ELSEVIER

Information Processing Letters 65 (1998) 163-168

Information Processing Letters

# An improved fixed-parameter algorithm for vertex cover [1]

R. Balasubramanian [a], Michael R. Fellows [b], Venkatesh Raman [a,*]

[a] *The Institute of Mathematical Sciences, C.I.T. Campus, 600 113 Chennai, India*

[b] *Department of Computer Science, University of Victoria, Victoria, British Columbia, Canada*

Received 27 August 1996

Communicated by F. Dehne

**Abstract**

The VERTEX COVER problem asks, for input consisting of a graph $G$ on $n$ vertices, and a positive integer $k$, whether there is a set of $k$ vertices such that every edge of $G$ is incident with at least one of these vertices. We give an algorithm for this problem that runs in time $O(kn + (1.324718)^k k^2)$. In particular, this gives an $O((1.324718)^n n^2)$ algorithm to find the minimum vertex cover in the graph. © 1998 Published by Elsevier Science B.V.

*Keywords:* Algorithms; Vertex cover; Fixed-parameter tractability

## 1. Introduction

For many computational problems the input consists of several parts, and it is useful to study how the different parts contribute to overall problem complexity. For example, many well-known decision problems concerning graphs including CLIQUE, DOMINATING SET, GRAPH GENUS, MIN CUT LINEAR ARRANGEMENT, BANDWIDTH and the problem VERTEX COVER that we consider here, take as input a graph $G$ and a positive integer $k$.

The parameter $k$ appears to contribute to the complexity of these problems in two qualitatively distinct ways. GRAPH GENUS, MIN CUT LINEAR ARRANGEMENT and VERTEX COVER can all be solved in time $O(f(k)n^c)$ where $c$ is a constant independent of $k$ and $f$ is some (arbitrary) function. This "good behavior" is termed *fixed-parameter tractability* in the theory introduced by Downey and Fellows in [3]. As is the case with the polynomial-time complexity, the exponent $c$ is typically small.

Contrasting complexity behaviour is exhibited by the problems CLIQUE, DOMINATING SET and BANDWIDTH, for which the best-known algorithms have running times $\Theta(n^{ck})$. These problems have been shown to be complete or hard for the parameterized complexity class $W[1]$ and are considered unlikely to be fixed-parameter tractable [4].

The VERTEX COVER problem asks whether a given graph on $n$ vertices has a vertex cover (a set of vertices such that every edge is incident with some vertex of the set) of size at most $k$. This was one of the first problems shown to be fixed-parameter tractable. The first algorithm due to S. Buss (see [1]) had an $O(kn + 2^k k^{2k+2})$-time complexity. Papadimitriou and Yannakakis [7] while proving that the vertex cover problem is in P when $k$ is restricted to $O(\log n)$ provided an $O(3^k n)$ algorithm. By using the observa-

* Corresponding author: Email: vraman@imsc.ernet.in.

[1] Part of this work was done while the third author was visiting the University of Victoria, Canada.



*PII* S0020-0190(97)00213-5

tion of Buss, the running time of this algorithm can be improved to $O(3^k k^2 + kn)$. Downey and Fellows [5] have given a different algorithm that runs in time $O(2^k k^2 + kn)$ (see also [6, p. 216]). In this paper, we describe an algorithm with an improved running time bound of $O(kn + (1.324718)^k k^2)$.

## 2. Previous work

We first describe the algorithm due to Buss [1]. Given a simple graph and an integer $k$, it checks whether it has a vertex cover of size $k$. The algorithm is based on a technique which Downey and Fellows [5] classify as "the method of reduction to problem kernel". The main idea is to reduce (in polynomial time) the problem to an equivalent problem where the problem size is bounded by a function of $k$. We assume that the given graph $G$ is given in the adjacency list representation.

**Algorithm** (*Buss*).

*Step* 1: Find all vertices of degree more than $k$ in $G$. Let the number of those vertices be $b$. If $b > k$ then answer "NO". Otherwise include those $b$ vertices in the vertex cover, remove them and the edges incident with them from $G$. Let $k' = k - b$. Remove any resulting isolated vertices.

*Step* 2: If the resulting graph has more than $kk'$ edges, then answer "NO".

*Step* 3: Find by brute-force whether the resulting graph has a vertex cover of size $k'$. If so then answer "YES". Otherwise answer "NO".

If the graph has a vertex cover of size $k$, then all vertices of degree more than $k$ must be in the vertex cover. This justifies the first step. Step 1 can be performed in $O(kn)$ time given the adjacency list representation of the graph. As the vertices of the resulting graph have degrees bounded by $k$, $k'$ vertices can cover at most $kk'$ edges. This justifies Step 2. In Step 3, checking whether the resulting graph has a vertex cover of size $k'$ is done by brute-force by finding all possible subsets of size at most $k'$ of the resulting graph. Since the total number of edges is at most $kk'$ and there is no isolated vertex, the number of vertices of the resulting graph is at most $2kk'$. As $k'$ is at most $k$, Step 3 can be performed in $O((2k^2)^k k^2)$.

Note that when $k$ is $O(\log n)$ the complexity of the algorithm is superpolynomial. Papadimitriou and Yannakakis [7], when investigating the complexity of some NP-hard problems when the parameter $k$ is restricted to be logarithmic in the input size, designed the following algorithm to show that the LOG vertex cover problem is in P.

**Algorithm** (*PY*).

*Step* 1: Find a maximal matching in the graph. Let the size of the matching (the number of edges) be $m$. If $m > k$ answer "NO". If $2m \leqslant k$, then answer "YES". The $2m$ vertices form a vertex cover.

*Step* 2: Let $U$ be the set of the endpoints of the $m$ edges of the maximal matching. For every edge of the matching, either one of the endpoints or both are in any vertex cover of $G$. Furthermore, once a subset of $U$ is picked in a vertex cover, the rest of the vertex cover is uniquely determined: for every vertex in $V - U$, it is included in the vertex cover if and only if there is an edge incident with it whose other endpoint (which is in $U$) has not been picked in the vertex cover. So, cycle through the $3^m$ subsets of $U$ (by picking either one or both of the endpoints of each edge in the matching) and check, for each subset whether it along with its unique extension to $V$ is of size at most $k$. If it is so for any subset, answer "YES", otherwise answer "NO".

A maximal matching in a graph can be found in $O(n)$ time by simply picking the first edge in the list, removing the edges incident on both endpoints, and repeating this process until there are no edges.

In Step 2, for a subset of $U$, the vertex cover which is an extension of that subset can be determined in $O(e)$ time where $e$ is the number of edges in the graph. Since every vertex of $V - U$ has degree at most $2m$, $e$ is $O(mn)$ and as $m \leqslant k$, Step 2 can be performed in $O(3^k kn)$ time. So the entire algorithm requires $O(n + 3^k kn)$. By preprocessing the entire graph by applying Steps 1 and 2 of Buss' algorithm, we could assume that the resulting graph has $O(k^2)$ vertices and edges after spending $O(kn)$ time. Thus the bound for the algorithm reduces to $O(kn + 3^k k^2)$.

Instead of trying the $3^m$ subsets of $U$, one could simply run through $2^m$ subsets of $U$ by picking only one of the endpoints of each edge in the matching, and re-

cursively checking whether the resulting graph (after deleting the subset of vertices and the edges incident with them) has a vertex cover of size $k-m$. This results in an algorithm whose running time is $O(kn + 2^k k^2)$. Downey and Fellows [5] have achieved this bound independently by using a search tree technique. Their idea is to choose for each edge in the graph, one of the endpoints in the vertex cover, and check recursively whether any of the resulting two graphs (obtained after deleting the chosen vertex and all incident edges) has a vertex cover of size at most $k-1$. See also [6] for a similar algorithm.

In the next section, we present our algorithm that has a running time of $O(kn + (1.324718)^k k^2)$.

## 3. Improved search tree algorithms

Our algorithm is based on the "bounded search tree" technique as that of Downey and Fellows [5]. Hereafter the term "node" refers to a vertex of the search tree, and the term "vertex" refers to a vertex of the graph. Basically, at each node of the search tree we have a partial vertex cover and the resulting graph which is obtained by deleting the vertices in the partial vertex cover and edges incident with them (and any resulting isolated vertices) from the original graph. The edges of the tree correspond to various possibilities for addition of vertices to the existing vertex cover. We stop growing the search tree at a node if the partial vertex cover corresponding to that node has size $k$ or the resulting graph at that node is empty (in which case we have already found a vertex cover of size $k$ or less). Our search tree will have the property that for every vertex cover of size $k$ or less in the original graph, there will be a node in the search tree that has a corresponding (perhaps not the same) vertex cover of size $k$ or less, and if the original graph has no vertex cover of size $k$ or less, no node in our search tree will have its resulting graph empty. Given the adjacency list representation of the graph, we spend $O(n)$ time at each node of the search tree. So, if $C(k)$ is the total number of nodes in the search tree, then the total time spent is $O(nC(k))$.

We first reduce the problem to the kernel by applying Steps 1 and 2 of Buss' algorithm by spending $O(kn)$ time. So at the root node of the search tree, we have the $b$ vertices picked at Step 1 of Buss' algorithm in the partial vertex cover, and the number of vertices and edges of the resulting graph is $O(k^2)$.

First, to get the idea across, we describe a simple algorithm that has a running time of $O(kn+(\sqrt{3})^k k^2)$. At a node of the search tree, we pick a vertex $x$ of the resulting graph and grow a depth-first tree for three levels. We end up with a triangle $x, a, b, x$ or a path $x, a, b, c$ of length 3. In the first case, observe that any vertex cover will have the vertices $x, a$ or $a, b$ or $x, b$. In the second case, any vertex cover must have the vertices $x, b$ or $a, c$ or $a, b$. In either case, we try all the three possible pairs by including one pair at a time in the partial vertex cover and remove these vertices and the edges incident with them to reach the next node of the tree. If the depth-first tree stops at a (degree one) vertex at level one or two, then the neighbour of the degree one vertex is chosen in the partial vertex cover and we move to the next node of the search tree. We stop growing the search tree if the partial vertex cover has size $k$ or the resulting graph is empty. In the worst case, we can assume that each internal node of the search tree has (out)degree 3, and we pick two vertices to the partial vertex cover at each such branch. Thus the total number of nodes in the search tree $C(k)$ is $O(3^{k/2})$ and we have the following theorem.

**Theorem 1.** *Given a simple undirected $n$ vertex graph, and a positive integer $k$, we can find in $O((\sqrt{3})^k k^2 + kn)$ time a vertex cover of size at most $k$, or determine that no vertex cover of size $k$ exists.*

We extend this by analysing several cases and by picking subsets of different sizes at each node to prove the following theorem.

**Theorem 2.** *Given a simple undirected $n$ vertex graph, and a positive integer $k$, we can find in $O((1.324718)^k k^2 + kn)$ time a vertex cover of size at most $k$ or determine that no vertex cover of size $k$ exists.*

**Proof.** As before we preprocess the graph by applying the first two steps of the Buss' algorithm, and grow a search tree. At any node of the search tree, the adjacency list is scanned for the following cases in that order, and the suggested actions are taken depending

on the cases fulfilled by the graph. Each branch of the search tree is denoted by the set of vertices to be added to the partial vertex cover at the node from which the branch takes place. The partial vertex cover at the resulting node is the union of the partial vertex cover at its parent and the set of vertices in the branch. The partial vertex cover at the root is the set of $b$ vertices picked in Step 1 of Buss' algorithm.

Let $N(x)$ denote the neighbours of the vertex $x$, and $N(S)$ for a set $S$ is $\bigcup_{x \in S} N(x)$.

(1) If there is a degree 1 vertex, branch by picking its neighbour in the partial vertex cover. Thus

$$C(k) \leqslant 1 + C(k-1). \tag{1}$$

(2) If there is a degree 2 vertex say $x$, let its neighbours be $y$ and $z$. Then exactly one, or both, or neither of $y$ and $z$ is in any vertex cover. If exactly one of $y$ and $z$, say $y$ is in the vertex cover, then $x$ has to be in the cover to cover the edge $xz$. Since $x$ has no other role in that case, we might as well pick the other vertex $z$, and assume that either both or neither of $y$ and $z$ is in the vertex cover. If there is an edge between $y$ and $z$, this argument forces both $y$ and $z$ to be in the vertex cover. So we branch by simply including $y$ and $z$. This implies that

$$C(k) \leqslant 1 + C(k-2). \tag{2}$$

Otherwise,

(a) Let $y$ and $z$ together have at least two neighbours other than $x$. Then a vertex cover will have $\{y, z\}$ or $N(\{y, z\})$. Thus

$$C(k) \leqslant 1 + C(k-2) + C(k-3). \tag{3}$$

(b) The nodes $y$ and $z$ have together only one neighbour $a$ other than $x$. If both $y$ and $z$ are in the vertex cover, then we might as well pick $x$ and $a$ instead. So branch by simply picking $\{x, a\}$ to grow. Thus,

$$C(k) \leqslant 1 + C(k-2). \tag{4}$$

(3) If there is a vertex $x$ of degree at least 5, then either the vertex is in the vertex cover or all its neighbours are. So branch as $x$ or $N(x)$ which results in the recurrence

$$C(k) \leqslant 1 + C(k-1) + C(k-5). \tag{5}$$

(4) Now every vertex has degree exactly 3 or 4. Let $x$ be a vertex of degree three with neighbours 1, 2 and 3. In any vertex cover, if only two of these three vertices, say 1 and 2 are present then $x$ must be in the cover to cover the edge $x3$. Instead we might as well include 3 in the cover as $x$ has no other role. So without loss of generality, we assume that in any vertex cover, $\{1,2,3\}$ or $\{1\}$ or $\{2\}$ or $\{3\}$ or none of 1, 2 and 3 will be there.

(a) There is an edge (say 12) in the induced subgraph on 1, 2 and 3. Then in any vertex cover, if not all of 1, 2, 3 are there, then 3 cannot be there as at most one of $\{1,2,3\}$ can be in any vertex cover and at least one of 1 and 2 must be in the vertex cover. Then branch as $\{1,2,3\}$ or $N(3)$ and the recurrence is

$$C(k) \leqslant 1 + 2C(k-3). \tag{6}$$

since the degree of the vertex 3 is at least 3.

(b) There is a common neighbour other than $x$, say $y$, between a pair of the vertices $\{1,2,3\}$, say 1 and 2. Now if all of 1, 2 and 3 are not in the vertex cover, then $x$ and $y$ are since at most one of 1 and 2 can be in the vertex cover. So branch as $\{1,2,3\}$ or $\{x, y\}$ and the resulting recurrence is given by

$$C(k) \leqslant 1 + C(k-3) + C(k-2). \tag{7}$$

(c) There are no edges among 1, 2 and 3, and one of them say 1, has at least three neighbours other than $x$. Now if not all of 1, 2, 3 are there in a vertex cover, then either 1 is not there or 1 is there and 2 and 3 are not. So branch as $\{1,2,3\}$ or $N(1)$ or $\{1\} \cup N(\{2,3\})$ and the recurrence is given by

$$C(k) \leqslant 1 + C(k-3) + C(k-4) + C(k-6). \tag{8}$$

(d) There are no edges among 1, 2 and 3, and each of 1, 2 and 3 has, apart from $x$, exactly two private neighbours (say 4, 5; 6, 7 and 8, 9 respectively). At least one of these vertices say 4, has a neighbour say 10, in the complement of the induced subgraph on the vertices 1 to 9. (Otherwise the induced subgraph on the vertices 1 to 9 forms a component by itself and so the optimum vertex

cover in the component can be found in constant time.) In that case if all of 1, 2 and 3 are not in a vertex cover, then $x$ is. Consider vertices 4 and 5. Either 4 and 5 are there in the vertex cover or exactly one of them is there or neither of them is there. If exactly one of them (say 5) is there, then 1 has to be there apart from $x$. But the only role of 1 is to cover the edge 14, and we might as well pick the vertex 4 instead of 1. So without loss of generality either both of 4 and 5 are there in the vertex cover or neither of them. If neither of them is there, then 1 must be there and so 2 and 3 cannot be. Also neighbours of 4 and 5 must be in the cover. So branch as $\{1,2,3\}$ or $\{x,4,5\}$ or $N(\{2,3,4,5\})$ and the resulting recurrence is

$$C(k) \leqslant 1 + 2C(k-3) + C(k-7). \qquad (9)$$

since $N(\{2,3,4,5\}) = \{x,6,7,8,9,1,10\}$.

(5) Repeating the above steps, we will be left with a 4-regular graph. Pick any vertex $x$ and let its neighbours be 1, 2, 3 and 4. As we argued earlier in the previous cases, we can assume without loss of generality that if not all of 1, 2, 3 and 4 are in the vertex cover, then at most two of them are.

(a) The induced subgraph of 1, 2, 3 and 4 has an edge, say 12. Then 3 and 4 together cannot be in a vertex cover unless all of 1, 2, 3 and 4 are there. So branch as $\{1,2,3,4\}$ or $N(3)$ or $\{3\} \cup N(4)$ and the recurrence is

$$C(k) \leqslant 1 + 3C(k-4) \qquad (10)$$

since $|\{3\} \cup N(4)|$ can be four as 3 and 4 can be adjacent.

(b) There is no edge among 1, 2, 3 and 4. Three of these vertices (say 1, 2 and 3) share a common neighbour $y$ other than $x$. Then when not all of 1, 2, 3 and 4 are in a vertex cover, $x$ and $y$ must be. So branch as $\{1,2,3,4\}$ or $\{x,y\}$ and the recurrence is

$$C(k) \leqslant 1 + C(k-4) + C(k-2). \qquad (11)$$

(c) There is no edge among 1, 2, 3 and 4, each one of them has three neighbours other than $x$, and no three of them have a common neighbour other than $x$. Here there is a pair of vertices who have totally 5 neighbours other than $x$. (Otherwise consider 1 and 2; they should have two common neighbours other than $x$, and so 1 and 3 can have at most one common neighbour other than $x$ as any other common neighbour will also be a neighbour of 2. So together 1 and 3 have five neighbours other than $x$). Let it be 1 and 3. Consider vertices 2 and 4. If all of 1, 2, 3, 4 are not there in the vertex cover then 2 is not there, or 2 is there and 4 is not there, or both 2 and 4 are there (in which case 1 and 3 are not). So branch as $\{1,2,3,4\}$ or $N(2)$ or $\{2\} \cup N(4)$ or $\{2,4\} \cup N(\{1,3\})$ and the recurrence is

$$C(k) \leqslant 1 + 2C(k-4) + C(k-5) + C(k-8). \qquad (12)$$

It can be verified using induction on $k$ that $C(k) = d((1.324718)^k - 1)$ for some constant $d$ satisfies all the inequalities, the critical one being the inequality (3) (which is the same as the inequality (7)). 1.324718 is the positive real root (rounded to 6 decimal places) to the equation $c^3 - c - 1 = 0$ which comes from the recurrence in inequality (3). Note that it suffices to verify the solution on inequalities (3), (5), (8), (9), (10) and (12) only. For, $d((1.324718)^k - 1)$ satisfies inequality (1) follows from the proof of the inequality (5) and that it satisfies inequalities (2), (4), (6) and (11) follows from the proof of the inequality (3). It is also easy to construct example graphs where the search tree constructed in our algorithm has $C(k) = d((1.324718)^k - 1)$ nodes by making the algorithm to go through Case 2(a) always.

Thus the total running time of the algorithm is $\Theta((1.324718)^k k^2 + kn)$. □

## 4. Conclusions

Once a problem is known to be fixed-parameter tractable, it is still an interesting question to find the best possible $f(k)$ for the fixed-parameter complexity of the problem. As the dependence on $k$ in the runtime improves, this gives feasible algorithm for a larger range of the parameter. We have demonstrated this by providing an algorithm for the VERTEX COVER problem that runs in time $O((1.324718)^k k^2 + kn)$ by using the bounded search tree technique. This bound

improves the best previous bound of $O(2^k k^2 + kn)$ for the problem. Our algorithm also provides the best-known algorithm to solve the Minimum Vertex Cover problem. To find the minimum vertex cover, it takes $O((1.324718)^n n^2)$ time.

An immediate open question is whether the bound can be further improved. Perhaps a more interesting question is whether approaches similar to that developed here can be applied to other fixed-parameter tractable problem to improve the parameter functions $f(k)$.

## Acknowledgements

We thank Rod Downey for pointing out that the size of the input graph can be assumed to be $O(k^2)$ by reducing the problem to the kernel, i.e., by applying the first two steps of Buss' algorithm.

## References

[1] J.F. Buss, J. Goldsmith, Nondeterminism within P, SIAM J. Comput. 22 (1993) 560–572.

[2] L. Cai, J. Chen, Fixed parameter tractability and approximability of NP-hard optimization problems, in: Proc. 2nd Israel Symp. on Theory and Computing Systems, 1993, pp. 118–126.

[3] R.G. Downey, M.R. Fellows, Fixed parameter tractability and completeness I: Basic theory, SIAM J. Comput. 24 (1995) 873–921.

[4] R.G. Downey, M.R. Fellows, Fixed parameter tractability and completeness II: Completeness for $W[1]$, Theoret. Comput. Sci. 141 (1995) 109–131.

[5] R.G. Downey, M.R. Fellows, Parametrized Computational feasibility, in: P. Clote, J. Remmel (Eds.), Feasible Mathematics II, Birkhauser, Boston, MA, 1995, pp. 219–244.

[6] K. Mehlhorn, Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness, Springer, Berlin, 1984.

[7] C.H. Papadimitriou, M. Yannakakis, On limited nondeterminism and the complexity of the V–C dimension, J. Comput. System Sci. 53 (1996) 161–170.