Refined Memorisation for Vertex Cover

L. Sunil Chandran and Fabrizio Grandoni

Max-Planck-Institut für Informatik,
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
{sunil,grandoni}@mpi-sb.mpg.de

Abstract. Memorisation is a technique which allows to speed up exponential recursive algorithms at the cost of an exponential space complexity. This technique already leads to the currently fastest algorithm for fixed-parameter vertex cover, whose time complexity is $O(1.2832^k k^{1.5} + kn)$, where n is the number of nodes and k is the size of the vertex cover. Via a refined use of memorisation, we obtain a $O(1.2759^k k^{1.5} + kn)$ algorithm for the same problem. We moreover show how to further reduce the complexity to $O(1.2745^k k^4 + kn)$.

1 Introduction

A *vertex cover* of an undirected graph G is a subset C of nodes such that any edge is incident on at least one node in C. The *vertex cover problem* consists in deciding whether G admits a vertex cover of at most k nodes.

It is well known [6] that vertex cover is *fixed-parameter tractable*, that is it can be solved in $O(f(k)n^\alpha)$ steps, where n is the number of nodes, f is a function of the *parameter* k only, and α is a constant. A lot of effort was recently devoted to develop faster and faster *fixed-parameter* algorithms for vertex cover. The first fixed-parameter algorithm for vertex cover is the $O(2^k n)$ algorithm of Fellows [8] (see also [9]), which is based on a *bounded search tree* strategy. Buss and Goldsmith [2] proposed a $O(2^k k^{2k+2} + kn)$ algorithm, in which they introduced the *kernel reduction* technique. Combining their bounded search tree strategy with the kernel reduction of Buss and Goldsmith, Downey and Fellows [5] obtained a $O(2^k k^2 + kn)$ algorithm. The complexity was later reduced to $O(1.3248^k k^2 + kn)$ by Balasubramanian, Fellows and Raman [1], to $O(1.2918^k k^2 + kn)$ by Niedermeier and Rossmanith [10], and to $O(1.2906^k k + kn)$ by Downey, Fellows and Stege [7]. The current fastest polynomial-space algorithm is the $O(1.2852^k k + kn)$ algorithm of Chen, Kanj and Jia [3]. Thanks to the *interleaving technique* of Niedermeier and Rossmanith [11], it is possible to get rid of the polynomial factor in the exponential term of the complexity. For example the complexity of the algorithm of Chen et al. can be reduced to $O(1.2852^k + kn)$. It is worth to notice that, though such polynomial factor is not relevant from the asymptotic point of view (the base of the exponential factor is overestimated), it is usually indicated. The reason is that the value of k is assumed to be not "too" big.

Memorisation is a technique developed by Robson [14,15] in the context of maximum independent set, which allows to reduce the time complexity of many exponential time recursive algorithms at the cost of an exponential space complexity. The key-idea behind memorisation is that, if the same subproblem appears many times, it may be convenient to store its solution instead of recomputing such solution from scratch. With this technique Robson [15] managed to derive a $O(1.1889^n)$ exponential-space algorithm for maximum independent set from his own $O(1.2025^n)$ polynomial-space algorithm for the same problem. Memorisation cannot be applied to the algorithm of Chen et al., since their algorithm branches on subproblems involving graphs which are not induced subgraphs of the kernel. Niedermeier and Rossmanith [12,13] applied memorisation to their $O(1.2918^k k^2 + kn)$ polynomial-space algorithm for vertex cover, thus obtaining a $O(1.2832^k k^{1.5} + kn)$ exponential-space algorithm for the same problem, which is also the currently fastest algorithm for vertex cover. It is worth to notice that the polynomial factor in the exponential term cannot be removed any more with the interleaving technique of Niedermeier and Rossmanith. In fact, such technique is based on the idea that most of subproblems concern graphs with few nodes. This is not the case when memorisation is applied.

1.1 Our Results

The kind of memorisation which is currently applied to vertex cover is in some sense a weaker version of the technique originally proposed by Robson for maximum independent set. This is mainly due to the structural differences between the two problems. In this paper we present a simple technique which allows to get rid of these structural differences, thus allowing to apply memorisation to vertex cover in its full strength. By applying our refined technique to the $O(1.2918^k k^2 + kn)$ algorithm of Niedermeier and Rossmanith, we obtain a $O(1.2759^k k^{1.5} + kn)$ exponential-space algorithm for vertex cover. With a further refined technique, we reduce the complexity to $O(1.2745^k k^4 + kn)$.

2 Preliminaries

We use standard graph notation as contained for instance in [4]. An *undirected graph* G is a pair (V, E), where V is a set of n *nodes* and E is a set of m pairs of distinct nodes (*edges*). The *order* of G is n while its *size* is $n + m$. Two nodes v and w are *adjacent* if $\{v, w\} \in E$. The edge $\{v, w\}$ is *incident* on v and w. The set of nodes adjacent to a node v is denoted by $N(v)$. Given a subset W of nodes, by $G - W$ we denote the graph obtained by removing from G all the nodes in W and all the edges incident on them. The *induced subgraphs* of G are the graphs of the kind $G - W$, for any $W \subseteq V$. A *vertex cover* of an undirected graph $G = (V, E)$ is a subset C of V such that every edge in E is incident on at least one node in C. The *vertex cover problem* consists in deciding whether G admits a vertex cover of at most k nodes. A *minimum vertex cover* is a vertex cover of minimum cardinality. From now on we will consider a variant of vertex cover problem, in which the size $mvc(G)$ of the minimum vertex covers has to be returned, if it does not exceed k.

All the currently fastest algorithms for vertex cover are based on two key-ideas: *kernel reduction* and *bounded search trees*. The idea behind kernel reduction is to reduce (in polynomial time) the original problem (G, k) to an equivalent problem (G', k'), where $k' \leq k$ and the size of G', the *kernel*, is a function of k only. Buss and Goldsmith [2] showed how to obtain a kernel with $O(k^2)$ nodes and edges in $O(kn)$ time. Chen, Kanj and Jia [3] showed how to reduce the number of nodes in the kernel to at most $2k$ in $O(m\sqrt{n})$ steps. By combining the kernel reductions of Buss and Goldsmith and of Chen et al., one obtains [3] a $O(kn + k^3)$ algorithm to reduce the original problem to an equivalent problem (G', k'), where $k' \leq k$ and the order of G' is at most $2k'$.

Let us now consider the idea behind bounded search trees. For this purpose, let us define a function $fpvc(\cdot, \cdot)$ as follows:

$$fpvc(G, k) = \begin{cases} mvc(G) & \text{if } mvc(G) \leq k; \\ +\infty & \text{otherwise.} \end{cases}$$

Given a problem (F, h), the value of $fpvc(F, h)$ can be computed with a recursive algorithm of the following kind. If $h = 0$, $fpvc(F, h)$ is equal to zero if F contains no edges and it is equal to $+\infty$ otherwise. Otherwise, one solves (F, h) by *branching* on a set of subproblems $(F_1, h_1), (F_2, h_2) \ldots (F_b, h_b)$, with b upper bounded by a constant and $h_i < h$ for each $i \in \{1, 2 \ldots b\}$. These subproblems must satisfy the condition:

$$fpvc(F, h) = \min_{i \in \{1,2 \ldots b\}} \{h - h_i + fpvc(F_i, h_i)\}. \tag{1}$$

Thus the solution of the subproblems directly leads to the solution of (F, h).

To give an intuition of how subproblems are generated, let us consider a node v of F with neighborhood $N(v) = \{w, u\}$. Every vertex cover contains v or $N(v)$. This means that F admits a vertex cover of size at most h if and only if $F - \{v\}$ admits a vertex cover of size at most $h - 1$, or $F - \{w, u\}$ admits a vertex cover of size at most $h - 2$. In other words:

$$fpvc(F, h) = \min\{1 + fpvc(F - \{v\}, h - 1), 2 + fpvc(F - \{w, u\}, h - 2)\}.$$

Thus one can branch with the subproblems $(F - \{v\}, h-1)$ and $(F - \{w,u\}, h-2)$.

Let $C(h)$ denote the total number of search paths in the search tree which is created to solve a problem (F,h). For $h=0$, $C(h) = C(0) = 1$. Otherwise, let $(F_1,h_1),(F_2,h_2)\ldots(F_b,h_b)$ be the subproblems generated to solve (F,h). The following relation holds:

$$C(h) \leq \sum_{i=1}^{b} C(h_i).$$

The inequality above is satisfied by $C(h) = \hat{c}^h$, where $\hat{c}$ is the *branching factor* of the branching considered, that is the (unique [3]) positive root of the polynomial:

$$x^h - \sum_{i=1}^{b} x^{h_i}.$$

For example the branching factor associated with the branching at (F,h) on the subproblems $(F - \{v\}, h-1)$ and $(F - \{w,u\}, h-2)$ is the positive root $\hat{c} = 1.6180...$ of the polynomial $x^h - x^{h-1} - x^{h-2} = x^{h-2}(x^2 - x - 1)$.

The *branching factor* c of the algorithm is the maximum over all the branching factors associated to the branchings that the algorithm may execute. By induction, one obtains that $C(h)$ is $\mathrm{O}(c^h)$. This is also a valid upper bound on the number of nodes of the search tree.

The idea behind bounded search trees is to exploit the structure of the graph such as to obtain a branching factor c as small as possible. For this purpose, a tedious case analysis is often required.

Combining the recursive algorithm above with the kernel reduction algorithm of Chen et al., one obtains a $\mathrm{O}(c^k k^\beta + kn)$ algorithm for vertex cover, where the cost of branching at a given subproblem is $\mathrm{O}(k^\beta)$. This complexity can be reduced to $\mathrm{O}(c^k + kn)$ with the interleaving technique of Niedermeier and Rossmanith [11].

Let us now shortly describe how Niedermeier and Rossmanith applied memorisation to vertex cover [12,13]. Let $\mathcal{A}$ be an algorithm of the kind described above, with the extra condition that, when it branches at a problem (F,h), the corresponding subproblems (F_i,h_i) involve graphs F_i which are induced subgraph of F. In particular, let us consider the fastest algorithm which satisfies this extra condition, which is the $\mathrm{O}(c^k k^2 + kn) = \mathrm{O}(1.2918^k k^2 + kn)$ algorithm of Niedermeier and Rossmanith [10]. From $\mathcal{A}$ one derives a new algorithm $\mathcal{A}'$ in the following way. For any induced subgraph F of the kernel of at most $2\alpha k$ nodes, for a given $\alpha \in (0,1)$, $\mathcal{A}'$ solves (by using $\mathcal{A}$) the problem $(F, \alpha k)$ and it stores the pair $(F, fpvc(F, \alpha k))$ in a database. Note that, since F is an induced subgraph of the kernel, one can simply store the set of nodes of F instead of F. Then $\mathcal{A}'$ works in the same way as $\mathcal{A}$, with the following difference. When the parameter in a subproblem reaches or drops below the value αk, $\mathcal{A}'$ performs a kernel reduction. This way, $\mathcal{A}'$ generates a new subproblem (F,h), where $h \leq \alpha k$ and the order of F is at most $2\alpha k$. Thus $\mathcal{A}'$ can easily derive the value of $fpvc(F,h)$ from the value of $fpvc(F, \alpha k)$ which is stored in the database.

The cost of solving each subproblem $(F, \alpha k)$ is $\mathrm{O}(c^{\alpha k})$. The number of pairs stored in the database is at most $2\binom{2k}{2\alpha k}$, which is $\mathrm{O}(k^{-0.5}(\alpha^\alpha(1-\alpha)^{1-\alpha})^{-2k})$ from Stirling's approximation. Thus the database can be created in $\mathrm{O}(c^{\alpha k} k^{-0.5}(\alpha^\alpha(1-\alpha)^{1-\alpha})^{-2k})$ time. The search tree now contains $\mathrm{O}(c^{(1-\alpha)k})$ nodes (that is the upper bound on $C(k)$ which is obtained by assuming $C(\alpha k) = 1$). The cost associated to each node is $\mathrm{O}(k^2)$, not considering the leaves for which the costs of the query to the database and of the kernel reduction have to be taken into account. In particular, the database can be implemented such as that the cost of each query is $\mathrm{O}(k)$. Moreover each kernel reduction costs $\mathrm{O}(k^{2.5})$ (since each graph contains $\mathrm{O}(k)$ nodes and $\mathrm{O}(k^2)$ edges). Thus the cost to create the search tree is $\mathrm{O}(c^{(1-\alpha)k} k^{2.5})$. The value of α has to be chosen such as to balance the cost of creating the search tree and the database. The optimum is reached when $\alpha > 0.0262$ satisfies:

$$c^{1-\alpha} = c^\alpha \left(\frac{1}{\alpha^\alpha (1-\alpha)^{1-\alpha}} \right)^2.$$

Thus one obtains a $\mathrm{O}(c^{(1-\alpha)k} k^{2.5} + kn) = \mathrm{O}(1.2832^k k^{2.5} + kn)$ time complexity.

The complexity can be slightly reduced in the following way. All the nodes of degree greater than 6 are filtered out in a preliminary phase (in which the algorithm stores no solution in the database and does not perform kernel reductions). In particular, let (F, h) be a subproblem where F contains a node v with $|N(v)| > 6$. The algorithm branches on the subproblems $(F - \{v\}, k-1)$ and $(F - N(v), k - |N(v)|)$. The number of subproblem generated in the preliminary phase is $O(\tilde{c}^k)$, where $\tilde{c} < 1.256 < c$. The cost of branching at these subproblems is $O(k^2)$. Thus the total cost to remove "high" degree nodes is $O(1.256^k k^2)$. All the subproblems generated after the preliminary phase, involve subgraphs with $O(k)$ nodes and edges. This means that the kernel reductions can be performed in $O(k^{1.5})$ time only. This way the complexity of the algorithm is reduced to $O(1.2832^k k^{1.5} + kn)$.

3 Refined Memorisation for Vertex Cover

In this section we present a refined way to apply memorisation to vertex cover. The complexity analysis are based on the $O(c^k k^2 + kn) = O(1.2918^k k^2 + kn)$ algorithm of Niedermeier and Rossmanith [10], which is the currently fastest algorithm for vertex cover which is compatible with memorisation (since it branches on subproblems involving only induced subgraphs of the kernel). It is worth to mention that the technique proposed can be easily adapted to other algorithms of similar structure.

The rest of this section is organized as follows. In Section (3.1), we present a variant of the exponential-space algorithm of Niedermeier and Rossmanith [12,13], of complexity $O(1.2829^k k^{1.5} + kn)$. The reduction in the complexity is achieved via a more efficient use of the database. In Section (3.2), we show how to reduce the complexity to $O(1.2759^k k^{1.5} + kn)$ by branching on subproblems involving connected induced subgraphs only. In Section (3.3), the complexity is further reduced to $O(1.2745^k k^4 + kn)$.

3.1 A More Efficient Use of the Database

Our algorithm works as follows. No database is created a priori. Then, the algorithm works as the algorithm of Niedermeier and Rossmanith (as described in Section (2)), with the following differences. There is no value $h > 0$ of the parameter for which the recursion is stopped. Let (F, h) be a problem generated after the preliminary phase. Before branching on the corresponding subproblems $(F_1, h_1), (F_2, h_2) \ldots (F_b, h_b)$, the algorithm applies a kernel reduction to each subproblem (no matter which is the value of the parameters h_i). In particular, the algorithm branches on a set of subproblems $(F'_1, h'_1), (F'_2, h'_2) \ldots (F'_b, h'_b)$, where the order of F'_i is at most $2h'_i$ (*order-condition*). The reason of enforcing the order-condition will be clearer in the analysis. Observe that this does not modify the branching factor of the algorithm. When (F, h) is solved, the triple $(F, h, fpvc(F, h))$ is stored in a database. In this case also, before solving (F, h), the algorithm checks whether the solution is already available in the database. This way one ensures that a given subproblem is solved at most once.

The cost of the preliminary phase (in which "high" degree nodes are removed from the kernel) is not modified. The cost associated to each node of the search tree, after the preliminary phase, is $O(k^{1.5})$. Let $P(h)$ denote the number of subproblems with parameter h (after the preliminary phase). From the analysis of Section (2), $P(h)$ is $O(c^{k-h})$. The subproblems considered involve induced subgraphs of the kernel of order at most $2h$. Let $N(2k, 2h)$ denote the number of such induced subgraphs. Since no subproblem is solved more than once, $P(h)$ is $O(\min\{c^{k-h}, N(2k, 2h)\})$. For $h > k/4$:

$$\min\{c^{k-h}, N(2k, 2h)\} \leq c^{k-h} \leq c^{3k/4} = O(1.2118^k).$$

For $h \leq k/4$, $N(2k, 2h) \leq 2\binom{2k}{2h}$. As h decreases from $k/4$ to 0, the function $\min\{c^{k-h}, 2\binom{2k}{2h}\}$ first increases, then reaches a peak, and eventually decreases. The peak is reached for $\tilde{h} = \alpha k + O(1)$, where $\alpha > 0.0271$ satisfies:

$$c^{1-\alpha} = \left(\frac{1}{\alpha^\alpha (1-\alpha)^{1-\alpha}}\right)^2.$$

Thus $P(h)$ is $O(c^{(1-\alpha)k}) = O(1.2829^k)$. This is also an upper bound on the number of nodes in the search tree. Then the complexity of the algorithm proposed is $O(c^{(1-\alpha)k}k^{1.5} + kn) = O(1.2829^k k^{1.5} + kn)$.

Besides the time complexity, an important difference between the algorithm proposed in this section and the exponential-space algorithm of Niedermeier and Rossmanith is the role played by the parameter α. In the algorithm of Niedermeier and Rossmanith, α influences the behavior of the algorithm, and thus it has to be fixed carefully. In our algorithm instead, α only appears in the complexity analysis.

3.2 Branching on Connected Induced Subgraphs

In previous section we described an algorithm $\mathcal{A}$ which makes use of a database in which it stores triples of the kind $(F, h, fpvc(F, h))$, where F is an induced subgraph of the kernel of order at most $2h$ (order-condition). The graphs F stored in the database may not be connected. We will now show how to derive from $\mathcal{A}$ an algorithm $\mathcal{A}_C$ with the same branching factor as $\mathcal{A}$, which branches only on subproblems involving induced subgraphs which are connected (besides satisfying the order-condition).

The difference between $\mathcal{A}_C$ and $\mathcal{A}$ is the way $\mathcal{A}_C$ branches at a given subproblem after the preliminary phase. Let (F, h) be a problem generated after the preliminary phase and (F_1, h_1), $(F_2, h_2) \dots (F_b, h_b)$ be the corresponding subproblems generated by $\mathcal{A}$. Let moreover $F_{i,1}, F_{i,2} \dots F_{i,p_i}$ be the connected components of F_i. A naive idea could be to branch on the subproblems $(F_{i,j}, h_i)$, for each $i \in \{1, 2 \dots b\}$ and for each $j \in \{1, 2 \dots p_i\}$ (the graphs $F_{i,j}$ are connected and satisfy the order-condition). In fact, it is not hard to show that:

$$fpvc(F_i, h_i) = \begin{cases} \sigma_i & \text{if } \sigma_i \leq h_i; \\ +\infty & \text{otherwise,} \end{cases}$$

where σ_i is the sum of the solutions returned by the subproblems $(F_{i,j}, h_i)$:

$$\sigma_i = \sum_{j=1}^{p_i} fpvc(F_{i,j}, h_i).$$

Once the values $fpvc(F_i, h_i)$ are available, the value of $fpvc(F, h)$ can be easily derived from Equation (1).

Though this approach is correct in principle, it may lead to a bad branching factor. The reason is that one may generate many more than b subproblems, without decreasing the parameter in each subproblem properly. To avoid this problem, one can use the following simple observation. The size of the minimum vertex covers of the connected components with less than $6\ell + 2$ nodes, for a fixed positive integer ℓ, can be computed in constant time. Thus one does not need to branch on them. The size of the minimum vertex covers of the remaining connected components is lower bounded by ℓ (since they contain at least $6\ell + 1$ edges and their degree is upper bounded by 6). This lower bound can be used to reduce the parameter in each subproblem.

In more details, for each connected component H of F_i with less than $6\ell+2$ nodes, the algorithm computes $mvc(H)$ by brute force, and this value is added to a variable Δ_i (which is initialized to 0). Let $F_{i,1}, F_{i,2} \dots F_{i,b_i}$ be the remaining connected components of F_i. The minimum vertex covers of each $F_{i,j}$ have cardinality at least ℓ. This implies that, if the size of the minimum vertex covers of some $F_{i,j}$ is greater than $h_i - (b_i - 1)\ell$, then $fpvc(F_i, h_i) = +\infty$. Thus one can replace the subproblem $(F_{i,j}, h_i)$ with a subproblem $(F_{i,j}, h_i - (b_i - 1)\ell)$. Note that the order-condition is satisfied by the new subproblem since each $F_{i,j}$ contains at most $2h_i - (b_i - 1)(6\ell + 2)$ nodes. Let σ'_i be the sum of the solutions returned by the subproblems $(F_{i,j}, h_i - (b_i - 1)\ell)$:

$$\sigma'_i = \sum_{j=1}^{b_i} fpvc(F_{i,j}, h_i - (b_i - 1)\ell).$$

The value of $fpvc(F_i, h_i)$ is given by:

$$fpvc(F_i, h_i) = \begin{cases} \Delta_i + \sigma'_i & \text{if } \Delta_i + \sigma'_i \leq h_i; \\ +\infty & \text{otherwise.} \end{cases}$$

In this case also one does not need to compute the solutions of the subproblems $(F_{i,j}, h_i - (b_i - 1)\ell)$ which are already available in the database.

Let $\hat{c}$ and $\hat{c}_C$ be the branching factors corresponding to the branching on (F, h) of the algorithms $\mathcal{A}$ and $\mathcal{A}_C$ respectively. We can decompose the branching of $\mathcal{A}_C$ on (F, h) in $b+1$ branchings. First there is one branching on the subproblems $(F_1, h_1), (F_2, h_2) \dots (F_b, h_b)$. Then, for each subproblem (F_i, h_i), $i \in \{1, 2 \dots b\}$, there is one branching on the "big" connected components of F_i. The first branching has branching factor $\hat{c}$. The other branchings have branching factors $\hat{c}_1, \hat{c}_2 \dots \hat{c}_b$ respectively, where:

$$\hat{c}_i = \begin{cases} 1 & \text{if } b_i = 1; \\ b_i^{\frac{1}{(b_i-1)\ell}} \leq 2^{1/\ell} & \text{otherwise.} \end{cases}$$

The value of $\hat{c}_C$ is upper bounded by the maximum over $\hat{c}$ and $\hat{c}_1, \hat{c}_2 \dots \hat{c}_b$. Thus, to ensure that the branching factor of $\mathcal{A}_C$ is not greater than the branching factor c of $\mathcal{A}$, $c \cong 1.2918$, it is sufficient to fix ℓ such that $2^{1/\ell} < c$. In particular, we can assume $\ell = 3$.

Let us now consider the complexity of $\mathcal{A}_C$. The cost of the preliminary phase is not modified. We can both filter out small connected components and search in the database in $O(k)$ time (with a careful implementation of the database). Thus the cost associated to each branching is the same as in $\mathcal{A}$, that is $O(k^{1.5})$. Let $R_d(m, n)$ be the number of connected induced subgraphs of order m which are contained in a graph of order n and degree upper bounded by d. Robson [14] showed that $R_d(m, n)$ is $O\left(\frac{n}{m}\left(\frac{(d-1)^{d-1}}{(d-2)^{d-2}}\right)^m\right)$. In particular, $R_6(2h, 2k)$ is $O(\frac{k}{h}(\frac{5^5}{4^4})^{2h})$. It turns out that this is also a valid upper bound for the number $N(2k, 2h)$ of the subproblems concerning graphs of order at most $2h$ which are generated after the preliminary phase. From this bound and the analysis of Section (3.1), the time complexity of $\mathcal{A}_C$ is $O(c^{(1-\alpha)k}k^{1.5} + kn) = O(1.2759^k k^{1.5} + kn)$, where

$$\alpha = \frac{\log(c)}{\log(c) + 2\log(\frac{5^5}{4^4})} > 0.04867.$$

3.3 A Further Refinement

In Section (3.2) we showed that restricting the class of graphs which are stored in the database, without increasing the branching factor, leads to a faster algorithm. In particular, the algorithm of Section (3.2) stores connected induced subgraphs only (instead of general induced subgraphs). In this section we show how it is possible to store only connected induced subgraphs of degree lower bounded by 2. This leads to a $O(1.2745^k k^4 + kn)$ algorithm for vertex cover.

Nodes of degree zero can be safely removed (since they do not belong to any minimum vertex cover). Then we just need to take care of nodes of degree one. Let v be a node of degree one, with $N(v) = \{w\}$. It is not hard to show that there exists a minimum vertex cover which contains w and does not contain v. Since we do not need to find all the minimum vertex covers (of a given size), but only one, we can simply assume that w is in the vertex cover and remove v and w from the graph. Thus nodes of degree one can be filtered out in linear time. Observe that, if one starts with a problem (F, h) which satisfies the order-condition, the graph obtained after removing nodes of degree at most one satisfies the order-condition too. Moreover this filtering out does not increase the branching factor of the algorithm.

Let $R_{2,d}(m, n)$ be the number of connected induced subgraphs of order m and degree lower bounded by 2 which are contained in a graph of order n and degree upper bounded by d. Robson [15] showed that $R_{2,d}(m, n)$ is $O(r_d^m \, n \, poly_d(m))$, where, for any fixed d, $poly_d(m)$ is a polynomial of m and r_d is a constant which comes from a maximization problem (in particular, $r_6 < 9.927405$). Following the proof of Robson, it is not hard to derive that $poly_d(m)$ is $O(m^d)$. With a more

careful analysis, one obtains that $poly_d(m)$ is $O(m^{\frac{d-3}{2}})$. Thus $R_{2,6}(2h, 2k)$ is $O(r_6^{2h} poly_6(h)k) = O(9.927405^{2h} h^{1.5} k)$. By replacing $R_6(2h, 2k)$ with $R_{2,6}(2h, 2k)$ in the analysis of Section (3.2), one obtains:

$$\alpha = \frac{\log(c)}{\log(c) + 2\log(r_6)} > 0.05282,$$

and thus a $O(c^{(1-\alpha)k} k^{2.5} k^{1.5} + kn) = O(1.2745^k k^4 + kn)$ time complexity.

References

1. R. Balasubramanian, M. Fellows, and V. Raman. An improved fixed-parameter algorithm for vertex cover. *Information Processing Letters*, 65:163–168, 1998.
2. J. F. Buss and J. Goldsmith. Nondeterminism within P. *SIAM Journal on Computing*, 22(3):560–572, 1993.
3. J. Chen, I. Kanj, and W. Jia. Vertex cover: further observations and further improvements. *Journal of Algorithms*, 41:280–301, 2001.
4. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT Press and McGraw-Hill Book Company, 6th edition, 1992.
5. R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness. II. On completeness for $W[1]$. *Theoretical Computer Science*, 141(1-2):109–131, 1995.
6. R. G. Downey and M. R. Fellows. *Parameterized complexity*. Monographs in Computer Science. Springer-Verlag, 1999.
7. R. G. Downey, M. R. Fellows, and U. Stege. Parameterized complexity: A framework for systematically confronting computational intractability. In J. K. F. Roberts and J. Nesetril, editors, *Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future*, volume 49 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 49–99, 1999.
8. M. R. Fellows. On the complexity of vertex set problems. Technical report, Computer Science Department, University of New Mexico, 1988.
9. K. Mehlhorn. *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*. Springer-Verlag, 1984.
10. R. Niedermeier and P. Rossmanith. Upper bounds for vertex cover further improved. In *Symposium on Theoretical Aspects of Computer Science*, pages 561–570, 1999.
11. R. Niedermeier and P. Rossmanith. A general method to speed up fixed-parameter-tractable algorithms. *Information Processing Letters*, 73(3–4):125–129, 2000.
12. R. Niedermeier and P. Rossmanith. Private communication, 2003.
13. R. Niedermeier and P. Rossmanith. On efficient fixed-parameter algorithms for weighted vertex cover. *Journal of Algorithms*, 47(2):63–77, 2003.
14. J. M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7(3):425–440, 1986.
15. J. M. Robson. Finding a maximum independent set in time $O(2^{n/4})$. Technical Report 1251-01, LaBRI, Université Bordeaux I, 2001.