FVS of size \( k - 1 \) that does not contain \( v \). If there is a unique path \( P \) between \( u_1 \) and \( u_2 \) in \( G \setminus F \), then the path \( P \) must contain at least one vertex \( w \) in \( V_1 \) (since \( u_1 \) and \( u_2 \) are in different connected components in \( G[V_2] \)). Every cycle \( C \) in \( (G \setminus F) \cup \{ v \} \) must contain \( v \), thus, also contain \( u_1 \) and \( u_2 \). Therefore, the partial path \( C \setminus v \) from \( u_1 \) to \( u_2 \) in \( C \) must be the unique path \( P \) between \( u_1 \) and \( u_2 \) in \( G \setminus F \), which contains the vertex \( w \). This shows that \( w \) must be contained in all cycles in \( (G \setminus F) \cup \{ v \} \). In consequence, the set \( F' = (F \setminus v) \cup \{ w \} \) is a \( V_1 \)-FVS of size bounded by \( k \) that does not contain \( v \).

Note that the second case of Rule 2 cannot be applied simultaneously on more than one vertex in \( V_1 \). For example, let \( v_1 \) and \( v_2 \) be two degree-2 vertices in \( V_1 \) that are both adjacent to two vertices \( u_1 \) and \( u_2 \) in \( V_2 \). Then it is obvious that we cannot move both \( v_1 \) and \( v_2 \) to \( V_2 \). In fact, if we first apply the second case of Rule 2 on \( v_1 \), then the first case of Rule 2 will become applicable on the vertex \( v_2 \).

We show that the simple Rules 1-2 reduce an instance \( (G; V_1, V_2; k) \) of DISJOINT-FVS to a small kernel. 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.

We remark that in case 1 in the algorithm FindingFVS, the leaf \( w \) in \( G[V_1] \) can have at most one neighbor \( u \) in \( G[V_1] \), which by our assumption has degree at least 3. Therefore, after removing \( w \), the degree of \( u \) is at least 2.

Lemma 2.2 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 \( \tau \) is the number of connected components in \( G[V_1] \).

Proof. There are only two cases in Algorithm FindingFVS, namely:
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.

1. F = ∅;
2. while |V1| > 0 do
3.   pick a leaf w in G[V1];
4.   case 1: \( w \) is in the objective V1-FVS F;
5.      add \( w \) to F and remove \( w \) from V1; \( k = k - 1 \);
6.      if the neighbor u of \( w \) in G[V1] becomes degree-2 then apply Rule 2 on \( u \);
7.   case 2: \( w \) is not in the objective V1-FVS F.
8.      move \( w \) from V1 to V2.

Figure 2: The computational path \( \mathcal{P} \) that finds the V1-FVS F of size bounded by k in G

**case 1.** \( w \in F \): \( w \) is removed. If the degree of the neighbor \( u \) is larger than 3 before the removal of \( w \), \( w \) will be the only vertex deleted at this step. If the degree of \( u \) is equal to 3, then after removing \( w \), \( u \) becomes of degree 2, and is moved from V1 to V2 by Rule 2. The degree of any other vertex in V1 keeps unchanged. Here we consider four subcases based on the other two neighbors \( v_1 \) and \( v_2 \) of \( u \) other than \( w \):

1.1 \( v_1 \) and \( v_2 \) are in the same tree in \( G[V_2] \): \( u \) is added to \( F \), and \( k \) decreases by 1.
1.2 \( v_1 \) and \( v_2 \) are in different trees in \( G[V_2] \): \( u \) is moved to \( V_2 \), and \( l \) decreases by 1.
1.3 \( v_1 \) and \( v_2 \) are in \( V_1 \): \( u \) is moved to \( V_2 \) and becomes a single-vertex tree in \( G[V_2] \), which increases \( l \) by 1. Moreover, moving \( u \) from \( V_1 \) to \( V_2 \) splits \( v_1 \) and \( v_2 \) into different trees in \( G[V_1] \), and therefore increases \( \tau \) by 1.
1.4 \( v_1 \) is in \( V_1 \), and \( v_2 \) is in \( V_2 \): \( u \) is moved to \( V_2 \), with no any other influence.

**case 2.** \( w \notin F \): \( w \) is moved to \( V_2 \), which decreases \( l \) by at least 1. No other vertex in \( V_1 \) is impacted by this.

For case 1, one or two vertices are removed from \( V_1 \), with \( k \) decreased by 1 or 2. For case 2, exactly one vertex is removed with \( l \) decreased by 1. Case 1 can only occur at most \( k \) times, since the output V1-FVS F has its size bounded by \( k \), and each step of case 1 can remove at most 2 vertices from \( V_1 \). The second case removes exactly 1 vertex from \( V_1 \). However, the number of times case 2 is executed is more complicated because the subcase 1.3 might increase \( l \), so case 2 might happen more than \( l \) times. If we assume subcase 1.3 happens \( x \) times, then we can bound case 2 to no more than \( l + x \) times. The total number of vertices removed from this process is at most \( 2 \times k + 1 \times (l + x) = 2k + l + x \).

Note that subcase 1.3 also increases \( \tau \), which will counteract the influence by increasing of \( l \). We observe that in each step at most two vertices are removed from \( G_1 \), and then to remove a whole tree from \( G_1 \), the last step removes 1 or 2 vertices:

- A trivial tree in \( G_1 \), i.e. a single vertex \( u \) with degree at least 3. If \( u \) is in \( F \), i.e. case 1, it removes only one vertex from \( V_1 \) with \( k \) decreased by 1. Otherwise, it is in case 2, which moves \( u \) from \( V_1 \) to \( V_2 \) and removes one vertex from \( V_1 \) but decreases the number \( l \) of trees in \( G[V_2] \) by at least 2. In both cases, the total number loses at least 1.

- Both vertices, say \( u_1 \) and \( u_2 \), have to be leaves, so we can pick any of them, w.l.o.g., \( u_1 \). Here we ignore the case \( u_1 \notin F \), namely, because it is not the final step. So \( u_1 \in F \), and the disposal of \( u_2 \) becomes very subtle, namely, they can only be subcases 1.1 or 1.2,

1.1 two vertices are removed from \( V_1 \) with \( k \) decreased by 2: the total number loses 2;
1.2 one vertex is removed from \( V_1 \) with \( l \) decreased by at least 2: the total number loses at least 1.
From the above analysis, whatever the situation is, at the final step of each tree, the total number loses at least 1. We know there are $\tau + x$ of such steps, where $\tau$ is the number of trees in the original induced subgraph $G[V_1]$, and $x$ is the number of trees created by subcase 1.3. So the bound becomes $2k + l + x - (\tau + x) = 2k + l - \tau$.

If $|G_1| > 2k + l - \tau$, after $2k + l - \tau$ vertices have been disposed of, let $v$ be a leaf still left in $G[V_1]$. Then $G[V_2]$ has already become a single connected component, and adding any new vertex with more than two neighbors in $V_2$ will incur cycles, so $v$ cannot be put into $G_2$. But the $V_1$-FVS $F$ already has $k$ vertices, and $v$ cannot be added to $F$. So it must be a “no” instance. \hfill\qed

Note that for those DISJOINT-FVS instances we will meet in Section [1] we always have $|V_2| = k + 1$, this 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, 13]. 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 3.1** Let $G$ be a 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 [7].

**Proof.** The lemma can be proved based on a process that is similar to the well-known Kruskal’s algorithm for constructing a minimum spanning tree for a given graph $G$ [7], which runs in time $O(m\alpha(n))$ if we do not have to sort the edges. Starting from a structure $G_0$ that initially consists of the forest $G[S]$ and all vertices in $G - S$, we repeatedly insert each of the remaining edges (in an arbitrary order) into the structure $G_0$ as long as the edge does not create a cycle. The resulting structure of this process must be a spanning tree that contains the forest $G[S]$. \hfill\qed

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 3.1, 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** 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 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 $G$) [13]. The following lemma is a nontrivial generalization of a result in [18] (the result in [18] is a special case for Lemma 3.2 in which all vertices in the set $V_2$ have degree 2).

**Lemma 3.2** For any 3-regular 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 $\mu(G)$.

**Proof.** Let $T$ be a $T_{G|V_2}$-tree such that there is a $V_1$-adjacency matching $M$ in $G - T$ that contains $\mu(G)$ 2-groups. Let $U$ be the set of edges that are in the 1-groups in $M$. We construct a $V_1$-FVS $F$ as follows: (1) for each edge $e$ in $U$, arbitrarily pick an end of $e$ that is in $V_1$ and include it in $F$; and (2) for each 2-group of two $V_1$-adjacent edges $e_1$ and $e_2$ in $M$, pick the vertex in $V_1$ that is a common end of $e_1$ and $e_2$ and include it in $F$. Note that every cycle in the graph $G$ must contain at least one edge in $G - T$, while every edge in $G - T$ has at least one end in $F$. Therefore, $F$ is an FVS. By the above construction, $F$ is a $V_1$-FVS. The number of vertices in $F$ is equal to $|U| + \mu(G)$. Since $|U| = |G - T| - 2\mu(G) = \beta(G) - 2\mu(G)$, we have $|F| = \beta(G) - \mu(G)$. This results in

\[ f_{V_1}(G) \leq \beta(G) - \mu(G). \]  

(1)

Now consider the other direction. Let $F$ be a $V_1$-FVS such that $|F| = f_{V_1}(G)$. Since $G - F$ is a forest, by Lemma 3.1, there is a spanning tree $T$ in $G$ that contains the entire induced subgraph $G - F$. We construct a $V_1$-adjacency matching in $G - T$ and show that it contains at least $\beta(G) - |F|$ 2-groups. Since $T$ contains $G - F$, each of the edges in $G - T$ has at least one end in $F$. Let $E_2$ be the set of edges in $G - T$ that have their both ends in $F$, and let $E_1$ be the set of edges in $G - T$ that have exactly one end in $F$.

**Claim.** Each end of an edge in $E_2$ is shared by exactly one edge in $E_1$. In particular, no two edges in $E_2$ share a common end.

To see this, let $u$ be an end of an edge $[u, v]$ in $E_2$, where both $u$ and $v$ are in $F$. Let $e_1$ and $e_2$ be the other two edges incident to $u$ (note that $u$ has degree 3). If $u$ is not shared by an edge in $E_1$, then either both $e_1$ and $e_2$ are in $T$ or one of $e_1$ and $e_2$ is in $E_2$. If both $e_1$ and $e_2$ are in $T$, then, since every edge in $G - T$ (including $[u, v]$) has at least one end in $F$, the set $F \setminus \{u\}$ would make a smaller $V_1$-FVS. Similarly, if the edge $e_1 = [u, w]$ is in $E_2$, where $w$ is also in $F$, then again $F \setminus \{u\}$ would make a smaller $V_1$-FVS (note that $u$ has degree 3 and that $[u, v]$ and $[u, w]$ are the only edges in $G - T$ that are incident to $u$). Therefore, both cases would contradict the assumption that $F$ is a minimum $V_1$-FVS. This proves the claim.

Suppose that there are $m_2$ vertices in $F$ that are incident to two edges in $G - T$. Thus, each of the rest $|F| - m_2$ vertices in $F$ is incident to at most one edge in $G - T$. By counting the total number of incidences between the vertices in $F$ and edges in $G - T$, we get

\[ 2|E_2| + (\beta(G) - |E_2|) \leq 2m_2 + (|F| - m_2), \]

or equivalently,

\[ m_2 - |E_2| \geq \beta(G) - |F|. \]  

(2)

Now we construct a $V_1$-adjacency matching in $G - T$, as follows. For each edge $e$ in $E_2$, by the above claim, we can make a 2-group that consists of $e$ and an edge in $E_1$ that shares an end with $e$. Besides the ends of the edges in $E_2$, there are $m_2 - 2|E_2|$ vertices in $F$ that are incident to two edges in $E_1$. For each $v$ of these vertices, we make a 2-group that consists of the two edges in $E_1$ that are incident to $v$. Note that this construction of 2-groups never re-uses any edges in $G - T$ more than once. Therefore, the construction gives $|E_2| + (m_2 - 2|E_2|) = m_2 - |E_2|$ disjoint 2-groups. We then make each of the rest edges in $G - T$ a 1-group. This gives a $V_1$-adjacency matching in $G - T$ that has $m_2 - |E_2|$ 2-groups. By Inequality (2) and by definition, we have

\[ \mu(G) \geq \mu(G, T) \geq m_2 - |E_2| \geq \beta(G) - |F| = \beta(G) - f_{V_1}(G). \]  

(3)

6
Combining (11) and (8), we conclude with \( f_{V_1}(G) = \beta(G) - \mu(G) \). The first paragraph also illustrates how to construct a minimum \( V_1 \)-FVS from a \( T_{G[V_2]} \)-tree whose \( V_1 \)-adjacency matching number is \( \mu(G) \).

By Lemma 3.2 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 [13]. 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. More detailed discussion on matroid theory can be found in [19].

A matroid is a pair \((E, \mathcal{I})\), where \( E \) is a finite set and \( \mathcal{I} \) is a collection of subsets of \( E \) that satisfies the following properties:

1. If \( A \in \mathcal{I} \) and \( B \subseteq A \), then \( B \in \mathcal{I} \);
2. If \( A, B \in \mathcal{I} \) and \( |A| > |B| \), then there is an element \( a \in A - B \) such that \( B \cup \{a\} \in \mathcal{I} \).

The matroid parity problem is stated as follows: given a matroid \((E, \mathcal{I})\) 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 \( P \in \mathcal{I} \) such that for all \( i, 1 \leq i \leq n \), 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 \((E_G, \mathcal{I}_G)\), where \( E_G \) is the edge set of \( G \), and an edge set \( S \) is in \( \mathcal{I}_G \) 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 [19]. The fastest known algorithm for cographic matroid parity problem is by Gabow and Stallmann [14], which runs in time \( O(mn \log^6 n) \).

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 \( (e_i_0) \) for all segment edges (note that \( (e_0e_i) \) is a segment edge from the edge \( e_0 \) in \( G_1 \) and that \( (e_i_0) \) is a segment edge from the edge \( e_i \) in \( G_1 \)). By the above remarks, this is a perfect pairing \( P \) of the edges in \( G_2 \). Now with this edge pairing \( P \) in \( G_2 \), and with the cographic matroid \((E_{G_2}, \mathcal{I}_{G_2})\) for the graph \( G_2 \), we call Gabow and Stallmann’s algorithm [14] for the cographic matroid parity problem. The algorithm produces a maximum edge subset \( P \) in \( \mathcal{I}_{G_2} \) that, for each segment edge \( (e_0e_i) \) in \( G_2 \), either contains both \( (e_0e_i) \) and \( (e_i_0) \), or contains neither of \( (e_0e_i) \) and \( (e_i_0) \).
Lemma 3.3 From the edge subset \( P \) in \( \mathcal{G}_2 \) 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 \( m \) and \( n \) are the number of vertices and the number of edges, respectively, of the graph \( G \).

PROOF. Let the edge subset \( P \) consist of the edge pairs \( \{(e_1,e'_1),(e'_1,e_1),\ldots,(e_k,e'_k),(e'_k,e_k)\} \). Since \( P \in \mathcal{G}_2 \), \( G_2 - P \) is connected. Thus, for each edge \( e_i \) in \( G_1 \), there is at most one segment edge in \( P \) that is from \( e_i \). Therefore, the edge subset \( P \) corresponds to a edge subset \( P' \) of exactly \( 2h \) edges in \( G_1 \) (thus exactly \( 2h \) edges in \( G \)): \( P' = \{e_1,e'_1;\ldots,e_h,e'_h\} \), where for \( 1 \leq i \leq h \), the edges \( e_i \) and \( e'_i \) are \( V_1 \)-adjacent. Since \( G_2 - P \) is connected, it is easy to verify that the graph \( G_1 - P' \) (thus the graph \( G - P' \)) is also connected. Also note that the graph \( G - P' \) contains the induced subgraph \( G[V_2] \). Therefore, by Lemma 3.1 we can construct, in time \( O(m\alpha(n)) \), a \( T_{G[V_2]} \)-tree \( T_1 \) for the graph \( G \) from \( G - P' \). Now if we make each pair \([e_i,e'_i]\) a 2-group for \( 1 \leq i \leq h \), and make each of the rest edges in \( G - T_1 \) a 1-group, we get a \( V_1 \)-adjacency matching with \( h \) 2-groups in \( G - T_1 \).

To complete the proof of the lemma, we only need to show that \( h = \mu(G) \). For this, it suffices to show that no \( T_{G[V_2]} \)-tree can have a \( V_1 \)-adjacency matching with more than \( h \) 2-groups. Let \( T_2 \) be a \( T_{G[V_2]} \)-tree with \( q \) 2-groups \([e_1,e'_1],\ldots,[e_q,e'_q]\) 2-groups in \( G - T_2 \). Since \( T_2 \) is entirely contained in \( G - \bigcup_{i=1}^{h}\{e_i,e'_i\} \), \( G - \bigcup_{i=1}^{h}\{e_i,e'_i\} \) is also connected. From this, it is easy to verify that the graph \( G_2 - \bigcup_{i=1}^{h}\{e_i,e'_i\} \) is connected. Therefore, the edge subset \([e_1,e'_1),(e'_1,e_1);\ldots,(e_q,e'_q),(e'_q,e_q)\] is in \( \mathcal{G}_2 \). Now since \( P \) is the solution of the matroid parity problem for the cographic matroid \( (E_{G_2},\mathcal{G}_2) \) and since \( P \) consists of \( h \) edge pairs, we must have \( h \geq g \). This completes the proof of the lemma. \( \square \)

Now we are ready to present our main result in this section.

Theorem 3.4 There is an \( O(n^2\log^6 n) \) 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.

PROOF. For the 3-regular \( V_1 \) instance \((G;V_1,V_2;k)\) of DISJOINT-FVS, we first construct the graph \( G_1 \) in linear time by shrinking each connected component of \( G[V_2] \) into a single vertex. Note that since each vertex in \( V_1 \) has degree 3, the total number of edges in \( G_1 \) is bounded by \( 3|V_1| \). From the graph \( G_1 \), we construct the labeled subdivision graph \( G_2 \). Again since each vertex in \( V_1 \) has degree 3, each edge in \( G_1 \) is subdivided into at most 4 segment edges in \( G_2 \). Therefore, the number \( n_2 \) of vertices and the number \( m_2 \) of edges in \( G_2 \) are both bounded by \( O(|V_1|) = O(n) \). From the graph \( G_2 \), we apply Gabow and Stallmann's algorithm [14] on the cographic matroid \( (E_{G_2},\mathcal{G}_2) \) that produces the edge subset \( P \) in \( \mathcal{G}_2 \) in time \( O(m_2n_2\log^6 n_2) = O(n^2\log^6 n) \). By Lemma 3.3 from the edge subset \( P \), we can construct in time \( O(m\alpha(n)) \) a \( T_{G[V_2]} \)-tree \( T \) for the graph \( G \) whose \( V_1 \)-adjacency matching number is \( \mu(G) \). Finally, by Lemma 3.2 from the \( T_{G[V_2]} \)-tree \( T \), we can construct a minimum \( V_1 \)-FVS \( F \) in linear time. Now the solution to the 3-regular \( V_1 \) instance \((G;V_1,V_2;k)\) of DISJOINT-FVS can be trivially derived by comparing the size of \( F \) and the parameter \( k \). Summarizing all these steps gives the proof of the theorem. \( \square \)

Combining Theorem 3.4 and Lemma 3.1 we have

Corollary 3.5 There is an \( O(n^2\log^6 n) \) 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.

We remark this special structure is the best we can expect, in the sense that the DISJOINT-FVS problem becomes NP-hard when maximum degree in \( V_1 \) is no less than 4.

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
Therefore, step 8 handles this case correctly, and after step 8, the following conditions hold:

- the subgraph $G[V_1]$ 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 $m = k + \frac{1}{2} - p$ will be used in the analysis of our algorithm.

**Lemma 4.1** 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 $m = k + \frac{1}{2} - p \leq 0$, 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 $p - k' \geq p - k \geq \frac{1}{2}$. 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 3.4 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 4.1 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 Lemma 2.1 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 Lemma 2.1 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, and after step 8, the following conditions hold:
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, or ‘No’ otherwise.

1. If ($k < 0$) or ($k = 0$ and $G$ is not a forest) then return ‘No’;
2. If $k ≥ 0$ and $G$ is a forest then return $\emptyset$;
3. Else let $l$ be the number of connected components in $G[V_2]$,
   and let $p$ be the number of nice $V_1$-vertices;
4. If $p + k + \frac{l}{4} < 0$ then return ‘No’;
5. Else solve the problem in polynomial time;
6. If a vertex $w ∈ V_1$ has degree not larger than 1 then
   return Feedback($G - w, V_1 \setminus \{w\}, V_2, k$);
7. Else if a vertex $w ∈ V_1$ has degree 2 then
   return Feedback($G, V_1 \setminus \{w\}, V_2 \cup \{w\}, k$);
8. Else if a vertex $w ∈ V_1$ has degree 3 then
   return Feedback($G, V_1 \setminus \{w\}, V_2 \cup \{w\} \cup \{y\}, k$);
9. Else return Feedback($G, V_1 \setminus \{w\}, V_2 \cup \{w\}, k$).

Figure 3: Algorithm for DISJOINT-FVS

1. $k > 0$ and $G$ is not a forest (by steps 1 and 2);
2. $p ≤ k + \frac{l}{4}$ and not all vertices of $V_2$ are nice vertices (by step 3);
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 $y ∈ V_1$ 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 $F_1 ⊆ V_1 \setminus \{w\}$ and $|F_1| ≤ k - 1$. Note that after removing $w$, the leaf $w_1$ becomes degree-2, and by Lemma 2.1 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.

Now we are ready to present the following theorem.

**Theorem 4.2** The algorithm Feedback($G, V_1, V_2, k$) correctly solves the DISJOINT-FVS problem. The running time of the algorithm is $O(2^{k+1/l}·n^2)$, 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 $T$. We first count the number of leaves in the search tree $T$. Note that only steps 7, 8 and 9 of the algorithm correspond to branches in the search tree $T$. Let $T(m)$ be the number of leaves in the
Proof. find an FVS of size $\sum_j$ disjoint fvs problem in time $O(\log k)$. The branch of step 7.1 has that $k' = k - 1$, $l' = l - 1$ and $p' \geq p$. Thus we have $m' = k' + l' - p' \leq k - 1 + l/2 - p = m - 1$. The branch of step 7.2 has that $k'' = k$, $l'' \leq l - 2$ and $p'' = p$. Thus we have $m'' = k'' + l'' - p'' \leq m - 1$. 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 $m' = k' + l' - p' \leq k - 1 + (l - 1)/2 - p = m - 1.5$. The branch of step 8.2 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 $m'' = k'' + l'' - p'' \leq k + (l + 1)/2 - (p + 2) = m - 1.5$. Thus, for step 8, the recurrence is $T(m) \leq 2T(m - 1.5)$.

The worst case happens at step 9. 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 lemma 2.1, 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 $T$ 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 $O(2^{k+l/2}n^2)$. This completes the proof of the lemma.

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. [23], 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 5.1 The FVS REDUCTION problem on an $n$-vertex graph $G$ can be solved in time $O(3.83^k n^2)$.

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 4.2 (note that $l \geq j + 1$), can be constructed in time $O(2^j(2j+1)/j^2) = O(2.83^j n^2)$. 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 $\sum_{j=0}^{k} \binom{k+1}{j} \cdot O(2.83^j n^2) = O(3.83^k n^2)$.

Finally, by combining Lemma 5.1 with iterative compression [3], we obtain the main result of this paper.

Theorem 5.2 The FVS problem on an $n$-vertex graph is solvable in time $O(3.83^k n^2)$.

The proof of Theorem 5.2 is exactly similar to that of Theorem 3 in [5], with the complexity $O(5^k n^2)$ for solving the FVS REDUCTION problem being replaced by $O(3.83^k n^2)$, as given in Lemma 5.1.
References

[1] Bafna, V., Berman, P., Fujito, T.: A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM J. Discrete Math.* 12(3), 289–297 (1999)

[2] Becker, A., Bar-Yehuda, R., Geiger, D.: Randomized algorithms for the loop cutset problem. *J. Artif. Intell. Res.* 12, 219–234 (2000)

[3] Bodlaender, H.: On disjoint cycles. *Int. J. Found. Comput. Sci.* 5(1), 59–68 (1994)

[4] Chen, J.: Minimum and maximum imbeddings. In: Gross, J., Yellen, J.(eds.) *The Handbook of Graph Theory*, pp.625–641, CRC Press (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] Chen, J., Liu, Y., Lu, S.: An improved parameterized algorithm for the minimum node multiway cut problem. *Algorithmica* 55, 1–13 (2009)

[7] Cormen, T., Leiserson, C., Rivest, R., Stein, C.: *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company (2001)

[8] 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: *COCOON*, LNCS, vol. 3595, pp. 859–869. Springer (2005)

[9] Downey, R., Fellows, M.: Fixed parameter tractability and completeness. In: *Complexity Theory: Current Research*, pp. 191–225. Cambridge University Press (1992)

[10] Downey, R., Fellows, M.: *Parameterized Complexity*. Springer-Verlag, New York (1999)

[11] Festa, P., Pardalos, P., Resende, M.: Feedback set problems. In: *Handbook of Combinatorial Optimization, Supplement Vol. A*, pp. 209–258. Kluwer Acad. Publ., Dordrecht (1999)

[12] Fomin, F., Gaspers, S., Pyatkin, A.: Finding a minimum feedback vertex set in time $O(1.7548^n)$. In: *IWPEC*, LNCS vol. 4169, pp. 184–191. Springer (2006)

[13] Furst, M., Gross, J., McGeoch, L.: Finding a maximum-genus graph imbedding. *Journal of the ACM.* 35(3): 523–534 (1988)

[14] Gabow, H., Stallmann, M.: Efficient algorithms for graphic matroid intersection and parity. In: *Automata, Languages and Programming: 12th Colloquium*, LNCS vol. 194, pp. 210–220, Springer-Verlag (1985)

[15] 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)

[16] Kanj, I., Pelsmajer, M., Schaefer, M.: Parameterized algorithms for feedback vertex set. In: *IWPEC*, LNCS vol. 3162, pp. 235–247. Springer (2004)

[17] Karp, R.: Reducibility among combinatorial problems. In: *Complexity of Computer Computations*, pp. 85–103. Plenum Press, New York (1972)

[18] Li, D., Liu, Y.: A polynomial algorithm for finding the minimal feedback vertex set of a 3-regular simple graph. *Acta Mathematica Scientia* 19(4), 375-381 (1999)

[19] Lovász, L.: The matroid matching problem. In: *Algebraic Methods in Graph Theory*, Colloquia Mathematica Societatis János Bolyai, Szeged (Hungary) (1978)
[20] Raman, V., Saurabh, S., Subramanian, C.: Faster fixed parameter tractable algorithms for finding feedback vertex sets. *ACM Trans. Algorithms* 2(3), 403–415 (2006)

[21] Raman, V., Saurabh, S., Subramanian, C.: Faster fixed parameter tractable algorithms for undirected feedback vertex set. In: *ISAAC 2002*, LNCS vol. 2518, pp. 241–248. Springer (2002)

[22] Razgon, I.: Exact computation of maximum induced forest. In: *SWAT*, LNCS, vol. 4059, pp. 160–171. Springer (2006)

[23] Reed, B., Smith, K., Vetta, A.: Finding odd cycle transversals. *Oper. Res. Lett.* 32(4), 299–301 (2004)

[24] Silberschatz, A., Galvin, P.: *Operating System Concepts, 4th edition*. Addison-Wesley (1994)