REPORTS
IN
INFORMATICS

ISSN 0333-3590

Improved Algorithms for the Feedback Vertex Set Problems

Jianer Chen, Fedor V. Fomin, Yang Liu,
Songjian Lu, and Yngve Villanger

REPORT NO 348 February 2007

Improved Algorithms for the Feedback Vertex Set Problems [*†]

Jianer Chen [‡] Fedor V. Fomin[§] Yang Liu[‡] Songjian Lu[‡]
Yngve Villanger[§]

Abstract

We present improved parameterized algorithms for the Feedback Vertex Set problem on both unweighted and weighted graphs. Both algorithms run in time $O(5^k kn^2)$. For unweighted graphs, our algorithm either constructs a feedback vertex set of size bounded by k in a given graph G, or reports that no such a feedback vertex set exists in G. For weighted graphs, our algorithm either constructs a minimum-weight feedback vertex set of size bounded by k in a given graph G, or reports that no feedback vertex set of size bounded by k exists in G.

1 Introduction

Let G be a graph. A *feedback vertex set* (FVS) F in G is a set of vertices in G whose removal results in an acyclic graph (or equivalently, every cycle in G contains at least one vertex in F). The problem of finding a minimum feedback vertex set in a graph is one of the classical NP-complete problems [12] and has many applications. The history of the problem can be traced back to the 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 of Festa et al. [8]). There are also exact algorithms finding a minimum FVS in a graph on n vertices in time $O(1.9053^n)$ [15] and in time $O(1.7548^n)$ [9].

An important application of the FVS problem is *deadlock recovery* in operation systems [17], 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. The problem also has a version on weighted graphs, where the weight of a vertex can be interpreted as the cost of aborting the corresponding process. In this case, we are looking for an FVS in G whose weight is minimized.

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 *parameterized algorithms* for the FVS problem that find an FVS of k vertices

[*]This work was supported in part by the National Science Foundation of USA under the Grant CCF-0430683.
[†]This work is supported by the Research Council of Norway
[‡]Department of Computer Science, Texas A&M University, College Station, Texas 77843-3112, USA {chen,yangliu,sjlu}@cs.tamu.edu
[§]Department of Informatics, University of Bergen, N-5020 Bergen, Norway. {fomin,yngvev}@ii.uib.no

in a graph of n vertices (where the weight of the FVS is minimized, in the case of weighted graphs), and run in time $f(k)n^{O(1)}$ for a fixed function f (thus, the algorithms become practically efficient when the value k is small).

This line of research has received considerable attention, most are on the FVS problem on unweighted graphs. The first group of parameterized algorithms of running time $f(k)n^{O(1)}$ for the FVS problem on unweighted graphs were given by Bodlaender [3] and by Downey and Fellows [6]. Since then a chain of dramatic improvements was obtained by different researchers (see Figure 1 for references.)

Bodlaender, Fellows [3, 6]	$O(17k^4!n^{O(1)})$
Downey and Fellows [7]	$O((2k+1)^k n^2)$
Raman et al.[13]	$O(\max\{12^k, (4\log k)^k\} n^{2.376})$
Kanj et al.[11]	$O((2\log k + 2\log\log k + 18)^k n^2)$
Raman et al.[14]	$O((12\log k/\log\log k + 6)^k n^{2.376})$
Guo et al.[10]	$O((37.7)^k n^2)$
Dehne et al.[5]	$O((10.6)^k n^3)$

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

Our results. In this paper we use the technique of iterative compression that was already applied for several similar problems including the FVS problem [5, 10, 16]. The novel part of our approach is the new recursive procedure and its analysis which allow us to reduce the running time of the algorithm significantly. We show that the problem of finding an FVS of size k of minimum weight in a weighted graph G of n vertices can be solved in time $O(5^k kn^2)$. This improves and generalizes a long chain of results in parameterized algorithms.

We remark that randomized parameterized algorithms have also been studied in the literature for the FVS problem, for both unweighted and weighted graphs. The best known randomized parameterized algorithms for the FVS problems are due to Becker et al. [2], who developed a randomized algorithm of running time $O(4^k kn^2)$ for the FVS problem on unweighted graphs, and a randomized algorithm of running time $O(6^k kn^2)$ for the FVS problem on weighted graphs. Compared to these results, the running time of our (deterministic) algorithm comes close to that of the best randomized algorithm for the FVS problem on unweighted graphs, while our (deterministic) algorithm has been better than the previous best randomized algorithm for the FVS problem on weighted graphs.

To simplify the explanation we describe first the algorithm for unweighted graphs, which is significantly easier. Then we explain how this algorithm can be adopted to the weighted case (with the same running time).

2 On feedback vertex sets in unweighted graphs

In this section, we consider the FVS problem on unweighted graphs.

We start with some terminologies. A *forest* is a graph that contains no cycles. A *tree* is a forest that is connected (therefore, a forest can be equivalently defined as a collection of disjoint trees). Let W be a subset of vertices in a graph G. We will denote by $G[W]$ the subgraph of G that is induced by the vertex set W. A pair (V_1, V_2) of vertex subsets in a graph $G = (V, E)$ is a *forest bipartition* of G if $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \emptyset$, and both induced subgraphs $G[V_1]$ and $G[V_2]$ are forests.

Let G be a graph and let F be a subset of vertices in G. The set F is a *feedback vertex set* (shortly, FVS) of G if $G - F$ is a forest (or equivalently, if every cycle in G contains at least one vertex in F). The *size* of an FVS F is the number of vertices in F.

Our main problem is formally defined as follows.

> FEEDBACK VERTEX SET: given a graph G and an integer k, either find an FVS of size bounded by k in G, or report that no such an FVS exists.

Before we present our algorithm for the FEEDBACK VERTEX SET problem, we first consider a special version of the problem, defined as follows:

> F-BIPARTITION FVS: given a graph G, a forest bipartition (V_1, V_2) of G, and an integer k, either find an FVS of size bounded by k for the graph G *in the subset* V_1, or report that no such an FVS exists.

Note that the main difference between the F-BIPARTITION FVS problem and the original FEEDBACK VERTEX SET problem is that we require that the FVS in the F-BIPARTITION FVS is contained in the given subset V_1.

A *bypass* operation will be used heavily in our process. Let w be a degree-2 vertex with two neighbors u and v in a graph G. We say that a graph G' is obtained from G by *bypassing* the degree-2 vertex w if G' is obtained from G by first removing the vertex w then adding a new edge between u and v.

```
Algorithm-1 Feedback($G, V_1, V_2, k$)
Input: $G = (V, E)$ is a graph with a forest bipartition $(V_1, V_2)$, $k$ is an integer
Output: An FVS $F$ of $G$ such that $|F| \leq k$ and $F \subseteq V_1$; or report "No" (i.e.,
            no such an FVS)

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 $\emptyset$;
3.     if a vertex $w$ in $V_1$ has at least two neighbors in $V_2$ then
3.1.       if two of the neighbors of $w$ in $V_2$ belong to the same tree in $G[V_2]$
            then $F' =$ Feedback($G - w, V_1 - w, V_2, k - 1$);
                 if $F' =$ "No" then return "No" else return $F' + w$;
3.2.       else $F_1 =$ Feedback($G - w, V_1 - w, V_2, k - 1$);
                $F_2 =$ Feedback($G, V_1 - w, V_2 + w, k$);
                if $F_1 \neq$ "No" then return $F_1 + w$
                else if $F_2 \neq$ "No" then return $F_2$
                else return "No";
4.     else pick any vertex $w$ that has degree $\leq 1$ in $G[V_1]$;
4.1.       if $w$ has degree $\leq 1$ in the original graph $G$
            then return Feedback($G - w, V_1 - w, V_2, k$)
4.2.       else let $G_w$ be the graph obtained from $G$ by bypassing $w$;
                return Feedback($G_w, V_1 - w, V_2, k$)
```

Figure 2: Algorithm for unweighted FVS problem

The algorithm, **Feedback**(G, V_1, V_2, k), for the F-BIPARTITION FVS problem is given in Figure 2. We first discuss the correctness of the algorithm. The correctness of step 1 and step 2 of the algorithm is obvious. Now consider step 3. Let w be a vertex in V_1 that has at least two neighbors in V_2.

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 restricted to find an FVS in the vertex subset V_1, the only way to break the cycles in $T + w$ is to include the vertex w in the objective FVS. Moreover, the objective FVS of size bounded by k exists in G if and only if the remaining graph $G - w$ has an FVS of size bounded by $k-1$ in the subset $V_1 - w$ (note that $(V_1 - w, V_2)$ is a valid forest bipartition of the graph $G - w$). Therefore, step 3.1 correctly handles this case.

If no two neighbors of the vertex w belong to the same tree in the induced subgraph $G[V_2]$, then the vertex w is either in the objective FVS or not in the objective FVS. If w is in the objective FVS, then we should be able to find an FVS F_1 in the graph $G - w$ such that $|F_1| \leq k-1$ and $F_1 \subseteq V_1 - w$ (again note that $(V_1 - w, V_2)$ is a valid forest bipartition of the graph $G - w$). On the other hand, if w is not in the objective FVS, then the objective FVS for G must be contained in the subset $V_1 - w$. Also note that in this case, the subgraph $G[V_2 + w]$ induced by the subset $V_2 + w$ is still a forest since no two neighbors of w in V_2 belong to the same tree in $G[V_2]$. In consequence, $(V_1 - w, V_2 + w)$ still makes a valid forest bipartition for the graph G. Therefore, step 3.2 handles this case correctly.

Now we consider step 4. At this point, every vertex in V_1 has at most one neighbor in V_2. Moreover, since the induced subgraph $G[V_1]$ is a forest, there must be a vertex w in V_1 that has degree bounded by 1 in $G[V_1]$ (note that V_1 cannot be empty at this point since otherwise the algorithm would have stopped at step 2). If the vertex w also has degree bounded by 1 in the original graph G, then removing w does not help breaking any cycles in G. Therefore, the vertex w can be discarded. This case is correctly handled by step 4.1. Otherwise, the vertex w has degree bounded by 1 in the induced subgraph $G[V_1]$ but has degree larger than 1 in the original graph G. Observing also the fact that w has at most one neighbor in V_2, we can easily derive in this case that the degree of w in the original graph G must be 2, and that w has two neighbors u_1 and u_2 such that u_1 is in V_1 and u_2 is in V_2. Therefore, if w is in the objective FVS F, then the set $F' = F - w + u_1$ will also make a valid solution to the given problem instance. Thus, by bypassing the degree-2 vertex w in G, we obtain a graph G_w, with the forest bipartition $(V_1 - w, V_2)$, such that G_w has an FVS of size bounded by k in $V_1 - w$ if and only if the original graph G has an FVS of size bounded by k in V_1. In conclusion, step 4.2 correctly handles this case.

Now we are ready to present the following lemma.

Lemma 2.1 *The algorithm* **Feedback**(G, V_1, V_2, k) *solves the* F-BIPARTITION FVS *problem correctly. The running time of the algorithm is bounded by* $O(2^{k+l}n^2)$*, where* n *is the number of vertices in the graph G, and* l *is the number of connected components (i.e. trees) 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 step 3.2 of the algorithm corresponds to branches in the search tree $\mathcal{T}$. Let $T(k, l)$ be the total number of leaves in the search tree $\mathcal{T}$ for the algorithm **Feedback**(G, V_1, V_2, k), where l is the number of trees in the forest $G[V_2]$. Inductively, the number of leaves in the search tree $\mathcal{T}_1$ corresponding to the recursive call **Feedback**$(G-w, V_1-w, V_2, k-1)$ is bounded by $T(k-1, l)$. Moreover, we assume at step 3.2 that w has at least two neighbors in V_2 and that no two neighbors of w in V_2 belong to the same tree in $G[V_2]$. Therefore, the vertex w "merges" at least two trees in $G[V_2]$ into a single tree in $G[V_2+w]$. Therefore, the number of trees in $G[V_2+w]$ is bounded by $l-1$. In consequence, the number of leaves in the search tree $\mathcal{T}_2$ corresponding to the recursive call **Feedback**(G, V_1-w, V_2+w, k) is bounded by $T(k, l-1)$. This gives the following recurrence relation:

$$T(k, l) \leq T(k-1, l) + T(k, l-1)$$

It is easy to derive from this relation that $T(k, l) = O(2^{k+l})$. Finally, observe that along each root-leaf path in the search tree $\mathcal{T}$, the total number of executions of steps 1, 2, 3.1, and

4 of the algorithm is bounded by $O(n)$ because each of these steps either stops immediately, or reduces the input graph size by at least 1. Moreover, it is also easy to verify that each of these steps takes time $O(n)$.

Therefore, the computation time along each root-leaf path in the search tree $\mathcal{T}$ is bounded by $O(n^2)$. In conclusion, the running time of the algorithm **Feedback**(G, V_1, V_2, k) is bounded by $O(2^{k+l}n^2)$. This completes the proof of the theorem. ▬

Following the idea of *iterative compression* proposed by Reed et al. [16], 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 bounded by k for G, or report that no such an FVS exists.

Lemma 2.2 *The* FVS REDUCTION *problem can be solved in time* $O(5^k n^2)$.

Proof. We use the algorithm **Feedback** to solve the FVS REDUCTION problem. Let F be the FVS of size $k+1$ in the graph $G = (V, E)$. Every FVS F' of size bounded by k for G is a union of a subset F_1 of at most $k - j$ vertices in $V - F$ and a subset F_2 of j vertices in F, for some integer j, $0 \le j \le k$. Note that since we assume that no vertex in $F - F_2$ is in the FVS F', the induced subgraph $G[F - F_2]$ must be a forest. Therefore, for each j, $0 \le j \le k$, we enumerate all subsets of j vertices in F. For each such a subset F_2 in F such that $G[F - F_2]$ is a forest, we seek a subset F_1 of at most $k - j$ vertices in $V - F$ such that $F_1 \cup F_2$ makes an FVS for the graph G.

Fix a subset F_2 in F, where $|F_2| = j$. Note that the graph G has an FVS $F_1 \cup F_2$ of size bounded by k, where $F_1 \subseteq V - F$, if and only if the subset F_1 of $V - F$ is an FVS for the graph $G - F_2$ and the size of F_1 is bounded by $k - j$. Therefore, to solve the original problem, we can instead consider how to construct an FVS F_1 for the graph $G - F_2$ such that $|F_1| \le k - j$ and $F_1 \subseteq V - F$.

Since F is an FVS for G, we have that the induced subgraph $G[V - F] = G - F$ is a forest. Moreover, by our assumption, the induced subgraph $G[F - F_2]$ is also a forest. Note that $(V - F) \cup (F - F_2) = V - F_2$, which is the vertex set for the graph $G' = G - F_2$. Therefore, $(V - F, F - F_2)$ is a forest bipartition of the graph G'. Thus, an FVS F_1 for the graph G' such that $|F_1| \le k - j$ and $F_1 \subseteq V - F$ can be constructed by the algorithm **Feedback**$(G', V - F, F - F_2, k - j)$.

Since $|F| = k + 1$ and $|F_2| = j$, we have that $|F - F_2| = k + 1 - j$. Therefore, the forest $G[F - F_2]$ contains at most $k+1-j$ trees. By Lemma 2.1, the running time of the algorithm **Feedback**$(G', V - F, F - F_2, k - j)$ is bounded by $O(2^{(k-j)+(k+1-j)}n^2) = O(4^{k-j}n^2)$. Now for all integers j, $0 \le j \le k$, we enumerate all subsets F_2 of j vertices in F and apply the algorithm **Feedback**$(G', V - F, F - F_2, k - j)$ for those F_2 such that $G[F - F_2]$ is a forest. As we discussed above, the graph G has an FVS of size bounded by k if and only if for some $F_2 \subseteq F$, the algorithm **Feedback**$(G', V - F, F - F_2, k - j)$ produces an FVS F_1 for the graph G'. The running time of this process is bounded by

$$\sum_{j=0}^{k} \binom{k+1}{j} \cdot O(4^{k-j}n^2) = \sum_{j=0}^{k} \binom{k+1}{k-j+1} O(4^{k-j+1}n^2) = O(5^k n^2).$$

This completes the proof of the lemma. ▬

Finally, by combining Lemma 2.2 with iterative compression, we obtain the main result of this section.

Theorem 2.3 *The* FEEDBACK VERTEX SET *problem is solvable in time* $O(5^k kn^2)$.

Proof. To solve the FEEDBACK VERTEX SET problem, for a given graph $G = (V, E)$, we start by applying Bafna et al.'s 2-approximation algorithm for the MINIMUM FEEDBACK VERTEX SET problem [1]. This algorithm runs in $O(n^2)$ time, and either returns an FVS F' of size bounded by $2k$, or verifies that no FVS of size bounded by k exists. If no FVS is returned, the algorithm is terminated with the result that no FVS of size bounded by k exists. In the case of the opposite result, we use any subset $V' \subseteq F'$ of k vertices, and let $V_0 = V' \cup (V - F')$. Of course, the induced subgraph $G[V_0]$ has an FVS of size k, since $G[V_0 - V']$ is a forest. Let $F' - V_0 = \{v_1, v_2, \ldots, v_{|F'|-k}\}$, and let $V_i = V_0 \cup \{v_1, \ldots, v_i\}$ for $i = 0, 1, \ldots, |F'| - k$. Inductively, suppose that we have constructed an FVS F_i for the graph $G[V_i]$, where $|F_i| = k$. Then the set $F'_{i+1} = F_i + v_{i+1}$ is obviously an FVS for the graph $G[V_{i+1}]$ and $|F'_{i+1}| = k + 1$.

Now the pair $(G[V_{i+1}], F'_{i+1})$ is an instance for the FVS REDUCTION problem. Therefore, in time $O(5^k n^2)$, we can either construct an FVS F_{i+1} of size k for the graph $G[V_{i+1}]$, or report no such an FVS exists. Note that if the graph $G[V_{i+1}]$ does not have an FVS of size k, then the original graph G cannot have an FVS of size k. In this case, we simply stop and claim the non-existence of an FVS of size k for the original graph G. On the other hand, with an FVS F_{i+1} of size k for the graph $G[V_{i+1}]$, our induction proceeds to the next graph $G[V_{i+1}]$, until we reach the graph $G = G[V_{|F'|-k}]$. Clearly, this process runs in time $O(5^k kn^2)$ since $|F'| - k \le k$, and solves the FEEDBACK VERTEX SET problem. ▬

3 On feedback vertex sets in weighted graphs

In this section, we discuss the FVS problem on weighted graphs. A weighted graph $G = (V, E)$ is an undirected graph, where each vertex $u \in V$ is assigned a non-negative weight. The weight of a vertex set $A \subseteq V$ is the sum of the vertex weights of all vertices in A. We denote by $|A|$ the size of, i.e., the number of vertices in, the set A . The (parameterized) feedback vertex set problem on weighted graphs is formally defined as follows:

> WEIGHTED-FVS: given a weighted graph G and an integer k, either find an FVS F of minimum weight for G such that $|F| \le k$, or report that no FVS of size bounded by k exists in G.

Our algorithm for the weighted case has several similarities with the unweighted case, but also has a significant difference. The difference is that the bypass operation for unweighted graphs can no longer be used in the weighted case. Indeed, a degree-2 vertex in a weighted graph may be necessarily included in the objective FVS if its weight is very small.

On the other hand, if two degree-2 vertices v and w are adjacent, then we can always bypass the one with a larger weight. This is because every cycle in the graph either contains both v and w or contains neither of them, so we can always assume that the one with a larger weight is not included in the objective FVS. We call this operation that bypasses the vertex of a larger weight in two adjacent degree-2 vertices in a weighted graph the *restricted bypass* operation.

However, since the restricted bypass operation cannot guarantee to eliminate all degree-2 vertices in a weighted graph, step 4.2 in the algorithm **Feedback** is not always possible. To overcome this difficulty, we introduce a new partition structure of the vertices in a weighted graph.

A triple (V_0, V_1, V_2) is a *independent-forest partition* (briefly, an *IF-partition*) of a graph $G = (V, E)$ if (V_0, V_1, V_2) is a partitioning of V such that (1) $G[V_1]$ and $G[V_2]$ are forests;

(2) $G[V_0]$ is an independent set; and (3) every vertex $u \in V_0$ is of degree 2 in G, and all neighbors of u are in V_2.

We consider the following problem on weighted graphs that is similar to the F-BIPARTITION FVS problem on unweighted graphs.

> WEIGHTED IF-PARTITION FVS: given a weighted graph G, an IF-partition (V_0, V_1, V_2) of G, and an integer k, either find an FVS F of minimum weight for G that satisfies the conditions $|F| \leq k$ and $F \subseteq V_0 \cup V_1$, or report that no such an FVS exists.

To study WEIGHTED IF-PARTITION FVS, we introduce the following concept.

Definition 3.1 Let (G, k) be an instance of WEIGHTED IF-PARTITION FVS, where an IF-partition (V_0, V_1, V_2) of G is given. The *deficiency* of (G, k) is defined by

$$\tau(k, V_0, V_1, V_2) = k - (|V_0| - \#c(V_2) + 1)$$

where $\#c(V_2)$ is the number of connected components in the subgraph $G[V_2]$.

Intuitively, the deficiency $\tau(k, V_0, V_1, V_2)$ of the instance (G, k) is the maximum number of vertices in the objective FVS that are in the set V_1 (this will become clearer during our discussion below). Our algorithm for the WEIGHTED IF-PARTITION FVS problem is based on the following observation: once we have correctly determined all vertices in the objective FVS that are in the set V_1, the problem will become solvable in polynomial time, as shown in the following lemma.

Lemma 3.2 *Let (G, k) be an instance of* WEIGHTED IF-PARTITION FVS, *with an IF-partition (V_0, V_1, V_2) of G. If $V_1 = \emptyset$ or $\tau(k, V_0, V_1, V_2) \leq 0$, then the solution to the instance (G, k) can be constructed in time $O(n^2)$.*

Proof. Construct a new graph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, where each vertex μ in $\mathcal{V}$ corresponds to a connected component in the induced subgraph $G[V_2]$, and each edge $[\mu, \nu]$ in $\mathcal{E}$ corresponds to a vertex v in the set V_0 such that the two edges incident to v in G are connected to the connected components in $G[V_2]$ that correspond to the two vertices μ and ν, respectively, in $\mathcal{H}$. Equivalently, the graph $\mathcal{H}$ can be obtained from the induced subgraph $G[V_0 \cup V_2]$ by shrinking each connected component in $G[V_2]$ into a single vertex and bypassing each degree-2 vertex in V_0. Moreover, we give each edge in $\mathcal{H}$ a weight that is equal to the weight of the corresponding vertex in V_0. Thus, the graph $\mathcal{H}$ is a graph with edge weights.

First consider the case of $V_1 = \emptyset$. If $k < 0$ then the solution to (G, k) is "No": we cannot remove a negative number of vertices. Assuming $k \geq 0$. Then we need to find a minimum-weight subset of at most k vertices in the set V_0 whose removal from $G = G[V_0 \cup V_2]$ results in an acyclic graph. Note that removing vertices in V_0 in the graph G corresponds to removing edges in the graph $\mathcal{H}$. Therefore, this problem is equivalent to finding a minimum-weight subset of at most k edges in the graph $\mathcal{H}$ whose removal from $\mathcal{H}$ results in an acyclic graph (note that each connected component in $G[V_2]$ is a tree). Let $\mathcal{H}_1$, ..., $\mathcal{H}_s$ be the connected components of the graph $\mathcal{H}$, where for each i, the component $\mathcal{H}_i$ has n_i vertices and m_i edges. In order to get an acyclic graph from $\mathcal{H}$, it is necessary and sufficient to remove $m_i - n_i + 1$ edges from $\mathcal{H}_i$ for each i (i.e., to make each connected component in $\mathcal{H}$ a tree). In consequence, in order to get an acyclic graph from $\mathcal{H}$, it is necessary and sufficient to remove $\sum_i^s (m_i - n_i + 1) = |\mathcal{E}| - |\mathcal{V}| + s$ edges from the graph $\mathcal{H}$.

Correspondingly, in case $V_1 = \emptyset$, a minimum-weight FVS in V_0 for the graph G contains exact $|\mathcal{E}| - |\mathcal{V}| + s$ vertices. Note that $|\mathcal{E}| = |V_0|$, and $|\mathcal{V}|$ is equal to the number $\#c(V_2)$ of connected components in the induced subgraph $G[V_2]$. Thus, every FVS in the graph G contains at least $|V_0| - \#c(V_2) + s$ vertices. Therefore, if $\tau(k, V_0, V_1, V_2) = k - (|V_0| - \#c(V_2) + 1) < 0$, or $\tau(k, V_0, V_1, V_2) = 0$ but $s > 1$, then we have $k < |V_0| - \#c(V_2) + s$. That is, the graph G has no FVS of size bounded by k and the solution to the instance (G, k) is a "No".

The remaining case is that $s = 1$, and $\tau(k, V_0, V_1, V_2) = k - (|V_0| - \#c(V_2) + 1) = 0$. In this case, to find a minimum-weight FVS of k vertices in V_0, we construct a maximum-weight spanning tree in the graph $\mathcal{H}$ (this can be done by a modified minimum spanning tree algorithm of time $O(n^2)$[4]). The remaining $|\mathcal{E}| - |\mathcal{V}| + 1 = |V_0| - \#c(V_2) + 1 = k$ edges in $\mathcal{H}$ then correspond to k vertices in the set V_0 that make a minimum-weight FVS for the graph G. Summarizing the above discussion, we conclude that if $V_1 = \emptyset$, then the solution to the instance (G, k) can be constructed in time $O(n^2)$.

Now consider the case $\tau(k, V_0, V_1, V_2) \leq 0$. As shown above, even to break all cycles in the induced subgraph $G[V_0 \cup V_2]$ requires removing at least $|V_0| - \#c(V_2) + 1$ vertices in the set V_0. Therefore, if $\tau(k, V_0, V_1, V_2) \leq 0$, then $k \leq (|V_0| - \#c(V_2) + 1)$, and all k vertices in the objective FVS must be in the set V_0 in order to break all cycles in the induced subgraph $G[V_0 \cup V_2]$, and no vertex in the objective FVS can be in the set V_1. Therefore, if the induced subgraph $G[V_1 \cup V_2]$ contains a cycle, then the solution to (G, k) is a "No". On the other hand, suppose that $G[V_1 \cup V_2]$ is a forest, then the graph G has another IF-partition (V_0', V_1', V_2'), where $V_0' = V_0$, $V_1' = \emptyset$, and $V_2' = V_1 \cup V_2$. It is easy to verify that in this case the instance (G, k) with the IF-partition (V_0', V_1', V_2') has the same solution set as the same instance with the IF-partition (V_0, V_1, V_2). Now since $V_1' = \emptyset$, by the first part of this lemma, the solution to (G, k) with the IF-partition (V_0', V_1', V_2') can be constructed in time $O(n^2)$. This completes the proof of the lemma. ▬

Now we are ready for our main algorithm, which is given in Figure 3 and solves the WEIGHTED IF-PARTITION FVS problem. As explained for the unweighted case, vertices of degree less than 2 cannot contribute to the objective FVS, thus can be directly deleted. Moreover, each restricted bypass operation takes time $O(n)$ and eliminates a degree-2 vertex in a pair of adjacent degree-2 vertices. Therefore, we can perform a preprocessing of time $O(n^2)$ and assume that the input graph G of the algorithm contains no vertex of degree less than 2, and no two adjacent degree-2 vertices. Moreover, for each tree in the forest $G[V_1]$, we fix a root so that we can talk about the "lowest leaf" in a tree in $G[V_1]$.

We first discuss the correctness of the algorithm. Step 1 of the algorithm is justified by Lemma 3.2. Justifications for steps 2, 3.1, and 3.2 are exactly the same as those for the unweighted case. Now consider step 4. When the algorithm reaches step 4, the following conditions hold: (1) every vertex in G has degree at least 2; (2) there are no two adjacent degree-2 vertices in G; (3) $V_1 \neq \emptyset$; and (4) every vertex in V_1 is incident to at most one edge whose other end is in V_2. Conditions (1) and (2) hold because of our assumption on the input graph G; condition (3) holds because of step 1; and condition (4) holds because of step 3.

By condition (3) and because the induced subgraph $G[V_1]$ is a forest, step 4 can always pick the vertex w_1. By conditions (1) and (4), the vertex w_1 is adjacent to a unique vertex v_1 in V_2. Then by condition (1) again, w_1 must have a parent w in the tree T in $G[V_1]$. In consequence, the vertex w_1 has degree exactly 2 in the graph G. Finally, since w_1 is the lowest leaf in the tree T, all children w_1, ..., w_t of w in T are also leaves in T. By conditions (1) and (4) again, each child w_i of w has a unique neighbor v_i in the set V_2, and every child w_i of w has degree exactly 2 in the graph G.

```
Algorithm-1 W-Feedback($G, V_0, V_1, V_2, k$)
Input: $G = (V, E)$ is a graph with an IF-partition $(V_0, V_1, V_2)$, $k$ is an integer
Output: a minimum-weight FVS $F$ of $G$ such that $|F| \leq k$ and $F \subseteq V_0 \cup V_1$;
            or report "No" (i.e., no such an FVS).

1.     if $V_1 = \emptyset$ or $\tau(k, V_0, V_1, V_2) \leq 0$ then solve the problem in time $O(n^2)$;
2.     if ($k < 0$) or ($k = 0$ and $G$ is not a forest) then return "No";
3.     if a vertex $w$ in $V_1$ is incident to 2 edges whose other ends are in $V_2$ then
3.1        if 2 edges incident to $w$ have their other ends in the same tree in $G[V_2]$
           then return $w$ + W-Feedback($G - w, V_0, V_1 - w, V_2, k - 1$);
3.2        else $F_1 = w$ + W-Feedback($G - w, V_0, V_1 - w, V_2, k - 1$);
                $F_2$ = W-Feedback($G, V_0, V_1 - w, V_2 + w, k$);
                return the one of $F_1$ and $F_2$ that has a smaller weight;
4.     else pick a lowest leaf $w_1$ in any tree $T$ in $G[V_1]$;
4.1        let $w$ be the parent of $w_1$ in $T$, and let $w_1, \ldots, w_t$ be the children
              of $w$ in $T$, where for each $i$, $w_i$ has a neighbor $v_i$ in $V_2$;
4.2        if $w$ has a neighbor $v$ in $V_2$ then
4.2.1          if for some $i$, $v$ and $v_i$ are in the same tree in $G[V_2]$
4.2.2          then $F_1 = w$ + W-Feedback($G - w, V_0, V_1 - w, V_2, k - 1$);
                    $F_2 = w_i$ + W-Feedback($G - w_i, V_0, V_1 - w_i, V_2, k - 1$);
                    return the one of $F_1$ and $F_2$ that has a smaller weight;
4.2.3          else $F_1 = w$ + W-Feedback($G - w, V_0, V_1 - w, V_2, k - 1$);
4.2.4               $F_2$ = W-Feedback($G, V_0 + w_1 + \ldots + w_t, V_1 - w, V_2 + w, k$);
                    return the one of $F_1$ and $F_2$ that has a smaller weight;
4.3        else $F_1 = w$ + W-Feedback($G - w, V_0, V_1 - w, V_2, k - 1$);
                $F_2$ = W-Feedback($G, V_0 + w_1 + \ldots + w_t, V_1 - w, V_2 + w, k$);
                return the one of $F_1$ and $F_2$ that has a smaller weight;
```

Figure 3: Algorithm for weighted FVS problem

If the vertex w has a (unique) neighbor v in V_2, and the vertices v and v_i for some i belong to the same tree T' in $G[V_2]$, as given in step 4.2.1, then this tree T' plus the edges $[v_i, w_i]$, $[w_i, w]$, and $[w, v]$ must contain a cycle, and in this cycle all vertices are in V_2 except w_i and w. Therefore, to break this cycle using vertices not in V_2, one of the vertices w and w_i must be included in the objective FVS. Thus, step 4.2.2 correctly handles this case. On the other hand, suppose that the vertex v is in a tree in $G[V_2]$ that does not contain any of the vertices $v_1, \ldots, v_t$. Then we simply branch on the vertex w: step 4.2.3 includes w in the objective FVS, and step 4.2.4 excludes w from the objective FVS, by moving w from V_1 to V_2. Note that in case of step 4.2.4, each of the degree-2 vertices $w_1, \ldots, w_t$ now is incident to two edges whose other ends are in V_2. Thus, we can correctly move these vertices from V_1 to V_0. Moreover, it is easy to verify that the new partition in each of the cases still makes a valid IF-partition of the graph G.

Finally, suppose that the vertex w has no neighbor in V_2. By condition (2) and because the vertex w_1 has degree 2 in G, the vertex w must have at least two children in the tree T (i.e., $t \geq 2$). Again we branch on w by either including or excluding w in the objective FVS, as given by step 4.3.

Since all possible cases are covered in the algorithm, we conclude that when the algorithm **W-Feedback** stops, it must output a correct solution to the given instance (G, k).

Lemma 3.3 *The algorithm* **W-Feedback**(G, V_0, V_1, V_2, k) *solves the* WEIGHTED IF-BIPARTITION FVS *problem correctly, and runs in time* $O(2^{\tau(k,V_0,V_1,V_2)}n^2)$, *where* n *is the number of vertices in the graph* G.

Proof. The correctness of the algorithm is verified by the preceding discussion.

As for the unweighted case, we first count the number of leaves in the search tree corresponding to the execution of the algorithm. Let $T(k, V_0, V_1, V_2)$ be the number of leaves in the search tree for algorithm **W-Feedback**(G, V_0, V_1, V_2, k). We prove by induction that $T(k, V_0, V_1, V_2) \le 2^{\tau(k,V_0,V_1,V_2)}$.

Most cases are similar to the corresponding cases for the unweighted case, and can be easily verified. For example, suppose that step 3.2 of the algorithm is executed. Then

$$T(k, V_0, V_1, V_2) \le T(k-1, V_0, V_1 - w, V_2) + T(k, V_0, V_1 - w, V_2 + w)$$

By induction $T(k-1, V_0, V_1 - w, V_2) \le 2^{\tau_1}$, and $T(k, V_0, V_1 - w, V_2 + w) \le 2^{\tau_2}$, where $\tau_1 = \tau(k-1, V_0, V_1 - w, V_2)$ and $\tau_2 = \tau(k, V_0, V_1 - w, V_2 + w)$. By the definition, we have

$$\tau_1 = (k-1) - (|V_0| - \#c(V_2) + 1) = \tau(k, V_0, V_1, V_2) - 1$$

Moreover, in this case, the vertex w is adjacent to at least two vertices in V_2, and no two vertices in V_2 that are adjacent to w are in the same tree in $G[V_2]$. Therefore, when w is added to the induced subgraph $G[V_2]$, w merges at least two connected components in $G[V_2]$. That is, the number of connected components in the induced subgraph $G[V_2 + w]$ is at least one fewer than that in the induced subgraph $G[V_2]$: $\#c(V_2 + w) \le \#c(V_2) - 1$. This gives

$$\tau_2 = k - (|V_0| - \#c(V_2 + w) + 1) \le k - (|V_0| - (\#c(V_2) - 1) + 1) = \tau(k, V_0, V_1, V_2) - 1$$

This verifies that in this case (i.e., in the case of step 3.2), we have

$$T(k, V_0, V_1, V_2) \le 2^{\tau_1} + 2^{\tau_2} \le 2^{\tau(k,V_0,V_1,V_2)-1} + 2^{\tau(k,V_0,V_1,V_2)-1} = 2^{\tau(k,V_0,V_1,V_2)}.$$

The two least trivial cases that are different from the unweighted cases occur in steps 4.2.3-4.2.4 and in step 4.3, for which we give more detailed analysis.

Suppose that steps 4.2.3-4.2.4 of the algorithm are executed. In this case, the vertex w has children $w_1, \ldots, w_t$ in the tree T in the induced subgraph $G[V_1]$, w has a unique neighbor v in V_2, and for each i, w_i has a unique neighbor v_i in V_2, such that no v_i shares the same connected component with v in the induced subgraph $G[V_2]$. Steps 4.2.3-4.2.4 give

$$T(k, V_0, V_1, V_2) \le T(k-1, V_0, V_1 - w, V_2) + T(k, V_0 + w_1 + \cdots + w_t, V_1 - w, V_2 + w)$$

As we have verified above, we have $T(k-1, V_0, V_1 - w, V_2) \le 2^{\tau(k,V_0,V_1,V_2)-1}$. Moreover, $|V_0 + w_1 + \cdots + w_t| \ge |V_0| + 1$, and, since v is the only neighbor of w in V_2, we have $\#c(V_2 + w) = \#c(V_2)$. Therefore,

$$\begin{aligned} & \tau(k, V_0 + w_1 + \cdots + w_t, V_1 - w, V_2 + w) \\ = \; & k - (|V_0 + w_1 + \cdots + w_t| - \#c(V_2 + w) + 1) \\ \le \; & k - ((|V_0| + 1) - \#c(V_2) + 1) \\ = \; & \tau(k, V_0, V_1, V_2) - 1. \end{aligned}$$

This gives

$$\begin{aligned} T(k, V_0 + w_1 + \cdots + w_t, V_1 - w, V_2 + w) & \le 2^{\tau(k,V_0+w_1+\cdots+w_t,V_1-w,V_2+w)} \\ & \le 2^{\tau(k,V_0,V_1,V_2)-1}. \end{aligned}$$

Thus, if steps 4.2.3-4.2.4 are executed, we also have $T(k, V_0, V_1, V_2) \le 2^{\tau(k,V_0,V_1,V_2)}$.

Finally, consider step 4.3, which gives the following recurrence

$$T(k, V_0, V_1, V_2) \le T(k-1, V_0, V_1 - w, V_2) + T(k, V_0 + w_1 + \cdots + w_t, V_1 - w, V_2 + w)$$

Again we have $T(k-1, V_0, V_1 - w, V_2) \le 2^{\tau(k,V_0,V_1,V_2)-1}$. Moreover, in this case, the vertex w has no neighbor in V_2. Since w_1 is of degree 2 in G, and the graph G contains no two adjacent degree-2 vertices, the vertex w has degree at least 3, i.e., w must have at least two children (i.e, $t \geq 2$). Therefore, $|V_0 + w_1 + \cdots + w_t| \geq |V_0| + 2$. Moreover, since w has no neighbor in V_2, we have $\#c(V_2 + w) = \#c(V_2) + 1$. Therefore,

$$\begin{aligned} & \tau(k, V_0 + w_1 + \cdots + w_t, V_1 - w, V_2 + w) \\ = \ & k - (|V_0 + w_1 + \cdots + w_t| - \#c(V_2 + w) + 1) \\ \leq \ & k - ((|V_0| + 2) - (\#c(V_2) + 1) + 1) \\ = \ & \tau(k, V_0, V_1, V_2) - 1 \end{aligned}$$

Thus, again we have

$$\begin{aligned} T(k, V_0 + w_1 + \cdots + w_t, V_1 - w, V_2 + w) & \leq 2^{\tau(k, V_0 + w_1 + \cdots + w_t, V_1 - w, V_2 + w)} \\ & \leq 2^{\tau(k, V_0, V_1, V_2) - 1}, \end{aligned}$$

which gives $T(k, V_0, V_1, V_2) \leq 2^{\tau(k, V_0, V_1, V_2)}$ for the case of step 4.3.

This completes the proof that the number of leaves in the search tree for the algorithm **W-Feedback**(G, V_0, V_1, V_2, k) is bounded by $2^{\tau(k, V_0, V_1, V_2)}$. Moreover, it is similar to that for the unweighted case to verify that along each root-leaf path in the search tree, the running time of the algorithm is bounded by $O(n^2)$. Therefore, the running time of the algorithm **W-Feedback**(G, V_0, V_1, V_2, k) is bounded by $O(2^{\tau(k, V_0, V_1, V_2)} n^2)$. ■

With Lemma 3.3, we can now proceed the same way as for the unweighted case to solve the original WEIGHTED-FVS problem. Consider the following weighted version of the FVS REDUCTION problem.

> WEIGHTED FVS REDUCTION: given a weighted graph G and an FVS F of size $k + 1$ for G, either construct an FVS F' of minimum weight that satisfies $|F'| \leq k$, or report that no such an FVS exists.

Note that in the definition of WEIGHTED FVS REDUCTION, we do not require that the given FVS F' of size $k + 1$ have the minimum weight.

Lemma 3.4 *The* WEIGHTED FVS REDUCTION *problem is solvable in time $O(5^k n^2)$.*

Proof. The proof proceeds very similarly to that for Lemma 2.2. For the given FVS F of size $k+1$ in the graph $G = (V, E)$, every FVS F' of size bounded by k for G (including the one with the minimum weight) is a union of a subset F_1 of at most $k - j$ vertices in $V - F$ and a subset F_2 of j vertices in F, for some integer j, $0 \leq j \leq k$, where $(V - F, F - F_2)$ is a forest bipartition of the graph $G_0 = G - F_2$. Therefore, we can enumerate all subsets F_2 of j vertices in F, for each j, $0 \leq j \leq k$, such that $(V - F, F - F_2)$ is a forest bipartition of the graph $G_0 = G - F_2$, and construct the minimum-weight FVS F_0 of G_0 satisfying $|F_0| \leq k - j$. Note that the forest bipartition $(V - F, F - F_2)$ of G_0 is in fact a special IF-partition (V_0, V_1, V_2) of G_0, where $V_0 = \emptyset$, $V_1 = V - F$, and $V_2 = F - F_2$. Therefore, by Lemma 3.3, a minimum-weight FVS F_0 of G_0 satisfying $|F_0| \leq k - j$ can be constructed in time

$$O(2^{\tau(k-j, V_0, V_1, V_2)} n^2) = O(2^{(k-j)-(0-\#c(F-F_2)+1)} n^2) = O(4^{k-j} n^2)$$

Now the proof proceeds exactly the same as that of Lemma 2.2, and concludes that the WEIGHTED FVS REDUCTION problem can be solved in time $O(5^k n^2)$. ■

Now using Theorem 2.3 and Lemma 3.4, we obtain the main result of this paper.

Theorem 3.5 *The* WEIGHTED-FVS *problem is solvable in time* $O(5^k kn^2)$.

Proof. Let (G, k) be a given instance of the WEIGHTED-FVS problem. As we explained in the proof of Theorem 2.3, we can first construct, in time $O(5^k kn^2)$, an FVS F of size $k+1$ for the graph G (the weight of F is not necessarily the minimum). Then we simply apply Lemma 3.4. ▬

Acknowledgment:
The authors wish to thank Hans L. Bodlaender for the suggestion of combining iterative compression and an approximation algorithm, to further reduce the polynomial factor in the complexity of the algorithms.

References

[1] V. BAFNA, P. BERMAN, AND T. FUJITO, *A 2-Approximation Algorithm for the Undirected Feedback Vertex Set Problem.*, SIAM J. Discrete Math., 12 (1999), pp. 289–297. 2

[2] A. BECKER, R. BAR-YEHUDA, AND D. GEIGER, *Randomized algorithms for the loop cutset problem*, J. Artificial Intelligence Res., 12 (2000), pp. 219–234. 1

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

[4] T. CORMEN, C. LEISERSON, R. RIVEST, AND C. STEIN, *Introduction to Algorithms*, 2nd ed., McGraw-Hill Book Company, Boston, Mass., 2001. 3

[5] F. K. H. A. DEHNE, M. R. FELLOWS, M. A. LANGSTON, F. A. ROSAMOND, AND K. STEVENS, *An O($2^{O(k)}n^3$) FPT algorithm for the undirected feedback vertex set problem.*, in Proceedings of the 11th Annual International Conference on Computing and Combinatorics (COCOON 2005), vol. 3595 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2005, pp. 859–869. 1, 1

[6] R. G. DOWNEY AND M. R. FELLOWS, *Fixed parameter tractability and completeness*, in Complexity Theory: Current Research, Cambridge University Press, 1993, pp. 191–225. 1

[7] R. G. DOWNEY AND M. R. FELLOWS, *Parameterized complexity*, Springer-Verlag, New York, 1999. 1

[8] P. FESTA, P. M. PARDALOS, AND M. G. C. RESENDE, *Feedback set problems*, in Handbook of combinatorial optimization, Supplement Vol. A, Kluwer Acad. Publ., Dordrecht, 1999, pp. 209–258. 1

[9] F. V. FOMIN, S. GASPERS, AND A. V. PYATKIN, *Finding a minimum feedback vertex set in time $O(1.7548^n)$*, in Proceedings of the 2nd International Workshop on Parameterized and Exact Computation (IWPEC 2006), vol. 4169 of Lecture Notes in Comput. Sci., Springer, Berlin, 2006, pp. 184–191. 1

[10] J. GUO, J. GRAMM, F. HÜFFNER, R. NIEDERMEIER, AND S. WERNICKE, *Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization.*, J. Comput. Syst. Sci., 72 (2006), pp. 1386–1396. 1, 1

[11] I. A. KANJ, M. J. PELSMAJER, AND M. SCHAEFER, *Parameterized algorithms for feedback vertex set.*, in Proceedings of the 1st International Workshop on Parameterized and Exact Computation (IWPEC 2004), Berlin, 2004, Springer, pp. 235–247. 1

[12] R. M. KARP, *Reducibility among combinatorial problems*, in Complexity of computer computations, Plenum Press, New York, 1972, pp. 85–103. 1

[13] V. RAMAN, S. SAURABH, AND C. R. SUBRAMANIAN, *Faster fixed parameter tractable algorithms for undirected feedback vertex set*, in Proceedings of the 13th International Symposium on Algorithms and Computation (ISAAC 2002), vol. 2518 of LNCS, Springer-Verlag, Berlin, 2002, pp. 241–248. 1

[14] V. RAMAN, S. SAURABH, AND C. R. SUBRAMANIAN, *Faster fixed parameter tractable algorithms for finding feedback vertex sets*, ACM Trans. Algorithms, 2 (2006), pp. 403–415. 1

[15] I. RAZGON, *Exact computation of maximum induced forest*, in Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT 2006), vol. 4059 of Lecture Notes in Comput. Sci., Berlin, 2006, Springer, pp. 160–171. 1

[16] B. REED, K. SMITH, AND A. VETTA, *Finding odd cycle transversals.*, Oper. Res. Lett., 32 (2004), pp. 299–301. 1, 2

[17] A. SILBERSCHATZ AND P. GALVIN, *Operating System Concents*, Addison-Wesley, Reading, Massachusetts, 1994. 1