Journal of Computer and System Sciences 74 (2008) 1188–1198

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Improved algorithms for feedback vertex set problems [☆]

Jianer Chen [a], Fedor V. Fomin [b], Yang Liu [a], Songjian Lu [a], Yngve Villanger [b,*]

[a] *Department of Computer Science, Texas A&M University, College Station, TX 77843-3112, USA*
[b] *Department of Informatics, University of Bergen, N-5020 Bergen, Norway*

ARTICLE INFO

Article history:
Received 13 December 2007
Received in revised form 24 April 2008
Available online 13 June 2008

Keywords:
Parameterized complexity
Feedback vertex set

ABSTRACT

We present improved parameterized algorithms for the FEEDBACK VERTEX SET problem on both unweighted and weighted graphs. Both algorithms run in time $\mathcal{O}(5^k kn^2)$. The algorithms construct a feedback vertex set of size at most k (in the weighted case this set is of minimum weight among the feedback vertex sets of size at most k) in a given graph G of n vertices, or report that no such feedback vertex set exists in G.

© 2008 Elsevier Inc. All rights reserved.

1. Introduction

Let G be an undirected 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 classic NP-complete problems [14] 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. [10]). There are also exact algorithms that find a minimum FVS in a graph of n vertices in time $\mathcal{O}(1.8899^n)$ [17] and in time $\mathcal{O}(1.7548^n)$ [11].

An important application of the FVS problem is *deadlock recovery* in operating systems [19], in which a deadlock is presented by a cycle in a *system resource-allocation graph* G. 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 in a graph of n vertices (where the weight of the FVS is minimized, in the case of weighted graphs), and run in time $f(k)n^{\mathcal{O}(1)}$ 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, mostly on the unweighted version of the problem. The first group of parameterized algorithms of running time $f(k)n^{\mathcal{O}(1)}$ for the FVS problem on unweighted graphs was given by

[☆] This work was supported in part by the National Science Foundation of USA under the Grant CCF-0430683 and by the Research Council of Norway. The extended abstract of this paper was presented at WADS 2007.

[*] Corresponding author.

E-mail addresses: chen@cs.tamu.edu (J. Chen), fomin@ii.uib.no (F.V. Fomin), yangliu@cs.tamu.edu (Y. Liu), sjlu@cs.tamu.edu (S. Lu), yngvev@ii.uib.no (Y. Villanger).

0022-0000/$ – see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcss.2008.05.002

Bodlaender, Fellows [3,8]	$\mathcal{O}(17(k^4)!n^{\mathcal{O}(1)})$
Downey and Fellows [9]	$\mathcal{O}((2k+1)^k n^2)$
Raman et al. [16]	$\mathcal{O}(\max\{12^k, (4\log k)^k\} n^{2.376})$
Kanj et al. [13]	$\mathcal{O}((2\log k + 2\log\log k + 18)^k n^2)$
Raman et al. [15]	$\mathcal{O}((12\log k/\log\log k + 6)^k n^{2.376})$
Guo et al. [12]	$\mathcal{O}((37.7)^k n^2)$
Dehne et al. [7]	$\mathcal{O}((10.6)^k n^3)$

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

Bodlaender [3] and by Downey and Fellows [8]. Since then a chain of dramatic improvements was obtained by different researchers (see Fig. 1 for references).

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 $\mathcal{O}(4^k kn^2)$ for the FVS problem on unweighted graphs, and a randomized algorithm of running time $\mathcal{O}(6^k kn^2)$ for the FVS problem on weighted graphs. To our knowledge, no deterministic algorithm of running time $f(k)n^{O(1)}$ for any function f was known prior to our results for the weighted FVS problem.

1.1. Our results

The main result of this paper is an algorithm that for a given integer k and a weighted graph G, either finds a minimum weight FVS in G of at most k vertices, or correctly reports that G contains no FVS of at most k vertices. The running time of our algorithm is $\mathcal{O}(5^k kn^2)$. This improves and generalizes a long chain of results in parameterized algorithms. Let us remark that the running time of our (deterministic) algorithm comes close to that of the best randomized algorithm for the FVS problem on unweighted graphs and is better than the running time of the previous best randomized algorithm for the FVS problem on weighted graphs.

The general approach of our algorithm is based on the *iterative compression* method [18], which has been successfully used recently for improved algorithms for the FVS and other problems [7,12,18]. The method starts with an FVS of k vertices for a small subgraph of the given graph, and iteratively grows the small subgraph while keeping an FVS of k vertices in the grown subgraph until the subgraph becomes the original input graph. This method makes it possible to reduce the original FVS problem on general graphs to the FVS problem on graphs with a special decomposition structure. The main contribution of the current paper is the development of a general algorithmic technique that identifies a dual parameter in problem instances that limits the number of times where the original parameter k cannot be effectively reduced during a branch and search process. In particular, for the FVS problem on graphs of the above special decomposition structure, a measure is introduced that nicely combines the original parameter and the dual parameter and bounds effectively the running time of a branch and search algorithm for the FVS problem. This technique leads to a simpler but significantly more efficient parameterized algorithm for the FVS problem on unweighted graphs. Moreover, the introduction of the measure greatly simplifies the processing of degree-2 vertices in a weighted graph, and enables us to solve the FVS problem on weighted graphs in the same complexity as that for the problem on unweighted graphs. Note that this is significant because no previous algorithms for the FVS problem on unweighted graphs can be extended to weighted graphs mainly because of the lack of an effective method for handling degree-2 vertices. Finally, the technique of dual parameters seems to be of general usefulness for the development of parameterized algorithms, and has been used more recently in solving other parameterized problems [4,5].

The remaining part of this paper is organized as follows. In Section 2, we provide in full details a simpler algorithm and its analysis for unweighted graphs. This is done for clearer demonstration of our approach. We also indicate why this simpler algorithm does not work for weighted graphs. In Section 3, we obtain the main result of the paper, the algorithm for the weighted FVS problem. This generalization of the results from Section 2 is not straightforward and requires a number of new structures and techniques.

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 $G = (V, E)$. We will denote by $G[W]$ the subgraph of G that is induced by the vertex set W. For simplicity we will use the notation $G - w$ and $G - W$ for respectively $G[V \setminus \{w\}]$ and $G[V \setminus W]$ where $w \in V$ and $W \subseteq V$. 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. For a vertex $u \in V$ the degree of u will be the number of edges incident to u.

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. The *size* of an FVS F is the number of vertices in F.

```
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 $|F| \leqslant k$ and $F \subseteq V_1$; or report "No" (i.e.,
    no such an FVS exists).

1.   if ($k < 0$) or ($k = 0$ and $G$ is not a forest) then return "No";
2.   if ($k \geqslant 0$) 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
             $F_1 =$ Feedback($G - w, V_1 \setminus \{w\}, V_2, k - 1$);
             if $F_1 =$ "No" then return "No" else return $F_1 \cup \{w\}$;
3.2.     else
             $F_1 =$ Feedback($G - w, V_1 \setminus \{w\}, V_2, k - 1$);
             $F_2 =$ Feedback($G, V_1 \setminus \{w\}, V_2 \cup \{w\}, k$);
             if $F_1 \neq$ "No" then return $F_1 \cup \{w\}$ else return $F_2$;
4.   else pick any vertex $w$ that has degree $\leqslant 1$ in $G[V_1]$;
4.1.     if $w$ has degree $\leqslant 1$ in the original graph $G$ then
             return Feedback($G - w, V_1 \setminus \{w\}, V_2, k$);
4.2.     else return Feedback($G, V_1 \setminus \{w\}, V_2 \cup \{w\}, k$).
```

Fig. 2. Algorithm for unweighted feedback vertex set problem.

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 at most k in G, or report that no such 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 at most 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.

Observe that certain structures in the input graph G can be easily processed and then removed from G. For example, if a vertex v has a self-loop (i.e., an edge whose both ends are incident to v), then the vertex v is necessarily contained in every FVS in G. Thus, we can directly include v in the objective FVS. If two vertices v and w are connected by multiple edges (i.e., there are more than one edge whose one end is v and the other end is w), then one of v and w must be contained in the objective FVS. Thus, we can branch into two recursive calls, one includes v, and the other includes w, in the objective FVS. All these operations are more efficient than the algorithm of running time $\mathcal{O}(5^k kn^2)$ developed in the current paper. Therefore, for a given input graph G, we always first apply a preprocessing that applies the above operations and removes 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.

The algorithm, **Feedback**(G, V_1, V_2, k), for the F-BIPARTITION FVS problem is given in Fig. 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 our search for an FVS is restricted to V_1, the only way to break the cycles in $T \cup \{w\}$ is to include the vertex w in the objective FVS. Moreover, the objective FVS of size at most k exists in G if and only if the remaining graph $G - w$ has an FVS of size at most $k - 1$ in the subset $V_1 \setminus \{w\}$ (note that $(V_1 \setminus \{w\}, V_2)$ 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 $|F_1| \leqslant k - 1$ and $F_1 \subseteq V_1 \setminus \{w\}$ (again note that $(V_1 \setminus \{w\}, V_2)$ 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 $V_1 \setminus \{w\}$. Also note that in this case, the subgraph $G[V_2 \cup \{w\}]$ induced by the subset $V_2 \cup \{w\}$ is still a forest since no two neighbors of w in V_2 belong to the same tree in $G[V_2]$. In consequence, $(V_1 \setminus \{w\}, V_2 \cup \{w\})$ 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 at most 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 at most 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 at most 1 in the induced subgraph $G[V_1]$ but has degree larger than 1 in the original graph G. Observing that w has at most one neighbor in V_2, we can derive that the degree of w in the original graph G must be exactly 2. Moreover, w has exactly two neighbors u and v such that v is in V_1 and u is in V_2.

Since the vertex w has degree 2 in the original graph G, and the vertex v is adjacent to w, we have that every cycle in G that contains w has to contain v. In consequence, if w is contained in the objective FVS, then we can simply replace it by v. Therefore, in this case, we can safely assume that the vertex w is not in the objective FVS. This can be easily implemented by moving the vertex w from the set V_1 to the set V_2, and recursively working on the modified instance, as given in step 4.2 of the algorithm (note that $(V_1 \setminus \{w\}, V_2 \cup \{w\})$ is a valid forest bipartition of the graph G, because by our assumption, the vertex w will be a degree-1 vertex in the induced subgraph $G[V_2 \cup \{w\}]$).

Now we are ready to present the following lemma.

Lemma 1. *The algorithm* **Feedback**(G, V_1, V_2, k) *correctly solves the* F-BIPARTITION FVS *problem. The running time of the algorithm is* $\mathcal{O}(2^{k+l}n^2)$*, 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 $\mathcal{T}$. 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 connected components (i.e., 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**$(G - w, V_1 \setminus \{w\}, V_2, k-1)$ is at most $T(k-1, l)$. Moreover, we assumed 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 $G[V_2 \cup \{w\}]$. Hence, the number of trees in $G[V_2 \cup \{w\}]$ is at most $l-1$. In consequence, the number of leaves in the search tree $\mathcal{T}_2$ corresponding to the recursive call **Feedback**$(G, V_1 \setminus \{w\}, V_2 \cup \{w\}, k)$ is at most $T(k, l-1)$. This gives the following recurrence relation:

$$T(k,l) \leqslant T(k-1,l) + T(k,l-1).$$

Also note that none of the (non-branching) recursive calls in the algorithm (steps 3.1, 4.1, and 4.2) would increase the values k and l, and that $T(0,l) = 1$ for all l and $T(k,0) = 1$ for all k (by steps 1–2). From all these facts, we can easily derive that $T(k,l) = \mathcal{O}(2^{k+l})$.

Finally, observe that along each root-leaf path in the search tree $\mathcal{T}$, the total number of executions of steps 1, 2, 3, 3.1, 4.1, and 4.2 of the algorithm is $\mathcal{O}(n)$ because each of these steps either stops immediately, or reduces the size of the set V_1 by at least 1 (and the size of V_1 is never increased during the execution of the algorithm). It remains to explain how each of the steps can be executed in $\mathcal{O}(n)$ time.

Before the first call to the **Feedback** algorithm, we use $\mathcal{O}(n^2)$ time, because this will happen only once. The three graphs, $G_1 = G[V_1]$, $G_2 = G[V_2]$, and $G_{12} = (V, E \setminus (E(G_1) \cup E(G_2)))$ can be trivially constructed in $\mathcal{O}(n^2)$ time. G_1 and G_2 are forests, and G_{12} is a bipartite graph with the two vertex sets V_1 and V_2 as independent sets.

Steps 1, 2, 4.1, and 4.2 can be easily performed in $\mathcal{O}(n)$ time. For step 3, we simply search for a vertex of V_1 that has degree at least 2 in G_{12}, and for step 4 we search in G_1 for a leaf (vertex of degree at most 1). The condition for step 3.1 is that no two neighbors of w belong to the same tree in $G[V_2]$, which can be checked by simply marking each neighbor of w, and doing a search in the forest $G[V_2]$.

Each of the steps 3.1, 3.2, 4.1, and 4.2 changes one or more of the graphs G_1, G_2, G_{12}, and we have to argue that these manipulations can also be done in $\mathcal{O}(n)$ time. Looking closely at these steps, we can observe that only two operations are required. The first is to delete a vertex in V_1, which corresponds to deleting the vertex and all incident edges in G_1 and G_{12}. The second operation is to move a vertex w from V_1 to V_2, which corresponds to deleting w from G_1 and updating G_2 and G_{12} as follows: add w to $V(G_2)$ and to V_2 of G_{12}, and read the set of edges incident to w in G, and add edges between w and vertices in V_2 to G_2 and between w and V_1 to G_{12}. Using double linked lists and pointers it is possible to delete a vertex and all incident edges in $\mathcal{O}(n)$ time, and to insert edges in $\mathcal{O}(1)$ time.

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

Following the idea of *iterative compression* proposed by Reed et al. [18], 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 2. *The* FVS REDUCTION *problem on an n-vertex graph G can be solved in time* $\mathcal{O}(5^k n^2)$.

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 at most k for G is a union of a subset F_1 of at most $k-j$ vertices in $V \setminus F$ and a subset

F_2 of j vertices in F, for some integer j, $0 \leqslant j \leqslant k$. Note that since we assume that no vertex in $F \setminus F_2$ is in the FVS F', the induced subgraph $G[F \setminus F_2]$ must be a forest. For each j, $0 \leqslant j \leqslant k$, we enumerate all subsets of j vertices in F. For each such subset F_2 in F such that $G[F \setminus F_2]$ is a forest, we seek a subset F_1 of at most $k - j$ vertices in $V \setminus F$ such that $F_1 \cup F_2$ is an FVS in 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 at most k, where $F_1 \subseteq V \setminus F$, if and only if the subset F_1 of $V \setminus F$ is an FVS for the graph $G - F_2$ and $|F_1| \leqslant k - j$. Therefore, to solve the original problem, we construct an FVS F_1 for the graph $G - F_2$ such that $|F_1| \leqslant k - j$ and $F_1 \subseteq V \setminus F$.

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

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

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

This completes the proof of the lemma. □

Finally, by combining Lemma 2 with iterative compression, we obtain the main result of this section.

Theorem 3. *The* FEEDBACK VERTEX SET *problem on an n-vertex graph is solvable in time* $\mathcal{O}(5^k kn^2)$.

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 $\mathcal{O}(n^2)$ time, and either returns an FVS F' of size at most $2k$, or verifies that no FVS of size at most k exists. If no FVS is returned, the algorithm is terminated with the conclusion that no FVS of size at most k exists. In the case of the opposite result, we use any subset V' of k vertices in F', and put $V_0 = V' \cup (V \setminus F')$. Of course, the induced subgraph $G[V_0]$ has an FVS of size k, namely the set V' ($G[V_0 \setminus V']$ is a forest). Let $F' \setminus V' = \{v_1, v_2, \ldots, v_{|F'|-k}\}$, and let $V_i = V_0 \cup \{v_1, \ldots, v_i\}$ for $i \in \{0, 1, \ldots, |F'| - k\}$. Inductively, suppose that we have constructed an FVS F_i for the graph $G[V_i]$, where $|F_i| = k$. Then the set $F'_{i+1} = F_i \cup \{v_{i+1}\}$ 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 $\mathcal{O}(5^k n^2)$, we can either construct an FVS F_{i+1} of size k for the graph $G[V_{i+1}]$, or report that 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 $G = G[V_{|F'|-k}]$. This process runs in time $\mathcal{O}(5^k kn^2)$ since $|F'| - k \leqslant k$, and solves the FEEDBACK VERTEX SET problem. □

3. Feedback vertex set in weighted graphs

In this section, we discuss the FEEDBACK VERTEX SET problem on weighted graphs. A weighted graph $G = (V, E)$ is an undirected graph, where each vertex $u \in V$ is assigned a *weight* that is a positive real number. 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 cardinality of 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 $|F| \leqslant k$, or report that no FVS of size at most k exists in G.

The algorithm for the weighted case has several similarities with the unweighted case, but has also a significant difference. The difference is that step 4.2 of **Algorithm-1** becomes invalid for weighted graphs. A degree-2 vertex w in the set V_1 cannot simply be excluded from the objective FVS. If the weight of w is smaller than that of its parent v in $G[V_1]$, it may become necessary to include w instead of v in the objective FVS.

To overcome this difficulty, we introduce a new partition structure of the vertices in a weighted graph.

Definition 4. A triple (V_0, V_1, V_2) is an *independent-forest partition* (*IF-partition*) of a graph $G = (V, E)$ if (V_0, V_1, V_2) is a partitioning of V (i.e., $V_0 \cup V_1 \cup V_2 = V$, and V_0, V_1, and V_2 are pairwise disjoint), such that

(1) $G[V_1]$ and $G[V_2]$ are forests;
(2) $G[V_0]$ is an independent set;
(3) Every vertex u in V_0 is of degree 2 in G, with both neighbors in V_2.

The following problem is the analogue of the F-BIPARTITION FVS problem on weighted 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 $|F| \leqslant k$ and $F \subseteq V_0 \cup V_1$, or report that no such an FVS exists.

To develop and analyze our algorithm for the WEIGHTED IF-PARTITION FVS problem, we need the following concept of *measure* for the problem instances. For a vertex subset V' in the graph G, we will denote by $\#c(V')$ the number of connected components in the induced subgraph $G[V']$.

Definition 5. Let (G, V_0, V_1, V_2, k) be an instance of the WEIGHTED IF-PARTITION FVS problem with an IF-partition (V_0, V_1, V_2). The *deficiency* of the instance (G, V_0, V_1, V_2, k) is defined as

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

Intuitively, $\tau(k, V_0, V_1, V_2)$ of the instance (G, V_0, V_1, V_2, k) is an upper bound on the 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 6. *Let (G, V_0, V_1, V_2, k) be an instance of the* WEIGHTED IF-PARTITION FVS *problem with an IF-partition (V_0, V_1, V_2) of an n-vertex graph G. If $V_1 = \emptyset$, or $V_2 = \emptyset$, or $\tau(k, V_0, V_1, V_2) \leqslant 0$, then a solution to the instance (G, V_0, V_1, V_2, k) can be constructed in time $\mathcal{O}(n^2)$.*

Proof. First of all, note that if $k < 0$, then the solution to the instance is "No": we cannot remove a negative number of vertices from G. Thus, in the following discussion, we assume that $k \geqslant 0$.

If $V_2 = \emptyset$, then by the definition, V_0 should also be an empty set. Thus, the graph $G = G[V_1]$ is a forest, and the solution to the instance (G, V_0, V_1, V_2, k) is the empty set $\emptyset$.

Now consider the case $V_1 = \emptyset$. Then we need to find a minimum-weight subset of at most k vertices in the set V_0 whose removal from the graph $G = G[V_0 \cup V_2]$ results in a forest. We will solve this problem by creating a new graph $\mathcal{H}$, such that an optimal solution for (G, V_0, V_1, V_2, k) corresponds to the edges which are not used in a maximum weight spanning tree of $\mathcal{H}$.

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 neighbors of v are in the connected components in $G[V_2]$ that correspond to the two vertices μ and ν, respectively, in $\mathcal{H}$. Intuitively, the graph $\mathcal{H}$ can be obtained from the graph $G = G[V_0 \cup V_2]$ by "shrinking" each connected component in $G[V_2]$ into a single vertex and "smoothing" each degree-2 vertex in V_0 (note that the graph $\mathcal{H}$ may contain multiple edges and self-loops). 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. Observe that there is a one-to-one correspondence between the connected components in the graph $\mathcal{H}$ and the connected components in the graph G. Moreover, since each connected component in the induced subgraph $G[V_2]$ is a tree, a connected component in the graph $\mathcal{H}$ is a tree if and only if the corresponding connected component in the graph G is a tree. Most importantly, removing a vertex in V_0 in the graph G corresponds to removing the corresponding edge in the graph $\mathcal{H}$. Therefore, the problem of constructing a minimum-weight vertex set in V_0 whose removal from G results in a forest is equivalent to the problem of constructing a minimum-weight edge set in the graph $\mathcal{H}$ whose removal from $\mathcal{H}$ results in a forest.

Let $\mathcal{H}_1, \ldots, \mathcal{H}_s$ 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. An edge set $\mathcal{E}_i$ in $\mathcal{H}_i$ whose removal from $\mathcal{H}_i$ results in a forest is of the minimum weight if and only if the complement graph $\mathcal{H}_i - \mathcal{E}_i$ is a spanning tree of the maximum weight in $\mathcal{H}_i$. Thus, the union $\mathcal{E}' = \bigcup_{i=1}^{s}(\mathcal{H}_i - \mathcal{T}_i)$ is a minimum-weight edge set whose removal from $\mathcal{H}$ results in a forest, where for each i, $\mathcal{T}_i$ is a maximum-weight spanning tree in the graph $\mathcal{H}_i$. Since the maximum-weight spanning tree $\mathcal{T}_i$ in $\mathcal{H}_i$ can be constructed in time $\mathcal{O}(n_i^2)$ by modifying the well-known minimum spanning tree algorithms (the algorithms work even for graphs with self-loops and multiple edges) [6], we conclude that the minimum-weight edge set $\mathcal{E}'$ in $\mathcal{H}$ can be constructed in time $\sum_{i=1}^{s} \mathcal{O}(n_i^2) = \mathcal{O}(n^2)$. Also note that the number of edges in the set $\mathcal{E}'$ is equal to $\sum_{i=1}^{s}(m_i - n_i + 1) = |\mathcal{E}| - |\mathcal{V}| + s$.

Correspondingly, in case $V_1 = \emptyset$, each minimum-weight FVS in V_0 for the graph G contains exactly $|\mathcal{E}| - |\mathcal{V}| + s$ vertices, and such an FVS can be constructed in time $\mathcal{O}(n^2)$. Note that $|\mathcal{E}| = |V_0|$, $|\mathcal{V}|$ is equal to the number $\#c(V_2)$ of connected components in the induced subgraph $G[V_2]$, and s (which is the number of connected components in $\mathcal{H}$) is equal to the number $\#c(G)$ of connected components in the graph $G = G[V_0 \cup V_2]$. Thus, each minimum-weight FVS in V_0 for the graph

```
Algorithm-2 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 $|F| \leqslant k$ and $F \subseteq V_0 \cup V_1$;
    or report "No" (i.e., no such an FVS exists).

1.   if ($V_1 = \emptyset$) or ($V_2 = \emptyset$) or ($\tau(k, V_0, V_1, V_2) \leqslant 0$) then
         solve the problem in time $\mathcal{O}(n^2)$;
2.   if ($k < 0$) or ($k = 0$ and $G$ is not a forest) then return "No";
3.   if a vertex $w$ in $V_1$ has degree less than 2 in $G$ then
         return W-Feedback($G - w, V_0, V_1 \setminus \{w\}, V_2, k$);
4.   else if a vertex $w$ in $V_1$ has at least two neighbors in $V_2$ then
4.1      if two neighbors of $w$ are in the same tree of $G[V_2]$ then
             return ($\{w\} \cup$ W-Feedback($G - w, V_0, V_1 \setminus \{w\}, V_2, k - 1$));
4.2      else
             $F_1 =$ W-Feedback($G - w, V_0, V_1 \setminus \{w\}, V_2, k - 1$);
             $F_2 =$ W-Feedback($G, V_0, V_1 \setminus \{w\}, V_2 \cup \{w\}, k$);
             return min-w($F_1 \cup \{w\}, F_2$);
5    else pick a lowest leaf $w_1$ in any tree $T$ in $G[V_1]$;
         let $w$ be the parent of $w_1$ in $T$, and let $w_1, \ldots, w_t$ be the children of $w$ in $T$;
5.1      if ($w$ has a neighbor in $V_2$) or ($w$ has more than one child in $T$) then
             $F_1 =$ W-Feedback($G - w, V_0, V_1 \setminus \{w\}, V_2, k - 1$);
             $F_2 =$ W-Feedback($G, V_0 \cup \{w_1, \ldots, w_t\}, V_1 \setminus \{w, w_1, \ldots, w_t\}, V_2 \cup \{w\}, k$);
             return min-w($F_1 \cup \{w\}, F_2$);
5.2      else
             if the weight of $w_1$ is larger than the weight of $w$ then
                 return W-Feedback($G, V_0, V_1 \setminus \{w_1\}, V_2 \cup \{w_1\}, k$);
             else return W-Feedback($G, V_0 \cup \{w_1\}, V_1 \setminus \{w, w_1\}, V_2 \cup \{w\}, k$).
```

Fig. 3. Algorithm for weighted FVS problem.

G contains exactly $|V_0| - \#c(V_2) + \#c(G)$ vertices, and such a minimum-weight FVS can be constructed in time $\mathcal{O}(n^2)$. Therefore, for the given instance (G, V_0, V_1, V_2, k) of the WEIGHTED IF-PARTITION FVS problem with $V_1 = \emptyset$, the solution is "No" if $k < |V_0| - \#c(V_2) + \#c(G)$; and the solution is an $\mathcal{O}(n^2)$ time constructible FVS of $|V_0| - \#c(V_2) + \#c(G)$ vertices in V_0 if $k \geqslant |V_0| - \#c(V_2) + \#c(G)$.

This completes the proof that when $V_1 = \emptyset$, a solution to the instance (G, V_0, V_1, V_2, k) can be constructed in time $\mathcal{O}(n^2)$.

Now consider the case $\tau(k, V_0, V_1, V_2) \leqslant 0$. If $V_2 = \emptyset$, then by the first part of the proof, the lemma holds. Thus, we assume that $V_2 \neq \emptyset$. As analyzed above, to break every cycle in the induced subgraph $G[V_0 \cup V_2]$ we have to remove at least $|V_0| - \#c(V_2) + \#c(V_0 \cup V_2)$ vertices in the set V_0. Therefore, if $\tau(k, V_0, V_1, V_2) \leqslant 0$, then $k \leqslant |V_0| - \#c(V_2) + 1 \leqslant |V_0| - \#c(V_2) + \#c(V_0 \cup V_2)$ (note that $V_2 \neq \emptyset$ so $\#c(V_0 \cup V_2) \geqslant 1$). Thus, in this case, 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. Hence, if the induced subgraph $G[V_1 \cup V_2]$ contains a cycle, then the solution to the instance is "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, V_0', V_1', V_2', k) with the IF-partition (V_0', V_1', V_2') has the same solution set as the instance (G, V_0, V_1, V_2, k) with the IF-partition (V_0, V_1, V_2). Since $V_1' = \emptyset$, by the second part of the proof, a solution to the instance (G, V_0', V_1', V_2', k) with the IF-partition (V_0', V_1', V_2') can be constructed in time $\mathcal{O}(n^2)$. This completes the proof of the lemma. □

We are now in a position to introduce our main algorithm, which is given in Fig. 3 and solves the WEIGHTED IF-PARTITION FVS problem. The subroutine **min-w**(S_1, S_2) on two vertex subsets S_1 and S_2 in the algorithm returns among S_1 and S_2 the one with a smaller weight (or any one of them if the weights are tied). To simplify our descriptions, we take the conventions that "No" is a special vertex set of an infinitely large weight and that any set plus "No" gives a "No." Therefore, the value of **min-w**(S_1, S_2) will be (1) "No" if both S_1 and S_2 are "No"; (2) S_1 if S_2 is "No"; (3) S_2 if S_1 is "No"; and (4) the one of smaller weight among S_1 and S_2 if both S_1 and S_2 are not "No."

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 $G[V_1]$.

Lemma 7. *The algorithm* **W-Feedback**(G, V_0, V_1, V_2, k) *correctly solves the* WEIGHTED IF-BIPARTITION FVS *problem, and its running time is* $\mathcal{O}(2^{\tau(k, V_0, V_1, V_2)} n^2)$*, where n is the number of vertices in the graph G.*

Proof. We first verify the correctness of the algorithm. Step 1 of the algorithm is justified by Lemma 6. Justifications for steps 2, 3, 4, 4.1, and 4.2 are exactly the same as that for steps 1, 4.1, 3, 3.1, and 3.2 in **Algorithm-1** for unweighted graphs. Now consider step 5. When the algorithm reaches step 5, the following conditions hold:

(1) the sets V_1 and V_2 are not empty;
(2) every vertex in the set V_1 has degree at least 2 in the graph G; and
(3) every vertex in the set V_1 has at most one neighbor in the set V_2.

Condition (1) holds because of step 1; condition (2) holds because of step 3; and condition (3) holds because of step 4.

By condition (1) and because the induced subgraph $G[V_1]$ is a forest, step 5 can always pick the vertex w_1. By conditions (2) and (3), the vertex w_1 has a unique neighbor in V_2. Also by conditions (2) and (3), the vertex 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 G. Finally, since w_1 is the lowest leaf in the tree T, all children $w_1, \dots, w_t$ of w in the tree T are also leaves in T. By conditions (2) and (3) again, each child w_i of w has a unique neighbor in the set V_2, and every child w_i of w has degree exactly 2 in the graph G.

Step 5.1 simply branches on the vertex w. To include the vertex w in the objective FVS, we simply remove w from the graph G (and from the set V_1), and recursively look for an FVS in $V_0 \cup (V_1 \setminus \{w\})$ of size at most $k-1$. Note that in this case, the sets V_0 and V_2 are unchanged, and the triple $(V_0, V_1 \setminus \{w\}, V_2)$ obviously makes a valid IF-partition for the graph $G - w$. On the other hand, to exclude the vertex w from the objective FVS, we move w from V_1 to V_2. First note that since the vertex w has at most one neighbor in V_2, the induced subgraph $G[V_2 \cup \{w\}]$ is still a forest. Moreover, since all children $w_1, \dots, w_t$ of w have degree 2 in the graph G and each w_i has a unique neighbor in the set V_2, after moving w from V_1 to V_2, all these degree-2 vertices $w_1, \dots, w_t$ have their both neighbors in the set $V_2 \cup \{w\}$. Therefore, these vertices $w_1, \dots, w_t$ now can be moved to the set V_0. In particular, the triple $(V_0 \cup \{w_1, \dots, w_t\}, V_1 \setminus \{w, w_1, \dots, w_t\}, V_2 \cup \{w\})$ is a valid IF-partition of the vertex set of the graph G. This recursive branching is implemented by the two recursive calls in step 5.1.

If we reach step 5.2, then the two conditions in step 5.1 do not hold. Therefore, in addition to conditions (1)–(3), the following two conditions also hold:

(4) the vertex w has no neighbor in V_2; and
(5) the vertex w has a unique child w_1 in the tree T.

By conditions (2), (4), and (5), the vertex w has degree exactly 2 in the graph G (and w is not the root of the tree T). Therefore, the vertices w_1 and w are two adjacent degree-2 vertices in the graph G. Observe that in this case, a cycle in the graph G contains the vertex w_1 *if and only if* it also contains the vertex w. Therefore, we can safely assume that the one of larger weight among w_1 and w is not in the objective FVS. If the larger weight vertex is w_1, then the first recursive call in step 5.2 is executed, which moves w_1 from set V_1 to set V_2 (note that the triple $(V_0, V_1 \setminus \{w_1\}, V_2 \cup \{w_1\})$ is a valid IF-partition of G because w_1 has a unique neighbor in V_2). If the larger weight vertex is w, then the second recursive call in step 5.2 is executed, which moves w from V_1 to V_2. Note that since both neighbors of the degree-2 vertex w_1 are in the set $V_2 \cup \{w\}$, after adding w to the set V_2, we can also move the vertex w_1 from V_1 to V_0. Thus, the triple $(V_0 \cup \{w_1\}, V_1 \setminus \{w, w_1\}, V_2 \cup \{w\})$ is a valid IF-partition of the graph G.

We also remark that by our assumption, the input graph G contains neither multiple edges nor self-loops. Moreover, the graph in each of the recursive calls in the algorithm is either the original G, or G with a vertex deleted. Therefore, the graph in each of the recursive calls in the algorithm also contains neither multiple edges nor self-loops.

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, V_0, V_1, V_2, k).

To analyze the running time, as in 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 on the value $\tau(k, V_0, V_1, V_2)$ that $T(k, V_0, V_1, V_2) \leqslant \max(1, 2^{\tau(k, V_0, V_1, V_2)})$. First of all, if $\tau(k, V_0, V_1, V_2) \leqslant 0$, then by step 1 of the algorithm, we have $T(k, V_0, V_1, V_2) = 1$.

First consider the branching steps, i.e., step 4.2 and step 5.1. In case step 4.2 of the algorithm is executed, we have recursively

$$T(k, V_0, V_1, V_2) \leqslant T\big(k-1, V_0, V_1 \setminus \{w\}, V_2\big) + T\big(k, V_0, V_1 \setminus \{w\}, V_2 \cup \{w\}\big). \tag{1}$$

Since

$$\begin{aligned}\tau_1 &= \tau\big(k-1, V_0, V_1 \setminus \{w\}, V_2\big)\\ &= (k-1) - \big(|V_0| - \#c(V_2) + 1\big)\\ &= \tau(k, V_0, V_1, V_2) - 1\\ &< \tau(k, V_0, V_1, V_2),\end{aligned}$$

and

$$\begin{aligned}\tau_2 &= \tau\big(k, V_0, V_1 \setminus \{w\}, V_2 \cup \{w\}\big)\\ &= k - \big(|V_0| - \#c\big(V_2 \cup \{w\}\big) + 1\big)\\ &\leqslant k - \big(|V_0| - \big(\#c(V_2) - 1\big) + 1\big)\\ &= \tau(k, V_0, V_1, V_2) - 1\\ &< \tau(k, V_0, V_1, V_2),\end{aligned}$$

where we have used the fact $\#c(V_2 \cup \{w\}) \leqslant \#c(V_2) - 1$ because in this case, we assume that the vertex w has two neighbors in two different trees in $G[V_2]$, therefore, adding w to V_2 merges at least two connected components in $G[V_2]$ and reduces the number of connected components by at least 1.

Therefore, by the inductive hypothesis, $T(k-1, V_0, V_1 \setminus \{w\}, V_2) \leqslant 2^{\tau_1}$, and $T(k, V_0, V_1 \setminus \{w\}, V_2 \cup \{w\}) \leqslant 2^{\tau_2}$. Combining these with Inequality (1), we get

$$\begin{aligned} T(k, V_0, V_1, V_2) &\leqslant T\big(k-1, V_0, V_1 \setminus \{w\}, V_2\big) + T\big(k, V_0, V_1 \setminus \{w\}, V_2 \cup \{w\}\big) \\ &\leqslant 2^{\tau_1} + 2^{\tau_2} \\ &\leqslant 2^{\tau(k, V_0, V_1, V_2) - 1} + 2^{\tau(k, V_0, V_1, V_2) - 1} \\ &= 2^{\tau(k, V_0, V_1, V_2)}. \end{aligned}$$

In conclusion, the induction goes through for step 4.2 of the algorithm.

Now we consider step 5.1, which is the least trivial case, and makes the major difference from the unweighted cases. Let $V_0' = V_0 \cup \{w_1, \ldots, w_t\}$, $V_1' = V_1 \setminus \{w, w_1, \ldots, w_t\}$, and $V_2' = V_2 \cup \{w\}$. The execution of step 5.1 gives the following inequality:

$$T(k, V_0, V_1, V_2) \leqslant T\big(k-1, V_0, V_1 \setminus \{w\}, V_2\big) + T\big(k, V_0', V_1', V_2'\big).$$

As we have shown above, by the inductive hypothesis, we have

$$T\big(k-1, V_0, V_1 \setminus \{w\}, V_2\big) \leqslant 2^{\tau(k, V_0, V_1, V_2) - 1}. \tag{2}$$

To estimate the value $T(k, V_0', V_1', V_2')$, first note that $|V_0'| = |V_0| + t$. Moreover, at this point, we must have either that the vertex w has a neighbor in V_2 or that the vertex w has more than one child in the tree T in $G[V_1]$.

If w has a neighbor in V_2, then adding w to V_2 will "attach" the vertex w to a connected component in $G[V_2]$. In consequence, the number of connected components in $G[V_2]$ will be equal to that in $G[V_2 \cup \{w\}]$ (recall that w has only one neighbor in V_2). In this case (note that $t \geqslant 1$), we have

$$\begin{aligned} \tau\big(k, V_0', V_1', V_2'\big) &= k - \big(|V_0'| - \#c\big(V_2'\big) + 1\big) \\ &= k - \big(\big(|V_0| + t\big) - \#c(V_2) + 1\big) \\ &\leqslant \tau(k, V_0, V_1, V_2) - 1. \end{aligned}$$

Now let us assume that the vertex w has no neighbor in V_2 but has more than one child in the tree T in $G[V_1]$ (i.e., $t \geqslant 2$). Then $|V_0'| = |V_0| + t \geqslant |V_0| + 2$. In this case, adding the vertex w to the set V_2 increases the number of connected components in $G[V_2]$ by 1 (since w has no neighbor in V_2, the vertex w will become a single-vertex connected component in the induced subgraph $G[V_2 \cup \{w\}]$). That is, $\#c(V_2 \cup \{w\}) = \#c(V_2) + 1$. Therefore,

$$\begin{aligned} \tau\big(k, V_0', V_1', V_2'\big) &= k - \big(|V_0'| - \#c\big(V_2'\big) + 1\big) \\ &= k - \big(\big(|V_0| + t\big) - \#c\big(V_2 \cup \{w\}\big) + 1\big) \\ &\leqslant k - \big(\big(|V_0| + 2\big) - \big(\#c(V_2) + 1\big) + 1\big) \\ &\leqslant \tau(k, V_0, V_1, V_2) - 1. \end{aligned}$$

In conclusion, in all cases in step 5.1, we will have $\tau(k, V_0', V_1', V_2') \leqslant \tau(k, V_0, V_1, V_2) - 1$. Therefore, now we can apply the induction and get

$$T\big(k, V_0', V_1', V_2'\big) \leqslant 2^{\tau(k, V_0', V_1', V_2')} \leqslant 2^{\tau(k, V_0, V_1, V_2) - 1}.$$

Combining this with the inequalities (1) and (2), we conclude that

$$T(k, V_0, V_1, V_2) \leqslant 2^{\tau(k, V_0, V_1, V_2)}$$

holds for the case of step 5.1.

We should also remark that it can be verified that for all non-branching recursive calls in the algorithm, i.e., steps 3, 4.1, and 5.2, the instance deficiency is never increased. In particular, if the first recursive call in step 5.2 is executed, then since the vertex w_1 has a unique neighbor in V_2, $\#c(V_2) = \#c(V_2 \cup \{w_1\})$. Thus,

$$\begin{aligned} \tau\big(k, V_0, V_1 \setminus \{w_1\}, V_2 \cup \{w_1\}\big) &= k - \big(|V_0| - \#c\big(V_2 \cup \{w_1\}\big) + 1\big) \\ &= k - \big(|V_0| - \#c(V_2) + 1\big) \\ &= \tau(k, V_0, V_1, V_2). \end{aligned}$$

If the second recursive call in step 5.2 is executed, then $\#c(V_2 \cup \{w\}) = \#c(V_2) + 1$ because w has no neighbor in V_2 and w will become a single-vertex connected component in the induced subgraph $G[V_2 \cup \{w\}]$. Therefore,

$$\begin{aligned}
&\tau\big(k, V_0 \cup \{w_1\}, V_1 \setminus \{w, w_1\}, V_2 \cup \{w\}\big) \\
&\quad = k - \big(|V_0 \cup \{w_1\}| - \#c\big(V_2 \cup \{w\}\big) + 1\big) \\
&\quad = k - \big(\big(|V_0| + 1\big) - \big(\#c(V_2) + 1\big) + 1\big) \\
&\quad = \tau(k, V_0, V_1, V_2).
\end{aligned}$$

Summarizing all the above discussions, we complete the inductive proof that the number of leaves in the search tree for the algorithm **W-Feedback**(G, V_0, V_1, V_2, k) is at most $2^{\tau(k,V_0,V_1,V_2)}$.

In the same way as in the proof for the unweighted case, we observe that along each root-leaf path in the search tree, the total number of executions of steps 1, 2, 3, 4, 4.1, 4.2, 5, 5.1, and 5.2 of the algorithm is $\mathcal{O}(n)$ because each of these steps either stops immediately, or reduces the size of the set V_1 by at least 1. Step 1 is only preformed in leaf nodes of the tree, and thus only adds $\mathcal{O}(n^2)$ time to the total. By similar arguments as the one used for the unweighted case, all steps except step 1 can be preformed in $\mathcal{O}(n)$ time. Therefore, the running time of the algorithm **W-Feedback**(G, V_0, V_1, V_2, k) is $\mathcal{O}(2^{\tau(k,V_0,V_1,V_2)}n^2)$. □

With Lemma 7, we can now proceed in 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 $|F'| \leqslant k$, 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 8. *The* WEIGHTED FVS REDUCTION *problem on an n-vertex graph is solvable in time $\mathcal{O}(5^k n^2)$.*

Proof. The proof proceeds similarly to the proof of Lemma 2. For the given FVS F of size $k + 1$ in the graph $G = (V, E)$, every FVS F' of size at most 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 $V \setminus F$ and a subset F_2 of j vertices in F, for some integer j, $0 \leqslant j \leqslant k$, where $(V \setminus F, F \setminus F_2)$ 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, $0 \leqslant j \leqslant k$, such that $(V \setminus F, F \setminus F_2)$ is a forest bipartition of the graph $G_0 = G - F_2$, and construct the minimum-weight FVS F_0 of G_0 satisfying $|F_0| \leqslant k - j$. Note that the forest bipartition $(V \setminus F, F \setminus F_2)$ of G_0 is in fact a special IF-partition (V_0, V_1, V_2) of G_0, where $V_0 = \emptyset$, $V_1 = V \setminus F$, and $V_2 = F \setminus F_2$. Therefore, by Lemma 7, a minimum-weight FVS F_0 of G_0 satisfying $|F_0| \leqslant k - j$ can be constructed in time

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

where we have used the fact $\#c(F \setminus F_2) \leqslant |F \setminus F_2| = k + 1 - j$. Now the proof proceeds exactly the same way as in Lemma 2, and concludes that the WEIGHTED FVS REDUCTION problem can be solved in time $\mathcal{O}(5^k n^2)$. □

Using Theorem 3 and Lemma 8, we obtain the main result of this paper.

Theorem 9. *The* WEIGHTED-FVS *problem on an n-vertex graph is solvable in time $\mathcal{O}(5^k kn^2)$.*

Proof. Let (G, k) be a given instance of the WEIGHTED-FVS problem. As we explained in the proof of Theorem 3, we can first construct, in time $\mathcal{O}(5^k kn^2)$, 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 8. □

Acknowledgment

The authors 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, T. Fujito, A 2-approximation algorithm for the undirected feedback vertex set problem, SIAM J. Discrete Math. 12 (3) (1999) 289–297.
[2] A. Becker, R. Bar-Yehuda, D. Geiger, Randomized algorithms for the loop cutset problem, J. Artif. Intell. Res. (JAIR) 12 (2000) 219–234.

[3] H. Bodlaender, On disjoint cycles, Internat. J. Found. Comput. Sci. 5 (1) (1994) 59–68.
[4] J. Chen, Y. Liu, S. Lu, An improved parameterized algorithm for the minimum node multiway cut problem, Algorithmica (2008), in press.
[5] J. Chen, Y. Liu, S. Lu, B. O'Sullivan, I. Razgon, A fixed-parameter algorithm for the directed feedback vertex set problem, in: Proc. 40th ACM Symp. on Theory of Computing, 2008, pp. 177–186.
[6] T. Cormen, C. Leiserson, R. Rivest, C. Stein, Introduction to Algorithms, second ed., MIT Press, McGraw–Hill Book Company, 2001.
[7] F. Dehne, M. Fellows, M. Langston, F. Rosamond, K. Stevens, An $O(2^{O(k)}n^3)$ fpt algorithm for the undirected feedback vertex set problem, in: COCOON, in: Lecture Notes in Comput. Sci., vol. 3595, Springer, 2005, pp. 859–869.
[8] R. Downey, M. Fellows, Fixed parameter tractability and completeness, in: Complexity Theory: Current Research, Cambridge University Press, 1992, pp. 191–225.
[9] R. Downey, M. Fellows, Parameterized Complexit, Springer, New York, 1999.
[10] P. Festa, P. Pardalos, M. Resende, Feedback set problems, in: Handbook of Combinatorial Optimization, Supplement vol. A, Kluwer Acad. Publ., Dordrecht, 1999, pp. 209–258.
[11] F. Fomin, S. Gaspers, A. Pyatkin, Finding a minimum feedback vertex set in time $O(1.7548^n)$, in: IWPEC, in: Lecture Notes in Comput. Sci., vol. 4169, Springer, 2006, pp. 184–191.
[12] J. Guo, J. Gramm, F. Hüffner, R. Niedermeier, S. Wernicke, Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization, J. Comput. System Sci. 72 (8) (2006) 1386–1396.
[13] I. Kanj, M. Pelsmajer, M. Schaefer, Parameterized algorithms for feedback vertex set, in: IWPEC, in: Lecture Notes in Comput. Sci., vol. 3162, Springer, 2004, pp. 235–247.
[14] R. Karp, Reducibility among combinatorial problems, in: Complexity of Computer Computations, Plenum Press, New York, 1972, pp. 85–103.
[15] V. Raman, S. Saurabh, C. Subramanian, Faster fixed parameter tractable algorithms for finding feedback vertex sets, ACM Trans. Algorithms 2 (3) (2006) 403–415.
[16] V. Raman, S. Saurabh, C. Subramanian, Faster fixed parameter tractable algorithms for undirected feedback vertex set, in: ISAAC 2002, in: Lecture Notes in Comput. Sci., vol. 2518, Springer, 2002, pp. 241–248.
[17] I. Razgon, Exact computation of maximum induced forest, in: SWAT, in: Lecture Notes in Comput. Sci., vol. 4059, Springer, 2006, pp. 160–171.
[18] B. Reed, K. Smith, A. Vetta, Finding odd cycle transversals, Oper. Res. Lett. 32 (4) (2004) 299–301.
[19] A. Silberschatz, P. Galvin, Operating System Concepts, fourth ed., Addison–Wesley, 1994.