A Naive Algorithm for Feedback Vertex Set [1]

Seminar Advanced Algorithms and Data Structures - Student Report

Manuel Nowack

November 10, 2018

1 Introduction

The feedback vertex set problem originates from the constraint satisfaction problem, which is a subject of intense research in artificial intelligence. For each instance of the constraint satisfaction problem one can define a constraint graph and it is well known that the problem can be solved in polynomial time when the constraint graph is a forest [3]. Therefore, one way to solve the constraint satisfaction problem is to first find a minimum feedback vertex set of the constraint graph, enumerate all possible assignments on them and then solve the remaining instance. We do not go into more details here because it is not related to the algorithm presented in this paper.

The feedback vertex set problem is NP-complete [4]. The aforementioned approach for solving the constraint satisfaction problem only makes sense when the feedback vertex set is fairly small. This motivates the study of parameterized algorithms for the feedback vertex set problem, i.e. algorithms that find a feedback vertex set of size at most k in time $f(k) \cdot n^{O(1)}$.

The algorithm presented in this report is not groundbreaking because parameterized algorithms have been known for decades. However it is interesting that such a naive algorithm has not been discovered earlier especially because it is faster than most other algorithms!

1.1 Feedback Vertex Set Problem

In this report all graphs are assumed to be undirected and simple. A graph G is given by its vertex set $V(G)$ and edge set $E(G)$, whose cardinalities will be denoted by n and m respectively.

Definition 1. *A set V' of vertices is a feedback vertex set of graph G if $G - V'$ is acyclic, i.e. a forest. Given a graph G and an integer k, the feedback vertex set problem asks whether G has a vertex set of at most k vertices.*

Definition 2. *(G, k, F) is an extended instance of the feedback vertex set problem, if $F \subseteq V(G)$ and a solution of the feedback vertex set problem can only be picked from $V(G) \backslash F$.*

1.2 Examples

Figure 1 shows a graph and feedback vertex sets (marked red) of different sizes. In this graph only a feedback vertex set of size two is optimal and there are multiple optimal feedback vertex sets.

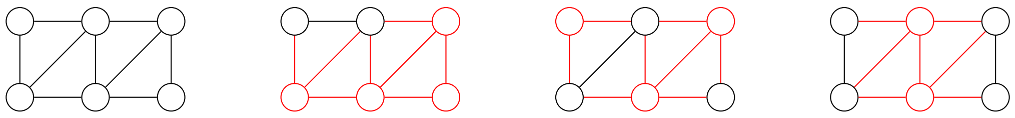

Figure 1: Feedback vertex sets in a graph

2 Algorithms

Before we look at the actual algorithm, we consider a truly trivial branching algorithm. This will give us a good understanding of the idea behind the algorithm and how its running time is analyzed. Then we look at the naive algorithm, that solves the problem in time $f(k) \cdot n^{O(1)}$, and afterwards we explore possibilities to further improve the running time.

2.1 Trivial Branching Algorithm

This algorithm simply goes through every vertex in the graph and recursively decides if it is part of a feedback vertex set.

The algorithm can be viewed as two parts. The first part covers the three base cases of the recursion.

1. For $k < 0$ there exists no solution and the algorithm returns "NO".
2. If G is already a forest, then the empty set is a trivial solution.
3. If there are no undecided vertices left, i.e. all vertices in G are also in F, there exists no solution and the algorithm returns "NO".

In the second part the algorithm picks an arbitrary undecided vertex v and branches on either including it in the solution V' or marking it "undeletable" (i.e. adding it to F). It does this by recursively solving the extended instance $(G - \{v\}, k - 1, F)$. If the recursive call returns a feedback vertex set, v and this feedback vertex set form a solution. If the recursive call returns "NO", v cannot be part of a feedback vertex set of size at most k. In this case v is added to F and the algorithm continues, i.e. recursively solves the extended instance $(G, k, F \cup \{v\})$.

Algorithm 1 trivial-fvs(G, k, F)

```
Require: a graph $G$, an integer $k$ and a set $F \subseteq V(G)$ inducing a forest
Ensure: a feedback vertex set $V' \subseteq V(G) \backslash F$ of size $\leq k$ or “NO”
  if $k < 0$ then
      return “NO”
  if $V(G)$ is a forest then
      return $\emptyset$
  if $V(G) \backslash F = \emptyset$ then
      return “NO”
  pick a vertex $v$ from $V(G) \backslash F$
  $X \leftarrow$ trivial-fvs($G - \{v\}, k - 1, F$)
  if $X$ is not “NO” then
      return $X \cup \{v\}$
  return trivial-fvs($G, k, F \cup \{v\}$)
```

Lemma 1. *Calling Algorithm 1* `trivial-fvs` *with* $(G, k, \emptyset)$ *solves the instance* (G, k) *of the feedback vertex set problem.*

Proof. The first two base cases are clearly correct. The third base case is correct because there are no more undecided vertices left and G is still not a forest. The correctness of the second part is also easy to see. If there is a feedback vertex set that contains v, then it is found after the first recursive call. Otherwise the second recursive call gives the correct answer because v is now ineligible for any feedback vertex set. □

Lemma 2. *Algorithm 1* `trivial-fvs` *can be implemented in* $\mathcal{O}(2^n)$ *time to decide whether a graph* G *has a feedback vertex set of size at most* k.

Proof. The execution of the algorithm can be described as a search tree in which each node corresponds to an extended instance of the problem. A leaf node of the search tree returns either a solution or “NO”. We say that a path from the root of the search tree to a leaf node is an *execution path.*

A single node can be processed in polynomial time. Thus our goal is to bound the number of nodes. Since the tree is binary, it suffices to bound its depth. Let us fix an arbitrary execution path in the search tree. Each non-leaf node in this execution path branches on exactly one vertex by putting it into V' or F. If no solution was found after branching on all n vertices, the algorithm terminates by returning “NO”. Thus the length of this execution path is at most n. This result in a total running time of $\mathcal{O}(2^n)$. □

2.2 Naive Algorithm

One might suspect that a vertex of a larger degree has a larger chance to be in a minimum feedback vertex set. This is in fact the main idea behind the naive algorithm. First it eliminates vertices with a very small degree and then it recursively branches on an undecided vertex with the largest degree.

The algorithm handles three base cases, which are slightly different from the ones in the trivial algorithm.

1. For $k < 0$ there exists no solution and the algorithm returns "NO".
2. If G contains no vertices, then the empty set is the only solution.
3. As we will see in Proposition 2, any solution is found before more than $3k$ vertices are added to F. Thus the algorithm can terminate after more than $3k$ vertices were added to F and return "NO" directly.

Next the algorithm performs some simple operations when the situation is clear. The algorithm covers the following three situations.

1. G contains a vertex v with degree less than two.
   Since v is not part of any cycle, it is ignored by any (optimal) solution and can be deleted.
2. G contains a vertex v that is part of a cycle, in which all vertices except v are marked as undeletable.
   Since a feedback vertex set makes the graph acyclic, v must be in the solution.
3. All vertices in G have degree two, i.e. all components are cycles.
   Since a feedback vertex set makes the graph acyclic, one vertex from every cycle, that is not marked as undeletable, is added to the solution. If the solution contains no more than k vertices, it is returned, otherwise "NO" is returned.

Finally the algorithm picks an undecided vertex with maximum degree[1] and branches on either including it in the solution V' or marking it "undeletable" (i.e. adding it to F). It does this using the same procedure as the trivial algorithm.

For the running time analysis we take the same approach as before and model the execution of the algorithm as a search tree. However this time each node corresponds to two extended instances of the problem, the *entry instance* and the *exit instance*. The entry instance of a node is the instance passed to it. The exit instance of a node is the one after the three situations have been exhaustively applied on the entry instance. If the algorithm branches on a vertex, one child node with entry instance $(G-\{v\}, k-1, F)$ is generated. If no solution is found in the child node, another child node with entry instance $(G, k, F \cup \{v\})$ is generated. A leaf node of the search tree returns either a feedback vertex set or "NO".

[1]The actual algorithm already picks a vertex with maximum degree before the third situation, but this is only syntactic sugar.

Algorithm 2 naive-fvs(G, k, F)

```
Require: a graph $G$, an integer $k$ and a set $F \subseteq V(G)$ inducing a forest
Ensure: a feedback vertex set $V' \subseteq V(G) \backslash F$ of size $\leq k$ or "NO"
  if $k < 0$ then
      return "NO"
  if $V(G) = \emptyset$ then
      return $\emptyset$
  if more than $3k$ vertices were added to $F$ already then
      return "NO"
  if a vertex $v \in V(G)$ has degree less than two then
      return naive-fvs$(G - \{v\}, k, F \backslash \{v\})$
  if a vertex $v \in V(G) \backslash F$ has two neighbors in the same component of $G[F]$ then
      $X \leftarrow$ naive-fvs$(G - \{v\}, k - 1, F)$
      return $X \cup \{v\}$
  pick a vertex $v$ from $V(G) \backslash F$ with the maximum degree
  if $d(v) = 2$ then
      $X \leftarrow \emptyset$
      while there is a cycle $C$ in $G$ do
          take any vertex $x$ in $C \backslash F$
          add $x$ to $X$ and delete it from $G$
      if $|X| \leq k$ then
          return $X$
      else
          return "NO"
  $X \leftarrow$ naive-fvs$(G - \{v\}, k - 1, F)$
  if $X$ is not "NO" then
      return $X \cup \{v\}$
  return naive-fvs$(G, k, F \cup \{v\})$
```

We say that a path from the root of the search tree to a leaf node is an execution path. Let F' denote all vertices moved into F in this execution path. Note that in this algorithm vertices can be deleted from F and hence $|F'|$ may be larger than $|F|$.

Proposition 1.

$$|V'| \geq \sum_{v \in F'} \frac{d^*(v) - 2}{d^*(v)}$$

where $d^*(v)$ denotes the degree of v at the moment it is moved into F.

We do not prove this proposition and refer to [1] instead for a full proof.

Proposition 2. *In an execution path that leads to a solution, $|F'| \leq 3|V'|$.*

Proof. The algorithm only moves a vertex into F when G has no vertices with degree less than three. Thus $d^*(v)$ must be larger or equal to three. Putting this and Proposition 1 together, we have

$$\begin{aligned} |V'| &\geq \sum_{v \in F'} \frac{d^*(v) - 2}{d^*(v)} \\ &\geq \sum_{v \in F'} \frac{1}{3} \\ &\geq \frac{|F'|}{3} \end{aligned}$$

□

Lemma 3. *Calling Algorithm 2 `naive-fvs` with $(G, k, \emptyset)$ solves the instance (G, k) of the feedback vertex set problem.*

Proof. The first two base cases are clearly correct. The correctness of the third base case follows from Proposition 2.

In the first situation, the vertex v is not part of any cycle and hence can be ignored by any optimal solution. Thus the recursive call returns a solution if and only if the new extended instance $(G - \{v\}, k, F \backslash \{v\})$ has a solution.

In the second situation, there is a cycle consisting of the vertex v and vertices in F and hence any solution has to contain v. Thus the recursive call returns a solution if and only if the new extended instance $(G - \{v\}, k - 1, F)$ has a solution.

In the third situation, all vertices in G have degree two because all vertices with degree less than two have already been removed in the first situation and v has the maximum degree. This means all components are cycles and in order to make G acyclic, it suffices to remove a vertex from each cycle. Thus G has a solution if and only if the number of removed vertices is smaller or equal to k.

The correctness of the final part is easy to see. If there is a feedback vertex set that contains v, then it is found after the first recursive call. Otherwise the other recursive call gives the correct answer because v is now ineligible for any feedback vertex set. □

Theorem 1. *Algorithm 2 `naive-fvs` can be implemented in $\mathcal{O}(16^k \cdot n^2)$ time to decide whether a graph G has a feedback vertex set of size at most k.*

Proof. The processing time of a node is bounded by transforming the entry instance into the exit instance, i.e. applying the three situations. The first situation can be processed in time $\mathcal{O}(n)$ and there are at most n recursive calls. The second situation can be processed in time $\mathcal{O}(n^2)$ and there are a constant amount of recursive calls because each child has at most one more vertex in F than its parent. The third situation can be processed in time $\mathcal{O}(n^2)$ too. A depth-first search can be used to find a cycle in $\mathcal{O}(n+m)$ time. Since every vertex has degree two, G has exactly n edges and we can reduce the time to $\mathcal{O}(n)$. There are less than n cycles in G. Thus a single node can be processed in $\mathcal{O}(n^2)$ time.

All that is left to do is bounding the number of nodes. Since the tree is binary, it suffices to bound its depth. Any execution path must terminate after it has put k vertices into V' or $3k$ vertices into F. Since each non-leaf puts exactly one vertex into V' or F, the length of any path is at most $4k$. This gives a total running time of $\mathcal{O}(2^{4k} \cdot n^2) = \mathcal{O}(16^k \cdot n^2)$. □

2.3 Improved Naive Algorithm

We can further improve the running time of the naive algorithm by exploiting a lemma from another paper [2].

Lemma 4. *Given a graph G and a set F of vertices such that every vertex in $V(G)\backslash F$ has degree at most three, there is a polynomial-time algorithm for finding a minimum set $V' \subseteq V(G)\backslash F$ such that $G - V'$ is a forest.*

We can use this to change the third situation in Algorithm 2 to the following:

```
if $d(v) \leq 3$ then
    invoke Lemma 4 to find a minimal solution $X$
    if $|X| \leq k$ then
        return $X$
    else
        return "NO"
```

As a result of this we can tighten the depth in the third base case of Algorithm 2 and terminate the algorithm after it has put $2k$ vertices into F by returning "NO" directly. We call this new algorithm `improved-naive-fvs`.

Theorem 2. *Algorithm `improved-naive-fvs` can be implemented in $\mathcal{O}(8^k \cdot n^{O(1)})$ time to decide whether a graph G has a feedback vertex set of size at most k.*

Proof. Removing the third situation does not affect correctness and the newly added part is clearly correct. Since vertices can now only be moved into F if their degree is greater than 3, i.e. $d^*(v) \geq 4$, the result of Proposition 2 improves to $|F'| \leq 2|V'|$. This proves the correctness of the tightened termination condition.

Consequently any execution path must terminate after it has put k vertices into V' or $2k$ vertices into F. Since each non-leaf puts exactly one vertex into V' or F, the length of any path is at most $3k$. This gives a total running time of $\mathcal{O}(2^{3k} \cdot n^{O(1)}) = \mathcal{O}(8^k \cdot n^{O(1)})$. $\square$

References

[1] Yixin Cao. A Naive Algorithm for Feedback Vertex Set . In Raimund Seidel, editor, *1st Symposium on Simplicity in Algorithms (SOSA 2018)*, volume 61 of *OpenAccess Series in Informatics (OASIcs)*, pages 1:1–1:9, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

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

[3] Eugene C. Freuder. A sufficient condition for backtrack-free search. *J. ACM*, 29(1):24–32, January 1982.

[4] John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is np-complete. *Journal of Computer and System Sciences*, 20(2):219 – 230, 1980.