A Naive Algorithm for Feedback Vertex Set

Problem description

Given an undirected simple graph on n vertices and an integer k, the feedback vertex set problem asks for the deletion of at most k vertices to make the graph acyclic. We show that a greedy branching algorithm, which always branches on an undecided vertex with the largest degree, runs in single-exponential time, i.e., $O(c^k \cdot n^2)$ for some constant c.

What are we going to store?

To solve the problem, we will use the bruteforce. We will store:

G: the graph

k: the upper bound on the size of the feedback vertex set that we need to find

F: the set of vertices of G that we are **NOT** going to use in the feedback vertex set (they form a forest)

Simple cases

If k is < 0, then the answer is “NO”.

if G has no vertices, then empty set is the answer.

If there is a vertex with degree at most 1, then we can delete it from F and from G, because this vertex is not a part of any cycle (and proceed recursively to G-{v},k,F\{v})

If there is a vertex that has two neighbors in the same component of F, then we should add it to the answer set, because this vertex can’t be added to F (so we should proceed recursively to G-{v},k-1,F)

Main case

Consider the vertex v with the largest degree.

If degree of this vertex is at most 2, then our graph is a union of cycles and paths and we can find the answer greedily.

Otherwise, we need to branch to two cases:

1) v is in the answer → branch to (G-{v}, k-1, F), and return the answer + {v}, if it exists
2) v is not in the answer → branch to (G, k, F + {v}).

Algorithm naive-fvs(G, k, F)

INPUT: a graph G, an integer k, and a set $F \subseteq V(G)$ inducing a forest.

OUTPUT: a feedback vertex set $V_- \subseteq V(G) \setminus F$ of size $\leqslant k$ or "NO."

```
0.    if $k < 0$ then return "NO"; if $V(G) = \emptyset$ then return $\emptyset$;
1.    if a vertex $v$ has degree less than two then
          return naive-fvs$(G - \{v\}, k, F \setminus \{v\})$;
2.    if a vertex $v \in V(G) \setminus F$ has two neighbors in the same component of $G[F]$ then
          $X \leftarrow$ naive-fvs$(G - \{v\}, k - 1, F)$;
          return $X \cup \{v\}$;
3.    pick a vertex $v$ from $V(G) \setminus F$ with the maximum degree;
4.    if $d(v) = 2$ then
4.1.      $X \leftarrow \emptyset$;
4.2.      while there is a cycle $C$ in $G$ then
              take any vertex $x$ in $C \setminus F$;
              add $x$ to $X$ and delete it from $G$;
4.3.      if $|X| \leqslant k$ then return $X$; else return "NO";
5.    $X \leftarrow$ naive-fvs$(G - \{v\}, k - 1, F)$; \\case 1: $v \in V_-$.
      if $X$ is not "NO" then return $X \cup \{v\}$;
6.    return naive-fvs$(G, k, F \cup \{v\})$.  \\case 2: $v \notin V_-$.
```

Time Complexity

Our goal is to estimate the number of nodes in the search tree.

For each branch, we are adding a new vertex to the answer |A| or to the |F|.

|A| is at most k, so our goal is to estimate |F|.

Invariant 1: Degree of no vertex can increase

Invariant 2: When a branch happens (in step 5 or 6 of the previous code), all vertices have degrees at least 2.

For each vertex v, let d*(v) be the degree of vertex v at the moment it is deleted from the graph (put to A or to F).

Initial degree is at least d*(v), and also d*(v) is at least 3 when it put into F.

Consider the sequence of feedback set vertices (vertices from |A|) in order of deletion

$x_1, x_2, \ldots, x_{|A|}$

We say that the decrements of degree of a vertex v from F from $d^*(v)$ to 2 are **effective (ED)**.

Effective decrement is incurred by $x_i \in A$, if it happens between deleting xi and x_{i+1} or after deletion of $x_{|A|}$ if $i=|A|$

Let g(u, i) be the number of effective decrements of u incurred by x_i.

Proposition 2.2: if g(u, x_i) > 0, then d*(u) is at least d*(x_i)

Because if g(u,x_i) > 0 then u was added to F before x_i was added to A, it means that our greedy algorithm picked u first, so d*(u) is at least d*(x_i)

Lemma 2.3: for vertex x_i in A, sum g(u, x_i) is at most d*(x_i)

When you delete x_i from the graph it, consider its d*(x_i) neighbors at this moment.

If there are **d** vertices with degree two among them, may happen at most d*(x_i) - d effective decrements before you will proceed to the next node of your recursion tree (because vertices of degree 2 already have degree 2 so they won’t be affected).

But after that, you will have **d** vertices with degree one (note that they will be the only vertices with degree one in the graph at the moment because of the invariant 2).

In the next steps of your recursion, you will delete these vertices from the graph. You may have two cases

1) The only neighbor of the deleted vertex has degree 1, in that case the number of vertices of degree 1 will stay the same and no ED will happen
2) The only neighbor of the deleted vertex has degree more than one, in that case, the number of vertices of degree 1 will decrease, but one ED may happen

It means that sum g(u,x_i) is at most d*(x_i) - d + d = d*(x_i). Thus, lemma 2.3 is proved.

Main lemma: |F| is at most 3|A|

We will prove it on the next slide, but if it is correct it means that we can exit our tree if |F| > 3|A|, thus the depth of the tree is at most 4k+1, and we can proceed each vertex in O(n^2) thus our sol works in O(2^(4k) \cdot n^2) →

O(16^k \cdot n^2)

Lemma 2.4. *In an execution path that leads to a solution, $|F'| \leqslant 3|V_-|$.*

Proof. Since this execution path leads to a solution, all vertices must be deleted from the graph at the end of the path. In the algorithm, a vertex in F can only be deleted from the graph in step 1, when the degree of the vertex has to be one or zero. On the other hand, $d^*(u) \geqslant 3$. Thus, all the $d^*(u) - 2$ effective decrements must have happened on this vertex, i.e., $\sum_{v \in V_-} \delta(u, v) = d^*(u) - 2$. Putting everything together, we have

$$\begin{aligned}
|V_-| = \sum_{v \in V_-} 1 &= \sum_{v \in V_-} \frac{d^*(v)}{d^*(v)} \\
&\geqslant \sum_{v \in V_-} \frac{1}{d^*(v)} \sum_{u \in F'} \delta(u, v) && \text{(Lemma 2.3)} \\
&= \sum_{v \in V_-} \sum_{u \in F'} \frac{\delta(u, v)}{d^*(v)} \\
&\geqslant \sum_{v \in V_-} \sum_{u \in F'} \frac{\delta(u, v)}{d^*(u)} && \text{(Proposition 2.2)} \\
&= \sum_{u \in F'} \frac{1}{d^*(u)} \sum_{v \in V_-} \delta(u, v) \\
&= \sum_{u \in F'} \frac{d^*(u) - 2}{d^*(u)} \\
&\geqslant \sum_{u \in F'} \frac{1}{3} && (d^*(u) \geqslant 3) \\
&= \frac{|F'|}{3},
\end{aligned}$$

and the proof is complete. □

Further Optimization

There is a **well-known** fact: if all degrees are at most 3, then we can find the feedback vertex set in the polynomial time. (by reduction to the maximum matching of a linearly represented 2-polymatroid).

If we will use this fact and use this algorithm in our brute when degrees are at most 3, then |A| is at most 2|F| and we solve the problem in O(n^O(1) \prod 8^k).