ELSEVIER

Information Processing Letters 73 (2000) 125–129

Information Processing Letters

www.elsevier.com/locate/ipl

A general method to speed up fixed-parameter-tractable algorithms

Rolf Niedermeier [a], Peter Rossmanith [b,*]

[a] *Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13, D-72076 Tübingen, Germany*

[b] *Institut für Informatik, Technische Universität München, Arcisstr. 21, D-80290 München, Germany*

Received 14 June 1999; received in revised form 17 November 1999

Communicated by K. Iwama

Abstract

A fixed-parameter-tractable algorithm, or FPT algorithm for short, gets an instance (I,k) as its input and has to decide whether $(I,k) \in \mathcal{L}$ for some parameterized problem $\mathcal{L}$. Many parameterized algorithms work in two stages: reduction to a problem kernel and bounded search tree. Their time complexity is then of the form $\mathrm{O}(p(|I|) + q(k)\xi^k)$, where $q(k)$ is the size of the problem kernel. We show how to modify these algorithms to obtain time complexity $\mathrm{O}(p(|I|) + \xi^k)$, if $q(k)$ is polynomial.

Keywords: Algorithms; Parameterized complexity

1. Introduction

A parameterized problem usually consists of two components—the input and aspects of the input that constitute a parameter. For example, the NP-complete VERTEX COVER problem has an undirected graph G as its input and a positive integer k as its parameter; the question is whether there is a set of at most k vertices that cover all edges in G. The central question of parameterized complexity theory [5] is as follows: Given a parameterized problem $\mathcal{L}$ with input size n and parameter k, is there an algorithm solving $\mathcal{L}$ in time $f(k)n^\alpha$, where α is a constant independent of k and n and f is an arbitrary function depending only on k. A problem with such an algorithm is called *fixed-parameter-tractable* and the corresponding complexity class of problems is called FPT. VERTEX COVER is in FPT [1,4,5], the currently best known FPT algorithm running in time faster than $\mathrm{O}(kn + 1.3^k k^2)$ [3, 11,12].

There is, however, a problem concerning the definition of FPT—the function f may grow arbitrarily fast. Thus, there are currently only a few parameterized problems known that have an (exponential) function f that grows as "slowly" as in the case of VERTEX COVER. The development of efficient FPT algorithms hence is an active field of research [5,6,9]. To the authors' best knowledge, at least the majority of *efficient* FPT algorithms known so far (e.g., [1,6,7,10, 11]) is based on the combination of two standard methods: *bounded search trees* and *reductions to problem kernel* [5]. Here, we show how to significantly improve all FPT algorithms based on the combination of these two techniques. Hence, we contribute to the positive toolkit for designing FPT algorithms, which according to Downey and Fellows [5, p. 20] belongs

* Corresponding author. Email: rossmani@informatic.tu-muenchen.de.

0020-0190/00/$ – see front matter
PII: S0020-0190(00)00004-1

to the current research horizons in parameterized complexity: "The positive toolkit for designing FPT algorithms contains several key methods that are very deep and general—but for which practicality is still not yet clearly established". In the following, we provide a simple, practical, and generally applicable method to speed up FPT algorithms.

The basic idea of improvement is in a sense to *interleave* reduction to problem kernel and bounded search tree method. More specifically, assume that we have an FPT algorithm running in time $\mathrm{O}(p(n) + q(k)\xi^k)$, where ξ is a (small) constant and p and q are polynomials. Moreover, ξ^k shall be the size of the bounded search tree and $q(k)$ the size of the problem kernel. Then our new technique shows how to get rid of the factor $q(k)$, thus transforming the above algorithm into a time $\mathrm{O}(p(n) + \xi^k)$ one. It is important here to note that this improvement is not due to asymptotic tricks, but that $q(k)$ can be replaced by a *small* constant. We shall just swiftly mention that our technique leads to a significant improvement of the so-called klam values (cf. [5, pp. 13–14]) for many problems. For example, consider VERTEX COVER again. The first nontrivial parameterized algorithm for this problem had running time $\mathrm{O}(kn + 1.32472^k k^2)$ [1], recently further developed to $\mathrm{O}(kn + 1.29175^k k^2)$ and recently even further to $\mathrm{O}(kn + 1.271^k k^2)$ [3]. Compare the growth of the two functions $1.32472^k k^2$ and $1.29175^k k^2$. For instance, for $k = 100$ the first one is bounded by 1.64×10^{16} and the second by 1.32×10^{15}, hence where $k = 100$, it improves by a factor of roughly 12. By way of contrast, the improvement from $1.29175^k k^2$ to 1.29175^k is a factor of about 10 000. In this context, observe that the second improvement involves, due to our technique, a small constant factor, which, however, is by magnitudes smaller than 10 000. Summarized, this shows that for practical parameter sizes ($k \approx 100$ is very natural in the case of VERTEX COVER and many other parameterized problems) our improvement has a potentially much higher benefit than small (but in no way trivial) improvements in the exponential base ξ of the search tree size may have.

2. FPT algorithms

Many FPT algorithms work in two stages [5]: Firstly, the instance is transformed into an equivalent one that is smaller in size. To be specific, its size is bounded by a function that depends *on the parameter only.* This stage is called *reduction to problem kernel.* Secondly, the new small instance is solved recursively by solving several derived instances with *smaller parameters.* Since the parameters in the recursive calls are smaller, the recursion eventually terminates (either by finding a solution or by realizing that no solution exists because of $k \leqslant 0$). That stage is called *bounded search tree.*

In the following we describe each stage in more detail and introduce all necessary notation that is needed to improve the algorithm. We also illustrate some concepts within the example of VERTEX COVER. The instance of VERTEX COVER is an undirected graph and a parameter k. The question the algorithm must answer is whether or not a vertex cover of size at most k exists. (A vertex cover is a subset of vertices C such that every edge in the graph is incident to at least one vertex in C.)

2.1. Reduction to problem kernel

Let $\mathcal{L}$ be a parameterized problem, i.e., $\mathcal{L}$ consists of pairs (I, k), where I has a solution of size k. In the case of VERTEX COVER $\mathcal{L}$ consists of all (G, k), where G is an undirected graph that has a vertex cover of size k. Reduction to problem kernel consists of replacing the original instance (I, k) with a new one (I', k') so that

$$k' \leqslant k, \quad |I'| \leqslant q(k'),$$

and

$$(I, k) \in \mathcal{L} \Leftrightarrow (I', k') \in \mathcal{L}.$$

What is particularly important is that the size of the new instance I' is bounded by a function of the parameter alone. We call this function q. In general, q might be arbitrary, but in this paper we restrict q to being polynomial as is usually the case for efficient FPT algorithms. [1] In the case of VERTEX COVER, reduction to problem kernel replaces an instance (G, k) with an instance (G', k'), where the new graph G' consists of at most $k(k + 1)$ vertices and $k' \leqslant k$ (see [1,5] for details).

[1] In general, $k' \leqslant k$ is not necessary. All results also hold if k' is bounded by some polynomial in k.

Let $\mathcal{R}$ denote the function that performs the reduction to problem kernel, i.e., $\mathcal{R}(I,k) = (I',k')$ and let $P(|I|)$ be the number of steps required to perform the reduction. We demand P be bounded by some polynomial. For VERTEX COVER $P(|G|) = \mathrm{O}(|G|)$ if the graph is represented by an adjacency list.

2.2. *Bounded search trees*

Let (I',k') be an instance after reduction to problem kernel. Many algorithms solve the problem by constructing a search tree that looks exhaustively for solutions. In order to gain efficiency, branches will be pruned. Pruning of branches is subject chiefly to two conditions: Either we can be sure that the branch contains no solution or, if there are two branches A and B we can prune B if we can be sure that a solution in B implies a solution in A of the same size at most. The main objective to find ever more efficient FPT algorithms involved decreasing the size of the search tree. In the following we analyze first the *size* of the tree as well, but then take a look at the *time* taken for processing the tree. The next section building on this analysis improves the overall *time* to traverse the search tree, but not its *size*, which will not be affected at all.

In general, let (I,k) be a node of the search tree. To solve (I,k), it is replaced by several instances $(I_1, k-d_1), (I_2, k-d_2), \ldots, (I_m, k-d_m)$ so that $d_i > 0$ and $|I_i| \leqslant |I|$ for all $i \in \{1,\ldots,m\}$ and $(I,k) \in \mathcal{L}$ iff $(I_i, k-d_i) \in \mathcal{L}$ for some $i \in \{1,\ldots,m\}$. The leaves consist commonly of those instances with $k \leqslant 0$. Since all $d_i > 0$, the children's parameters are strictly smaller and the tree has a finite size. An upper bound on the size of the tree is easy to obtain by solving the corresponding recurrence for the number of leaves:

$$S_k = S_{k-d_1} + S_{k-d_2} + \cdots + S_{k-d_m}.$$

The solution has the general form $S_k = \Theta(p(k)\xi^k)$, where $1/\xi$ is the smallest positive, real root of the reflected characteristic polynomial

$$1 - z^{d_1} - z^{d_2} - \cdots - z^{d_m}$$

and $p(k)$ is a polynomial [8]. If ξ is a unique root, as is almost always the case, p is simply a constant and therefore $S_k = \Theta(\xi^k)$. In the following we assume that ξ is a unique root. If that were not the case, then p is not a constant, but some polynomial of degree > 0. In that case ξ^k should be replaced by $p(k)\xi^k$ in the next section.

Finally, let $R(|I|)$ be the time needed to compute $(I_1, k-d_1), (I_2, k-d_2), \ldots, (I_m, k-d_m)$ from (I,k). Again we demand that $R(|I|)$ be bounded polynomially. The overall time complexity for the second stage *bounded search tree* is $\mathrm{O}(R(q(k))\xi^k)$.

3. Accelerating FPT algorithms

In the following, we will deal with a large class of fixed-parameter-tractable algorithms. Let us summarize the conditions that these algorithms have to undergo: They have to be FPT algorithms that work in two stages, *reduction to problem kernel* and *bounded search tree*. Reduction to problem kernel takes $P(|I|)$ steps and results in an instance of size at most $q(k)$, where both P and q are polynomially bounded. The expansion of a node in the search tree takes $R(|I|)$ steps, which must also be bounded by some polynomial, the search tree size being $\mathrm{O}(\xi^k)$. The overall time complexity of the algorithm is then

$$\mathrm{O}\big(P(|I|) + R(q(k))\xi^k\big),$$

where (I,k) is the instance to be solved. In the following we show how to modify the second stage of the algorithm in order to improve the time complexity to

$$\mathrm{O}\big(P(|I|) + \xi^k\big).$$

Generally, we now use the following algorithm to expand a node (I,k) in the search tree:

if $|I| > c \cdot q(k)$
  then replace (I,k) with $\mathcal{R}(I,k)$ **fi**;
replace (I,k) with
  $(I_1, k-d_1), (I_2, k-d_2), \ldots, (I_m, k-d_m)$

Here $c \geqslant 1$ is a constant that can be chosen with the aim of further optimizing the running time. There is a tradeoff in choosing c: The optimal choice depends on the implementation of the algorithm, but in the end it affects only the constant factor in the overall time complexity. Therefore we neglect optimizing c in this paper.

A closer look shows that we in fact seem to *increase* the time needed to expand a node in the search tree. This is generally speaking true: Sometimes we apply reduction to problem kernel prior to splitting into recursive calls. However, these additional reductions to problem kernel also *decrease* the instance size in the middle of the search tree. Since the time for splitting is bounded polynomially in the *instance size*, this also helps to *decrease* the time to expand a node. It proves to be the case that while we waste time near the root of the search tree, we gain much more time near the leaves. Note that the technique of *interleaving* reduction to problem kernel and bounded search trees was already used for developing efficient FPT algorithms for VERTEX COVER [6,12]. There, however, it was used to reduce the number of case distinctions in the search tree; it was not considered with the aim of removing the factor $R(q(k))$ as we do.

In order to analyze the running time of the above mathematically, we describe the time to expand a node (I,k) and all its descendants by a recurrence. Let T_k denote an upper bound on the *time* to process (I,k). The following recurrence exists for T_k:

$$T_k = T_{k-d_1} + T_{k-d_2} + \cdots + T_{k-d_m} + \mathrm{O}\big(P(q(k)) + R(q(k))\big).$$

The time to expand (I,k) itself is at most $\mathrm{O}(P(q(k))+R(q(k)))$, since $|I| = \mathrm{O}(q(k))$ since $|I| > c \cdot q(k)$ is constantly prevented. In order to solve this non-homogeneous linear recurrence we need a special solution. To get its general solution we add the general solution of the corresponding homogeneous recurrence $T_k = T_{k-d_1} + T_{k-d_2} + \cdots + T_{k-d_m}$. However, we already know that all solutions of this homogeneous recurrence are bounded by $\mathrm{O}(\xi^k)$. Consequently we are only required to find a small special solution of the non-homogeneous recurrence. In our case the inhomogeneity is a polynomial. Therefore, there exists a special solution that is also a polynomial in k. It is easy to construct such a special solution explicitly. There is always a polynomial solution that has the same degree as the inhomogeneity p. (If r is a polynomial special solution then

$$r(k) - \sum_{i=1}^{m} r(k-d_i) = p(k)$$

and the highest degree monomials on the left side cannot cancel each other.) All solutions of T_k are therefore bounded by $\mathrm{O}(\xi^k)$.

In order to illustrate this, let us consider the following recurrence.

$$T_k = 2T_{k-1} + C \cdot k^2 + D \cdot k + E,$$

where C, D and E are constants that depend on the implementation of the algorithm. The initial conditions are simple, say, $T_0 = 0$. The reflected characteristic polynomial is $1 - 2z$ and its unique root is $\frac{1}{2}$. The general solution of the homogeneous recurrence is $\lambda 2^k$ for $\lambda \in \mathbb{R}$. Since it is a recurrence of first order, the dimension of its space of solutions is one, too.

A special solution is

$$T_k = -Ck^2 - (4C + D)k - (6C + 2D + E).$$

The general solution is then

$$\lambda 2^k - Ck^2 - (4C + D)k - (6C + 2D + E)$$

and the solution for $T_0 = 0$ is

$$T_k = (6C + 2D + E) \cdot 2^k - Ck^2 - (4C + D)k - (6C + 2D + E).$$

4. The modification is necessary

In this section, we show that an improved analysis alone cannot achieve the speedup of the last section. That is, the interleaving of reduction to problem kernel and the bounded search tree really is necessary to get the claimed improvements. Without modification, the algorithms in general have a running time of $\Omega(P(|I|) + R(f(k))\xi^k)$. As an example, we can again use VERTEX COVER. Look at Fig. 1 for a definition of a family of instances of VERTEX COVER defined for odd k. There is no solution of size $\leqslant k$, since the optimal vertex cover has size $\frac{5}{2}k - \frac{3}{2}$ (in the head $k-2$ vertices and half the vertices of the tail). The graph contains exactly $(k-1)(k-2)+1$ vertices in the head and $3k+1$ vertices in the tail (altogether k^2+4). Reduction to problem kernel does not affect this graph since the degree of every vertex is at most k, although its size is very near the maximum possible $k(k+1)$. Now assume that the unmodified algorithm chooses edges from right to left. This leads to a search tree of size 2^k, the largest possible. While the algorithm examines this graph, it removes nodes and edges, but the *head* remains unchanged. Consequently, instances have size $\Omega(k^2)$ during *each* splitting step. The overall

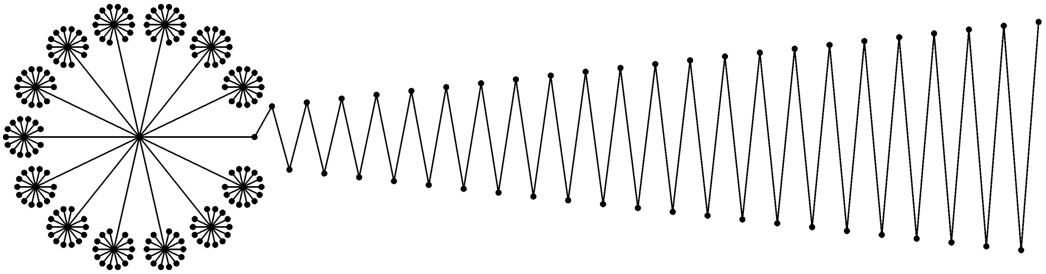

Fig. 1. An instance of VERTEX COVER. The following graph is the $k = 15$ member of a family of instances (G_k, k) for VERTEX COVER. The graph G_k consists of a tree with degree $k - 1$ and depth 2 to which a path with $3k + 1$ vertices is attached (called the *tail*). It is easy to see that the smallest vertex cover for G_k has size $\frac{5}{2}k - \frac{3}{2}$ and therefore the whole family has no members in VERTEX COVER.

time complexity therefore is the worst possible—$\Omega(k^2 2^k)$. Of course, a better time complexity can also be achieved by changing the order of choosing edges. Nevertheless, the time bound is $\Theta(k^2 2^k)$ in the worst case.

After the modification the running time is decreased tremendously. After the *second* edge is removed and k decreased by two, the whole head is removed from the graph.

5. Conclusion

We introduced a new, simple, and prospective technique for speeding up FPT algorithms based on reductions to problem kernel and bounded search trees. As a rule, the potential for improvement due to our method increases the larger the problem kernel in the underlying parameterized problem is. For example, associated candidate problems (see [5] for details) are k-Leaf Spanning Tree (problem kernel size $O(k^2)$) [2] and Hitting Set for Size Three Sets (problem kernel size $O(k^3)$) [10]. Thus, our method belongs in the toolkit of every designer of efficient FPT algorithms.

References

[1] R. Balasubramanian, M.R. Fellows, V. Raman, An improved fixed parameter algorithm for vertex cover, Inform. Process. Lett. 65 (3) (1998) 163–168.

[2] L. Cai, J. Chen, R.G. Downey, M.R. Fellows, Advice classes of parameterized tractability, Ann. Pure Appl. Logic 84 (1997) 119–138.

[3] J. Chen, I. Kanj, W. Jia, Vertex cover: Further observations and further improvements, in: Proc. 25th International Workshop on Graph-Theoretic Concepts in Computer Science, Ascona, Switzerland, Lecture Notes in Computer Sci., Springer, Berlin, 1999.

[4] R.G. Downey, M.R. Fellows, Parameterized computational feasibility, in: P. Clote, J. Remmel (Eds.), Feasible Mathematics II, Birkhäuser, Boston, MA, 1995, pp. 219–244.

[5] R.G. Downey, M.R. Fellows, Parameterized Complexity, Springer, Berlin, 1999.

[6] R.G. Downey, M.R. Fellows, U. Stege, Parameterized Complexity: A Framework for Systematically Confronting Computational Intractability, in: R.L. Graham, Kratochvi, J. Nesetril, F.S. Roberts (Eds.), Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 49, AMS, Providence, RI, 1999, pp. 49–100.

[7] H. Fernau, R. Niedermeier, An efficient exact algorithm for constraint bipartite vertex cover, in: Proc. 24th Conference on Mathematical Foundations of Computer Science, Szklarsk Poreba, Poland, Lecture Notes in Computer Sci., Vol. 1672, Springer, Berlin, 1999, pp. 387–397.

[8] D.H. Greene, D.E. Knuth, Mathematics for the Analysis of Algorithms, 2nd edn., Progress in Computer Science, Birkhäuser, Boston, MA, 1982.

[9] R. Niedermeier, Some prospects for efficient fixed parameter algorithms (Invited Paper), in: B. Rovan (Ed.), Proc. 25th Conference on Current Trends in Theory and Practice of Informatics (SOFSEM), Lecture Notes in Computer Sci., Vol. 1521, Springer, Berlin, 1998, pp. 168–185.

[10] R. Niedermeier, P. Rossmanith, An efficient fixed parameter algorithm for 3-Hitting Set, Technical Report WSI-99-18, WSI für Informatik, Universität Tübingen, Germany, 1999.

[11] R. Niedermeier, P. Rossmanith, Upper bounds for Vertex Cover further improved, in: C. Meinel, S. Tison (Eds.), Proc. 16th Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Sci., Vol. 1563, Springer, Berlin, 1999, pp. 561–570.

[12] U. Stege, M. Fellows, An improved fixed-parameter-tractable algorithm for Vertex Cover, Technical Report 318, Department of Computer Science, ETH Zürich, April 1999.