An $O(2^{O(k)n^3})$ FPT Algorithm for the Undirected Feedback Vertex Set Problem

Paper by Frank Dehne, Michael Fellows, Michael Langston, Frances Rosamond and Kim Stevens

Speaker: Vered Cohen
Outline

- Iterative Compression
- Feedback Vertex Set Problem
- An FPT Algorithm for FVS Solution Compression
Iterative Compression

- Introduced in 2004 by Reed, Smith and Vetta.

**Core Idea:**

- Inductive approach: Compute a solution for a problem instance using the information provided by a solution for a smaller instance.

**In terms of a minimization problem on graphs:**

- Compute a solution $X$ for a problem instance $(G, k)$ using the information provided by a solution $X_0$ for a subinstance $(G - v, k)$. 
General Iterative Compression Scheme

Let $V(G) = \{v_1, \ldots, v_n\}$, $G_i = G[v_1, \ldots, v_i]$ for all $i \in \{1, \ldots, n\}$.

$S_1 = \text{solution set for } G_1$

For $i := 1 \ldots n - 1$:

- If $S_i$ is a solution set for $G_{i+1}$
  
  then $S_{i+1} := S_i$

- else find a solution for $S_{i+1}$ based on $S_i$

If $|S_{i+1}| > k$:

- Use an FPT algorithm $A$ to compute a $k$-solution-set $S_{i+1}$ for $G_{i+1}$,
  or return ‘NO’ and halt if it does not exist.

Return ‘YES’.
What kind of problems is this approach suitable?

For any minimization problem on instances $G$, with integer objective value, where we can construct a sequence $G_1, \ldots, G_n$ of polynomial length with $G_n = G$ and:

1. A $k$-solution for $G_1$ exists and can be found in polynomial time.
2. If $G_i$ has a $k$-solution, then $G_{i+1}$ has a $(k+1)$-solution, which can be found in polynomial time.
3. If $G_{i+1}$ has a $k$-solution, then $G_i$ has a $k$-solution.
4. If a $(k+1)$-solution $S$ for $G_{i+1}$ is given, then there is an FPT algorithm for parameter $k$ that decides whether $G_{i+1}$ has a $k$-solution (The compression step).
Iterative Compression is suitable for Vertex Cover

1. A k-solution for $G_1$ exists and can be found in polynomial time.

2. If $G_i$ has a k-solution, then $G_{i+1}$ has a (k+1)-solution, which can be found in polynomial time.

3. If $G_{i+1}$ has a k-solution, then $G_i$ has a k-solution.

4. If a (k+1)-solution $S$ for $G_{i+1}$ is given, then there is an FPT algorithm for parameter k that decides whether $G_{i+1}$ has a k-solution (The compression step).
Outline

• Iterative Compression

• Feedback Vertex Set Problem

• An FPT Algorithm for FVS Solution Compression
Undirected Feedback Vertex Set

**Input:** An undirected graph $G=(V,E)$ and a positive integer $k$.

**Output:** Is there a subset $S \subseteq V$ with $|S| \leq k$, such that $G - S$ is acyclic?

In other words, a feedback vertex set contains at least one vertex of any cycle in the graph.
Undirected FVS - Example
Undirected FVS - Example
Undirected FVS - Example
Undirected FVS - Example
Undirected FVS - Example
Undirected FVS - Example
Undirected FVS - Example
Undirected FVS - Example
Applications of FVS

- In operating systems – deadlock recovery
- VLSI chip design
- Various contexts in computational biology (e.g. Genome Assembly)
Generalization of FVS

Input: An undirected **multigraph** $G=(V,E)$ (i.e., loops and multiple edges are allowed), a **forbidden** subset $U \subseteq V$ of vertices, a positive integer $k$

Parameter: $k$

Output: Is there a subset $S \subseteq V - U$, with $|S| \leq k$, such that $G - S$ is acyclic?
Generalized FVS - Example

Forbidden vertex
Generalized FVS - Example

Forbidden vertex
Generalized FVS - Example

Forbidden vertex
Generalized FVS - Example
Generalized FVS - Example
Outline

• Iterative Compression

• Feedback Vertex Set Problem

• An FPT Algorithm for FVS Solution Compression
Iterative Compression Applied to FVS

1. A k-solution for $G_1$ exists and can be found in polynomial time. ($S_1 = \emptyset$, or $S_1 = \{v_1\}$)

2. If $G_i$ has a k-solution, then $G_{i+1}$ has a $(k+1)$-solution, which can be found in polynomial time.

3. If $G_{i+1}$ has a k-solution, then $G_i$ has a k-solution.

4. If a $(k+1)$-solution $S$ for $G_{i+1}$ is given, then there is an FPT algorithm for parameter k that decides whether $G_{i+1}$ has a k-solution (The compression step).
General Iterative Compression Scheme

Let $V(G) = \{v_1, \ldots, v_n\}$, $G_i = G[v_1, \ldots, v_i]$ for all $i \in \{1, \ldots, n\}$.

$S_1 = \{v_1\}$ (solution set for $G_1$)

For $i := 1 \ldots n - 1$:

- If $S_i$ is a solution set for $G_{i+1}$
  - then $S_{i+1} := S_i$
  - else find a solution for $S_{i+1}$ based on $S_i$

- If $|S_{i+1}| > k$:
  - Use an FPT algorithm $A$ to compute a $k$-solution-set $S_{i+1}$ for $G_{i+1}$,
  - or return ‘NO’ and halt if it does not exist.

Return ‘YES’.
Solution Compression for Feedback Vertex Set

**Input:** An undirected multigraph $G=(V,E)$ (loops and multiple edges are allowed) 
a forbidden subset $U \subseteq V$ of vertices 
a solution set $S \subseteq V - U$ such that $G - S$ is acyclic, where $|S| = k+1$

**Parameter:** $k$

**Output:** Either: (1) a solution set $S'$ of size $k$, 
(2) NO (i.e., no solution of size $k$ is possible)
Reduction Rules

We will use the following reduction rules to simplify an instance of the problem.

Recall that some vertices (U) may be forbidden to belong to a solution set.
Rule 1: The Degree One Rule
If $v$ is a vertex (forbidden or not) of degree 1 in $G$, then delete $v$.
The parameter $k$ is unchanged.
Rule 2: The Degree Two Rule

If $v$ is a vertex (forbidden or not) of degree 2 in $G$, with neighbors $a$ and $b$ (allowing possibly $a = b$), then modify $G$ by replacing $v$ and its two incident edges with a single edge between $a$ and $b$ (or loop on $a = b$). The parameter $k$ is unchanged.
Rule 2: The Degree Two Rule

If \( v \) is a vertex (forbidden or not) of degree 2 in \( G \), with neighbors \( a \) and \( b \) (allowing possibly \( a = b \)), then modify \( G \) by replacing \( v \) and its two incident edges with a single edge between \( a \) and \( b \) (or loop on \( a = b \)). The parameter \( k \) is unchanged.
Rule 3: Annotation Contraction

If $u$ and $v$ are adjacent forbidden vertices (that is, $u, v \in U$) then contract one of the edges between $u$ and $v$.

The parameter $k$ is unchanged.
Rule 4: The Loop Rules

If there is a loop on a forbidden vertex $v$ then answer NO.

If there is a loop on an unfornbidden vertex $v \in V - U$ then take $v$ into the solution set, and reduce to the instance $(G - v, U, k-1)$.

No Solution!
Rule 5: Multiedge Reduction

If there are more than two edges between $u$ and $v$ (forbidden or not) then delete all but two of these.

The parameter $k$ is unchanged.
**Rule 6: Multiedge Selection**

If there is a forbidden vertex \( u \) that is connected by two edges to an unforbidden vertex \( v \), then take \( v \) into the solution set, that is, reduce to the instance \((G - v, U, k-1)\).
Reduction Rules - Summary

In time $O(n)$ we can determine if any of the above reduction rules can be applied to a problem instance.

Note that applications of the rules may cascade.

We say that an instance is reduced if none of the reduction rules can be applied.
Algorithm for Solution Compression for FVS

Input: An undirected multigraph $G=(V,E)$,
a forbidden subset $U \subseteq V$ of vertices
a solution set $S \subseteq V - U$ of size $k+1$

Parameter: $k$

Output: Either: (1) a solution set $S'$ of size $k$,
(2) NO (i.e., no solution of size $k$ is possible)
Algorithm for Solution Compression for FVS

- **Step 1**
  Branch on all $2^{k+1}$ subsets of $S$.
  A subset $A \subseteq S$ represents the search for a size $k$ solution $S'$ that includes any of the vertices of $S - A = A'$. Thus, in the instance $(G', U', k')$ that represents this branch:
  1. the vertices of $A$ are deleted,
  2. the vertices of $A'$ are forbidden,
  3. $k' = k - |A|,$
  4. the instance is further reduced according to Reduction Rules.
Algorithm for Solution Compression for FVS

- **Step 1**

  For the reduced instance \((G', U', k')\) considered on any of these \(2^{k+1}\) branches of Step 1, we have either:

  1. \(|V' - U'| \leq 4k\), or
  2. we can immediately determine that the answer is NO.

(Proof later)
Algorithm for Solution Compression for FVS

- **Step 2**

  On each branch of Step 1, *exhaustively analyze* the resulting reduced instance by checking each \(k\)'-element subset of the unforbidden vertices to see if any provides a solution.

  This step requires checking at most \(\binom{4k}{k} \approx (9.4815)^k\) subsets.

**Total running time of our algorithm is** \(O(18.963^k n^2)\).

Later: a refined version that runs in time \(O(10.567^k n^2)\).
Several Definitions

• \( A \subseteq S \)

• \( A' = S - A \)

• The instance graph \( G' \) on the A-branch consists of two sets of vertices:
  
  1. The (now) forbidden vertices of \( A' \), where \( |A'| \leq k+1 \)
  
  2. The other vertices, which we denote \( F \).
     Some of these may be forbidden.
Lemma 1

The subgraph \(<F>\) induced by F is acyclic.

**Proof:** Otherwise S would not be a solution for G.
Lemma 2

Each leaf \( l \) of the forest \( F \) is adjacent to at least two distinct vertices in \( A' \).

**Proof:** In view of Lemma 1 and Reduction Rules 1 and 2 (Degree One Rule and Degree Two Rule), there must be at least two edges connecting \( l \) to vertices in \( A' \).

Reduction Rule 6 (Multiedge Selection Rule) would apply if \( l \) were connected to only one vertex of \( A' \).
Few Notations

The vertices of the forest $F$ can be partitioned into three sets:

1. $L$ – the leaves of $F$

2. $J$ – the vertices of degree 2 in the forest subgraph $F$
   (subdivision vertices of $F$)

3. $B$ – the vertices of degree at least 3 in the subgraph $F$
   (branch vertices of $F$)
Lemma 3

Each vertex $j \in J$ is connected to at least one vertex of $A'$.

Proof: Otherwise, in view of Lemma 1 ($F$ is acyclic), the Degree Two Rule would apply.
Definition 1 – path-matching

Let F be a forest with vertex set partitioned into the three sets:
1. the leaves L,
2. the subdivision vertices J, and
3. the branch vertices B of F.

A path-matching of the J-vertices of F of size r consists of:
1. r mutually disjoint 2-element subsets \( \{x_i, y_i\} \subseteq J, 1 \leq i \leq r \).
2. For each i, 1 \leq i \leq r, a path \( p_i \) in F from \( x_i \) to \( y_i \), subject to the requirement that for i \neq j, the paths \( p_i \) and \( p_j \) are vertex disjoint.
Definition 2 – $\pi(F)$

The potential $\pi(F)$ of the forest $F$ is defined to be

the sum of the number of leaves $|L|$ of $F$ and

the size of a maximum path-matching of the $J$-vertices.
Definition 2 – $\pi(F)$

The potential $\pi(F)$ of the forest $F$ is defined to be the sum of the number of leaves $|L|$ of $F$ and the size of a maximum path-matching of the $J$-vertices.
Definition 2 – $\pi(F)$

The potential $\pi(F)$ of the forest $F$ is defined to be the sum of the number of leaves $|L|$ of $F$ and the size of a maximum path-matching of the J-vertices.

$\pi(F) = 11 + 3 = 14$
Lemma 4

Suppose that for the reduced instance \((G', U', k')\) with vertex set partitioned into \(A'\) and \(F\) (as above) we have \(\pi(F) \geq k' + |A'|\). Then the answer for this instance is NO.

Proof:

- There's a set \(P\) of at least \(k' + |A'|\) paths in \(G'\) that begin and end at vertices of \(A'\), and that are pairwise internally vertex-disjoint.
- Assuming there's a set \(S'\) (FVS for \(G'\)), \(|S'| \leq k, S' \cap A' = \emptyset\).
Lemma 4 (cont.)

If $\pi(F) \geq k' + |A'|$ then the answer for this instance is NO.

Proof (cont.):

• Let $P' \subseteq P$ denote the subset of paths that are disjoint from $S'$. Since the paths in $P$ are pairwise internally vertex-disjoint, so that any vertex of $S'$ can hit at most one of them, then $|P| - |P'| \leq k'$ (or: $|P'| \geq |P| - k'$).

• $|P'| \geq |P| - k' \geq k' + |A'| - k' \geq |A'| \implies |P'| \geq |A'|$

• *Contradiction:* the vertices of $A'$ together with the paths in $P'$ must form a cycle disjoint from $S'$!
Lemma 5

For any forest $F$ on $m$ vertices, $\pi(F) \geq (m+1)/2$. 

(Proof later)
Lemma 6

If on the branch (of Step 1 of the algorithm) corresponding to \( A \subseteq S \) we have a reduced instance \((G', U', k')\) where the vertices of \( G' \) are partitioned into \( A' \) and \( F \) (as defined above), and where \(|F| \geq 4k+1\), then this is a NO-instance.

Proof: By Lemma 5 (\(|F|=m, \pi(F) \geq (m+1)/2\)) we get \(\pi(F) \geq 2k+1\). The rest follows by Lemma 4 (if: \(\pi(F) \geq k'+|A'|\) then: NO), since \(|A'| \leq k+1\) and \(k' \leq k\).
Proof of Lemma 5

Lemma 5: For any forest $F$ on $m$ vertices, $\pi(F) \geq (m+1)/2$.

Two parts to that argument:
1. For trees of maximum degree 3 (by structural induction)
2. For arbitrary trees (by minimum counterexample)
Proof of Lemma 5 (cont.)

For any tree of maximum degree 3 on $m$ vertices, $\pi(T) \geq \frac{(m+1)}{2}$

- We induct the structure of such trees.
- Each such tree $T$ is considered to be rooted at a vertex $r$:
  1. $r$ is a leaf of $T$
  2. $r$ has degree 2 in $T$

- Those trees are generated by two operations:
  1. $x(T)$ (extension of $T$), applied to either type 1 or 2. 
     Adds a new vertex $r'$ to $r$, which becomes the new root.
  2. $T_1 \oplus T_2$ (join), applied only when both trees of type 1. 
     The two roots of the trees are identified, resulting in a rooted tree of type 2.
Proof of Lemma 5 (cont.)

For any tree of maximum degree 3 on $m$ vertices, $\pi(T) \geq \frac{(m+1)}{2}$
Proof of Lemma 5 (cont.)

For any tree of maximum degree 3 on \( m \) vertices, \( \pi(T) \geq (m+1)/2 \)

Induction hypothesis. One of the following claims holds:

1. \(|J|\) is even and the J-vertices of T admit a perfect path-matching.

2. \(|J|\) is odd and the J-vertices can be path-matched in T with the exception of one vertex \( u \in J \), and furthermore the path-matching can be accomplished so that there is a path from \( u \) to the root \( r \) that is disjoint from the paths in T that realize the path-matching of J.
Proof of Lemma 5 (cont.)

For any tree of maximum degree 3 on $m$ vertices, $\pi(T) \geq \frac{(m+1)}{2}$

Case 2 of induction hypothesis:

$|J|$ is odd and the $J$-vertices can be path-matched in $T$ with the exception of one vertex $u \in J$, and furthermore the path-matching can be accomplished so that there is a path from $u$ to the root $r$ that is disjoint from the paths in $T$ that realize the path-matching of $J$. 

$|J| = 5$, odd
Proof of Lemma 5 (cont.)

For any tree of maximum degree 3 on $m$ vertices, $\pi(T) \geq (m+1)/2$
Proof of Lemma 5 (cont.)

For any tree of maximum degree 3 on $m$ vertices, $\pi(T) \geq (m+1)/2$

It follows that there can be at most one unmatched $J$-vertex in a maximum path-matching of $J$ in a tree $T$ of maximum degree 3. Since $|B| = |L| - 2$ and therefore

$$|T| = m = |L| + |J| + |B| = 2|L| + |J| - 2$$

$$\Rightarrow \quad \pi(T) \geq |L| + \left( |J| - 1 \right)/2 = (m+1)/2$$

which proves:
For any tree of maximum degree 3 on $m$ vertices, $\pi(T) \geq (m+1)/2$
Proof of Lemma 5 (cont.)

For any arbitrary tree on $m$ vertices, $\pi(T) \geq (m+1)/2$

By minimum counterexample:
- Let $T$ be a minimum counterexample, $|T|=m$.
- $T$ must have at least one vertex $v$ of degree 4 or more.
- We “break” the vertex $v$ into two copies (in $T_1$ $v$ is a leaf).
- In $T_2$, the degree of $v$ is decreased by 1.
- We get: $m_1 + m_2 = m + 1$
- $\pi(T_i) \geq (m_i + 1)/2$, and there are suitable path-matchings of the sets of $J_i$.
- By putting $T$ back together, we get: $\pi(T) \geq (m_1 + 1)/2 + (m_2 + 1)/2 - 1 = (m+3)/2 - 1 = (m+1)/2$
Proof of Lemma 5 (cont.)

For any **arbitrary tree** on $m$ vertices, $\pi(T) \geq (m+1)/2$

$T_1$: $v$ is a leaf

$T_2$: $d(v)$ is decreased by 1
Lemma 6 (revisited)

If on the branch (of Step 1 of the algorithm) corresponding to \( A \subseteq S \) we have a reduced instance \((G', U', k')\) where the vertices of \( G' \) are partitioned into \( A' \) and \( F \) (as defined above), and where \(|F| \geq 4k+1\), then this is a NO-instance.

**Proof:** By Lemma 5 (\(|F| = m, \pi(F) \geq (m+1)/2\)) we get \( \pi(F) \geq 2k+1 \). The rest follows by Lemma 4 (if: \( \pi(F) \geq k' + |A'| \) then: NO), since \(|A'| \leq k+1\) and \( k' \leq k \).
Complexity of the Algorithm

- **Step 1**
  Branch on all $2^{k+1}$ subsets of $S$.
- **Step 2**
  Exhaustive analysis. $\binom{4k}{k} \approx (9.4815)^k$ subsets.

Total running time of our algorithm is $O(18.963^k n^2)$. 
A More Efficient Version

- Lemma 4 yields: if $\pi(F) \geq k'+|A'|$ then it’s a NO-instance.
- $k' = k - |A|, \quad |A'| = k + 1 - |A|$
- Lemma 5 yields: $\pi(F) \geq (m+1)/2$
- $m \leq 2((k - |A|) + (k + 1 - |A|)) - 1$

Total bound on the number of possible solutions explored in Steps 1 and 2 is:

$$\sum_{i=0}^{k} \binom{k+1}{i} \left(2\left((k + 1 - i) + (k - i) - 1\right) - 1\right)$$

$$= \sum_{i=0}^{k} \binom{k+1}{i} \left(4k - 4i - 1\right)$$
A More Efficient Version (cont.)

\[
\sum_{i=0}^{k} \binom{k + 1}{i} \binom{4k - 4i - 1}{k - i}
\]

Define: \( f(x, k) = \binom{k}{x} \binom{4(k - x)}{k - x} \)

Suppose \( f(x, k) \) is maximized for \( x^* = x(k) \).

Then our sum above is bounded by \((k+1) \cdot f(x^*, k+1)\).
A More Efficient Version (cont.)

\[ f(x, k) = \binom{k}{x} \binom{4(k - x)}{k - x} \]  
bound: \((k+1) \cdot f(x^*, k+1)\)

We work out two estimates \(x_1(k) \leq x^*(k) \leq x_2(k)\).

Therefore we have a bound on our sum of:

\[ (k + 1) \cdot \binom{k + 1}{x_2(k + 1)} \left(\frac{4((k + 1) - x_1(k + 1))}{(k + 1) - x_1(k + 1)}\right) \]
A More Efficient Version (cont.)

Studying the ratio \( f(x, k)/f(x+1, k) \) shows that the maximizing value \( x^* \) is located essentially at the point where this ratio is equal to 1.

Moreover, the partial derivative of \( f'(x, k) \) with respect to \( x \) is positive in the range \([0, k)\).

Hence, \( f'(x, k) \) is an increasing function of \( x \) in that range.

It follows that, over the reals, there is a unique \( x^* \) such that \( f'(x^*, k) = 1 \).

The ratio is approximately:

\[
\frac{f(x, k)}{f(x + 1, k)} \approx \left( \frac{x + 1}{k - x} \right) \cdot 4 \cdot \left( \frac{4}{3} \right)^3
\]
A More Efficient Version (cont.)

\[ \frac{f(x, k)}{f(x + 1, k)} \approx \left( \frac{x + 1}{k - x} \right) \cdot 4 \cdot \left( \frac{4}{3} \right)^3 \]

This yields the estimates:
\[ x_1(k) = \frac{27}{283} k \]
\[ x_2(k) = \frac{28}{283} k \]

Using Stirling’s approximation we obtain the bound on our total cost sum of \((k+1)(10.567)^k\).
Optimality

- **Solution Compression For FVS** – $O(10.567^k n^2)$
- This yields an FPT algorithm for the parameterized
  \textit{Feedback Vertex Set} problem – $O(10.567^k n^3)$
- In qualitative terms it’s a running time of the form $O^*(2^{O(k)})$
  (This notation focuses attention on the exponential time costs due to the
  parameter and ignores the polynomial time costs due to the overall input size)
- Best previous running time: $O^*(2^{O(k \log k / \log \log k)})$
Theorem: There can be no FPT algorithm for FVS with a running time of the form $O^*(2^{o(k)})$ unless $\text{FPT} = \text{M}[1]$.

Proof: The standard reduction from VC to FVS doesn’t change the value of the parameter. Hence, there exists an algorithm for VC with a running time of $2^{o(k)}p(n)$, for some polynomial $p$. This implies $\text{FPT} = \text{M}[1]$ and has other consequences considered unlikely.
Open Problems?

- Is the *Feedback Vertex Set* problem for directed graphs in FPT?  
  J. Chen et al., 2008

- Is there a polynomial-time algorithm that kernelizes FVS on undirected graphs to a kernel of size polynomial in $k$?  
  K. Burrage et al., 2006
Summary

• Iterative Compression technique for minimization problems

• Generalized Undirected Feedback Vertex Set

• An optimal $O(2^{O(k)} n^3)$ FPT Algorithm for the Undirected Feedback Vertex Set Problem

• An FPT Algorithm for Solution Compression for FVS
Questions