A General Method to Speed Up
Fixed-Parameter-Traceable Algorithms

By
Niedermeier and Rossmanith

Presented By
Chris Standish
chriss@cs.tamu.edu

22 April 2004
Topics To Be Covered

- The problem, speeding up FPT algorithms.
- Two standard techniques, Reduction to Problem Kernel (RPK) and Bounded Search Tree (BST).
- Main result: The FPT speed up algorithm.
- Analysis of the running time of the speed up algorithm.
- Examples of its application to the Vertex Cover problem.
Speeding Up FPT Algorithms

The basic idea is that during each iteration of some FPT algorithm we first do a Reduction to Problem Kernel step followed by a Bounded Search Tree step. If certain assumptions are satisfied, the factor $R(q(k))$ in the equation for the running time is replaced by a constant $[1]$.

That is:

$$O(P(|I|) + R(q(k))\varepsilon^k) \rightarrow O(P(|I|) + \varepsilon^k)$$

where:

- $P(|I|)$ - the number of steps to reduce a problem instance $(I, k)$ to the problem kernel $(I', k')$,
- $R(|I|)$ - the number of steps to process a node in the search tree,
- $q(k)$ - polynomial bound on the size of the problem kernel, i.e., $|I'| \leq q(k)$, $|I'|$ is bounded by a function of the parameter $k$ alone,
- $\varepsilon^k$ - upper bound on the size of the bounded search tree.

The method:

- requires that $P(|I|)$, $q(k)$ and $R(|I|)$ are each bounded by a polynomial in the parameter $k$.
- is applicable to all FPT algorithms that work in two stages, Reduction to Problem Kernel followed by Bounded Search Tree.
- improves the overall running time of an FPT algorithm by focusing on the Bounded Search Tree stage.
Standard Techniques

Reduction to Problem Kernel - consists of replacing the original instance \((I, k)\) with a new instance \((I', k')\) such that:

\[
k' \leq k, \quad |I'| \leq q(k'), \quad (I, k) \in \mathcal{L} \quad \iff \quad (I', k') \in \mathcal{L}.
\]

where \(\mathcal{L}\) is a parameterized problem, i.e., the set of pairs \((I, k)\) for which problem instance \(I\) has a solution of size \(k\). This stage has time complexity \(O(P(|I|))\). The point of RPK is to reduce the size of the search space.

Bounded Search Trees - the instance \((I, k)\) is replaced by several smaller instances

\[(I_1, k - d_1), (I_2, k - d_2), \ldots, (I_m, k - d_m)\]

such that:

\[
d_i > 0 \quad \text{for all} \quad i \in \{1, 2, \ldots, m\}\\
|I_i| \leq |I| \quad \text{for all} \quad i \in \{1, 2, \ldots, m\}\\
(I, k) \in \mathcal{L} \quad \iff \quad (I_i, k - d_i) \in \mathcal{L} \quad \text{for some} \quad i \in \{1, 2, \ldots, m\}
\]

This stage has time complexity \(O(R(q(k)) \varepsilon^k)\).

Recurrence Relation - The BST stage has the following recurrence relation:

\[
S_k = S_{k - d_1} + S_{k - d_2} + \cdots + S_{k - d_m}
\]

where \(S_k\) denotes the number of search paths in the search tree.

Characteristic Polynomial - The recurrence relation corresponds to the characteristic polynomial:

\[
z^d = z^{d - d_1} + z^{d - d_2} + \cdots + z^{d - d_m}
\]

where \(d = \max\{d_1, \ldots, d_m\}\).

Key Observation - The solution of the recurrence relation has the general form \(S_k = \Theta(p(k) \varepsilon^k)\) where \(\varepsilon\) is a real root of the characteristic polynomial with maximum absolute value, and \(p(k)\) is a polynomial in \(k\) [2].

If \(\varepsilon\) is the unique real root then \(p(k)\) is a constant and so the BST stage will have time complexity \(\Theta(\varepsilon^k)\) [1, 3].

\[
O(P(|I|) + R(q(k)) \varepsilon^k) \longrightarrow O(P(|I|) + \varepsilon^k)
\]
FTP Speed Up Algorithm

The following algorithm is used to process a node \((I, k)\) in the search tree:

Algorithm 0.1 FPT Speed Up

\[
\begin{align*}
\text{If } |I| &> c \cdot q(k) \\
\text{Then replace } (I, k) \text{ with } \mathcal{R}(I, k) \\
\text{replace } (I, k) \text{ with } & (I_1, k_1), (I_2, k_2), \ldots, (I_m, k_m)
\end{align*}
\]

Where \(c \geq 1\) is a constant chosen to further optimize the running time, \(\mathcal{R}\) denotes a function that performs the reduction, i.e., \(\mathcal{R}(I, k) = (I', k')\).

Notice the abuse of notation, i.e., when a new instance \((I', k')\) is created by \(\mathcal{R}\) it is denoted \((I, k)\).

However, the recurrence relation for the BST stage becomes more complicated.
Running Time of FPT Speed Up Alg

- The basic idea is that the RPK stage will be called for nodes near the root of the tree, where $k$ is large. This will result in new instances that quickly become too small for the RPK stage to be called. The time needed to process these small instances will then be bounded by some small constant.

- Since the speed up algorithm interleaves the RPK and BST stages, during each recursive call, we need to take into consideration the time for the RPK stage.

- To analyze the running time we need to consider the cost of the RPK stage in addition to the size of the bounded search tree.

Running Time -
- Let $T_k$ denote the time to process a node $(I, k)$ in the tree.
- The inhomogeneous recurrence relation for the running time is given by:

$$T_k = T_{k-d_1} + \ldots + T_{k-d_m} + O(P(q(k)) + R(q(k))) \tag{1}$$

- To solve Eqn (1) we need a special solution for the inhomogeneous part.
- The general solution to Eqn (1) will then be given by the sum of the solutions for the homogeneous and inhomogeneous parts.

General Solution -
- We have already seen that the solution to the homogeneous part is bounded by $O(\varepsilon^k)$.
- Further, we have required that $P(|I|)$, $R(|I|)$ and $q(k)$ all be bounded by a polynomial in $k$, and $|I| \leq q(k)$.
- So the special solution will also be bounded by a polynomial in $k$, say $p(k)$.
- Then the general solution will be bounded by $O(\varepsilon^k + p(k)) = O(\varepsilon^k)$ [2].

The Result -

$$O(P(|I|) + R(q(k))\varepsilon^k) \longrightarrow O(P(|I|) + \varepsilon^k)$$
Example: VC-Degree-3

Algorithm for vertex cover for graphs with maximum degree 3, See Chen, et al. [4].

Algorithm 0.2 VC-Degree-3

Input: an instance \((G, k)\) with maximum degree 3
Output: a vertex cover \(C\) of size bounded by \(k\), if one exists.

\[C = \emptyset;\]
\[\text{while } |C| \leq k \text{ and } G \text{ is not empty do}\]
\[\text{pick a degree-2 vertex } v \text{ in } G;\]
\[\text{fold } v;\]
\[\text{if the new vertex } v_0 \text{ has degree larger than 2 then branch at } v_0;\]

Notice each time through the while loop we only look at local structure.

The algorithm has time complexity \(O(kn + 1.273^k)\) after the speed up technique is applied(not shown above).
Example: VC3-solver

An improved algorithm for vertex cover for graphs with maximum degree 3, See Chen, et al. [5].

Algorithm 0.3 VC3-solver

**Input:** an instance \((G, k)\) with maximum degree 3

**Output:** a vertex cover \(C\) of size bounded by \(k\), if one exists.

1. while there is a deg-2 vertex \(v\) where folding \(v\) is safe do Fold\((v)\); 
   if Reducing is applicable then apply Reducing and go to step 1 
   else if there is a deg-2 vertex \(v\) then branch on the two neighbors of \(v\) 
   else branch on a deg-3 vertex \(v\)

**Reducing**

1. if there is a component \(H\) of size bounded by 50 
   then compute a minimum vertex cover of \(H\) by brute force
2. else if there are two adjacent triangles \((u, v, w)\) and \((u, v, z)\) 
   then include \(v\) in the cover
3. else if there is an alternating cycle \(K\) in \(G\) 
   then include all deg-3 vertices in \(K\) in the cover
4. else if removing a cut-vertex or a two-edge cut results in a component \(H\) with \(2 \leq |V(H)| \leq 50\) 
   then remove \(H\) without any branching as explained in Theorem 1
5. else if there is a maximal alternating tree \(T\) of at least 4 vertices in \(G\) 
   then branch on the vertices in \(T\) that are of deg-3 in \(G\)

Notice that both local and global structure are examined.

The algorithm has amortized time complexity \(O(n + 1.194^k)\) after the speed up technique is applied (not shown above).
References


