Theoretical Computer Science 411 (2010) 3736–3756

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Improved upper bounds for vertex cover

Jianer Chen [a], Iyad A. Kanj [b], Ge Xia [c,*]

[a] Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843, USA
[b] The School of CTI, DePaul University, 243 S. Wabash Avenue, Chicago, IL 60604, USA
[c] Department of Computer Science, Acopian Engineering Center, Lafayette College, Easton, PA 18042, USA

A R T I C L E   I N F O

Article history:
Received 7 August 2009
Received in revised form 16 June 2010
Accepted 22 June 2010
Communicated by D.-Z. Du

Keywords:
Vertex cover
Parameterized algorithm
NP-complete problem
Branch-and-search

A B S T R A C T

This paper presents an $O(1.2738^k + kn)$-time polynomial-space algorithm for Vertex Cover improving the previous $O(1.286^k + kn)$-time polynomial-space upper bound by Chen, Kanj, and Jia. Most of the previous algorithms rely on exhaustive case-by-case branching rules, and an underlying conservative worst-case-scenario assumption. The contribution of the paper lies in the simplicity, uniformity, and obliviousness of the algorithm presented. Several new techniques, as well as generalizations of previous techniques, are introduced including: *general folding*, *struction*, *tuples*, and *local amortized analysis*. The algorithm also improves the $O(1.2745^k k^4 + kn)$-time exponential-space upper bound for the problem by Chandran and Grandoni.

1. Introduction

Deriving upper bounds for NP-hard problems is important from both the practical and theoretical perspectives. Practically, an algorithm of running time $O(1.01^n)$ (n is the input size) could render an NP-hard problem computational feasible for many practical instances (say for $n \leq 1000$) as opposed to an $O(2^n)$ time algorithm for the problem. Theoretically, deriving upper bounds for an NP-hard problem helps studying the inherent structural complexity of the problem which can lead to a deeper understanding of the problem itself, and in general, of the structure of NP-hard problems. As a result, the study of exact algorithms for NP-hard problems has been attracting a lot of attention recently [1,24,34]. In particular, for many well-known NP-hard problems with important applications such as Satisfiability, Independent Set, Vertex Cover, and Graph Coloring, exact algorithms have been extensively studied and developed.

The current paper focuses on the parameterized Vertex Cover problem, abbreviated VC henceforth: given a graph G and a parameter k, decide if G has a vertex cover of at most k vertices. This problem was amongst the first few problems that were shown to be NP-hard [22]. In addition, the problem has been a central problem in the study of parameterized algorithms [16], and has applications in areas such as computational biochemistry and biology [7]. Since the development of the first parameterized algorithm for the problem by Buss and Goldsmith which runs in $O(kn + 2^k k^{2k+2})$ time [4], there has been an impressive list of improved algorithms for the problem [2,8,9,15,27,29,33]. The most recent algorithm for the problem running in polynomial space, was presented in 1999 and gives the currently best time upper bound of $O(kn + 1.286^k)$ [8]. Algorithms using exponential space for the problem have also been proposed [6,8,29], amongst which the best runs in time $O(1.2745^k k^4 + kn)$ [6]. Most of the previous algorithms rely on exhaustive case-by-case branching rules, and work under a conservative worst-case-scenario assumption. The analysis of these algorithms would consider the worst-case branch over numerous combinatorial cases, and derive an upper bound accordingly. In particular, the design phase of these algorithms (usually) did not provide the appropriate ground that the analysis phase could take advantage of to derive better upper

* Corresponding author. Tel.: +1 610 330 5415; fax: +1 610 330 5059.
E-mail addresses: chen@cs.tamu.edu (J. Chen), ikanj@cs.depaul.edu (I.A. Kanj), gexia@cs.lafayette.edu, xiag@lafayette.edu (G. Xia).

0304-3975/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2010.06.026

bounds than the ones claimed. Consequently, to improve the upper bounds, larger and larger sets of local structures had to be examined and processed differently. Examining these numerous structures and processing them differently on a case-by-case basis became very tedious, rendering the verification and implementation of these algorithms very complicated and impractical.

On the other hand, progress has been recently made on deriving computational lower bounds for the problem. It has been shown that unless all SNP problems are solvable in sub-exponential time, there is a constant $c_0 > 1$ such that the VC problem cannot be solved in time $c_0^k n^{O(1)}$ [5,23]. Therefore, from both the algorithmic and the complexity points of view, it becomes important to study how far we can push to lower the constant $c > 1$, such that the VC problem can be solved in time $c^k n^{O(1)}$.

In this paper we adopt a different approach to improve the time upper bound for the VC problem. Our goal was to design an algorithm that is simple and uniform, and that provides the tools and the basis for a tighter analysis of its running time. We came up with an algorithm that is very simple. The algorithm keeps a list of prioritized "advantageous" structures at its disposal. At each stage it will pick the structure of highest priority (most advantageous structure). Picking such a structure can be easily done following few simple rules. When this structure is picked, the algorithm processes this structure very *uniformly*, and *obliviously*, in a way that is almost independent of what the structure is. As a matter of fact, there are *only* two different ways for processing *any* structure – that is, only two different branches – that the algorithm needs to distinguish. All the other operations performed by the algorithm are non-branching operations that process certain simple structures in the graph such as degree-1 and degree-2 vertices, and that set the stage for the subsequent branch performed by the algorithm to be efficient. The interleaving and ordering of these operations in the algorithm is very crucial, and is fully exploited by the analysis phase. The analysis phase, however, is lengthy, showing that regardless of the structure picked, the oblivious branching performed by the algorithm will yield the desired upper bound.

We note that a very effective technique introduced by Fomin et al. [21], called "measure and conquer", has been recently used to analyze the time complexity of exact algorithms for NP-hard problems. This technique allows for the design of extremely simple algorithms, and then for performing a careful, yet complicated, analysis to derive upper bounds on their time complexity that are much better than the upper bounds derived under a conservative approach. Using this technique, researchers were able to obtain the currently-best time upper bound for several important NP-hard problems, such as Dominating Set [21] and Independent Set [20]. Our approach shares the spirit of this technique in terms of the simplicity of the presented algorithm and its careful analysis, but our approach applies to parameterized problems, which have not been addressed by "measure and conquer".

To be able to carry out all the above, a set of new techniques and generalizations of some well-known and classical techniques have been introduced. We developed a graph operation that is a generalization of the *folding* operation [8], and a graph operation that is a specialization of the *struction* operation by Ebengger et al. [17]. These operations help the algorithm remove several simple structures from the graph without the need to perform any branching. This makes analyzing the two branching operations performed in the resulting graph more insightful. The notion of a *tuple*, which was implicitly used by Robson [32], has been fully developed and exploited to prune the search space. Finally we perform a "local" amortized analysis to balance expensive branching operations by combining them with more efficient operations. Being able to perform this local amortized analysis is due to the careful interleaving and ordering of the operations in the algorithm, and not to the different way of processing each structure.

The presented algorithm runs in polynomial space, and has its running time bounded by $O(1.2738^k + kn)$. This is a significant improvement over the previous polynomial-space algorithm for the problem which runs in $O(1.286^k + kn)$ time. This also improves the exponential space $O(1.2745^k k^4 + kn)$-time algorithm by Chandran and Grandoni [6].

2. Preliminaries and structural results

For a graph G we denote by $|G|$ the number of vertices in G. For a vertex v in G we denote by $N(v)$ the set of neighbors of v, $N[v]$ the set $N(v) \cup \{v\}$, and $d(v)$ the degree of v in G. For a set of vertices S in G, let $N(S) = \bigcup_{v \in S} N(v) - S$ denote the set of neighbors of the vertices in S, and $N[S]$ the set $N(S) \cup S$. Let $\tau(G)$ denote the size of a minimum vertex cover of G. The following proposition from [8] is based on a theorem by Nemhauser and Trotter [26], usually referred to as the NT-theorem or the NT-decomposition.

Proposition 2.1 ([8]). *There is an algorithm of running time $O(kn + k^3)$ that, given an instance (G, k) of the* VC *problem where $|G| = n$, constructs another instance (G_1, k_1) of* VC *with $k_1 \le k$ and $|G_1| \le 2k_1$, such that $\tau(G) \le k$ if and only if $\tau(G_1) \le k_1$.*

We say that the instance (G_1, k_1) is the *kernel* of the instance (G, k). The NT-decomposition of (G, k) into (G_1, k_1) is said to be *non-trivial* if $|G_1| < |G|$. Proposition 2.1 allows us to assume, without loss of generality, that in an instance (G, k) of the VC problem the graph G contains at most $2k$ vertices.

For two vertices u and v we say that $\{u, v\}$ is an *anti-edge* in G if (u, v) is not an edge in G. Let v_0 be a vertex in G with a set of neighbors $\{v_1, \ldots, v_p\}$. Construct a graph G' as follows: (1) remove the vertices $\{v_0, v_1, \ldots, v_p\}$ from G and introduce a new node v_{ij} for every anti-edge $\{v_i, v_j\}$ in G where $0 < i < j \le p$; (2) add an edge (v_{ir}, v_{js}) if $i = j$, $r \ne s$ and (v_r, v_s) is an edge in G; (3) if $i \ne j$ add an edge (v_{ir}, v_{js}); and (4) for every $u \notin \{v_0, \ldots, v_p\}$, add the edge (v_{ij}, u) if (v_i, u) or (v_j, u) is an edge in G. This completes the construction of G'. We say that the graph G' is obtained from G by applying the *struction* operation to the vertex v_0 in G [17] (see Fig. 1 for an illustration).

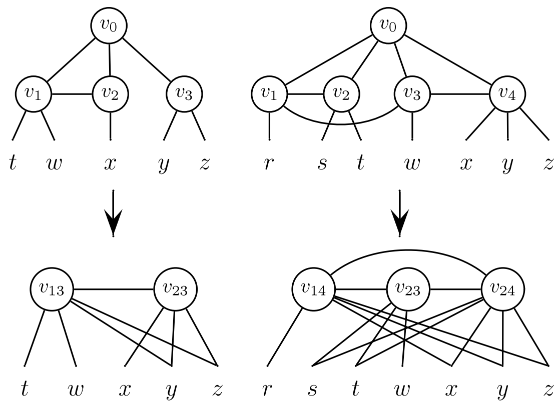

Fig. 1. The struction operation.

Lemma 2.2. *Let v_0 be a vertex in G with a set of neighbors $\{v_1, \ldots, v_p\}$. Suppose that there are at most $p - 1$ anti-edges among the vertices $\{v_1, \ldots, v_p\}$, and let G' be the graph obtained from G by applying the struction operation to the vertex v_0. Then $\tau(G') \leq \tau(G) - 1$.*

Proof. Let $\alpha(G)$ and $\alpha(G')$ denote the size of a maximum independent set in G and G', respectively. It was shown in [17] that $\alpha(G') = \alpha(G) - 1$. Let n and n' denote the number of vertices in G and G', respectively. Since there are at most $p - 1$ anti-edges among the vertices $\{v_1, \ldots, v_p\}$, the number of newly introduced vertices in G' is at most $p - 1$. Since $p + 1$ vertices were removed from G, namely $\{v_0, v_1, \ldots, v_p\}$, we have $n' \leq n - 2$. It is well-known [22] that for any graph H we have $\alpha(H) + \tau(H) = |H|$. Therefore $\tau(G') = n' - \alpha(G') \leq (n - 2) - (\alpha(G) - 1) = \tau(G) - 1$. This completes the proof. □

Lemma 2.2 gives a generic setting in which the application of the struction operation reduces the size of the minimum vertex cover of the graph. We will assume that we have a subroutine called **Struction()** that applies the struction operation to a vertex v in G when the assumptions of Lemma 2.2 holds. This operation turns out to be very useful in the algorithm presented in this paper. Two possible scenarios in which the operation will be applied are illustrated in Fig. 1. Note that the time spent by this operation on a vertex v is proportional to $|N(v)|$.

Remark 2.3. Note that when the struction operation is applied to a degree-3 vertex u in G with neighbors v, w, and z, and with an edge between v and w, the only vertices removed from G are u, v, w, and z, and the only vertices of G in the resulting graph whose degree could have increased are the neighbors of z.

Next we present an operation that generalizes the folding operation introduced in [8].

Lemma 2.4. *Let $I \neq \emptyset$ be an independent set in G and let $N(I)$ be the set of neighbors of I. Suppose that $|N(I)| = |I| + 1$, and that for every subset $\emptyset \neq S \subseteq I$ we have $|N(S)| \geq |S| + 1$.*

1. *If the graph induced by $N(I)$ is not an independent set, then there exists a minimum vertex cover in G that includes $N(I)$ and excludes I.*
2. *If the graph induced by $N(I)$ is an independent set, let G' be the graph obtained from G by removing $I \cup N(I)$ and adding a vertex u_I, then connecting u_I to every vertex $v \in G'$ such that v was a neighbor of a vertex $u \in N(I)$ in G. Then $\tau(G') = \tau(G) - |I|$.*

Proof. We first prove the following claim: There exists a minimum vertex cover C for G such that C contains I and excludes $N(I)$, or such that C contains $N(I)$ and excludes I. To see why this is true, suppose that $C \cap I = X$ and $C \cap N(I) = Y$. Since C is a vertex cover for G, if $X = \emptyset$ then $Y = N(I)$, and similarly if $Y = \emptyset$ then $X = I$. Therefore we can assume that both X and Y are nonempty. Since C is a vertex cover for G, we have $N(I - X) \subseteq Y$. If $(I - X) \neq \emptyset$, $|Y| \geq |N(I - X)| \geq |I - X| + 1 = |I| - |X| + 1$, from the statement of the lemma. If $I - X = \emptyset$, since $Y \neq \emptyset$, we also have $|Y| \geq |I| - |X| + 1$. Therefore $|Y| + |X| \geq |I| + 1 = |N(I)|$. Since I is an independent set, if we replace $Y \cup X$ by $N(I)$ in C we get a vertex cover C' for G of size not larger than that of C. It follows that C' is a minimum vertex cover for G that includes $N(I)$ and excludes I, and the claim follows.

Let C be a minimum vertex cover that satisfies the conditions in the claim. If the graph induced by $N(I)$ is not an independent set, then any vertex cover of G, and in particular C, cannot exclude $N(I)$. It follows from the above claim that C a minimum vertex cover for G that includes $N(I)$ and excludes I. This proves part (1) in the statement of the lemma.

Suppose now that $N(I)$ is an independent set. If C contains I, then C excludes $N(I)$ and must include $N(N(I))$ in G'. Then $C' = C - I$ is a vertex cover for G' of size $|C| - |I| = \tau(G) - |I|$, and $\tau(G') \leq \tau(G) - |I|$. If C contains $N(I)$, then $(C - N(I)) \cup \{u_I\}$ is a vertex cover for G' of size $\tau(G) - (|I| + 1) + 1 = \tau(G) - |I|$, and $\tau(G') \leq \tau(G) - |I|$. This shows that $\tau(G') \leq \tau(G) - |I|$.

On the other hand, let C' be a minimum vertex cover for G'. Then either C' contains u_I or contains $N(u_I)$ and excludes u_I. If C' contains u_I, then $(C' - \{u_I\}) \cup N(I)$ is a vertex cover for G of size $|C'| + |I|$, and $\tau(G') \geq \tau(G) - |I|$. If C' contains $N(u_I)$ and excludes u_I, then $C' \cup I$ is a vertex cover for G of size $|C'| + |I|$, and $\tau(G') \geq \tau(G) - |I|$. This shows that $\tau(G') \geq \tau(G) - |I|$.

It follows from the above that $\tau(G') = \tau(G) - |I|$. This proves part (2) in the statement of the lemma, and the proof is complete. □

Let us call a structure $(I, H = N(I))$ satisfying the conditions in Lemma 2.4 an *almost-crown* structure. Before we discuss how an almost-crown structure can be found in a graph, we need to review a related technique called "crown reduction" that was first introduced by Chor et al. [12] (also briefly described in [18]), and has been subsequently applied to a number of parameterized problems, for example see [13,19,30,31,25].

Let $I \neq \emptyset$ be an independent set in a graph and let $H = N(I)$. If there exists a matching in G that matches H into I, the structure (I, H) is called a *crown* [12]. Note that this implies that $|H| \leq |I|$. We will call (I, H) a *strict crown* if $|H| < |I|$ and an *equal crown* if $|H| = |I|$. The graph G is said to be *crown-free* if G does not contain any crown [11]. It was shown in [11] that G is crown-free if and only if the NT-decomposition of G is trivial. Note that an *almost-crown* is not a crown because an almost-crown allows $|H| = |I| + 1$ while a crown requires $|H| \leq |I|$.

Lemma 2.5. *Let G be a crown-free graph. Then G has an almost-crown structure if and only if there exists a vertex $v \in G$ such that $G - v$ has an equal crown.*

Proof. Let G be a crown-free graph. Suppose first that G has an almost-crown $(I, H = N(I))$. Then $|H| = |I| + 1$, and for every $\emptyset \subsetneq S \subseteq I$ we have $|N(S)| \geq |S| + 1$. Let v be any vertex in H, and let $H' = H - v$. Note that $|H'| = |I|$. It is not difficult to see that (I, H') is an equal crown in $G - v$. In effect, $H' = N(I)$ in $G - v$. Moreover, since G is crown-free, for every nonempty set $S \subseteq I$ we have $|N(S)| \geq |S| + 1$ in G. This implies that $|N(S)| \geq |S|$ for every nonempty set $S \subseteq I$ in $G - v$. By Hall's theorem [14], I is matched into H', and equivalently H' is also matched into I since $|H'| = |I|$. It follows that $G - v$ has an equal crown.

Conversely, suppose that G is a crown-free graph, and suppose further that there exists a vertex v in G such that $G' = G - v$ has an equal crown (I, H'). We claim that $(I, H = H' \cup \{v\})$ is an almost-crown in G. Observe first that v must belong to $N(I)$ in G, otherwise $H' = N(I)$ in G, and (I, H') would also be a crown in G, contradicting the fact that G is crown-free. It follows that $H = N(I)$ in G and $|H| = |I| + 1$. Let S be a nonempty subset of I, and note that since (I, H') is an equal crown in G', and hence I is perfectly matched into H', $|N(S)| \geq |S|$ in G'. If $v \in N(S)$ in G, then $|N(S)| \geq |S| + 1$ in G. If $v \notin N(S)$ in G, then $N(S)$ in G' is the same as $N(S)$ in G. If $|N(S)| = |S|$, then since S is matched into $N(S)$, it would follow that $N(S)$ is matched into S too, and $(S, N(S))$ would be a crown in G, contradicting the fact that G is crown-free. It follows that if $v \notin N(S)$ in G then we must have $|N(S)| \geq |S| + 1$. Consequently, whether $v \in N(S)$ or not, $|N(S)| \geq |S| + 1$ in G for every nonempty subset S of I. This shows that $(I, H = N(I))$ is an almost-crown in G and the proof is complete. □

Proposition 2.6. *There is an algorithm A of time $O(k^3\sqrt{k})$ that, for any instance (G, k) of VC with $|G| \leq 2k$, reduces (G, k) to an instance (G', k') with $|G'| \leq |G|$ and $k' \leq k$ such that G' is crown-free. Moreover, if G' contains an almost-crown, then the algorithm A also finds an almost-crown in G'.*

Proof. Apply the NT-decomposition to G until the application of this decomposition is trivial. Each application of the NT-decomposition takes $O(m\sqrt{n})$ time [3], where m and n are the number of edges and vertices, respectively. Since $n = |G| = O(k)$ and $m = O(k^2)$, each application of the NT-decomposition takes $O(k^2\sqrt{k})$ time. Each such non-trivial application reduces the number of vertices in the graph, and therefore the number of non-trivial applications of the NT-decomposition is $O(k)$. Since a graph is crown-free if and only if the application of the NT-decomposition is trivial [11], it follows that in $O(k^3\sqrt{k})$ time, we can reduce (G, k) to an instance (G', k') where G' is a crown-free graph. Note that each non-trivial application of the NT-decomposition reduces the number of vertices in the graph as well as the parameter value.

By Lemma 2.5, G' has an almost-crown if and only if there exists a vertex $v \in G'$ such that $G' - v$ has an equal crown. Then an almost-crown in G' can be constructed as follows (in case it exists). For every vertex v in G', check if $G' - v$ has a crown. By [11], this can be done by applying the NT-decomposition to $G' - v$. If the NT-decomposition yields a crown, then this crown must be an equal crown (otherwise G' would not be crown-free), and hence by Lemma 2.5, we have constructed an almost-crown structure in G'. This step takes time $O(k'^3\sqrt{k'}) = O(k^3\sqrt{k})$ due to the possible $O(k') = O(k)$ applications of the NT-decomposition. It follows from the above that an almost-crown structure in G' can be constructed in time $O(k^3\sqrt{k})$, in case such a structure exists.

Since G' is crown-free, and equivalently the application of the NT-decomposition to G' is trivial [11], we have $|G'| \leq 2k'$. It follows from the above that in time $O(k^3\sqrt{k})$ we can reduce the instance (G, k) to an instance (G', k') satisfying the statement in the proposition. This completes the proof. □

We will refer to the operation described in Lemma 2.4 by the *general folding* operation. The reason behind this nomenclature is that this operation generalizes the folding operation that appeared in [8,10], and which deals with the case when $|I| = 1$. Two scenarios in which this operation is applicable are given in Fig. 2. The left figure is the special case in which the general folding reduces to the folding operation. We will assume that we have a subroutine called **General-Fold()** that searches for a structure in the graph to which the general folding operation applies, and applies the operation to it in case it exists. From the way we search for an almost-crown structure in the graph, we always reduce the graph to a crown-free graph along the way as proved by Proposition 2.6. Therefore, if the subroutine **General-Fold()** is not applicable to the graph, i.e., if its application does not change the structure of the graph, then we can assume that the graph is both crown-free and *almost-crown free* (i.e., does not contain an almost-crown). Using Proposition 2.6, this subroutine **General-Fold()** can be implemented to run in $O(k^3\sqrt{k})$ time.

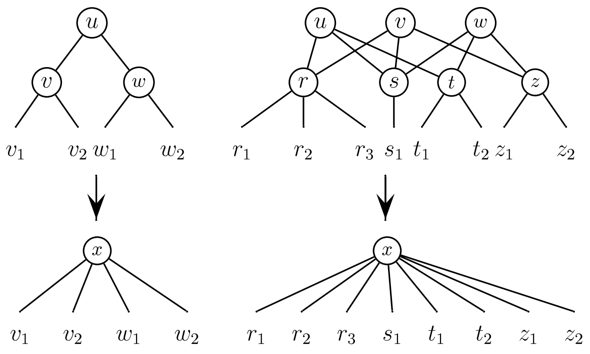

Fig. 2. General folding.

3. Prelude to the algorithm

The main algorithm is a branch-and-search process. Each stage of the algorithm starts with an instance (G, k) of VC, and tries to reduce the parameter k by identifying a set S of vertices that are entirely contained in a minimum vertex cover of G, and including the vertex set S in the objective minimum vertex cover, which will be called *the partial cover* (or simply the cover) for G, then recursively works on the reduced instances. We will assume that we have the subroutine **General-Fold(**G**)** described above, and the subroutine **Struction(**G**)**, which applies the struction operation to G.

If a vertex set S is identified such that either there is a minimum vertex cover containing the entire S or there is a minimum vertex cover containing no vertex in S, then we can *branch on the set* S. This means that the algorithm constructs two instances of the VC problem, one by including the set S in the partial cover and the other by excluding the set S from the partial cover, and in the latter case, every vertex that is adjacent to a vertex in S should be included in the partial cover. The algorithm then recursively works on the two reduced instances. If the set S consists of a single vertex v, then we simply say we *branch on* v.

Definitions and observations

Observation 3.1. *Let v be a vertex in G. Then there exists a minimum vertex cover for G containing $N(v)$ or at most $|N(v)| - 2$ vertices from $N(v)$.*

Proof. If a minimum vertex cover C for G contains $|N(v)| - 1$ vertices from $N(v)$, then it has to contain v. We form another minimum vertex cover for G by replacing v in C by the single vertex in $N(v) - C$. We obtain a minimum vertex cover for G containing $N(v)$. □

Observation 3.2. *Let u and v be two adjacent vertices in G. Then there exists a minimum vertex cover for G that includes v or that excludes v and excludes at least another neighbor of u.*

Proof. Proceed by contradiction. Suppose that every minimum vertex cover C excludes v and does not exclude any other neighbor of u. Since C excludes v, C must contain u. Since C contains all the neighbors of u except v, $(C - \{u\}) \cup \{v\}$ is a minimum vertex cover for G containing v, a contradiction. □

A vertex u is said to be *dominated* by a vertex v, or alternatively, a vertex v is said to *dominate* a vertex u, if (u, v) is an edge in G and $N(u) \subseteq N[v]$.[1] A vertex u is said to be *almost-dominated* by a vertex v, or alternatively, a vertex v is said to *almost-dominate* a vertex u, if u and v are non-adjacent and $|N(u) - N(v)| \le 1$.

Observation 3.3. *Let u and v be two vertices in G such that v dominates u. Then there exists a minimum vertex cover of G containing v.*

Proof. Let C be a minimum vertex cover. If C does not contain v then C must contain $N(v)$ which includes u (since (u, v) is an edge). Since $(N(u) - \{v\}) \subseteq N(v)$, if we remove u from C and replace it with v, we get a minimum vertex cover for G containing v. This completes the proof. □

A *good pair* is a pair of vertices $\{u, z\}$ chosen as follows. For a vertex u in G with neighbors $\{u_1, \ldots, u_d\}$, define its *tag*, denoted *tag*(u), to be the vector $\eta = \langle \eta_1, \ldots, \eta_d \rangle$, where η_1 is the degree of the largest-degree neighbor of u, η_2 is the degree of the second largest-degree neighbor of u, ..., and η_d is the degree of the smallest-degree neighbor of u. To choose the first vertex in a good pair, we pick a vertex u of minimum degree in G such that the following conditions are satisfied in their respective order.

1 We note that the notion of domination used in the current paper is different from the standard notion of domination in which a vertex u dominates a vertex v if u is a neighbor of v. However, the notion of domination used in the current paper is common in the literature of exact algorithms for the maximum independent set problem. For example, this notion is used in [20] and [32], among others.

(i) The vector $tag(u)$ is maximum in lexicographic order over $tag(w)$ for every w in G with the same degree as u.
(ii) If G is regular, then the number of pairs of vertices $\{x, y\} \subseteq N(u)$ such that y is almost-dominated by x is maximized.
(iii) The number of edges in the subgraph induced by $N(u)$ is maximized.

Having chosen the first vertex u in a good pair, to choose the second vertex, we pick a neighbor z of u such that the following conditions are satisfied in their respective order.

(a) If there exist two neighbors of u, say v and w, such that v is almost-dominated by w, then z is almost-dominated by a neighbor of u.
(b) The degree of z is maximum among all neighbors of u satisfying part (a) above. (Note that if no vertex in $N(u)$ is almost-dominated by another vertex in $N(u)$, then (a) is vacuously satisfied by every vertex in $N(u)$, and z will be a neighbor of u of maximum degree.)
(c) The degree of z in the subgraph induced by $N(u)$ is minimum among all vertices satisfying (a) and (b) above. (That is, z is adjacent to the least number of neighbors of u.)
(d) The number of shared neighbors between z and a neighbor of u is maximized over all neighbors of u satisfying (a), (b), and (c) above.

We note that the intuition behind the choice of a good pair of vertices is that such a pair allows for efficient branching. A similar notion to the good pair of vertices was used in [32].

Tuples

Tuples will play a very crucial role in the algorithm by helping to reduce the search space. We define the notion of tuples next and describe how they will be updated and processed by the algorithm.

Definition and intuition

A *tuple* is a pair (S, q) where S is a set of vertices and q is an integer. The tuple will represent the information that in the instance of the problem (G, k) we can look for a minimum vertex cover for G excluding at least q vertices from S. This information will help the algorithm prune the search tree. The algorithm will only consider tuples (S, q) with $q \leq 2$, so we will only focus on such tuples here. A tuple (S, q), where $S = \{u, v\}$, is called a *2-tuple* if it satisfies the following conditions: (1) $q = 1$, (2) $d(u) \geq d(v) \geq 1$, and (3) u and v are non-adjacent. A 2-tuple $(\{u, v\}, 1)$ is a *strong-2-tuple* if it satisfies the additional condition: $d(u) \geq 4$ and $d(v) \geq 4$, or $2 \leq d(u) \leq 3$ and $2 \leq d(v) \leq 3$.

To see how tuples can be used to prune the search space, suppose that the algorithm branches on a vertex z with a set of neighbors $N(z)$. By Observation 3.1, there exists a minimum vertex cover in G that either contains $N(z)$, or that excludes at least two vertices from $N(z)$. Therefore, when the algorithm branches on z, on the side of the branch where z is included, we can restrict our search to a minimum vertex cover that excludes at least two neighbors of $N(z)$, and we know that this is safe because if such a minimum vertex cover does not exist, then on the other side of the branch where $N(z)$ has been included the algorithm will still be able to find a minimum vertex cover. Consequently, on the side of the branch where z is included, we can work under the assumption that at least two vertices in $N(z)$ must be excluded. This working assumption will be stipulated by creating the tuple $(N(z), q = 2)$. This information will be used by the algorithm to render the branching more efficient. Similarly, if the algorithm branches on a vertex z with a neighbor u, by Observation 3.2, either there exists a minimum vertex cover in G that includes z, or there exists a minimum vertex cover in G that excludes z and excludes at least another neighbor of u. Therefore, on the side of the branch where z is excluded, we can restrict our search to a minimum vertex cover that excludes at least two vertices in $N(u)$ (z and another vertex in $N(u)$). This working assumption can be stipulated by creating the tuple $(N(u), q = 2)$. Note that after the removal of $z \in N(u)$ from the graph, the created tuple $(N(u), q = 2)$ will be updated, as discussed in the next section.

Updating tuples

Let (S, q) be a tuple. If $q = 0$ then the tuple S will be removed because the information represented by (S, q) is satisfied by any minimum vertex cover. If one of the vertices in S is removed and is excluded from the cover, then the tuple is modified by removing the vertex from S and decrementing q by 1. The correctness of this step can be seen as follows. Suppose that a vertex $u \in S$ has been excluded from the cover. If there exists a minimum vertex cover C that excludes at least q vertices from S, then C excludes at least $q-1$ vertices from $S - \{u\}$. Therefore the above update to the tuple is valid. If a vertex $u \in S$ is removed from the graph by including it in the cover, the vertex is removed from S and q is kept unchanged. The justification of this step follows from the argument that if there exists a minimum vertex cover C that includes u and excludes at least q vertices from S, then C must exclude q vertices from $S - \{u\}$ (note that the validity of the inclusion of u in the cover is taken care of by the correctness of the steps performed by the algorithm when it includes u in the cover).

Since a tuple imposes certain constraints on the minimum vertex cover sought, one needs to be careful that the constraints imposed by the creation of a tuple do not conflict with the conditions imposed by other operations of the algorithm. The other operations that do impose constraints on the minimum vertex cover sought are the creation of (other) tuples, the struction operation, and the general folding operation. For example, the general folding operation assumes

```
VC($G, \mathcal{T}, k$)
    Input: a graph $G$, a set $\mathcal{T}$ of tuples, and a positive integer $k$.
    Output: the size of a minimum vertex cover of $G$ if the size is bounded by $k$; report failure otherwise.

    0. if $k \le 7$ then solve the instance by brute-force and stop;
    1. apply Reducing;
    2. pick a structure $\Gamma$ of highest priority;
    3. if ($\Gamma$ is a 2-tuple $(\{u,z\},1)$) or ($\Gamma$ is a good pair $(u,z)$ where $z$ is almost-dominated by a vertex $v \in N(u)$) or
            ($\Gamma$ is a vertex $z$ with $d(z) \ge 7$) then
            return $\min\{1+$VC$(G-z, \mathcal{T} \cup (N(z),2), k-1), d(z)+$ VC$(G-N[z], \mathcal{T}, k-d(z))\}$;
       else    /* $\Gamma$ is a good pair $(u,z)$ where $z$ is not almost-dominated by any vertex in $N(u)$ */
            return $\min\{1+$VC$(G-z, \mathcal{T}, k-1), d(z)+$ VC$(G-N[z], \mathcal{T} \cup (N(u),2), k-d(z))\}$;

  Reducing
    a. for each tuple $(S,q) \in \mathcal{T}$ do
            a.1. if $|S| < q$ then reject;
            a.2. for every vertex $u \in S$ do $\mathcal{T} = \mathcal{T} \cup \{(S-\{u\}, q-1)\}$;
            a.3. if $S$ is not an independent set then $\mathcal{T} = \mathcal{T} \cup (\bigcup_{(u,v)\in E, u,v\in S}\{(S-\{u,v\}, q-1)\})$;
            a.4. if there exists $v \in G$ such that $|N(v) \cap S| \ge |S|-q+1$ then return $(1+$VC$(G-v, \mathcal{T}, k-1))$; exit;

    b. if Conditional_General_Fold($G$) or Conditional_Struction($G$) in the given order is applicable then
            apply it; exit;
    c. if there are vertices $u$ and $v$ in $G$ such that $v$ dominates $u$ then return $(1+$ VC$(G-v, \mathcal{T}, k-1))$; exit;

  Conditional_General_Fold
    if there exists a strong 2-tuple $(\{u,z\},1)$ in $\mathcal{T}$ then
          if the repeated application of General_Fold reduces the parameter by at least 2 then apply it repeatedly;
          else if the application of General-Fold reduces the parameter by 1 and $(d(u) < 4)$
                then apply it until it is no longer applicable;
    else apply General-Fold until it is no longer applicable;

  Conditional_Struction
    if there exists a strong 2-tuple $\{u,v\}$ in $\mathcal{T}$ then
            if there exists $w \in \{u,v\}$ such that $d(w) = 3$ and the Struction is applicable to $w$ then apply it;
    else if there exists a vertex $u \in G$ where $d(u) = 3$ or $d(u) = 4$ and such that the Struction is applicable to $u$
          then apply it;
```

Fig. 3. The algorithm **VC**.

that when we are looking for a minimum vertex cover, we can look for one that either contains the set I or the set $N(I)$ in the structure $(I, N(I))$. This is mainly the reason why the set $N(I)$ can be folded. If the general folding operation is applied, then this constraint imposed by the operation on the minimum vertex cover might conflict with the constraints imposed by a certain tuple. Therefore, to be on the safe side, when we decide to apply the struction or the general folding operations, we will invalidate all the constraints imposed by the tuples. That is, we will basically remove all the tuples. The decision on whether to apply the general folding or the struction operations will be based on the reduction in the parameter resulting from applying these operations. Therefore, we will have two subroutines **Conditional_Struction** and **Conditional_General_Fold** that will apply the struction and general folding operations, respectively. These subroutines will be applied when the gain (reduction in the parameter) resulting from the application of either operation surpasses that resulting from branching on a certain tuple (in case it exists), which will be invalidated after the execution of these operations.

The tuples need to be updated as described above after each operation of the algorithm. We will assume that this step is performed implicitly by the algorithm after each operation.

Storing and branching on 2-tuples

When the algorithm creates tuples it will use them to generate 2-tuples using very simple rules described in steps a.2 and a.3 of the subroutine **Reducing** in Fig. 3. Steps a.2 and a.3 of **Reducing** disintegrate a tuple into smaller tuples. During this process, some vertices might be determined to be in a minimum vertex cover by step a.4 of **Reducing**. For example, if $(S = \{u, w, z\}, 1)$ is a tuple, then this tuple imposes the constraint that we can look for a minimum vertex cover that excludes at least one vertex from S. Now if a vertex v is a common neighbor of u, w, and z, then v can be included in a minimum vertex cover satisfying the constraint imposed by the tuple because one of the vertices in S has to be excluded from such a cover. Therefore v will be included by step a.4. Since steps a.2 and a.3 derive more tuples from the tuple S, we need to make sure that the constraints imposed by the tuples generated in these two steps are consistent. This will be shown in Lemma 4.1.

The algorithm, however, creates new tuples when branching. Therefore, if we maintain existing tuples, then the constraints imposed by the newly generated tuples may conflict with those imposed by existing ones. To overcome this hurdle, and since the algorithm only processes 2-tuples, when the subroutine **Reducing** finishes processing the tuples in step a, we will maintain only one 2-tuple and invalidate the rest. Therefore, if 2-tuples exist after step a of **Reducing**, we will pick any strong 2-tuple in case a strong 2-tuple exists and invalidate the rest, or we will pick any 2-tuple and invalidate the rest, otherwise. Since when the algorithm branches it considers 2-tuples first (if they exist), this ensures that when the algorithm creates a new tuple in the next branch, it will have destroyed the only existing tuple when it branched on it. Therefore, after step a of **Reducing**, we will assume that at most one 2-tuple exists.

The algorithm only processes 2-tuples of the form $(S, 1)$. A 2-tuple of the form $(\{u, z\}, 1)$ stipulates that at least one vertex in $\{u, z\}$ must be excluded from the cover. This means that if u is included in the cover then z should be excluded, and hence $N(z)$ must be included; similarly, if z is included in the cover then u should be excluded, and $N(u)$ must be included. Let $(S = \{u, z\}, 1)$ be a 2-tuple. When the algorithm branches on a vertex in this two tuple, this vertex is picked as follows. If there is a vertex $w \in S = \{u, z\}$ such that w has a neighbor u' where u' is almost-dominated by the vertex in $S - \{w\}$, then the algorithm will branch on the vertex in $S - \{w\}$ (that is, if there is a vertex in S with a neighbor that is almost-dominated by the other vertex in S, then the algorithm will pick the other vertex in S). Otherwise, it will pick a vertex in S arbitrarily and branch on it. Without loss of generality, we will always assume that the vertex in the 2-tuple $S = \{u, z\}$ that the algorithm branches on is z. The algorithm can be made oblivious to this choice by ordering the vertices in a 2-tuple as described above whenever the 2-tuple is created.

4. The algorithm VC

A tuple, a good pair, or a vertex of degree at least seven, will be referred to by the word *structure*. The algorithm will maintain a list of structures $\mathcal{T}$, and then it will pick a structure and process it. The structures in $\mathcal{T}$ will be considered in a certain (sorted) order according to their priorities. The higher the priority of a structure is, the higher the benefit out of this structure will be. The priority is assigned to a structure whenever this structure is created. If an operation in the algorithm affects a certain structure in $\mathcal{T}$, then the priority of this structure needs to be modified accordingly, and the structure may need to be removed from $\mathcal{T}$. If a structure Γ is a vertex, and if this vertex is removed by the algorithm, then Γ is also removed from $\mathcal{T}$. If Γ is a good pair, and if one of the vertices in Γ is removed by the algorithm, then Γ is removed from $\mathcal{T}$. If Γ is a tuple (S, q) then Γ will be updated as described in the previous subsection. We will assume that the algorithm implicitly updates the structures in $\mathcal{T}$ and their priorities after each operation. We give below a comprehensive list of the structures Γ that can exist at a certain point in $\mathcal{T}$ listed in a non-increasing order of their priorities.

1 Γ is a strong 2-tuple.
2 Γ is a 2-tuple.
3 Γ is a good pair (u, z) where $d(u) = 3$ and the neighbors of u are degree-5 vertices such that no two of them share any common neighbors besides u.
4 Γ is a good pair (u, z) where $d(u) = 3$ and $d(z) \geq 5$.
5 Γ is a good pair (u, z) where $d(u) = 3$ and $d(z) \geq 4$.
6 Γ is a good pair (u, z) where $d(u) = 4$, u has at least three degree-5 neighbors, and the graph induced by $N(u)$ contains at least one edge (i.e., there is at least one edge among the neighbors of u).
7 Γ is a good pair (u, z) where $d(u) = 4$ and all the neighbors of u are degree-5 vertices such that no two of them share a neighbor other than u.
8 Γ is a vertex z with $d(z) \geq 8$.
9 Γ is a good pair (u, z) where $d(u) = 4$ and $d(z) \geq 5$.
10 Γ is a good pair (u, z) where $d(u) = 5$ and $d(z) \geq 6$.
11 Γ is a vertex z such that $d(z) \geq 7$.
12 Γ is any good pair other than the ones appearing in 1–11 above.

The above list gives the structures that could exist in $\mathcal{T}$ and their respective priorities. Moreover, the above list is comprehensive in the sense that for any non-empty graph G, G must contain one of the structures listed above, and the algorithm will have a structure to process. This can be seen as follows. First if the maximum degree of G is bounded by 2 then **Reducing** must apply. So suppose that this is not the case. Suppose also that G is connected.[2] If G contains a vertex of degree at least 7, then the algorithm will have at least one structure to consider by items 8 and 11 on the list. If this is not the case, then G has degree bounded by 6. If G is regular, then any good pair (u, z) must satisfy $d(u) = d(z)$, and hence none of the items 3–7, 9–10, dealing with good pairs applies, and item 12 applies. Basically, item 12 deals with regular graphs. Suppose now that G is not regular. Let u be a vertex with minimum degree in G. If $d(u) = 3$ then one of items 3, 4, 5 must apply (since G is not regular). If $d(u) = 4$ then one of items 6, 7, 9 must apply. If $d(u) = 5$ then item 10 must apply. Note that since G is not regular and has degree bounded by 6, G must contain a vertex of degree bounded by 5. This shows that the above list is comprehensive.

The algorithm will return the size of a minimum vertex cover in case this size is bounded by k, or otherwise it will reject. The algorithm can be easily modified to return the desired minimum vertex cover itself in case it has size bounded by k. We present the algorithm and prove its correctness next, and we analyze its running time in the next section. The algorithm is given in Fig. 3. Note that the algorithm performs *only* two branches *regardless* of the structure picked, which are the ones given in step 3 of the algorithm.

[2] If G is disconnected, the algorithm will be called recursively on each connected component of G (see Theorem 5.7).

Lemma 4.1. *The operations in step a of* **Reducing** *are valid tuple operations and the newly generated tuples in this step are consistent.*

Proof. If $|S| < q$ then the information represented by the tuple (S, q) has been violated because there does not exist a minimum vertex cover that excludes q vertices from S, and hence the algorithm can reject the instance. This shows that step a.1 is valid. (Again we note here that it is the "responsibility" of the algorithm to guarantee that whenever it branches by creating a tuple on one side of the branch, then either there exists a minimum vertex cover that does not violate the tuples along this side of the branch, or there is a minimum vertex cover along the other side of the branch.) If (S, q) is a tuple and $u \in S$, and if there exists a minimum vertex cover C excluding at least q vertices from S, then C excludes at least $q - 1$ vertices from $S - \{u\}$. Therefore, the constraint imposed by (S, q) is consistent with the constraints imposed by all the tuples of the form $(S - \{u\}, q - 1)$, where $u \in S$, simultaneously, and step a.2. is correct. Now let us look at step a.3 in **Reducing**. Suppose (S, q) is a tuple such that there are two vertices u and v in S that are adjacent. If there exists a minimum vertex cover C for G that excludes at least q vertices from S, then C must exclude at least $q - 1$ vertices from $S - \{u, v\}$ because C must contain at least one of the vertices in $\{u, v\}$. Therefore $(S - \{u, v\}, q - 1)$ is a tuple that is consistent with S. It can be easily shown also that all such tuples generated in step a.3 are consistent simultaneously, and step a.3 is correct. Now let us look at step a.4. Again since (S, q) is a tuple, if there exists a minimum vertex cover C that excludes at least q vertices from S, then since $|N(v) \cap S| \geq |S| - q + 1$, C excludes at least one vertex in $N(v)$ and must include v. Therefore there exists a minimum vertex cover of G that includes v and the statement is correct. This shows that any minimum vertex cover C that respects the constraints imposed by (S, q) respects all the constraints imposed by the newly generated tuples in step a of **Reducing**, and all the newly generated tuples in step a are simultaneously consistent. □

Theorem 4.2. *The algorithm* **VC** *is correct.*

Proof. We look at the operations performed by the algorithm. Step a of **Reducing** is valid by Lemma 4.1. By Lemma 2.2, the struction operation is correct and hence the operation **Conditional_Struction** is correct as well, since it only applies the struction operation to certain vertices that meet some specified conditions. The same is also true for the subroutine **Conditional_General_Fold** by Lemma 2.4. Therefore step b in **Reducing** is correct. Step c of **Reducing** is correct by Observation 3.3.

Consider the operations in the algorithm **VC**. Step 0 is correct since if $|G| > 0$ and $k = 0$, G does not have a vertex cover of size bounded by k (assuming G does not consist of isolated vertices). Step 1 is correct by the above discussion of the subroutine **Reducing**. Step 2 simply picks a structure Γ of highest priority in $\mathcal{T}$. We argued above that such a structure must exist, and hence the algorithm in step 2 will pick a structure Γ. Let us look at step 3 of the algorithm. First observe that each structure in $\mathcal{T}$ is either a 2-tuple, a good pair, or a vertex of degree at least 7. Therefore, one of the conditions in step 3 will apply to Γ and the algorithm branches accordingly. In all the cases in step 3 the algorithm branches on z, and hence the branch is valid by the definition of branching on a vertex. What is left is showing that the tuples added in each branch are valid tuples. The tuple created in the first branch is valid by Observation 3.1, and the tuple created by the second branch in step 3 is valid by Observation 3.2 (note that the tuple $(N(u), 2)$ created in the second branch in step 3 will be updated, as discussed in the previous subsection, due to the removal of the vertices in $N(u)$ from G, and in particular the removal of the vertex z). This completes the proof. □

5. Analysis of the algorithm

In this section we analyze the running time of the algorithm. Since the algorithm is a branch-and-search process, its execution can be depicted by a search tree. The running time of the algorithm is proportional to the number of root-leaf paths, or equivalently the number of leaves in the search tree, multiplied by the time spent along a longest root-leaf path in the search tree. Therefore, the main step in the analysis of the algorithm is deriving an upper bound on the number of leaves in the search tree. We start with the following propositions.

Lemma 5.1. *Let v be a vertex that satisfies the statement in step a.4 in* **Reducing***. If the algorithm does not reject the instance (along this path of the search tree) then v must be included in the cover before any branching operation by the algorithm. Moreover, each recursive call to* **Reducing** *before v is included in the cover, results in the execution of step a.4 of* **Reducing** *that includes a vertex in the cover.*

Proof. By looking at the algorithm **VC** the algorithm only branches when **Reducing** is not applicable. Moreover, since step a.4 in **Reducing** invokes the algorithm recursively, which in turn invokes **Reducing**, steps b and c of **Reducing** will not apply as long as step a.4 is applicable to a vertex in G.

Now suppose that there exists a vertex v and a tuple (S, q) such that $|N(v) \cap S| \geq |S| - q + 1$, and that the algorithm does not reject. When **Reducing** is applied, step a.4 is checked (note that steps a.1–a.3 in **Reducing** do not affect the graph at all). Since v satisfies this step, if v is considered in this step then v will be included. Now suppose that another vertex $x \neq v$ to which this step applies is checked, and x is included in the cover. If $x \notin S$, then (S, q) is unaffected by the inclusion of x, and v still satisfies this step in the (nested) recursive call to **Reducing** (note that this is true even when $x \in N(v)$). If $x \in S$, then each tuple containing x, and in particular S, will be updated. The tuple (S, q) will be updated to become $(S' = S - \{x\}, q)$. (Note that if $S = \{x\}$ then step a.4 cannot apply to x.) Since $|S| = |S'| + 1$, $|N(v) \cap S'| \geq |N(v) \cap S| - 1 \geq |S| - q \geq |S'| - q + 1$, and step a.4 is still applicable to v in the nested recursive call to **Reducing**. This shows that v will be included in the cover ultimately, and that each preceding call to **Reducing** before v is included, will include one vertex in the cover by step a.4. □

Lemma 5.2. *Let G be a graph such that every vertex in G has degree at least 2. Suppose that G contains three pairwise nonadjacent degree-2 vertices. Then the invocation of the subroutine* **Conditional_General_Fold** *reduces the parameter by at least 2.*

Proof. Let r be a degree-2 vertex in G. We claim that the repeated application of **General-Fold** reduces the parameter by at least 2. According to the subroutine **Conditional_General_Fold**, if the repeated application of the subroutine **General-Fold** reduces the parameter by at least 2, then **Conditional_General_Fold** will invoke **General-Fold** repeatedly until it is not applicable, thus reducing the parameter by at least two. Consider the vertex r, for instance. Since $d(r) = 2$, $(\{r\}, N(r))$ is an almost-crown in G. Therefore **General-Fold** is applicable. Now **General-Fold** first reduces the graph to a crown-free graph by repeatedly finding a crown and eliminating it by including its head in the cover. Since G does not contain any vertex of degree one, the head of any crown in G must have size at least two. Therefore, if a crown is found in G, then the application of **General-Fold** reduces the parameter by at least two as desired. If G is crown-free, then since G contains an almost-crown, **General-Fold** will determine an almost-crown structure $(I, N(I))$, and apply the general folding operation to this structure. If $|I| \geq 2$, then this operation results in reducing the parameter by at least two. If this is not the case, then I consists of a single degree-2 vertex, and the application of general folding reduces the parameter by 1. When the general folding operation is applied to the almost-crown, a degree-2 vertex must remain in the resulting graph. This can be seen as follows. Let $I = \{u\}$ and $N(I) = \{v, w\}$. Since there are at least three pairwise nonadjacent degree-2 vertices, there must exist a degree-2 vertex $p \notin \{u, v, w\}$. Now after the application of the general folding operation, p remains a non-isolated vertex in the resulting graph of degree at most two, and hence, **General-Fold** is still applicable, reducing the parameter further by at least one. This shows that the repeated application of **General-Fold** reduces the parameter by at least 2, and hence the invocation of **Conditional_General_Fold** will reduce the parameter by at least 2. □

Lemma 5.3. *Let G be a graph with degree bounded by three and with no isolated vertices. If there are at least four vertices in G of degree at most two, then the invocation to* **Conditional_General_Fold** *reduces the parameter by at least two.*

Proof. Similar to the proof in Lemma 5.2, we will show that the repeated application of **General-Fold** reduces the parameter by at least 2. If **General-Fold** finds a crown (I, H) in G, and if H has size at least 2, then the inclusion of H in the cover reduces the parameter by at least two, and we are done. If H has size one, then since the degree of the graph is at most three, the single vertex in H can be adjacent to at most three vertices in G, and hence the removal of the crown will leave at least one non-isolated vertex of degree at most 2 in the resulting graph (note that if the vertex in H is adjacent to three vertices in G then it cannot be one of those vertices in G of degree at most 2). Therefore, **General-Fold** will still be applicable in the resulting graph, and the repeated application of **General-Fold** to G results in a total reduction of the parameter by at least two.

Suppose now that G is crown-free. Then all the vertices in G must have degree at least 2. Since we know that there are at least four vertices in G of degree 2, **General-Fold** will return an almost-crown $(I, N(I))$. If I has size at least two, then the application of general folding to the almost-crown results in reducing the parameter by at least two. If this is not the case, then the almost-crown I consists of a single degree-2 vertex, and $N(I)$ consists of the two neighbors of this degree-2 vertex. Now applying the general folding operation to the almost-crown structure in G will reduce the parameter by one, and at least one non-isolated vertex of degree at most two remains in the resulting graph, making **General-Fold** applicable again in the resulting graph. This will totally reduce the parameter by at least two. It follows that the repeated application of **General-Fold** reduces the parameter by at least 2, and hence the invocation to **Conditional_General_Fold** reduces the parameter by at least two. □

Lemma 5.4. *If there is no strong 2-tuple in the graph, and if* **Reducing** *is not applicable, then* **General-Fold** *and* **Conditional_Struction** *are not applicable. Consequently, every vertex in G has degree at least three, and no vertex of degree 3 or 4 to which the struction operation applies exists in G.*

Proof. This follows easily from the statements in the subroutines **Conditional_General_Fold** and **Conditional_Struction**. If there is no strong 2-tuple in G, then **Conditional_General_Fold** will apply **General-Fold** repeatedly until it is no longer applicable. This removes all vertices of degree bounded by 2 from the graph. Similarly, in this case **Conditional_Struction** will apply the struction operation to every vertex in G of degree bounded by 4 to which this operation applies. Therefore, given that no strong 2-tuple exists and that **Reducing** is not applicable, it follows that neither of these subroutines is applicable, and the graph does not contain any vertex of degree bounded by 2, or of degree 3 or 4 to which the struction operation is applicable. □

Lemma 5.5. *Let G be a graph of minimum degree at least 3. Let I be an independent set in G such that* $2 \leq |I| \leq 3$. *If* $|N(I)| \leq |I| + 1$ *then* **General-Fold** *applies to G.*

Proof. We will prove the lemma for the case when $|I| = 3$. The proof is very similar (and easier) when $|I| = 2$. Since I is an independent set, $|I| = 3$, and the minimum degree of G is at least 3, we must have $|N(I)| \geq |I|$ because any vertex in I has at least three neighbors in $N(I)$.

Suppose first that $|N(I)| = |I| + 1$. We claim that $(I, N(I))$ is an almost-crown structure in G and hence the general folding operation applies to $(I, N(I))$. Since I is an independent set, and $|N(I)| = |I| + 1$, by Lemma 2.4, all we need to show is that for every nonempty subset S of I, we have $|N(S)| \geq |S| + 1$. If $S = I$ then the statement follows from the assumption $|N(I)| = |I| + 1$ of this case. If $|S| \leq 2$, then since the minimum degree of G is at least 3, $|N(S)| \geq 3 \geq |S| + 1$. This shows that $(I, N(I))$ is an almost-crown structure in G, and **General-Fold** applies to G.

Suppose now that $|N(I)| = |I|$. We claim that $(I, H = N(I))$ is a crown in G. Since every vertex in I has degree at least 3, and since the neighbors of the vertices in I are in $N(I)$, it follows from the fact that $|N(I)| = |I| = 3$ that every vertex in I is adjacent to all the vertices in $N(I)$, and the graph induced by $I \cup N(I)$ contains a copy of $K_{3,3}$. Therefore, $H = N(I)$ is matched into I and $(I, N(I))$ is a crown in G. This shows that **General-Fold** is applicable to G in this case as well, and the proof is complete. □

Theorem 5.6. *Let (G, k) be an instance of the* **VC** *problem where G is a connected graph and $k \geq 2$. Then for any constant $c \geq 1.2738$, the search tree of the algorithm* **VC** *on the instance (G, k) has at most $F(k)$ leaves, where $F(k)$ is upper bounded by the following:*

0. c^{k-1} *if step a.4, step b, or step c of* **Reducing** *is applicable.*
1. $c^{k-1.536}$ *if there is a strong 2-tuple structure.*
2. c^{k-1} *if there exists a non-isolated vertex in G of degree bounded by two.*
3. c^{k-1} *if there is a 2-tuple structure.*
4. c^{k-1} *if G is 3-regular.*
5. $c^{k-0.897}$ *if there exist three non-adjacent degree-3 vertices in G such that the three of them do not share a common neighbor.*
6. c^{k-1} *if G has a degree-3 vertex u such that all the vertices in $N(u)$ are of degree 5, and no two vertices in $N(u)$ share a common neighbor other than u.*
7. $c^{k-0.605}$ *if the algorithm picks a good pair (u, z) such that z is almost-dominated by a vertex in $N(u)$.*
8. $c^{k-0.605}$ *if G has a degree-3 vertex u with at least one vertex in $N(u)$ of degree at least 5.*
9. $c^{k-0.536}$ *if G has a degree-3 vertex.*
10. $c^{k-0.450}$ *if G has a degree-4 vertex u such that at least three vertices in $N(u)$ have degree-5, and such that the graph induced by $N(u)$ contains an edge.*
11. $c^{k-0.450}$ *if G has a degree-4 vertex u such that all the vertices in $N(u)$ are of degree 5 and no two of them share a common neighbor other than u.*
12. $c^{k-0.302}$ *if G has a vertex of degree at least 8.*
13. $c^{k-0.255}$ *if G has a degree-4 vertex.*
14. $c^{k-0.116}$ *if G has a degree-5 vertex with at least one degree-6 neighbor.*
15. c^{k} *in all other cases.*

Proof. The proof is by induction on the parameter k. The base case is when $2 \leq k \leq 7$. In this case the instance is solved by brute-force, as indicated in step 0 of the algorithm **VC**. In this case there is only 1 leaf in the search tree corresponding to the execution of the algorithm. It follows that all the above statements are true because $F(k) = 1$, and therefore $F(k)$ is bounded above by all the values that appear in statements 0–15.

In the rest of the proof, we will prove that the statements 0–15 are true for any $k \geq 8$. Assume inductively that *all* the above statements are simultaneously true for any instance (G', k'), where $2 \leq k' < k$. In particular, since all exponents of c in items 0–15 above are at most k, we have $F(k') \leq c^{k'}$.

Before we prove the statements of the theorem we give a general remark.

Remark. If in the proof we showed that the graph contains a structure Γ with an inductively proven upper bound on the number of leaves when the structure Γ exists in the graph, then even if the algorithm does not pick Γ to process, this upper bound is still valid since the algorithm always picks a structure with the highest priority, and as it will be shown by the statements of the theorem, a structure of higher priority corresponds to a smaller upper bound on the number of leaves in its corresponding search tree. Therefore whenever a certain structure is present in the graph, we can safely claim the upper bound on the number of leaves corresponding to this structure that was inductively proved.

Now we are ready to prove the theorem.[3]

Part 0. Since **Reducing** consists of non-branching operations, and since the application of step a.4, step b, or step c of **Reducing** includes at least one vertex in the cover, we have $F(k) \leq F(k-1) \leq c^{k-1}$, by the inductive hypothesis.

Part 1. Suppose that there is a strong 2-tuple $(S = \{u, z\}, q = 1)$. Suppose first that **Reducing** applies to G. Note that steps a.2 and a.3 of **Reducing** will not affect this 2-tuple because these steps only add tuples to the existing ones. If step a.4 of **Reducing** applies, then a vertex v is included in the cover, thus reducing the parameter k by 1. Observe that in the resulting instance, $S = \{u, z\}$ remains a 2-tuple (not necessarily a strong 2-tuple) since $d(u) \geq 2$ and $d(z) \geq 2$, $q = 1$, and u

[3] We will use the parameter k to "measure" the efficiency of the algorithm. That is, the progress made by performing an operation of the algorithm is measured in the reduction of the parameter k. Although the initial parameter is a positive integer, the progress made by an operation may be more precisely measured by a real positive number. Therefore, instead of introducing a new (real value) parameter whose initial value is k to measure the progress after each operation, for simplicity, we will use the same parameter k with the understanding that the reduction in the parameter after an operation may assume a positive real value.

and z are non-adjacent. If $F(k')$, where $k' = k - 1$, is the number of leaves in the search tree of the resulting instance, then inductively by part (3) of the theorem, $F(k') \leq c^{k'-1}$. It follows that $F(k) \leq F(k') \leq c^{k'-1} = c^{k-2} \leq c^{k-1.536}$.

Now suppose that step b of **Reducing** applies. If **Conditional_General-Fold** is applicable, then the subroutine will always reduce the parameter k. If it reduces the parameter k by at least 2, then we have $F(k) \leq F(k-2) \leq c^{k-2} \leq c^{k-1.536}$. If the subroutine reduces the parameter by 1, then by looking at the statements in the subroutine, the repeated application of **General-Fold** reduces the parameter by 1 as well, and we must have $d(u) < 4$. In this case we also have $d(z) < 4$, by the definition of a strong 2-tuple. Note that **General-Fold** must apply only once in this case. We claim that after the single application of **General-Fold** a vertex of degree at most 3 remains in the graph. This can be seen as follows. There are only two cases in which **General-Fold** reduces the parameter by 1. The first case is when **General-Fold** finds a crown (I, H) with $|H| = 1$, and in this case H will be included in the cover. Since all the vertices in I have degree 1, and since both u and z have degree at least 2 (by the definition of a strong 2-tuple), neither of the vertices in I can belong to $\{u, z\}$. It follows that the removal of the crown (I, H) from G will leave at least one of the vertices in $\{u, z\}$ non-isolated, and of degree at most 3 (since each of u and z originally had degree at most 3), in the resulting graph as desired. The second case in which **General-Fold** reduces the parameter by exactly 1, is when it determines an almost-crown structure $(I, N(I))$, where I consists of a single degree-2 vertex. In this case **General-Fold** applies the general folding operation to this structure. Since u and z are non-adjacent by the definition of a strong 2-tuple, and since they do not share a common neighbor by virtue of the inapplicability of step a.4 to the strong 2-tuple S, at least one vertex in $\{u, z\}$ must be in $G - (I \cup N(I))$, and the application of general folding to the almost-crown will leave this vertex a non-isolated vertex of degree at most 3 in the resulting graph. Therefore, after the single application of **General-Fold**, a non-isolated vertex p of degree at most 3 remains. Let $F(k')$, where $k' = k - 1$, be the number of leaves in the search tree of the resulting instance after the single application of **General-Fold**. If p has degree bounded by 2, then inductively by part (2) of the theorem, $F(k) \leq F(k') \leq c^{k'-1} = c^{k-2} \leq c^{k-1.536}$, as desired. If p has degree 3, then inductively by part (9) of the theorem, we have $F(k) \leq F(k') \leq c^{k'-0.536} = c^{k-1.536}$.

If the **Conditional_Struction** operation applies and destroys the strong 2-tuple, then from the way the operation works, the operation must apply to a degree-3 vertex w such that w is in the strong 2-tuple. Note that this operation reduces the parameter by 1. Without loss of generality, assume that $w = u$. Since u and z are non-adjacent and do not share any neighbors, the operation will not affect the degree of z by Remark 2.3, and a similar analysis to the above goes through.

If step c of **Reducing** is applicable, the analysis is similar to the case when step b applies. The only way that the removal of this vertex can affect the strong 2-tuple is when the vertex is one of the two vertices in the tuple, or a neighbor of a vertex in the tuple. The same analysis performed above gives the bound.

Now suppose that **Reducing** does not apply. Since **Conditional_General_Fold**, which is part of reducing, does not apply, then either **General-Fold** reduces the parameter by exactly one and $d(u) \geq 4$, or **General-Fold** does not apply at all. Suppose first that **General-Fold** reduces the parameter by exactly 1. Then in this case we have $d(u) \geq 4$. Also it follows that $d(z) \geq 4$ (this follows from the definition of a strong 2-tuple). Since there is a strong 2-tuple, from the way the list of priorities was defined, the strong 2-tuple $(\{u, z\}, 1)$ must be picked by the algorithm as the structure Γ. The algorithm branches on the vertex z. Note that since **Reducing** is not applicable, u and z do not share any neighbors. Now on the side of the branch where z is included, z is removed from the tuple S and q is kept unchanged. The recursive call to the algorithm will invoke **Reducing** and the neighbors of u will be included in the cover by step a.4 of **Reducing**. Therefore this side of the branch reduces the parameter by at least 5 ($N(u) \cup \{z\}$ are included in the cover). On the other side of the branch $N(z)$ is included, reducing the parameter by at least 4. It follows that $F(k) \leq F(k-4) + F(k-5) \leq c^{k-4} + c^{k-5} \leq c^{k-1.536}$.

Now we can assume that **General-Fold** does not apply, and hence the graph is crown-free and almost-crown free. Consequently every vertex in the graph has degree at least 3. Moreover, since the case when $d(u) \geq 4$ was considered above, we can assume now that $d(u) \leq 3$, and so is $d(z)$ by the definition of a strong 2-tuple. Again, the algorithm will pick the strong 2-tuple and branch at z.

If $d(u) = d(z) = 3$, then since **Conditional_Struction** is not applicable, there are no edges between vertices in $N(u)$ and similarly for $N(z)$. Let $N(u) = \{u_1, u_2, u_3\}$ and $N(z) = \{z_1, z_2, z_3\}$. Suppose that there exists a vertex in $N(u)$, say u_1, such that $|N(u_1) - N(z)| \leq 2$. On the side of the branch where the algorithm excludes z and includes $N(z)$, u_1 becomes of degree 1 or 2, and we can claim a further reduction in the parameter of value at least 1 by part (2) of the theorem. Therefore, on this side of the branch, the parameter has been reduced by at least $|N(z)| + 1 = 4$. On the other side of the branch where the algorithm includes z, all the vertices in $N(u)$ will be included when **Reducing** is called by step a.4. Moreover, the algorithm creates the tuple $(N(z), 2)$. We first claim that at least two vertices in $N(z)$ do not become isolated in the resulting graph along this side of the branch. Suppose not, then two of the vertices in $N(z)$, say z_1 and z_2 become isolated in $G - (\{z\} \cup N(u))$. Since u and z do not share any neighbors, z_1 and z_2 are only connected to $N(u) \cup \{z\}$. But then $I = \{u, z_1, z_2\}$ is an independent set whose set of neighbors $N(I) = \{z\} \cup N(u)$ satisfies $|N(I)| = |I| + 1$, and **General-Fold** (and hence **Reducing**) would be applicable by Lemma 5.5, a contradiction (note that the minimum degree of G is at least three). Therefore two non-isolated vertices in $N(z)$, that are also non-adjacent, will remain in the resulting graph. These two vertices will create a 2-tuple by step a.2 of **Reducing** when applied to the tuple $(N(z), 2)$ created by this side of the branch. This leads to a further reduction of the parameter by at least 1 by part (1) of the theorem. Therefore, along this side of the branch the parameter is reduced by at least 5. It follows that $F(k) \leq F(k-4) + F(k-5) \leq c^{k-4} + c^{k-5} \leq c^{k-1.536}$ as required. Suppose now that $d(u) = d(z) = 3$ and that the above case does not apply. On the side of the branch where z is excluded, $N(z)$ is included and a degree-3 vertex u remains in the graph. By induction, and by part (9) in the theorem, the number of leaves in the search tree along this side of the branch is bounded by $c^{k-3.536}$. On the other side of the branch, $\{z\} \cup N(u)$ are included in the cover and the tuple $(N(z), 2)$ is created. When **Reducing** is called, it will end up creating a 2-tuple for every two vertices in $N(z)$ (note that no two vertices in $N(z)$ are adjacent because **Conditional_Struction** is not applicable). We claim that at least one of these 2-tuples is a strong 2-tuple. To see this, observe that all the vertices in $N(z)$ have degree at least 2 in the resulting graph. This is true because otherwise a neighbor of z would be almost-dominated by u, and by the way the algorithm branches on 2-tuples, u will be picked by the algorithm instead of z to be the vertex to branch on, and the previous discussion applies. If $\{z_1, z_2\}$ is not a strong 2-tuple, then one vertex in $\{z_1, z_2\}$, say z_1, has degree at least 4 and the other vertex z_2 has degree at most 3. Now if z_3 has degree at least 4 then $\{z_1, z_3\}$ is a strong 2-tuple, otherwise, $\{z_2, z_3\}$ is a strong 2-tuple. By induction, the number of leaves in the search tree resulting from this side of the branch is at most $c^{k-5.536}$ (since $\{z\} \cup N(u)$ were included and there is a strong 2-tuple). It follows that the number of leaves in the search tree is $F(k) \leq c^{k-3.536} + c^{k-5.536} \leq c^{k-1.536}$. This completes the proof of part (1).

In the remaining parts of the proof, since no strong 2-tuple exists (otherwise part (1) applies), we can assume by Lemma 5.4 that if **Reducing** is not applicable then every vertex in the graph has degree at least 3, and the **Struction** operation does not apply to any vertex of degree 3 or 4 in G.

Part 2. Let v be a non-isolated vertex in the graph of degree at most 2. If there exists a strong 2-tuple, then by part (1), we have $F(k) \leq c^{k-1.536} \leq c^{k-1}$. If there is no strong 2-tuple, since v has degree bounded by 2, then **Reducing** must be applicable by Lemma 5.4. In particular, **Conditional_General-Fold** must apply, and $F(k) \leq c^{k-1}$ by part (0) of the theorem. It follows that $F(k) \leq F(k-1) \leq c^{k-1}$ as desired.

Part 3. Let $(S = \{u, z\}, q = 1)$ be a 2-tuple. Since $q = 1$ and u and v are non-adjacent, the only way **Reducing** can destroy this 2-tuple is if step a.4, or if one of steps b–c applies. Each of these steps reduces the parameter by at least 1 and $F(k) \leq F(k-1) \leq c^{k-1}$ as desired.

Now we can assume that **Reducing** is not applicable. Since no strong 2-tuple exists at this point by the way the algorithm picks a structure from the priority list, this implies that $d(u) \geq 3$ and $d(z) \geq 3$ by Lemma 5.4, and the algorithm will branch on the 2-tuple S.

If there is a neighbor u' of u such that $|N(u') - N(z)| \leq 2$, then on the side of the branch where $N(z)$ is included u' becomes non-isolated (u' is still adjacent to u) of degree at most 2, and we can claim a further reduction in the parameter of value at least 1 by part (2) of the theorem. Therefore along this side of the branch the parameter is decreased by at least $d(z) + 1 \geq 4$. On the other side of the branch we include $\{z\} \cup N(u)$ and the parameter is again decreased by at least 4 (note that $d(u) \geq 3$). Therefore $F(k) \leq 2F(k-4) \leq 2c^{k-4} \leq c^{k-1}$.

Suppose now that the above case does not apply. This implies, from the choice of the vertices we branch on in the 2-tuple, that no vertex in $N(z)$ is almost dominated by u, and hence, the degree of every vertex in $N(z)$ in the graph $G - (\{z\} \cup N(u))$ is at least 1. If $d(u) = d(z) = 3$, on the side of the branch where the algorithm includes z, $N(u)$ is included and the tuple $(N(z), 2)$ is created. When **Reducing** is called this tuple will create a 2-tuple (note that every vertex in $N(z)$ has degree at least 1, and no two vertices are adjacent by virtue of the inapplicability of **Conditional_Struction** to z). Inductively, the number of leaves along this side of the branch is bounded by $F(k-5)$ (note that $|\{z\} \cup N(u)| \geq 4$). On the side of the branch where z is excluded we include $N(z)$. It follows that $F(k) \leq F(k-3) + F(k-5) \leq c^{k-3} + c^{k-5} \leq c^{k-1}$.

In the remaining cases we must have $d(u) > 3$ or $d(z) > 3$. On one side of the branch z and $N(u)$ are included, and on the other side of the branch $N(z)$ is included. This gives us a worst-case bound $F(k) \leq F(k-3) + F(k-5) \leq c^{k-3} + c^{k-5} \leq c^{k-1}$ as required.

Part 4. Suppose that G is 3-regular. If **Reducing** applies then the statement follows by part (0). If there exists a strong 2-tuple then the statement follows by part (1). Now we can assume by Lemma 5.4 that for every degree-3 vertex u no edges exist in the subgraph induced by $N(u)$ (this follows from the inapplicability of **Conditional_Struction**). Since no 2-tuple or strong 2-tuple exists, the algorithm branches on a good pair (u, z). On the side where z is included, the three neighbors of z become of degree 2, and no two of them are adjacent. Note also that no vertex in the resulting graph has degree less than two. If step a.4 is executed along this side of the branch (note that the algorithm can create a tuple along this side of the branch if z is almost-dominated by a vertex in $N(u)$), then if this step is executed twice, this will lead to a reduction in the parameter of value 2. If step a.4 is executed once, then this will lead to a reduction in the parameter of value 1, and a vertex of degree at most 2 must remain in the graph (since step a.4 includes a single vertex in the cover), which leads to a further reduction in the parameter of value at least 1 by part (2), and a total reduction in the parameter of value at least 2. On the other hand if step a.4 is not applicable, then the algorithm will invoke the subroutine **Conditional_General_Fold** which decreases the parameter by at least 2 by Lemma 5.2 (since the three degree-2 vertices are still in G). Therefore the total parameter reduction on this side of the branch is at least 3 (z is included). On the other side of the branch where $N(z)$ is included, we claim that there must exist at least four non-isolated vertices of degree at most 2. To see why this is the case, let $N(z) = \{u, z_1, z_2\}$ and note that $N(z)$ is an independent set (**Struction** does not apply). Let B be the set of vertices of degree at most 2 in the graph $G - N(z)$. If $|B| < 4$, then the set $N(z)$ is an independent set in G having at most 4 neighbors, namely the vertices in $\{z\} \cup B$, and **General-Fold** would be applicable to G by Lemma 5.5 (note that the minimum degree of G is at least 3 at this point), a contradiction (since **Reducing** is not applicable). Therefore, $|B| \geq 4$. Now if a vertex $w \in B$ is isolated in $G - N(z)$ then the set of vertices $I = \{z, w\}$ is an independent set in G having the neighboring set $N(I) = N(z)$ with $|N(I)| = |I| + 1$, and again **General-Fold** would be applicable to G by Lemma 5.5. Therefore the graph $G - N(z)$ contains at least four non-isolated vertices of degree at most two. Note also that no vertex in G has degree less than two. If step a.4 is executed along this side of the branch (note that the algorithm can create a tuple along this side of the branch if z is not almost-dominated by a vertex in $N(u)$), then if this step is executed twice, this will lead to a reduction in the parameter of value 2. If step a.4 is executed once, then this will lead to a reduction in the parameter of value 1, and a vertex of degree at most 2 must remain in the graph (since the included vertex has degree bounded by 3), which leads to a further reduction in the parameter of value at least 1 by part (2), and a total reduction in the parameter of value at least 2. On the other hand if step a.4 is not applicable, then the algorithm will invoke the subroutine **Conditional_General_Fold** which decreases the parameter by at least 2 by Lemma 5.3 (since the four vertices of degree at most 2 are still in G). The total reduction along this side of the branch is at least 5 ($N(z)$ is included). It follows that $F(k) \leq F(k-3) + F(k-5) \leq c^{k-3} + c^{k-5} \leq c^{k-1}$ as required.

Part 5. Let x_1, x_2, x_3 be three degree-3 vertices in G such that no two of them are adjacent, and such that the three vertices do not share a common neighbor. If the graph is 3-regular then the statement follows from part (4) above. If **Reducing** is applicable, or if there exists a 2-tuple, then the statement follows from Parts (0), (1), or (3) of the theorem. If this is not the case, then from the priority list of the structures, the structure Γ picked by the algorithm must be a good pair (u, z) where $d(u) = 3$ and $d(z) \geq 3$ (note that the minimum degree of a vertex in the graph is 3 by Lemma 5.4). Suppose first that z is almost-dominated by a vertex $v \in N(u)$.

If $d(z) = 3$, let $N(z) = \{u, z_1, z_2\}$ be the neighbors of z and observe that since **Conditional_Struction** does not apply, no two vertices in $N(z)$ are adjacent. The algorithm will branch on z. On the side of the branch where z is included, the algorithm will create the tuple $(\{u, z_1, z_2\}, 2)$ which will immediately be decomposed by step a.2 of **Reducing** into the tuples $(\{u, z_1\}, 1)$, $(\{u, z_2\}, 1)$, $(\{z_1, z_2\}, 1)$. Since z is almost-dominated by v, v is adjacent to all vertices in $N(z)$ except at most 1. Therefore there exists a tuple $(S, 1)$ between $(\{u, z_1\}, 1)$, $(\{u, z_2\}, 1)$, such that step a.4 of **Reducing** applies to v and $(S, 1)$, and v will be included in the cover. By Lemma 5.1, all preceding recursive calls to **Reducing** end up executing step a.4 of **Reducing** and include vertices in the cover. If these recursive calls include two vertices in the cover before v is included, then this side of the branch ends up including at least four vertices in the cover (including z and v), and hence reducing the parameter by at least 4. Suppose that exactly one vertex y is included in these recursive calls before v is included. Note that $y \neq u$ because u is not adjacent to any vertex in $\{z_1, z_2\}$, and hence step a.4 cannot apply to u and any of the created tuples. If $y \neq w$, where w is the third neighbor of u, then after v is included, u becomes a degree-1 vertex, and a further reduction in the parameter of value at least one can be claimed by part (2), again yielding a total reduction of the parameter by 4. Suppose now that w is the vertex that is included before v is included. Now in the resulting graph after v is included, $(\{z_1, z_2\}, 1)$ is a 2-tuple. To see why this is the case, note that z_1 and z_2 are non-adjacent and none of them can become isolated in $G - \{u, v, w, z\}$, otherwise, **General-Fold** would be applicable to the set consisting of u and that vertex by Lemma 5.5. By part (3) of the theorem, this reduces the parameter by 1 yielding a total reduction of the parameter by 4 along this side of the branch. On the other side of the branch $N(z)$ is included. Notice that along this side of the branch a non-isolated vertex of degree at most three must remain in the graph because there were three pairwise non-adjacent degree-3 vertices in the graph that do not share a common neighbor. One of these vertices must remain and cannot be isolated (otherwise **General-Fold** would apply to this vertex and z by Lemma 5.5). If this vertex has degree one or two, then a reduction in the parameter of value at least one can be claimed by part (2). If this vertex has degree three, then by part (9) of the theorem, a further reduction of the parameter by value 0.536 can be claimed. Therefore along this side of the branch we can claim a reduction of the parameter of value at least 3.536. It follows that $F(k) \leq F(k-3.536) + F(k-4) \leq c^{k-3.536} + c^{k-4} \leq c^{k-0.897}$ as claimed.

Suppose now that $d(z) \geq 4$. By a similar argument to the above, we can claim that along the side of the branch where z is included v satisfies step a.4 of **Reducing**. A similar (and easier) argument using Lemma 5.1 will show that either **Reducing** ends up including a total of three vertices along this side and leaving a vertex of degree at most three in the resulting graph, allowing us to claim a further reduction of value 0.536 in the parameter by part (9) of the theorem, or it will include at least four vertices. Therefore a total reduction in the parameter of value at least 3.536 can be claimed along this side of the branch. On the other side of the branch, $N(z)$ is included, reducing the parameter by at least 4 and the result follows using the same argument as above.

Suppose now that z is not almost-dominated by any vertex in $N(u)$. Then from the choice of a good pair and the fact that G is not regular, we have $d(z) \geq 4$. The algorithm now branches on z and in the side where $N(z)$ is included the algorithm will create a tuple $(N(u), 2)$. Suppose first that $d(z) = 4$. On the side of the branch where z is included, u becomes a degree-2 vertex. Since no strong 2-tuple is created along this side of the branch, **General-Fold** becomes applicable (since $d(u) = 2$). If the repeated application of **General-Fold** reduces the parameter by at least two, then we can claim a total reduction of value three along this side of the branch (z is included). If the repeated application of **General-Fold** reduces the parameter by 1, then **General-Fold** removes an almost-crown $(I, N(I))$ where I consists of a single degree-2 vertex (note that no vertices of degree 1 can result from the inclusion of z). It is easy to see that since no two vertices among x_1, x_2, and x_3 are adjacent, and since they do not share a common neighbor, at least one of them will remain a non-isolated vertex of degree at most 3 in the graph resulting from including z and applying **General-Fold** to the almost-crown. This is true since z cannot be one of the vertices in $\{x_1, x_2, x_3\}$ because $d(z) = 4$. This will lead to a further reduction in the parameter of value at least 0.536 by part (9) of the theorem (or of value at least 1 by part (2)) giving a total reduction along this side of the branch of value at least 2.536. On the other side of the branch where $N(z)$ is included, the tuple $(N(u), 2)$ will result by step a.2 of **Reducing** in the strong 2-tuple $(\{v, w\} = N(u) - \{z\}, 1)$. To see why $\{v, w\}$ is a strong 2-tuple observe that $\{v, w\}$ are non-adjacent since $d(u) = 3$ and **Conditional_Struction** does not apply. Moreover, from the choice of the good pair (u, z) and since z is not almost-dominated by any vertex in $N(u)$, none of the vertices v or w can be almost-dominated by z (otherwise that vertex would be chosen in place of z). It follows that the degree of v and w in the resulting graph is at least two. Moreover, the degree of these two vertices in G was bounded by 4 since $d(z) = 4$, and by the choice of z, z had the maximum degree among the neighbors of u. It follows that the degrees of v and w in $G - N(z)$ is bounded by 3 (since u was removed). This shows that $\{v, w\}$ is a strong 2-tuple. By part (1) of the theorem this gives a further reduction of the parameter of value at least 1.536, giving a total reduction of value at least 5.536 along this side of the branch. It follows that $F(k) \le F(k - 2.536) + F(k - 5.536) \le c^{k-2.536} + c^{k-5.536} \le c^{k-0.897}$ as required.

Suppose now that $d(z) \ge 5$. By a similar argument to the above, on the side of the branch where z is included, u becomes a degree-2 vertex and **General-Fold** is applicable. Moreover, if the repeated application of **General-Fold** does not decrease the parameter by at least two, then a non-isolated vertex of degree at most three will remain, resulting in a total reduction in the parameter of value at least 2.536 along this side of the branch. On the other side of the branch where $N(z)$ is included, $\{v, w\}$ become a 2-tuple (not necessarily a strong 2-tuple) yielding a further reduction in the parameter of value at least 1 by part (3) of the theorem. It follows that $F(k) \le F(k - 2.536) + F(k - 6) \le c^{k-2.536} + c^{k-6} \le c^{k-0.897}$ as required.

Part 6. Let Γ be a structure of highest priority picked by the algorithm. If Γ is a 2-tuple (or a strong 2-tuple) then the statement follows from the previous parts of the theorem. If this is not the case, then by the priority list of the algorithm, Γ is a good pair (u, z) such that $d(u) = 3$, and all the neighbors of u, say $\{v, w, z\}$ are degree-5 vertices such that no two of them share a common neighbor other than u. If two vertices in $\{v, w, z\}$ are adjacent, then **Conditional_Struction** is applicable, and hence **Reducing** is applicable, which reduces the parameter by at least 1 by part (0) and giving the desired result. Suppose that this is not the case. Since z is not almost-dominated by any vertex in $N(u)$ (no two vertices in $N(u)$ share a common neighbor other than u), the algorithm will branch on z by including z on one side of the branch, and excluding it and creating a tuple $(N(u), 2)$ on the other side of the branch. On the side of the branch where z is included, u becomes of degree 2. Since no strong 2-tuple is created along this side of the branch, **General-Fold** will be applied. If the repeated application of **General-Fold** reduces the parameter by at least 2, then we can claim a reduction along this side of the branch of value at least 3 (z is included). If **General-Fold** reduces the parameter by exactly 1, then **General-Fold** simply folds an almost-crown structure which is a degree-2 vertex p (since no vertices of degree 1 exist in $G - z$). If $p \ne u$, then since both neighbors of u have degree 4 in $G - z$, $p \notin N(u)$, and u will remain in the resulting graph after folding p, and **General-Fold** would be applicable again, a contradiction. Therefore, **General-Fold** folds u, creating a vertex of degree 8 (since both neighbors of u in $G - z$ have degree 4 and do not share any neighbors), and inductively by part (12) of the theorem, an additional reduction of the parameter of value at least 0.302 can be claimed. Therefore along this side of the branch we can claim a reduction of the parameter of value at least 2.302. On the side of the branch where $N(z)$ is included, the tuple $(N(u), 2)$ will be decomposed into the tuple $(\{v, w\}, 1)$ in step a.2 of reducing. Since v and w are non-adjacent and have degree exactly 4 in the resulting graph (since no two neighbors of u share a common neighbor besides u), this tuple is a strong 2-tuple giving a further reduction in the parameter of value at least 1.536 by part (1). Therefore the total reduction along this side of the branch is at least 6.536 and $F(k) \le F(k - 2.302) + F(k - 6.536) \le c^{k-2.302} + c^{k-6.536} \le c^{k-1}$ as required.

Part 7. Suppose the algorithm picks a structure Γ such that Γ is a good pair (u, z) and z is almost-dominated by a vertex $v \in N(u)$. Note that if part (0) or part (1) of the theorem applies, then the statement follows. Therefore, we can assume that there exists no strong 2-tuple, and **Reduce** is not applicable. Consequently, by Lemma 5.4, neither **General-Fold** nor **Conditional_Struction** is applicable.

Suppose first that $d(u) = 3$. Then all vertices in $N(u)$ have degree at least 3, and no two of them are adjacent (since **Conditional_Struction** is not applicable). If $d(z) = 3$, let z_1 and z_2 be the neighbors of z other than u. On the side of the branch where z is included, the algorithm forms the tuple $(N(z), 2)$, which will be decomposed into the tuples $(\{u, z_1\}, 1)$, $(\{z_1, z_2\}, 1)$, $(\{u, z_2\}, 1)$ by step a.2 of **Reducing**. Now since z is almost-dominated by v, v and at least one tuple S among these three tuples will satisfy step a.4 in **Reducing**. By Lemma 5.1, v will be included before any branching by the algorithm. If step a.4 in **Reducing** includes two vertices before v, then the total reduction in the parameter along this side of the branch is at least 4. If step a.4 in **Reducing** includes exactly one vertex before v, let this vertex be y. Note that $y \notin \{u, z_1, z_2\} = N(z)$, because there are no edges among vertices in $N(z)$, and the application of step a.4 requires y to be adjacent to a vertex in $N(z)$. Since $y \notin \{u, z_1, z_2\}$, at least one of the tuples $(\{u, z_1\}, 1)$, $(\{z_1, z_2\}, 1)$, $(\{u, z_2\}, 1)$, remains a 2-tuple in the resulting graph when y and v are included, and a reduction in the parameter of value at least 1 can be claimed by part (3) of the theorem (note that there is no edge between the vertices in $\{u, z_1, z_2\}$ because **Conditional_Struction** does not apply to z, and at least two of them are non-isolated in the graph $G - \{z, v, y\}$, otherwise general folding would be applicable to the two isolated vertices in G). Suppose now that step a.4 in **Reducing** includes v in the next execution. Let w be the other vertex in $N(u)$. When v is included, u becomes a degree-1 vertex. If step a.4 is still applicable, and does not include w, then a degree-1 vertex u remains in G after the application of step a.4 (note that u cannot be included by step a.4 since no edges exist among vertices in $N(z)$) and a further reduction of value 1 can be claimed by part (2). If step a.4 includes w then $(\{z_1, z_2\}, 1)$ remains a 2-tuple in the resulting graph, claiming a further reduction of value 1 by part (3) (note that none of z_1, z_2 can be isolated in the resulting graph because general folding would apply to u and that vertex, and no edge exists between z_1 and z_2 because **Conditional_Struction** does not apply to z). Suppose now that step a.4 does not apply. If the repeated application of **General-Fold** reduces the parameter by a least 2, then the invocation to **Conditional_General_Fold** reduces the parameter by at least 2. If the repeated application of **General-Fold** reduces the parameter by exactly 1, then **General-Fold** must include the head of the crown $(I = \{u\}, H = \{w\})$ (the vertex w), in the cover and **General-Fold** will not apply the general folding operation. In this case $(\{z_1, z_2\}, 1)$ remains a 2-tuple in the resulting graph, claiming a further reduction of value 1 by part (3). It follows that we can always claim an additional reduction in the parameter of value at least 2 after the inclusion of z and v, and a total reduction of value at least 4 along this side of the branch. On the other side of the branch the algorithm includes $N(z)$, reducing the parameter by 3. It follows that $F(k) \le F(k-4) + F(k-3) \le c^{k-3} + c^{k-4} \le c^{k-0.605}$ as required.

Suppose now that $d(u) = 3$ and $d(z) \ge 4$. By a similar argument to the above we can show that on the side of the branch where z is included, step a.4 applies to v and we can show that along this side of the branch the total reduction in the parameter is at least 3. On the other side of the branch $N(z)$ is included yielding a reduction in the parameter of value at least 4, and the statement follows as in the above case.

Suppose now that $d(u) = 4$. In this case $d(z) \ge 4$. Similar to the above analysis, when z is included step a.4 applies to v. If another vertex is included before v we get a reduction in the parameter of value 3; otherwise v is included and u becomes of degree 2, yielding a further reduction in the parameter of value at least 1 by part (2). Therefore we get a total reduction in the parameter of value at least 3 along this side of the branch. Along the other side of the branch we include $N(z)$ and the parameter is reduced by at least 4. The statement follows.

If $d(u) = 5$ we have $d(z) \ge 5$. When z is included, if v is not included immediately by step a.4 of **Reducing**, then the parameter will be reduced by at least 3 along this side of the branch using Lemma 5.1. If v is immediately included, then u becomes a degree-3 vertex and by part (9) of the theorem, an additional reduction in the parameter of value 0.536 can be claimed. Therefore the total reduction in the parameter along this side of the branch is at least 2.536. When $N(z)$ is included, the parameter is reduced by at least 5. We have $F(k) \le F(k-2.536) + F(k-5) \le c^{k-2.536} + c^{k-5} \le c^{k-0.605}$ as required.

If $d(u) \ge 6$, and hence $d(z) \ge 6$, by a similar token to the above, when z is included v will be included, reducing the parameter by at least 2. On the other side $N(z)$ is included, reducing the parameter by at least 6. Therefore $F(k) \le F(k-2) + F(k-6) \le c^{k-2} + c^{k-6} \le c^{k-0.605}$.

Part 8. Again we will assume that part (0) and part (1) do not apply, and hence **Reducing** does not apply. Let Γ be the structure with highest priority picked by the algorithm. If Γ is a 2-tuple or a good pair (u, z) such that z is almost-dominated by a neighbor of u, then the statement follows from the above parts of the theorem. If this is not the case, then from the priority list of the structures, the algorithm will pick a good pair (u, z) such that $d(u) = 3$ and $d(z) \ge 5$. Note that since **Reducing** is not applicable (and hence **Conditional_Struction** does not apply to u), no two neighbors of u are adjacent. Let $N(u) = \{v, w, z\}$. Note also that by the choice of z in a nice pair, if z almost-dominates a vertex in $\{v, w\}$, then z must also be almost-dominated by a vertex in $\{v, w\}$, and the statement follows by part (7) of the theorem. Therefore, we can assume that no vertex in $\{v, w\}$ is almost-dominated by z. The algorithm branches on z. When z is included, u becomes of degree 2, and part (2) is applicable, reducing the parameter by at least 1. Therefore, the total reduction along this side of the branch is at least 2. On the other side of the branch where $N(z)$ is included, the algorithm creates a tuple $(N(u), 2)$, which generates the tuple $(\{v, w\}, 1)$ by step a.4 of **Reducing**. Since no vertex in $\{v, w\}$ is almost-dominated by z, v and w have degree at least 2 in the resulting graph and are non-adjacent. A further reduction of the parameter of value 1 along this branch can be claimed by part (3) of the theorem. Therefore $F(k) \le F(k-2) + F(k-6) \le c^{k-2} + c^{k-6} \le c^{k-0.605}$.

Part 9. Suppose that G has a vertex of degree 3. Let Γ be the structure picked by the algorithm. Again, from the previous parts of the theorem, and from the priority list of the structures, we can assume that Γ is a good pair (u, z), where $N(u) = \{v, w, z\}$ such that $d(u) = 3$, $d(v) \le d(w) \le d(z) = 4$ (note that by part (4) of the theorem the graph is not 3-regular and is connected by the assumption of the theorem), no vertex among $N(u) = \{v, w, z\}$ is almost-dominated by another by part (7), and no two vertices in $N(u)$ are adjacent since **Conditional_Struction** is not applicable. The algorithm branches on z. On the side where z is included, u becomes of degree 2, and part (2) is applicable. Therefore a reduction in the parameter of value at least 2 can be claimed along this side of the branch. On the side of the branch where $N(z)$ is included, the tuple $(N(u), 2)$ is created and will be decomposed into the tuple $(\{v, w\}, 1)$ by step a.2 of **Reducing**. It is easy to see from the above conditions that this is a strong 2-tuple giving a further reduction in the parameter of value at least 1.536 by part (1). Therefore $F(k) \le F(k-2) + F(k-5.536) \le c^{k-2} + c^{k-5.536} \le c^{k-0.536}$.

Part 10. Suppose that G has a degree-4 vertex such that at least three of its neighbors are of degree 5 and such that the graph induced by this set of neighbors contains an edge. Again, we can assume that **Reducing** does not apply and hence **Conditional_Struction** does not apply either. Let Γ be the structure of highest priority picked by the algorithm. By the previous parts of the theorem, and from the priority list of the structures, we can assume that Γ is a good pair (u, z) such that $d(u) = 4$ and at least three vertices in $N(u) = \{v, w, r, z\}$ are of degree 5 and there is an edge among the vertices in $N(u)$. We can also assume by part (7) above and the choice of z in a good pair that no vertex in $N(u)$ is almost-dominated by another vertex in $N(u)$. By the choice of z in a good pair, and since at least two vertices in $\{v, w, r\}$ are of degree-5, there must exist an edge among the vertices $\{v, w, r\}$ (the vertex z in a good pair is chosen to be the neighbor of u with minimum degree in the subgraph induced by $N(u)$). The algorithm branches on z. In the side where z is included, u becomes a degree-3 vertex with at least one edge among its neighbors, and since no strong 2-tuple is created along this side of the branch, **Reducing** is applicable (since **Conditional_Struction** is applicable to u). Therefore we can claim a reduction in the parameter of value 2 along this side of the branch. On the side where $N(z)$ is excluded, since **Conditional_Struction** does not apply to u, and no vertex in $N(u)$ is almost-dominated by another, a non-isolated vertex of degree at most 4 remains in the graph (at least one vertex in $N(u) - \{z\}$ must satisfy these properties). If this vertex has degree at most 2, then we can claim a reduction of the parameter of value at least 1 by part (2). If this vertex has degree 3, then a reduction in the parameter of value 0.536 can be claimed by part (9) above. If this vertex has degree 4, then we can claim a reduction in the parameter of value at least 0.255 by part (13). Therefore, along this side of the branch the parameter is reduced by at least 5.255. It follows that $F(k) \leq F(k-2) + F(k-5.255) \leq c^{k-2} + c^{k-5.255} \leq c^{k-0.450}$.

Part 11. Suppose that G has a vertex of degree 4 such that all its neighbors are of degree 5 and no two of them share a common neighbor other than the vertex itself. Let Γ be the structure of highest priority picked by the algorithm. By the above parts of the theorem and by the list of priorities, we can assume that Γ is a good pair (u, z) where $d(u) = 4$, all vertices in $N(u)$ have degree 5, no two vertices in $N(u)$ share a neighbor other than u, no edge exists among the vertices in $N(u)$ (by part (10) above), and no vertex in $N(u)$ is almost-dominated by another vertex in $N(u)$ (by part (7)). The algorithm branches on z. In the side of the branch where z is included, u becomes a degree-3 vertex with three degree-5 neighbors such that no two of them share a common neighbor other than u. Therefore, by part (6) of the theorem, we can claim an additional reduction in the parameter of value at least 1 on this side of the branch. On the side of the branch where $N(z)$ is included, a degree-4 vertex remains (this can be easily seen from the fact that no edges exist among vertices in $N(u)$, no two vertices in $N(u)$ share a common neighbor other than u, and all the vertices in $N(u)$ are of degree 5) and we can claim a further reduction in the parameter of value 0.255 by part (13). It follows that $F(k) \leq F(k-2) + F(k-5.255) \leq c^{k-2} + c^{k-5.255} \leq c^{k-0.450}$.

Part 12. Suppose that G has a vertex of degree at least 8. Let Γ be the structure of highest priority picked by the algorithm. If Γ is not a vertex of degree at least 8, then it must have a higher ranking in the list, and the above parts of the theorem show that processing such a structure will give a search tree of size $F(k) \leq F(k-0.302)$. If Γ is a vertex z with $d(z) \geq 8$, then the algorithm branches on z. We get $F(k) \leq F(k-1) + F(k-8) \leq c^{k-1} + c^{k-8} \leq c^{k-0.302}$.

Part 13. Suppose that G has a degree-4 vertex. Again we can assume that none of the above parts applies, and hence we can assume that the algorithm will pick a good pair (u, z) with $d(u) = 4$ and $d(z) \geq 4$. Let $N(u) = \{v, w, t, z\}$. We can assume that no three edges exist among the vertices in $N(u)$ (otherwise **Conditional_Struction** applies) and no vertex in $N(u)$ is almost-dominated by another by part (7) of the theorem. The algorithm branches on z.

Suppose first that G is 4-regular, and note that by the choice of a good pair, we can assume that no vertex is almost-dominated by another, since otherwise a good pair (u, z) will be picked where z is almost-dominated by a vertex in $N(u)$ (because all tag vectors have the same value) and to which part (7) of the theorem applies. Let $N(u) = \{v, w, t, z\}$ and $N(z) = \{z_1, z_2, z_3, u\}$.

Suppose that there is at least one edge among the vertices $\{v, w, t\}$. On the side of the branch where the algorithm includes z, since no strong 2-tuples are created along this side of the branch, **Reducing** becomes applicable (because **Conditional_Struction** is applicable to u). If **Reducing** does not apply **Conditional_Struction**, then **Reducing** must apply **Conditional_General_Fold** (note again that no tuples exist) by the way the operations are structured in **Reducing**. If the application of **Conditional_General_Fold** reduces the parameter by at least 2, then we can claim a total reduction along this side of the branch of value at least 3. If **Conditional_General_Fold** reduces the parameter by exactly 1, then **General-Fold** applies exactly once and it either returns a crown (I, H), where $|H| = 1$, or an almost-crown $(I, N(I))$, where I consists of a single degree-2 vertex. Since the inclusion of z leaves four degree-3 vertices in the graph, namely u, z_1, z_2, z_3, it is easy to see that after the single application of **General-Fold**, at least one of these vertices remains a non-isolated vertex in the resulting graph of degree at most 3, claiming an additional reduction in the parameter of value at least 0.536 by part (9) of the theorem (or better if the degree of this vertex is less than 2 and part (1) is applicable). Therefore, in this case, on this side of the branch a total reduction in the parameter of value at least 2.536 can be claimed. If **Reducing** applies **Conditional_Struction**, we will show that a vertex of degree at most 3 remains in the graph, and hence, a total reduction of value 2.536 can be claimed as well. Note first that when z is included, the only vertices of degree 3 remaining in the graph are u, z_1, z_2, and z_3 (since G is 4-regular). Suppose that the struction applies to a vertex y, then y must be a vertex in $N(z)$. Let y_1, y_2, and y_3, be the other neighbors of y and assume, without loss of generality, that there is an edge between y_1 and y_2. If two vertices among $\{y_1, y_2, y_3\}$ are of degree 3, then these vertices have to be adjacent to z in G, and there are at least three edges between the vertices in $N(y)$ in G (note that z is in $N(y)$), which would make **Conditional_Struction** applicable to y in G, and this is not case by our assumption (since **Reducing** does not apply to G). Therefore, at most one vertex in $\{y_1, y_2, y_3\}$ is a degree-3 vertex. When **Conditional_Struction** is applied to y, at most two degree-3 vertices will be removed (y and another vertex). Now if no vertex of degree at most 3 remains in the graph, then by Remark 2.3, the operation will only increase the degree of the neighbors of one of the vertices in $\{y_1, y_2, y_3\}$, say y_3. Therefore at least two neighbors of y_3 other than y (which was removed) must be also neighbors of z, and y_3 and z share at least three neighbors in G. This means that y_3 is almost-dominated by z in G, contradicting our initial assumption. It follows that, in both cases, on this side of the branch a degree-3 vertex remains, and we can claim a total reduction in the parameter of value at least 2.536. On the other side of the branch where $N(z)$ is included, a non-isolated vertex of degree at most 3 remains in the graph (because G is 4-regular and $N[z]$ was removed from G), and a further reduction in the parameter of value at least 0.536 can be claimed by part (9). We get $F(k) \leq F(k-2.536) + F(k-4.536) \leq c^{k-2.536} + c^{k-4.536} \leq c^{k-0.255}$.

By the selection of the vertices u and z in a good pair, we can now assume that for any vertex y, no edge exists between the neighbors of y (otherwise, such a vertex will be selected as the vertex u in the good pair, and the above scenario applies). Moreover, since no vertex is almost-dominated by another vertex, no three vertices can share more than one common neighbor (if three vertices share two common neighbors, then each of these common neighbors almost-dominates the other). On the side of the branch where z is included, the vertices z_1, z_2 and z_3 become degree-3 vertices such that no two of them are adjacent (otherwise an edge exists among the neighbors of z in G), and such that they do not share any common neighbor in G—z (since z is a common neighbor of these vertices and no three vertices share more than one common neighbor in G). Therefore, by part (5) of the theorem we can claim a further reduction in the parameter of value at least 0.897, totally reducing the parameter by at least 1.897. On the side of the branch where $N(z)$ is included, if one of the vertices in $\{v, w, t\}$ become of degree at most 2 (note that this vertex cannot become isolated since this vertex would be almost-dominated by z) part (1) will apply. If all these vertices become of degree 3 in $G - N(z)$, then by a similar token to the above, no two of these vertices are adjacent and they do not share a common neighbor in $G - N(z)$, therefore part (5) applies, further reducing the parameter by a value of at least 0.897. We get $F(k) \le F(k-1.897) + F(k-4.897) \le c^{k-1.897} + c^{k-4.897} \le c^{k-0.255}$.

Now we can assume that G is not 4-regular. Since G is connected, we can assume that $d(z) \ge 5$ and $d(v) \le d(w) \le d(t) \le d(z)$ by the choice of z in a good pair.

Suppose that $d(z) \ge 6$. On the side of the branch where z is included, u becomes a degree-3 vertex and we can claim a further reduction in the parameter of value at least 0.536 by part (9). When $N(z)$ is included, the parameter is reduced by at least 6. We get $F(k) \le F(k-1.536) + F(k-6) \le c^{k-1.536} + c^{k-6} \le c^{k-0.255}$.

Suppose now that $d(z) = 5$. If there is an edge among the vertices in $\{v, w, t\}$, then on the side of the branch where z is included, **Reducing** is applicable (**Conditional_Struction** is applicable to u), and we can claim a further reduction in the parameter of value at least 1. On the other side of the branch $N(z)$ is included. We get $F(k) \le F(k-2) + F(k-5) \le c^{k-2} + c^{k-5} \le c^{k-0.255}$. If there are exactly two edges between z and two vertices in $\{v, w, t\}$, say w and t, then on the side of the branch where $N(z)$ is included, the algorithm creates a tuple $(N(u), 2)$. This tuple will subsequently generate the tuple $(\{v\}, 1)$ since z is excluded from $N(u)$ and t and r are included. By step a.4 of **Reducing** and Lemma 5.1, all neighbors of v will be included in the cover. Since v is not almost-dominated by z, the parameter will be decreased further by at least 1. When z is included u becomes of degree 3, and we can claim a further reduction in the parameter of value at least 0.536 by part (9). We get $F(k) \le F(k-1.536) + F(k-6) \le c^{k-1.536} + c^{k-6} \le c^{k-0.255}$. The analysis is very similar if there is exactly one edge between z and a vertex in $\{v, w, t\}$, say t, because on the side of the branch where z is excluded a 2-tuple will be created namely $\{v, w\}$. Now we can assume no edges exist between vertices in $N(u)$. If there exists a vertex in $\{v, w, t\}$ of degree 5, say t, and another vertex of degree 4, say v, then on the side of the branch where z is included, u becomes a degree-3 vertex with a at least one neighbor of degree 5, and we can claim a further reduction in the parameter of value at least 0.605 by part (8). When $N(z)$ is included v becomes a non-isolated vertex of degree at most 3 and we can claim a further reduction in the parameter of value at least 0.536 by part (9). We get $F(k) \le F(k-1.605) + F(k-5.536) \le c^{k-1.605} + c^{k-5.536} \le c^{k-0.255}$.

If all vertices in $\{v, w, t\}$ have degree 4, and if z shares a neighbor other than u with at least one vertex in $\{v, w, t\}$, say t, then on the side of the branch where $N(z)$ is included, t becomes a non-isolated vertex of degree at most 2 (no vertex dominates another), and we can claim a further reduction in the parameter of value 1 by part (2). On the side where z is included, u becomes of degree 3 and we can claim a further reduction in the parameter of value at least 0.536 by part (9). We get $F(k) \le F(k-1.536) + F(k-6) \le c^{k-1.536} + c^{k-6} \le c^{k-0.255}$. Suppose now that z does not share any neighbors with $\{v, w, t\}$. If $\{v, w, t\}$ share a common neighbor $y \ne u$, then on the side of the branch where $N(z)$ is included the algorithm will create the tuple $(N(u), 2)$, which will be reduced to $(\{v, w, t\}, 1)$. Now y satisfies step a.4 in **Reducing** with respect to this tuple, and hence, will be included by Lemma 5.1, further reducing the parameter by 1. When z is included u becomes of degree 3 and we can claim a further reduction in the parameter of value 0.536 by part (9). We get $F(k) \le F(k-1.536) + F(k-6) \le c^{k-1.536} + c^{k-6} \le c^{k-0.255}$. Now if v, w, and t do not share any common neighbor, then on the side of the branch where $N(z)$ is included these vertices become three vertices of degree 3 such that no two of them are adjacent and such that the three of them do not share a common neighbor. By part (5), we can claim a further reduction in the parameter of value at least 0.897. When z is included u becomes of degree 3, and we can claim a further reduction in the parameter of value 0.536 by part (9). We get $F(k) \le F(k-1.536) + F(k-5.897) \le c^{k-1.536} + c^{k-5.897} \le c^{k-0.255}$.

Now suppose that all the vertices in $\{v, w, t\}$ are of degree 5. If z shares a neighbor other than u with any vertex in $\{v, w, t\}$, say t, then on the side of the branch where $N(z)$ is included t becomes a non-isolated vertex of degree at most 3 and we can claim a further reduction of the parameter of value at least 0.536 by part (9). When z is included u becomes a degree-3 vertex with at least one degree-5 neighbor and we can claim a reduction in the parameter of value at least 0.605 by part (8). We get $F(k) \le F(k-1.605) + F(k-5.536) \le c^{k-1.605} + c^{k-5.536} \le c^{k-0.255}$. If z does not share any neighbors with $\{v, w, t\}$ other than u, then by the choice of z in a good pair (since all vertices in $N(u)$ have the same degree), no two vertices in $N(u)$ share a neighbor other than u. This case is actually part (11) in the theorem and we have $F(k) \le c^{k-0.450} \le c^{k-0.255}$ as required.

Part 14. Suppose that G has a degree-5 vertex with a neighbor of degree 6. Again, if the structure Γ picked by the algorithm is not a good pair (u, z) with $d(u) = 5$ and $d(z) = 6$, then the statement follows from the fact that the algorithm always picks a structure with the highest priority and from the above parts of the theorem. Suppose now that this is the case. The algorithm branches on z. When z is included, u becomes of degree 4, and we can claim a further reduction in the parameter of value at least 0.255 by part (13) (or better by parts (2) and (9)). When $N(z)$ is included, the parameter is reduced by at least 6. We get $F(k) \le F(k-1.255) + F(k-6) \le c^{k-1.255} + c^{k-6} \le c^{k-0.116}$.

Part 15. We can assume in this case that none of the previous parts applies. In particular, part (12) does not apply and the graph has degree bounded by 7. If there exists a vertex z of degree 7, then by looking at the list of priorities, the algorithm will branch on z (or any other vertex of degree 7). This gives $F(k) \le F(k-1) + F(k-7) \le c^{k-1} + c^{k-7} \le c^k$. Suppose now that the graph has degree bounded by 6. By part (2), there are no vertices in the graph of degree 1 and 2. By parts (9) and (13), there are no vertices in the graph of degree less than 5. By part (14), and the fact that G is connected, G is either 5-regular or 6-regular.

Suppose first that G is 6-regular. Since none of the above parts of the theorem applies, the algorithm in this case will pick a good pair (u, z) and branch on z. When z is included, u becomes a degree-5 vertex with a neighbor of degree 6, and we can claim a further reduction in the parameter of value at least 0.116 by part (14) of the theorem. When $N(z)$ is included, the parameter is reduced by at least 6. We get $F(k) \le F(k-1.116) + F(k-6) \le c^{k-1.116} + c^{k-6} \le c^k$.

Suppose now that G is 5-regular. Again, since none of the above parts applies, the algorithm will pick a good pair (u, z) and branch on z. Let $N(u) = \{v, w, r, t, z\}$. Note that in particular, no vertex in $N(u)$ is almost-dominated by another vertex in $N(u)$ by part (7).

If z is adjacent to at least two vertices in $\{v, w, r, t\}$, then by the choice of z in a good pair, the graph induced by $\{v, w, r, t\}$ must contain at least three edges (since no vertex in $N(u)$ almost dominates another, all of them are of the same degree, z is chosen to be a neighbor with minimum degree in the subgraph induced by $N(u)$). Therefore, on the side of the branch where z is included, and since no tuples are created along this side of the branch, **Reducing** applies (because **Conditional_Struction** applies to u; see Fig. 1). On the side of the branch where $N(z)$ is included, a non-isolated vertex of degree at most 4 remains (since no vertex in $N(u) - \{z\}$ is almost dominated by z, some vertex in $N(u) - \{z\}$ must satisfy this condition in the resulting graph), and a further reduction in the parameter of value at least 0.255 can be claimed by part (13) (or better if the degree is less than 4). We get $F(k) \le F(k-2) + F(k-5.255) \le c^{k-2} + c^{k-5.255} \le c^k$.

If z is adjacent to one vertex in $\{v, w, r, t\}$, then there is at least one edge in the subgraph induced by $\{v, w, r, t\}$. On the side of the branch where z is included, u becomes of degree 4 and at least three of its neighbors are of degree 5 with at least one edge among them. Therefore, we can claim a further reduction in the parameter of value at least 0.450 by part (10). On the side of the branch where $N(z)$ is included, a non-isolated vertex of degree at most 4 remains, and a further reduction in the parameter of value at least 0.255 can be claimed by part (13). We get $F(k) \le F(k-1.450) + F(k-5.255) \le c^{k-1.450} + c^{k-5.255} \le c^k$.

If z shares one or more neighbors with a vertex in $\{v, w, r, t\}$, say with t, then when z is excluded, t becomes a non-isolated (since no vertex in $N(u)$ dominates another) vertex of degree at most 3 (since u and t which belong to $N(t) \cap N(z)$ have been excluded), and a further reduction in the parameter of value 0.536 can be claimed by part (9). When z is included, u becomes of degree 4, and we can claim a further reduction in the parameter of value 0.255 by part (13). We get $F(k) \le F(k-1.255) + F(k-5.536) \le c^{k-1.255} + c^{k-5.536} \le c^k$.

Now from the choice of z in a good pair, we can assume that no two vertices in $\{v, w, r, t, z\}$ share a neighbor other than u (otherwise, z would satisfy this property and one of the scenarios discussed above applies). When z is included, part (11) applies to u, and we can claim a further reduction in the parameter of value at least 0.450. When $N(z)$ is included, a vertex of degree 4 remains in the graph, and we can claim a further reduction in the parameter of value 0.255 by part (13). We get $F(k) \le F(k-1.450) + F(k-5.255) \le c^{k-1.450} + c^{k-5.255} \le c^k$.

This completes the proof of the theorem. □

Theorem 5.7. *The algorithm* **VC** *solves the* VC *problem in* $O(1.2738^k + kn)$ *time.*

Proof. Let (G, k) be an instance of VC. By Theorem 4.2 the algorithm **VC** solves the VC problem correctly. Let T be the search tree of the algorithm on the instance (G, k), and let $F(k)$ be the number of leaves in T. If G is connected and $k \ge 7$, then by Theorem 5.6, the number of leaves in T is at most 1.2738^k. On the other hand, if G is connected and $k \le 7$, then the number of leaves in T is equal to 1, which is at most 1.2738^k. It follows that if G is connected then the number of leaves in T is at most 1.2738^k.

If G is not connected, suppose that G has two connected components G_1 and G_2. If G has more than two connected components, the statement follows by an inductive argument. The algorithm can be called recursively on G_1 and G_2. If any of the components G_1 and G_2 has fewer than c' vertices for a pre-specified constant c' (picking $c' = 16$ will work), we can compute the size of a minimum vertex cover in that component in constant time by brute-force and without any branching, and the search tree corresponding to this recursive call has one leaf. For example, if $|G_1| < 16$ and is not empty, the size of a minimum vertex cover in G_1 is at least 1. Therefore if G has a minimum vertex cover of size at most k, then the size of a minimum vertex cover for G_2 should be at most $k-1$, and the parameter passed in the recursive call to G_2 is $k-1$. We get $F(k) \le 1 + F(k-1) \le 1 + 1.2738^{k-1} \le 1.2738^k$ (note that we can assume that $k \ge 8$ otherwise the algorithm would compute the size of the minimum vertex cover of G_2 as well by brute-force, and $F(k) = 1$). On the other hand if $|G_1| \ge c' = 16$ and $|G_2| \ge c' = 16$, then by Proposition 2.6, the size of a minimum vertex cover for G_1 is at least $|G_1|/2 \ge c'/2 \ge 8$, and the size of a minimum vertex cover for G_2 is at least $|G_2|/2 \ge 8$. Therefore in the recursive calls of the algorithm on G_1 and G_2 we can pass the parameter $k-8$. This gives $F(k) \le 2F(k-8) \le 1.2738^k$. This shows that the number of leaves in T is bounded by 1.2738^k.

Now let us analyze the time spent along each root-leaf path in T. By Proposition 2.1, after $O(kn + k^3)$ processing time, the number of vertices in G is at most $2k$. Since in each branch the algorithm creates at most one tuple, and since along any root-leaf path of T the algorithm branches at most k times (since each branch decrements k by at least 1), the number of tuples created by the branches of the algorithm is $O(k)$. Now step a.2 and a.3 in **Reducing** can decompose the tuples created by the algorithm thus creating new tuples. Observe that if the algorithm creates a tuple (S, q) in a branch then $q = 2$, and that any decomposition of a tuple decrements q by 1, and when $q = 0$ the tuple is removed. Based on these observations, it can be easily seen that each tuple (S, q) may lead to the creation of at most $O(|S|^2)$ new tuples. Since each created tuple has the form (S, q), where $S = N(w)$ for some vertex w, and since $|G| \le 2k$, we have $|S| \le 2k$. This means that each tuple can create at most $O(k^2)$ new tuples, and the total number of tuples along any root-leaf path is $O(k^3)$. Therefore step a of **Reducing** can be implemented to run in $O(k^3)$ time. By Proposition 2.6, **General-Fold** runs in $O(k^3\sqrt{k})$ time. All the other operations in **Reducing** and in the algorithm, including the implicit maintenance of the structures in $\mathcal{T}$ and their priorities, can be implemented to run in $O(k^3)$ time using suitable data structures. If a node in the search tree corresponds to an instance in which the value of the parameter is at most 7, then the algorithm will solve the instance by brute-force, as indicated in step 0 of the algorithm **VC**. To solve an instance in which the parameter has value at most 7 by brute-force, we first spend $O(n) = O(k)$ time to kernelize the instance to an equivalent instance which has at most 14 vertices (see Proposition 2.1), and then we apply any brute-force algorithm which will decide the instance in constant time. Therefore, if the value of the parameter is at most 7 at a node in the search tree, then the algorithm decides the instance in $O(k)$ time.

It follows from the above that the amount of time spent along each node of the search tree is $O(k^3\sqrt{k})$, and hence along every root-leaf path of T is $O(k^4\sqrt{k})$.

By the standard analysis that uses the *interleaving technique* introduced by Niedermeier and Rossmanith [28], the $O(k^4\sqrt{k})$ multiplicative factor can be eliminated, and the running time of the algorithm is bounded by $O(1.2738^k + kn)$, where the term kn emerges from the application of Proposition 2.1 to the original instance of the problem. □

6. Concluding remarks

In this paper we presented a polynomial-space algorithm for the VC problem that runs in $O(1.2738^k + kn)$ time, improving the previous $O(1.286^k + kn)$-time polynomial-space upper bound by Chen et al. [8], and the $O(1.2745^k k^4 + kn)$-time exponential-space upper bound by Chandran and Grandoni [6]. We emphasize that the contribution of the paper lies in the simplicity and uniformity of the presented algorithm. The presented algorithm does not rely on case-by-case branching rules, and suggests that simple algorithms, if designed properly, can lead to a much tighter analysis on their running time than the straight-forward worst-case scenario analysis. Several ideas and elegant structural results had to be introduced and used to achieve this goal including structural graph operations and amortized analysis.

Many papers presenting exponential-time algorithms for NP-hard problems claim that the presented algorithms perform much better than their derived time upper bounds in practice. The inability to tighten the upper bound is mainly due to the inability of designing and analyzing the operations in an interleaved way where inefficient operations are balanced by more efficient ones. The current paper is an example illustrating how a careful design of a very simple and uniform algorithm for the VC problem sets the stage for a much tighter analysis than the one based on worst-case scenario. We believe that this should be the general approach used when designing exact algorithms for NP-hard problems.

Acknowledgements

The first author's research was supported in part by the NSF under grants CCF-0430683 and CCF-0830455. The second author's work was supported in part by DePaul University Competitive Research Grant. The third author's research was supported in part by a Lafayette College Research Grant.

References

[1] DIMACS Workshop on Faster Exact Algorithms for NP-hard problems. Princeton, NJ, 2000.
[2] R. Balasubramanian, M. Fellows, V. Raman, An improved fixed parameter algorithm for Vertex cover, Information Processing Letters 65 (1998) 163–168.
[3] R. Bar-Yehuda, S. Even, A local-ratio theorem for approximating the weighted vertex cover problem, Annals of Discrete Mathematics 25 (1985) 27–46.
[4] J. Buss, J. Goldsmith, Nondeterminism within P, SIAM Journal on Computing 22 (1993) 560–572.
[5] L. Cai, D. Juedes, On the existence of subexponential parameterized algorithms, Journal of Computer and System Sciences 67 (4) (2003) 789–807.
[6] L. Chandran, F. Grandoni, Refined memorisation for vertex cover, Information Processing Letters 93 (3) (2005) 125–131.
[7] J. Cheetham, F. Dehne, A. Rau-Chaplin, U. Stege, P. Taillon, Solving large FPT problems on coarse grained parallel machines, Journal of Computer and System Sciences 67 (4) (2003) 691–706.
[8] J. Chen, I. Kanj, W. Jia, Vertex cover: further observations and further improvements, Journal of Algorithms 41 (2001) 280–301.
[9] J. Chen, L. Liu, W. Jia, Improvement on vertex cover for low degree graphs, Networks 35 (2000) 253–259.
[10] Jianer Chen, Iyad A. Kanj, Ge Xia, Labeled search trees and amortized analysis: Improved upper bounds for NP-hard problems, Algorithmica 43 (4) (2005) 245–273.
[11] M. Chlebik, J. Chlebikova, Crown reductions for the minimum weighted vertex cover problem, Discrete Applied Mathematics 156 (2008) 292–312.
[12] Benny Chor, Mike Fellows, David W. Juedes, Linear kernels in linear time, or how to save k colors in O(n^2) steps, in: Proceedings of the 30th International Workshop on Graph-Theoretic Concepts in Computer Science, in: Lecture Notes in Computer Science, vol. 3353, Springer, 2004, pp. 257–269.
[13] Frank K.H.A. Dehne, Michael R. Fellows, Frances A. Rosamond, Peter Shaw, Greedy localization, iterative compression, modeled crown reductions: New FPT techniques, an improved algorithm for set splitting, and a novel $2k$ kernelization for vertex cover, in: Proceedings of the 1st International Workshop on Parameterized and Exact Computation, in: Lecture Notes in Computer Science, vol. 3162, Springer, 2004, pp. 271–280.
[14] R. Diestel, Graph Theory, in: Graduate Textbooks in Mathematics, vol. 173, Springer, 1997.
[15] R. Downey, M. Fellows, Fixed-parameter tractability and completeness, Congressus Numerantium 87 (1992) 161–187.
[16] R. Downey, M. Fellows, Parameterized Complexity, Springer, New York, 1999.
[17] C. Ebengger, P.L. Hammer, D. de Werra, Pseudo-Boolean functions and stability of graphs, Annals of Discrete Mathematics 19 (1984) 83–98.
[18] M. Fellows, Blow-ups, win/win's and crown rules: some new directions in FPT, in: 29th International Workshop on Graph-Theoretic Concepts in Computer Science, in: Lecture Notes in Computer Science, vol. 2880, 2003, pp. 1–12.
[19] Mike Fellows, Pinar Heggernes, Frances A. Rosamond, Christian Sloper, Jan Arne Telle, Finding k disjoint triangles in an arbitrary graph, in: Proceedings of the 30th International Workshop on Graph-Theoretic Concepts in Computer Science, in: Lecture Notes in Computer Science, vol. 3353, Springer, 2004, pp. 235–244.
[20] F. Fomin, F. Grandoni, D. Kratsch, A measure & conquer approach for the analysis of exact algorithms, J. ACM 56 (5) (2009) 1–32.
[21] Fedor V. Fomin, Fabrizio Grandoni, Dieter Kratsch, Measure and conquer: Domination — a case study, in: Proceedings of the 32nd International Colloquium on Automata, Languages and Programming, in: Lecture Notes in Computer Science, vol. 3580, Springer, 2005, pp. 191–203.
[22] M. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, New York, 1979.

[23] R. Impagliazzo, R. Paturi, Which problems have strongly exponential complexity? Journal of Computer and System Sciences 63 (2001) 512–530.
[24] D.S. Johnson, M.A. Tricks (Eds.), Cliques, Coloring and Satisfiability, Second DIMACS Implementation Challenges, vol. 26, American Mathematical Society, Providence, RI, 1996.
[25] Luke Mathieson, Elena Prieto, Peter Shaw, Packing edge disjoint triangles: A parameterized view, in: Proceedings of the 1st International Workshop on Parameterized and Exact Computation, in: Lecture Notes in Computer Science, vol. 3162, Springer, 2004, pp. 127–137.
[26] G. Nemhauser, L. Trotter, Vertex packings: structural properties and algorithms, Mathematical Programming 8 (1975) 232–248.
[27] R. Niedermeier, P. Rossmanith, Upper bounds for vertex cover further improved, in: Proceedings of the 16th International Symposium on Theoretical Aspects of Computer Science, in: Lecture Notes in Computer Science, vol. 1563, 1999, pp. 561–570.
[28] R. Niedermeier, P. Rossmanith, A general method to speed up fixed-parameter-tractable algorithms, Information Processing Letters 73 (3–4) (2000) 125–129.
[29] R. Niedermeier, P. Rossmanith, On efficient fixed-parameter algorithms for weighted vertex cover, Journal of Algorithms 47 (2003) 63–77.
[30] Elena Prieto, Christian Sloper, Either/or: Using vertex cover structure in designing FPT-algorithms — the case of k-internal spanning tree, in: Proceedings of the 8th International Workshop on Algorithms and Data Structures, in: Lecture Notes in Computer Science, vol. 2748, Springer, 2003, pp. 474–483.
[31] Elena Prieto, Christian Sloper, Looking at the stars, in: Proceedings of the 1st International Workshop on Parameterized and Exact Computation, in: Lecture Notes in Computer Science, vol. 3162, Springer, 2004, pp. 138–148.
[32] J.M. Robson, Algorithms for maximum independent sets, Journal of Algorithms 7 (1986) 425–440.
[33] U. Stege, M. Fellows, An improved fixed-parameter-tractable algorithm for vertex cover. Technical Report 318, Department of Computer Science, ETH Zürich, April 1999.
[34] Gerhard J. Woeginger, Open problems around exact algorithms, Discrete Applied Mathematics 156 (3) (2008) 397–405.