Simplicity is Beauty: Improved Upper Bounds for Vertex Cover

Jianer Chen*, Iyad A. Kanj†, and Ge Xia*
*Department of Computer Science, Texas A&M University, College Station, TX 77843
email: {chen, gexia}@cs.tamu.edu
†The corresponding author. School of CTI, DePaul University, 243 S. Wabash Avenue, Chicago, IL 60604
email: ikanj@cs.depaul.edu

***Abstract*— This paper presents an $O(1.2738^k + kn)$-time polynomial-space algorithm for VERTEX COVER improving both the previous $O(1.286^k + kn)$-time polynomial-space algorithm by Chen, Kanj, and Jia, and the very recent $O(1.2745^k k^4 + kn)$-time exponential-space algorithm, by Chandran and Grandoni. Most of the previous algorithms rely on exhaustive case-by-case analysis, and an underlying conservative worst-case-scenario assumption. The contribution of the paper lies in the *extreme* 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 induces improvement on the upper bound for the INDEPENDENT SET problem on graphs of degree bounded by 6.**

I. 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 most practical instances (say for $n \leq 1000$) as opposed to an $O(2^n)$ 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], [17], [24]. 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 [15]. In addition, the problem has been a central problem in the study of parameterized algorithms [12], and has applications in areas such as computational biochemistry and biology [6]. Since the development of the first parameterized algorithm for the problem by Sam Buss which runs in $O(kn + 2^k k^{2k+2})$ time [3], there has been an impressive list of improved algorithms for the problem [2], [7], [9], [11], [19], [21], [23]. The most recent algorithm for the problem running in polynomial space, was presented in 1999 and gives the currently best time upper bound of $O(kn + 1.286^k)$ [7]. Algorithms using exponential space for the problem have also been proposed [5], [7], [21], amongst which the best runs in time $O(1.2745^k k^4 + kn)$ [5]. Most of the previous algorithms rely on exhaustive case-by-case analysis, 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 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 meticulous, rendering the verification and implementation of these algorithms very complicated and unpractical.

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 subexponential time, there is a constant $c_0 > 1$ such that VERTEX COVER cannot be solved in time $c_0^k n^{O(1)}$ [4], [16]. 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 ground for an insightful analysis of its running time. We came up with an algorithm that is extremely 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.

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. A graph operation that is a generalizations of the *folding* operation [7], and a graph operation that is a specialization of the *struction* operation [13], have been developed. 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 [22], 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 indebted 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 [5]. As a by-product of this algorithm, we obtain a polynomial-space $O(1.224^n)$-time algorithm for the INDEPENDENT SET problem on graphs of degree bounded by 6, improving the previous best polynomial-space algorithm of running time $O(1.227^n)$ by Robson [22] on such graphs.

II. PRELIMINARIES

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 $N(S)$ 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 [7] is based on a theorem by Nemhauser and Trotter [18].

Proposition 2.1 ([7]): 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 $k_1 \leq k$ and $|G_1| \leq 2k_1$, such that $\tau(G) \leq k$ if and only if $\tau(G_1) \leq k_1$.

We say that the instance (G_1, k_1) is the *kernel* of the instance (G, k). 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 $\{v_1, \cdots, v_p\}$. Construct a graph G' as follows: (1) remove the vertices $\{v_0, v_1, \cdots, v_p\}$ from G and introduce a new node v_{ij} for every anti-edge $\{v_i, v_j\}$ in G where $0 < i < j \leq p$; (2) add an edge (v_{ir}, v_{js}) if $i = j$ and (v_r, v_s) is an edge in G; (3) if $i \neq j$ add an edge (v_{ir}, v_{js}); and (4) for every $u \notin \{v_0, \cdots, v_p\}$, 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 [13] (see Figure 1 for an illustration). We have the following lemma.

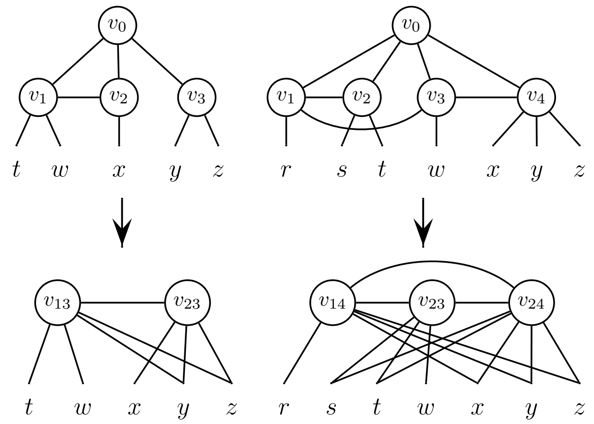

Fig. 1. The struction operation.

Lemma 2.2: Let v_0 be a vertex in G with a set of neighbors $\{v_1, \cdots, v_p\}$. Suppose that there are at most $p - 1$ anti-edges among the vertices $\{v_1, \cdots, v_p\}$, 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 [13] 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 $\{v_1, \cdots, v_p\}$, the number of newly introduced vertices in G' is at most $p - 1$. Since $p + 1$ vertices were removed from G, namely $\{v_0, v_1, \cdots, v_p\}$, we have $n' \leq n - 2$. It is well-known [15] 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. 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 Figure 1. We will assume that we have a subroutine called **Struction()** that applies the struction operation to a vertex v in G. Note that the time spent by this operation on a vertex v is proportional to $|N(v)|$.

Remark 2.3: 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 [7].

Lemma 2.4: Let I 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 $\emptyset \neq S \subset I$ 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.

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 $C \cap I = X \neq \emptyset$ and $C \cap N(I) = Y \neq \emptyset$. 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 os 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. ■

The following proposition can be proved using the results in [10], [14].

Proposition 2.5: Let (G, k) be an instance of **VC**. If a structure to which Lemma 2.4 applies exists in G, then such a structure can be found in $O(k^2\sqrt{k})$ time, otherwise, the number of vertices in G is at most $2k$.

We will refer to the operation 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 [7], [8], and which deals with the case when $|I| = 1$. Two scenarios in which this operation is applicable are given in Figure 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. Using Proposition 2.5, this subroutine can be implemented to run in $O(k^2\sqrt{k})$ time.

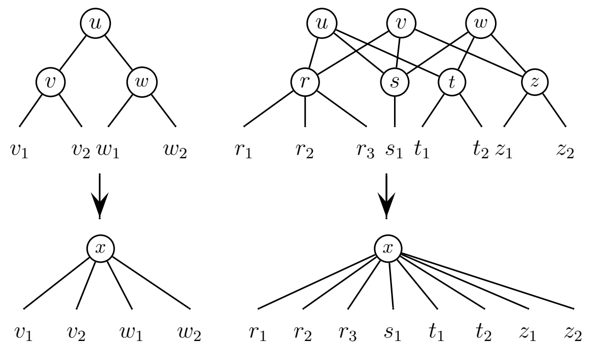

Fig. 2. General folding.

III. 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. The subroutine **General-Fold**(G) applies the general folding operation to G. Similarly, the subroutines **Struction**(G) and **Kernelize**(G) apply the struction operation and the kernelization procedure 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 v is said to *dominate* a vertex u if (u, v) is an edge in G and $N(u) \subseteq N[v]$. A vertex u is said to be *almost-dominated* by a vertex v if u and v are non-adjacent and $|N(u) - N(v)| \leq 1$.

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* of vertices is a pair of vertices $\{u, z\}$ chosen as follows. For a vertex u in G with neighbors $\{u_1, \cdots, u_d\}$, define its *tag*, denoted $tag(u)$, to be the vector $\eta = \langle \eta_1, \cdots, \eta_d \rangle$, 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. First choose a vertex u of minimum degree in G such that the following conditions are satisfied in their respective order.

(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.

Now choose a neighbor z of u such that the following conditions are satisfied.

(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.

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 neighbors $N(z)$. By Observation 3.1, either there exists a minimum vertex cover in G that either contains $N(z)$, or 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 $(N(u), q = 2)$.

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 by excluding it 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 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 $S - \{u\}$. Now 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). If a vertex in $u \in S$ is removed from the graph as a result of applying the struction operation or the general folding operation, then u is removed from S and q is decremented by 1. The reason is that if there exists a minimum vertex cover that excludes at least q vertices from S, then this vertex cover will exclude at least $q - 1$ vertices from $S - \{u\}$.

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.

Branching on 2-tuples

When the algorithm creates tuples it will use them to generate 2-tuples using very simple rules described in the algorithm (steps a.2 and a.3 of the subroutine **Reducing** in Figure 3). 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. The algorithm will branch on a vertex in this two tuple. This vertex is picked as follows. If there is a vertex $w \in \{u, z\}$ such that w has a neighbor u' and $|N(u') - N(S - \{w\})| \leq 1$, 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 that the algorithm branches on is z. The algorithm can be made anonymous to this choice by ordering the vertices in a 2-tuple as described above whenever the 2-tuple is created.

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 set of structures $\mathcal{T}$, and then it will pick a structure and processes 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 expected 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 before. We will assume that the algorithm implicitly updates the structures in $\mathcal{T}$ and their priorities after each operation. We give below a list of the structures Γ that can exist at a certain point in $\mathcal{T}$ listed in a non-increasing order of their priorities. Besides the structures listed below, $\mathcal{T}$ will contain tuples that are not 2-tuples, and those tuples will not be given any priorities. The algorithm will never process these tuples, and they are only used as intermediate structures which can result in the creation of 2-tuples by the algorithm.

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.

We note that the above list gives the structures that could exist in $\mathcal{T}$ and their priorities. Moreover, the above list is exhaustive 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 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[1]. 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

[1]If G is disconnected, the algorithm will be called recursively on each connected component of G (see Theorem 4.2).

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 the items 3, 4, 5 must apply (since G is not regular). If $d(u) = 4$ then one of the 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 Figure 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.

Proposition 3.4: The operations in step a of **Reducing** are valid tuple operations.

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 excluding q vertices from S, then there exists a minimum vertex cover excluding $q-1$ vertices from $S - \{u\}$. Therefore step a.2. is correct. Now let us look at step a.3 in **Reducing**. Suppose (S, q) is 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 $S - \{u, v\}$ since C must contain at least one of the vertices in $\{u, v\}$. Therefore $(S - \{u, v\}, q - 1)$ is a tuple, and the statement 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. ■

Theorem 3.5: The algorithm **VC** is correct.

Proof: We look at the operations performed by the algorithm. Step a of **Reducing** is valid by Proposition 3.4. Step-b in **Reducing** is correct because if $d(v) = 1$ then there exists a minimum vertex cover excluding v and including the neighbor of v. Therefore G has a vertex cover of size k if and only if $G - N[v]$ has a vertex cover of size $k - 1$. 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 operation **General-Fold** by Lemma 2.4. Therefore step c in **Reducing** is correct. Step d 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 $\mathcal{T}$. By the definition of a good pair, a good pair always exists in the graph as long as the graph is not empty. 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 condition 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. This completes the proof. ■

IV. Analysis of the Algorithm

In this section we analyze the running time of the algorithm. The algorithm is a branch-and-bound process and 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 each such path. Therefore, the main step in the analysis of the algorithm is deriving an upper bound on the number of leaves in the search tree. Let $F(k)$ be the number of leaves in the search tree of the algorithm when called on the instance (G, k).

First, we derive an upper bound on the number of leaves $F(k)$ of the search tree. This is the main theorem of this paper whose proof appears in Section V.

Theorem 4.1 (The Main Theorem): For any constant $c \geq 1.2738$, the search tree of the **VC** on an instance (G, k) where G is a connected graph, has at most $F(k)$ leaves where $F(k) \leq c^k$.

Proof: See Theorem 5.2, Section V. ■

Theorem 4.2: 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 3.5 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, then by Theorem 4.1, the number of leaves in T is bounded by 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 $c' = 16$

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** $|G| > 0$ and $k = 0$ **then** reject;
1. apply **Reducing**;
2. pick a structure Γ of highest priority;
3. **if** (Γ is a 2-tuple $(\{u, z\}, q)$) **or** (Γ is a good pair (u, z) such that z is almost-dominated by a vertex $v \in N(u)$) **or** (Γ is a vertex z with $d(z) \geq 7$) **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 if Γ is a good pair (u, z) **then**
   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 $(S, q) \in \mathcal{T}$ with $q = 2$ **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 $|N(v) \cap S| \geq |S| - q + 1$ **then return** $(1+$**VC**$(G - v, \mathcal{T}, k - 1))$; **exit**;
b. **if** there exists $v \in G$ such that $d(v) = 1$ **then return** $(1+$ **VC**$(G - N[v], \mathcal{T}, k - 1))$; **exit**;
c. **if General-Fold**(G) or **Conditional_Struction**(G) in the given order is applicable **then** apply it; **exit**;
d. **if** there are u and v in G such that v dominates u **then return** $(1+$ **VC**$(G - v, \mathcal{T}, k - 1))$; **exit**;

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

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 $F(k) \leq 1+F(k-1) \leq 1+1.2738^{k-1} \leq 1.2738^k$ (note that we can assume that $k \geq 8$ 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 $|G_1| \geq c' = 16$ and $|G_2| \geq c' = 16$, then since **Reducing** does not apply to G, and hence does not apply to G_1 and to G_2, by Proposition 2.5, the size of a minimum vertex cover for G_1 is at least $|G_1|/2 \geq c'/2 \geq 8$, and the size of a minimum vertex cover for G_2 is at least $|G_2|/2 \geq 8$. Therefore in the recursive calls of the algorithm on G_1 and G_2 we can pass the parameter $k - 8$. This gives $F(k) \leq 2F(k-8) \leq 1.2738^k$. 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, 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 shown that each tuple (S, q) may lead to the creation of at most $O(|S|)$ new tuples, each of them can no longer be decomposed. Since each created tuple has the form (S, q) where $S = N(w)$ for some vertex w, and since $|G| \leq 2k$, we have $|S| \leq 2k$. This means that each tuple can create at most $O(k)$ new tuples, and the total number of tuples along any root-leaf path is $O(k^2)$. Therefore step a of **Reducing** can be implemented to run in $O(k^3)$ time. By Proposition 2.5, **General-Fold** runs in $O(k^2\sqrt{k})$ 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. Therefore the amount of time spent along each node of the search tree is $O(k^3)$, and hence along every root-leaf path of T is $O(k^4)$.

Before any branching node in the search tree **General-Fold** does not apply (because **Reducing** does not apply). By Proposition 2.5, the size of the graph before any branching operation is bounded by twice the size of the parameter. By the standard analysis that uses the *interleaving technique* introduced by Niedermeier and Rossmanith [20], the running time of the algorithm is bounded by $O(1.2738^k + kn)$, where the term kn is due to the application of Proposition 2.1 to the original instance of the problem. ■

Using Theorem 4.2, and the fact that the size of a minimum vertex cover in a graph of degree bounded by 6 is at most $5n/6 + 1$, we get the following theorem.

Theorem 4.3: The INDEPENDENT SET problem on graphs of degree at most 6 can be solved in $O(1.224^n)$ time, where n is the number of vertices in the graph.

Theorem 4.3 improves the $O(1.227^n)$-time algorithm for INDEPENDENT SET on graphs of degree bounded by 6 [22].

V. PROOF OF THE MAIN THEOREM

In this section, we give a complete proof of Theorem 4.1. First, we have the following proposition which will be useful in the proof.

Proposition 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–d 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. 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 $(S' = S - \{x\}, q)$. 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. ■

Theorem 5.2: For any constant $c \geq 1.2738$, the search tree of the **VC** on an instance (G, k) where G is a connected graph, has at most $F(k)$ leaves where $F(k)$ is upper bounded by the following.

1. c^{k-1} if step a.4 or any of steps b–d of **Reducing** is applicable.
2. $c^{k-1.536}$ if there is a strong 2-tuple structure.
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 size of the instance (G, k). Assume inductively that *all* the above statements are simultaneously true for any instance (G', k') where $|G'| < |G|$ and $k' < k$.

Before we prove the statements of the theorem we give some general remarks. First, if during 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. Second, if the algorithm branches by reducing the parameter by a value p along one side, and along the other side the algorithm rejects without doing any branching, then the number of leaves in the search tree satisfies $F(k) \leq F(k-p) + 1$. Now we are ready to prove the theorem.

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

Part 2. 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 $q = 1$. 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 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, $F(k') \leq c^{k'-1} = c^{k-2}$. It follows that $F(k) \leq F(k') \leq c^{k-2} \leq c^{k-1.536}$.

Now if step b in **Reducing** applies, then a vertex v is included in the cover reducing the parameter k by 1. Since both u and z have degree at least 2, v is distinct from u and z. The only way v could affect the strong 2-tuple is when v is a neighbor of u or z, say u. If this is the case then u is included in the cover and now $S = \{z\}$ and $q = 1$. When the algorithm is called recursively in this step the neighbors of z

will be included by step a.4 in **Reducing** (since any neighbor of z will satisfy the statement in step a.4). Since $d(z) \geq 2$, and u and z did not share any neighbors (because step a did not apply), at least two vertices will be included in the cover. This is a total reduction in the parameter of value at least 3 giving $F(k) \leq c^{k-3} \leq c^{k-1.536}$. If the removal of v does not affect the strong 2-tuple, the strong 2-tuple will remain in the resulting graph. Letting $F(k')$ be the number of leaves in the resulting search tree, we have $F(k') \leq c^{k'-1.536}$ by induction. Hence $F(k) \leq F(k') \leq c^{k'-1.536} \leq c^{k-2.536}$.

If step d 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 step c of **Reducing** applies. If **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 the subroutine simply folds a degree-2 vertex w. If w is one of $\{u, z\}$, say u, then since u and z are non-adjacent, z will remain in the resulting graph. Since $d(u) = 2$, by the definition of a strong 2-tuple, $d(z) = 2$ or $d(z) = 3$. By induction, the former case leads to a further reduction in the parameter by value at least 1 by part (1) of the theorem, and the latter case to a reduction of the parameter of value 0.536 by part (9) of the theorem. Therefore the total reduction of the parameter is at least 1.536 and $F(k) \leq c^{k-1.536}$ as required. Now if $w \notin \{u, z\}$, then w cannot be adjacent to both u and z by step a.4 of **Reducing**. Folding w in this case will leave at least one vertex in $\{u, z\}$, and will similarly lead to a total reduction of the parameter of value at least 1.536. If the **Conditional_Struction** operation applies and destroys the strong 2-tuples, then from the way the operation works, the operation must apply to a degree-3 vertex w such that w is in a strong 2-tuple. Note that this operation reduces the parameter by 1. Without loss of generality, assume that the strong 2-tuple containing w is $\{u, z\}$, and suppose 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 cases goes through.

Suppose now that **Reducing** is not applicable. In this case we have $d(u) > 2$ and $d(z) > 2$. Since there is a strong 2-tuple, from the way the list of priorities was defined, a strong 2-tuple must be picked by the algorithm as the structure Γ. The algorithm branches in this case on the vertex z. Note that since **Reducing** is not applicable, u and z do not share any neighbors. Suppose first that $d(u) \geq 4$ and $d(z) \geq 4$. 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 $F(k) \leq F(k-4)+F(k-5) \leq c^{k-4}+c^{k-5} \leq c^{k-1.536}$.

If $d(u) = d(z) = 3$, then since the **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 $|N(u_1) - N(z)| \leq 2$. In the side of the branch where the algorithm excludes z and includes $N(z)$, u_1 becomes of degree 1 or 2, and when the subroutine **Reducing** is called the parameter will be further reduced by at least 1. Therefore in 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**) is applicable, a contradiction. 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 (2) 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 $F(k - 3.536)$. 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 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 $F(k - 5.536)$ (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 $F(k) \leq F(k-3.536) + F(k-5.536) \leq F(k-1.536)$.

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–d 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. This implies that $d(u) \geq 3$ and $d(z) \geq 3$.

If there is a neighbor u' of u such that $|N(u') - N(z)| \leq 2$, then by a similar token to the above, on the side of the branch where $N(z)$ is included u' becomes of degree at most 2, and **Reducing** will further decrease the parameter by at least 1. 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 and $d(u) = d(z) = 3$. On the side of the branch where the algorithm includes z, $N(u)$ is included and the tuple $(N(z), 2)$ is created. By a similar argument to the above, when **Reducing** is called this tuple will create a 2-tuple (note that every vertex in $N(z)$ has degree at least 2). 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 parameter is reduced by at least 1 and $F(k) \leq c^{k-1}$ as required. Now suppose that **Reducing** is not applicable. In this case for every degree-3 vertex u no edges exist in the subgraph induced by $N(u)$ (this follows from the inapplicability of **Conditional_Struction**). If there is a 2-tuple (or a strong 2-tuple) then the statement follows from above. 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. Then on this side of the branch **Reducing** will apply at least twice reducing the parameter by at least 2. On the 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. Let B be the set of vertices of degree at most 2 in the graph $G - N(z)$. If $|B| < 4$, then the set $N(z)$ has at most 4 neighbors namely the vertices in $\{z\} \cup B$, and **General-Fold** would be applicable, a contradiction. Therefore, $|B| \geq 4$. Now if a vertex $w \in B$ is isolated in $G - N(z)$ then the set of vertices $I = \{z, w\}$ has the neighboring set $N(I) = N(z)$ with $|N(I)| = |I| + 1$, and again **General-Fold** would be applicable. Therefore the graph $G - N(z)$ contains at least four non-isolated vertices of degree at most two. Again, in this side of the branch **Reducing** will be applied twice totally reducing the parameter along this side of the branch by at least 5. 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 either from the fact that **Reducing** reduces the parameter by at least 1, or from the parts (2) and (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). 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. If in the side of the branch where z is included the algorithm rejects before doing any branching, then we have $F(k) \leq F(k-3) + 1 \leq c^{k-0.897}$ as desired. Suppose now that this is not the case. 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 $(S, 1)$ among $(\{u, z_1\}, 1)$, $(\{u, z_1\}, 1)$, $(\{u, z_1\}, 1)$ such that step a.4 of **Reducing** applies to v and $(S, 1)$, and v will be included in the cover. By Proposition 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. If $y = u$, then the tuple $(\{u, z_1\}, 1)$ will be updated to $(\{z_1\}, 1)$ and step a.4 of **Reducing** will be applicable to all vertices in $N(z_1)$. Since $u \notin N(z_1)$, this means that at least four vertices will be included in the cover, namely $N(z_1) \cup \{u\}$, in this side of the branch. Now if $y \neq w$, where w is the third neighbor of u, then after v is included, u will be a degree-1 vertex and **Reducing** will end up further reducing the parameter by at least one, 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 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 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** will apply to this vertex and z). If this vertex has degree one or two, then **Reducing** will end up reducing the parameter by at least one. 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 Proposition 5.1 will show that either **Reducing** ends up including a total of three vertices along this side and leaving a vertex of degree 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 and **General-Fold** is applicable to u. Any operation in **Reducing**, other than **General-Fold** will leave u in the resulting graph a non-isolated vertex of degree at most 2, and **Reducing** will still be applicable (note that if **General-Fold** is applicable but was not applied then **Conditional_Struction** was not applied as well by the respective order of these two operations in the algorithm). This will reduce the parameter by at least 3 along this side of the branch. If instead **General-Fold** was applied, then it is easy to see that since no two of 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 a degree-2 vertex (if **General-Fold** was applied to a set I with $|I| > 1$, then we can claim a reduction in the parameter of at least 2 from this operation). This is true since z cannot be one of the vertices in $\{x_1, x_2, x_3\}$ since $d(z) = 4$. This will lead to a further reduction in the parameter of value at at least 0.536 by part (9) of the theorem 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 is 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 (2) 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 $F(k) \leq F(k-2.536) + F(k-5.536) \leq c^{k-2.536} + c^{k-5.536} \leq c^{k-0.897}$ as required.

Suppose now that $d(z) \geq 5$. 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 **Reducing** does not apply **General-Fold** then u will remain and **Reducing** will be applicable again claiming a total reduction in the parameter of value at least 3. If **Reducing** applies **General-Fold** then a non-isolated vertex of degree at most three remains claiming 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 $F(k) \leq F(k-2.536) + F(k-6) \leq c^{k-2.536} + c^{k-6} \leq c^{k-0.897}$ as required.

<u>Part 6.</u> Let Γ be a structure of highest priority picked by the algorithm. If Γ is a 2-tuple (or a strong 2-tuple) the 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 implying 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 two. If **Reducing** does not apply **General-Fold** to u, then u will remain in the graph (again note that **Reducing** did not apply **Conditional_Struction** by the way the algorithm works) non-isolated and having degree at most two. this means that **Reducing** will also be applicable and a total reduction of at least 3 in the value of the parameter can be claimed along this side of the branch. If **Reducing** applies **General-Fold** to a vertex other than u, then again a reduction of value 2 can be claimed if **General-Fold** applies to a set I of cardinality

at least two, or if **General-Fold** applies to another degree-2 vertex since u will remain (none of the neighbors of u could be the vertex folded since each has a degree larger than two). If **General-Fold** applies to u, then a vertex of degree eight results from folding u, and 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. Therefore the total reduction along this side of the branch is at least 6.536 and $F(k) \leq F(k-2.302) + F(k-6.536) \leq c^{k-2.302} + c^{k-6.536} \leq c^{k-1}$ 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 **Reduce** is not applicable at this point.

Suppose 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 other neighbors of z. 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 the algorithm. 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 Proposition 5.1, v will be included before any branching by the algorithm. If **Reducing** includes two vertices before v, then the total reduction in the parameter along this side of the branch is at least 4. If **Reducing** includes exactly one vertex before v, let this vertex be y. If $y \in \{u, z_1, z_2\} = N(z)$, then the neighbors of the vertices in $N(z) - \{y\}$ will be included by step a.4 of **Reducing** when applied the the vertices in $N(z) - \{y\}$ and the three tuples formed above. Note that the set $N(z) - \{y\}$ has at least two neighbors in the graph $G - \{z, y\}$ (otherwise **General-Fold** applies). Therefore the parameter is reduced by at least 4 in this case. If $y \notin \{u, z_1, z_2\}$, then $(\{z_1, z_2\}, 1)$ 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 none of z_1, z_2 can be isolated in the resulting graph because **General-Fold** does not apply). Suppose now that **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, and **Reducing** is still applicable. If **Reducing** in the following execution does not include u or w, then u remains a degree-1 vertex in the resulting graph, and **Reducing** will still be applicable. On the other hand, if **Reducing** includes u or w in the next execution, then $\{z_1, z_2\}$ remains a 2-tuple in the resulting graph, and a further reduction in the parameter of value 1 can be claimed. It follows that in all cases this side of the branch will reduce the parameter by at least 4. On the other side of the branch the algorithm includes $N(z)$ reducing the parameter by 3. It follows that $F(k) \leq F(k-4) + F(k-3) \leq c^{k-3} + c^{k-4} \leq c^{k-0.605}$ as required.

Suppose now that $d(u) = 3$ and $d(z) \geq 4$. 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 $d(z) \geq 4$. 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 become of degree 2 yielding a further reduction in the parameter of value at least 1 by **Reducing**. Therefore we get a total reduction in the parameter of value at least 3 along this side of the branch. Along the other side we include $N(z)$ and the parameter is reduced by at least 4. The statement follows.

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

If $d(u) \geq 6$, and hence $d(z) \geq 6$, 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 $F(k) \leq F(k-2) + F(k-6) \leq c^{k-2} + c^{k-6} \leq c^{k-0.605}$.

Part 8. 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 $d(z) \geq 5$. Note that since **Reduce** is not applicable 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 by part (7) of the theorem the statement follows. 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 **Reducing** 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 when $N(z)$ is included the algorithm creates a tuple $(N(u), 2)$ which reduces to $(\{v, w\}, 1)$ 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, and a further reduction of the parameter by value 1 can be claimed by part (3) of the theorem. Therefore $F(k) \leq F(k-2) + F(k-6) \leq c^{k-2} + c^{k-6} \leq c^{k-0.605}$.

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 $d(u) = 3$, $d(v) \leq d(w) \leq d(z) = 4$ (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 **Reducing** 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 $(\{v, w\}, 1)$ in 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. Therefore $F(k) \leq F(k-2) + F(k-5.536) \leq c^{k-2} + c^{k-5.536} \leq c^{k-0.536}$.

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. Note that **Reducing** does not apply and hence **Conditional_Struction** does not apply as well. 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 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 **Reducing** is applicable (since **Conditional_Struction** is applicable). 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. If this vertex has degree at most 2 then we can claim a reduction of the parameter of value at least 1 by **Reducing**. 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 a degree 4 vertex 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 $N(u)$, and no vertex in $N(u)$ is almost-dominated by another vertex in $N(u)$. 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 a further 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 remain 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 cases applies. 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. 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 vertex 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 (8) 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, **Reducing** becomes applicable (because **Conditional_Struction** is applicable to u). If **Reducing** does not apply **Conditional_Struction**, then **Reducing** reduces the parameter by at least 1, and a non-isolated vertex of degree at most 3 remains in the graph (namely, a vertex in $\{u, z_1, z_2, z_3\}$), and we can claim a further reduction in the parameter of value at least 0.536 by part (9) of the theorem (or better). Therefore, on the 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. Note first that when z is included the only vertices of degree 3 in the graph are u, z_1, z_2, and z_3. 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 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 $N(y)$ (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. 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. 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 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 there neighbors. This means that y_3 is almost-dominated by z, contradicting our assumption. It follows that on this side of the branch a degree-3 vertex remains, and we can claim a 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, 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 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. Moreover, note that since no vertex is almost-dominated by another vertex, no three vertices can share more than one common neighbor. 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, and such that they do not share any common neighbor in $G - z$ (since z is a common neighbor of these vertices). Therefore, by part (9) 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) **Reducing** 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 $F(k) \leq F(k-1.897) + F(k-4.897) \leq c^{k-1.897} + c^{k-4.897} \leq c^{k-0.255}$.

Now we can assume that G is not 4-regular. Since G is not connected, we can assume that $d(z) \geq 5$ and $d(v) \leq d(w) \leq d(t) \leq d(z)$ by the choice of z

Suppose that $d(z) \geq 6$. 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. When $N(z)$ is included the parameter is reduced by at least 6. We get $F(k) \leq F(k-1.536) + F(k-6) \leq c^{k-1.536} + c^{k-6} \leq c^{k-0.255}$.

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, 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 $F(k) \leq F(k-2) + F(k-5) \leq c^{k-2} + c^{k-5} \leq c^{k-0.255}$.

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 be reduced subsequently to the tuple $(\{v\}, 1)$ since z is excluded from $N(u)$ and t and r are included. By step a.4 of **Reducing** and Proposition 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. We get $F(k) \leq F(k-1.536) + F(k-6) \leq c^{k-1.536} + c^{k-6} \leq c^{k-0.255}$. 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 $\{v, w\}$.

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 $F(k) \leq F(k-1.605) + F(k-5.536) \leq c^{k-1.605} + c^{k-5.536} \leq c^{k-0.255}$.

If all vertices in $\{v, w, t\}$ have degree 4, and if v 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, and we can claim a further reduction in the parameter of value 1 since **Reducing** will be applicable. 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. We get $F(k) \leq F(k-1.536) + F(k-6) \leq c^{k-1.536} + c^{k-6} \leq c^{k-0.255}$. Suppose now that z does not share any neighbors with $\{v, w, t\}$. If $\{v, w, t\}$ share a common neighbor $y \neq u$, 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 Proposition 5.1 further reducing the parameter by 1. When z is included u becomes of degree 3. We get $F(k) \leq F(k-1.536) + F(k-6) \leq c^{k-1.536} + c^{k-6} \leq c^{k-0.255}$. 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. We get $F(k) \leq F(k-1.536) + F(k-5.897) \leq c^{k-1.536} + c^{k-5.897} \leq c^{k-0.255}$.

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. We get $F(k) \leq F(k-1.605) + F(k-5.536) \leq c^{k-1.605} + c^{k-5.536} \leq c^{k-0.255}$. 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 $F(k) \leq c^{k-0.450} \leq c^{k-0.255}$ 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 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). When $N(z)$ is included the parameter is reduced by at least 6. We get $F(k) \leq F(k-1.255) + F(k-6) \leq c^{k-1.255} + c^{k-6} \leq c^{k-0.116}$.

Part 15. We can assume in this case that none of the previous parts applies. In particular, part (11) 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 $F(k) \leq F(k-1) + F(k-7) \leq c^{k-1} + c^{k-7} \leq c^k$. Suppose now that the graph has degree bounded by 6. By part (1), 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 $F(k) \leq F(k-1.116) + F(k-6) \leq c^{k-1.116} + c^{k-6} \leq c^k$.

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 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. Therefore on the side of the branch where z is included **Reducing** applies (because **Conditional_Struction** applies). On the side of the branch where $N(z)$ is included a 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) (or better if the degree is less than 4). We get $F(k) \leq F(k-2) + F(k-5.255) \leq c^{k-2} + c^{k-5.255} \leq c^k$.

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 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 $F(k) \leq F(k-1.450) + F(k-5.255) \leq c^{k-1.450} + c^{k-5.255} \leq c^k$.

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 vertex of degree at most 3, 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 $F(k) \leq F(k-1.255) + F(k-5.536) \leq c^{k-1.255} + c^{k-5.536} \leq c^k$.

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. 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 we can claim a further reduction in the parameter of value 0.255 by part (13). We get $F(k) \leq F(k-1.450) + F(k-5.255) \leq c^{k-1.450} + c^{k-5.255} \leq c^k$.

This completes the proof. ∎

REFERENCES

[1] *DIMACS Workshop on Faster Exact Algorithms for NP-hard problems*. Princeton, NJ, 2000.

[2] R. Balasubramanian, M. Fellows, and V. Raman. An improved fixed parameter algorithm for Vertex Cover. *Information Processing Letters*, 65:163–168, 1998.

[3] J. Buss and J. Goldsmith. Nondeterminism within P. *SIAM Journal on Computing*, 22:560–572, 1993.

[4] L. Cai and D. Juedes. On the existence of subexponential parameterized algorithms. *Journal of Computer and System Sciences*, 67(4):789–807, 2003.

[5] L. Chandran and F. Grandoni. Refined memorisation for vertex cover. In *Proceedings of the 1st International Workshop on Parameterized and Exact Computation*, volume 3162 of *Lecture Notes in Computer Science*, pages 61–70, 2004.

[6] J. Cheetham, F. Dehne, A. Rau-Chaplin, U. Stege, and P. Taillon. Solving large FPT problems on coarse grained parallel machines. *Journal of Computer and System Sciences*, 67(4):691–706, 2003.

[7] J. Chen, I. Kanj, and W. Jia. Vertex cover: further observations and further improvements. *Journal of Algorithms*, 41:280–301, 2001.

[8] J. Chen, I. Kanj, and G. Xia. Labeled search trees and amortized analysis: improved upper bounds for NP-hard problems. In *14th International Symposium on Algorithms and Computation*, volume 2906 of *Lecture Notes in Computer Science*, pages 148–157. Springer, 2003.

[9] J. Chen, L. Liu, and W. Jia. Improvement on Vertex Cover for low degree graphs. *Networks*, 35:253–259, 2000.

[10] M. Chlebik and J. Chlebikova. Crown reductions for the minimum weighted vertex cover problem. In *Electronic Colloquium on Computational Complexity, Report No. 101*, 2004.

[11] R. Downey and M. Fellows. Fixed-parameter tractability and completeness. *Congressus Numerantium*, 87:161–187, 1992.

[12] R. Downey and M. Fellows. *Parameterized Complexity*. Springer, New York, 1999.

[13] Ch. Ebengger, P. Hammer, and D. de Werra. Pseudo-boolean functions and stability of graphs. *Annals of Discrete Mathematics*, 19:83–98, 1984.

[14] M. Fellows. Blow-ups, win/win's and crown rules: some new directions in FPT. volume 2880 of *Lecture Notes in Computer Science*, pages 1–12, 2003.

[15] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979.

[16] R. Impagliazzo and R. Paturi. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63:512–530, 2001.

[17] D. S. Johnson and M. A. Tricks Eds. *Cliques, Coloring and Satisfiability, Second DIMACS Implementation Challenges*, volume 26. American Mathematical Society Providence, RI, 1996.

[18] G. Nemhauser and L. Trotter. Vertex packing: structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975.

[19] R. Niedermeier and P. Rossmanith. Upper bounds for Vertex Cover further improved. In *Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pages 561–570, 1999.

[20] R. Niedermeier and P. Rossmanith. A general method to speed up fixed-parameter-tractable algorithms. *Information Processing Letters*, 73:3-4:125–129, 2000.

[21] R. Niedermeier and P. Rossmanith. On efficient fixed-parameter algorithms for weighted vertex cover. *Journal of Algorithms*, 47:63–77, 2003.

[22] J. M. Robson. Algorithms for maximum independent set. *Journal of Algorithms*, 6:425–440, 1977.

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

[24] G. Woeginger. Exact algorithms for NP-hard problems: a survey. In *Combinatorial Optimization - Eureka! You shrink!*, volume 2570 of *Lecture Notes in Computer Science*, pages 185–207, 2003.