Feedback Vertex Set via Iterative Compression

1 Iterative Compression

We show how to use the technique of iterative compression to design an FPT-algorithm for the Feedback Vertex Set problem. The problem setting is as follows: we are given a graph $G = (V, E)$, an integer parameter k and a set $S \subseteq V$ of size $k+1$ such that $G-S$ is a forest; the question is to decide whether G has a feedback vertex set of size at most k.

Before we proceed any further, we ought to clarify some points. Firstly, why should such a problem setting be helpful at all? It turns out that the solution S imposes some additional structure on the remaining graph that can be exploited. Secondly, given a solution to this problem how do we use it to solve the actual Feedback Vertex Set problem? To answer this, let us assume that we have an FPT-algorithm $\mathscr{A}$ that solves the above variant of the problem in $f(k) \cdot n^c$ time, where $n = |V|$. Order the vertices of G and consider the graph $G[v_1, \ldots, v_{k+3}]$ induced by the first $k+3$ vertices. This graph has a feedback vertex set of size $k+1$, trivially. Invoke algorithm $\mathscr{A}$ to test whether it has a feedback vertex set of size k. If no, then G cannot have a feedback vertex set of size k, and we are done (Why?). If yes, then let S' be a solution of size at most k. Now consider the graph $G[v_1, \ldots, v_{k+4}]$ on the first $k+4$ vertices. This graph has a feedback vertex set of size $k+1$: $S' \cup v_{k+4}$. Now we have a situation that is identical to the one we started out with. Invoke algorithm $\mathscr{A}$ on the graph $G[v_1, \ldots, v_{k+4}]$ and repeat as before. The term "iterative compression" should make more sense now: we iteratively include new vertices, obtain a solution of size $k+1$ in the process, and try to compress it. If we fail to compress the solution at some stage, it implies that the original instance is a no-instance.

2 The Forest Bipartition Problem

Suppose that we are given as input a graph $G = (V, E)$, an integer k as parameter and a solution S of size $k+1$. The goal is to decide whether there exists a solution of size at most k. Let us assume that there exists such a solution F. Now how does this solution interact with the rest of the graph? Presumably F includes some vertices from the solution S and the remaining from $G - S$. To capture this situation, consider all possible ways of partitioning the set S into $Y \uplus N$, where Y represents the vertices that are in the purported solution F and N represents the vertices *not* in F. What can we now say about the graph $G[N]$? It must be a forest! If for some partition $G[N]$ is not a forest, then we can disregard this. We therefore have a new problem setting:

> Forest Bipartition: Given a graph $G = (V, E)$, a partition of $V = V_1 \cup V_2$, and an integer k, such that $G[V_i]$ is a forest for $i \in \{1, 2\}$, does G have a feedback vertex set of size at most k contained entirely in V_1?

Here V_1 plays the role of the set $V \setminus S$ above and $V_2 = N$. Since $|S| = k+1$, there are only 2^{k+1} ways of partitioning the set S and each of these is an instance of the Forest Bipartition problem. Therefore if we can solve each of these instances in FPT-time we are still safe.

2.1 Reduction Rules

The following reduction rules are applied to simplify the problem instance.

Reduction Rule 1 *If $u \in V_1$ has two neighbors in the same tree in $G[V_2]$, include u in the solution and set $k = k - 1$.*

The soundness of this rule is easy to verify. If $u \in V_1$ has two neighbors in the same connected component of $G[V_2]$, then there exists a cycle such that the only vertex it includes from V_1 is u. Since our solution set must be from V_1, we have to choose u.

Before we state the next reduction rule, we introduce the *short circuiting* operation. Let w be a degree-two vertex with neighbors u and v in a graph G. We say that the graph G' is obtained by short circuiting the degree-two vertex w, if G' is obtained by first deleting w and adding a new edge between u and v. Note that short circuiting might introduce loops and parallel edges.

Reduction Rule 2 *If $w \in V_1$ has degree at most one in G, delete w. If $w \in V_1$ is degree at most one in $G[V_1]$ and has exactly one neighbor in V_2, short circuit w. The parameter remains unchanged.*

Again it is easy to verify the soundness of this rule. If a vertex $w \in V_1$ has degree at most one in G, it cannot be part of any cycle and can be safely deleted. If it has exactly one neighbor each in V_1 and V_2 then every cycle that passes through w also passes through its unique neighbor in V_1 and one may always choose the neighbor instead of w in a size-k solution.

2.2 An FPT-Algorithm

Consider the following algorithm for the Forest Bipartition problem in Figure 1. The reduction rules discussed before are made use of in the algorithm. The correctness follows from the discussion on the soundness of the reduction rules and we only have to verify the running time.

The recursive execution of the algorithm can be described by a serach tree $\mathcal{T}$. We first count the number of leaves in the search tree. Note that the only step in which the algorithm actually branches is in Step 4. Let $T(k, l)$ denote the 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 components in the forest $G[V_2]$. In Step 4, the vertex w has neighbors in two different trees of $G[V_2]$. If vertex w is included in the solution then this recursive call generates a serach tree with at most $T(k-1, l)$ leaves; if w is excluded from the solution, the recursive call generates a search tree with at most $T(k, l-1)$ leaves. This follows because when w is included in V_2, it *merges* two components of $G[V_2]$ into one thereby reducing the number of connected components.

This gives us the following recurrence:

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

Show that $T(k, l) = O(2^{k+l})$! Finally observe that each root-leaf path in the search tree is bounded by $O(n)$, because when the algorithm does not branch (Steps 3 and 5), it reduces the size of the graph. The operations in Steps 3 and 5 take polynomial time and hence the total time taken by the algorithm is $O(2^{k+l} \cdot \text{poly}(n))$. Since $l \leq k+1$, the running time of the algorithm is $O(4^k \cdot \text{poly}(n))$.

Now to solve the Feedback Vertex Set problem, we need to solve the compression version n times and the compression version reduces to 2^{k+1} instances of the Forest Bipartition problem. Together this gives:

Theorem 1 *The* Feedback Vertex Set *can be solved in time $O(8^k \cdot \text{poly}(n))$.*

As an exercise, show that the running time is actually $O(5^k \cdot \text{poly}(n))$.

Feedback(G, V_1, V_2, k)

Input: A graph $G = (V, E)$ with a forest bipartition $V_1 \uplus V_2$ and an integer parameter k.

Output: A feedback vertex set $F \subseteq V_1$ of size at most k if one exists; or report NO (that is, no such feedback vertex set exists).

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** $w \in V_1$ has two neighbors in the same tree of $G[V_2]$
   $F' =$ **Feedback**$(G - w, V_1 - w, V_2, k - 1)$;
   if $F' =$ NO **then** return NO; **else** return $F' + w$;
4. **if** $w \in V_1$ has at least two neighbors in V_2
   $F_1 =$ **Feedback**$(G - w, V_1 - w, V_2, k - 1)$;
   $F_2 =$ **Feedback**$(G, V_1 - w, V_2 \cup w, k)$;
   if $F_1 \neq$ NO **then** return $F_1 \cup w$; **else if** $F_2 \neq$ NO **then** return F_2;
   else return NO;
5. **else** pick any vertex w that has degree at most one in $G[V_1]$;
   if w has degree at most one in G
   then return **Feedback**$(G - w, V_1 - w, V_2, k)$;
   else let G_w be the graph obtained by short circuiting w;
   return **Feedback**$(G_w, V_1 - w, V_2, k)$;

Figure 1: Algorithm for the Forest Bipartition problem.