Laboratoire d'Analyse et Modélisation de Systèmes pour l'Aide à la Décision
CNRS UMR 7243

CAHIER DU LAMSADE

307

Mars 2011

On the MAX *k*-VERTEX COVER problem

Frederico Della Croce, Vangelis Th. Paschos

On the MAX k-VERTEX COVER problem*

Federico Della Croce[1] Vangelis Th. Paschos[2]

[1]D.A.I., Politecnico di Torino, Torino, Italy, federico.dellacroce@polito.it
[2]LAMSADE, CNRS and Université Paris-Dauphine
Institut Universitaire de France
paschos@lamsade.dauphine.fr

March 29, 2011

Abstract

We consider the MAX k-VERTEX COVER problem where we search for k vertices in an undirected graph G such that the total number of edges covered by these vertices is maximized. First, a combinatorial algorithm able to compute the optimal solution in $O^*(2^\tau)$ is proposed where τ is the size of the minimum vertex cover on the same graph. Such algorithm leads also to a running time $O^*(2^{\frac{\Delta-1}{\Delta}n})$ where Δ is the maximum degree of the graph. Then, an exact branch and reduce algorithm based on the measure and conquer paradigm is proposed requiring running time $O^*(2^{\frac{\Delta-1}{\Delta+1}n})$. Such algorithm is then tailored to graphs with maximum degree 3 inducing a running time $O^*(1.3339^n)$. We finally study approximation of MAX k-VERTEX COVER by moderately exponential algorithms.

1 Introduction

In the MAX k-VERTEX COVER problem a graph $G(V, E)$ with $|V| = n$ vertices $1, \ldots, n$ and $|E|$ edges (i, j) is given together with an integer value $k < n$. The goal is to find a subset $K \subset V$ with cardinality k, that is $|K| = k$, such that the total number of edges covered by K is maximized. The problem is well known to be **NP**-hard and has been shown to be fixed-parameter intractable in [8].

MAX k-VERTEX COVER is known to be polynomially approximable within approximation ratio $3/4$, while it cannot be solved by a polynomial time approximation schema unless $\mathbf{P} = \mathbf{NP}$. The interested reader can be referred to [14, 18] for more information about approximation issues for this problem.

In this paper we consider the worst-case complexity of exact algorithms for MAX k-VERTEX COVER. Throughout the paper we use notation $O^*(\cdot)$ to to measure running time of an algorithm ignoring polynomial factors. To the authors knowledge, the only available result is the one of [8], where an exact algorithm with complexity $O^*(n^{\omega\lceil k/3\rceil+O(1)})$ was proposed based upon a generalization of the $O^*(n^{\omega t})$ algorithm of [19] for finding a $3t$-clique in a graph, where $\omega = 2.376$. This turns to a complexity $O^*(n^{0.792k})$. Let us note that a trivial optimal algorithm for MAX k-VERTEX COVER takes time $O^*(\binom{n}{k}) = O^*(n^k)$ producing all the subsets of V of size k. This turns to a worst-case $O^*(2^n)$ time (since $\binom{n}{k} \leqslant 2^n$ with equality for $k = \frac{n}{2}$). To authors knowledge, no exact algorithm is known with running time $O^*(\gamma^n)$, for some $\gamma < 2$. Note finally that, as it is proved in [8], MAX k-VERTEX COVER is not fixed-parameter tractable (i.e., it does not belong to **FPT**, the class of the fixed parameter tractable problems); in other words, it cannot be solved by exact algorithms with running time $c^{\mathrm{opt}(G)}$ where c is a fixed constant and $\mathrm{opt}(G)$ is the value of an optimum solution, i.e., the maximum number of edges covered by k vertices in G.

Here, several exact and low-exponential complexity approximation algorithms are presented. Some of them are based on the so-called branch-and-reduce paradigm where for each level of the search tree we define as *fixed* those vertices that have already been selected or discarded, while we define as *free* the other vertices.

In what follows, we denote by α_j the total number of vertices adjacent to j that have been discarded in the previous levels of the search tree and we say that j is α-mutilated. We denote by d_j the degree

*Research supported by the French Agency for Research under the DEFIS program TODO, ANR-09-EMER-010.

of vertex j and by $N(j)$ the set of vertices adjacent to j, that is the neighborhood of j. Notice that, whenever a branch on a vertex j occurs with another $l \in N(j)$, if j is selected then d_l is decreased by one unit as edge (j,l) is already covered by j. Alternatively, j is discarded: correspondingly d_l is not modified and α_l is increased by one unit.

The above are the basic ingredients of the most of our algorithms for MAX k-VERTEX COVER. Based upon them, an exact branch and reduce algorithm based upon the measure and conquer paradigm by [16] is proposed in Section 4 requiring running time $O^*(2^{\frac{\Delta-1}{\Delta+1}n})$, where Δ denotes the maximum degree of G. Such algorithm is then tailored to graphs with maximum degree 3 inducing a running time $O^*(1.3339^n)$ (Section 5). Let us note that a purely combinatorial exact algorithm is proposed in Section 3 with complexity $O^*(2^\tau)$ where τ is the cardinality of a minimum vertex cover of G. This, to our knowledge, is the first result where the time-complexity of a problem is measured with respect to the optimal value of another problem. In Section 6, we address the question of approximating MAX k-VERTEX COVER within ratios "prohibited" for polynomial time algorithms, by algorithms running with moderately exponential complexity. The general goal of this issue is to catch-up on polynomial inapproximability, by developing algorithms achieving, with worst-case running times importantly smaller than those needed for exact computation, approximation ratios unachievable in polynomial time. This approach has already been considered for several other paradigmatic problems such as MINIMUM SET COVER [6, 11], MIN COLORING [2, 5], MAX INDEPENDENT SET and MIN VERTEX COVER [4], MIN BANDWIDTH [12, 17], ... Similar issues arise in the field of FPT algorithms, where approximation notions have been introduced, for instance, in [9, 13].

2 Preliminaries – General recurrence

Consider the vertex j with maximum degree Δ and neighbors $l_1, \ldots, l_\Delta$. As j has maximum degree, we may assume that if there exists an optimal solution of the problem where all neighbors of j are discarded, then there exists at least one optimal solution where j is selected and a branching scheme (called *basic branching scheme*) on j of type:

$$\left[l_1, \left(\overline{l_1} - l_2\right), \left(\overline{l_1} - \overline{l_2} - l_3\right), \ldots, \left(\overline{l_1} - \overline{l_2} - \ldots - \overline{l_{\Delta-1}} - l_\Delta\right), \left(\overline{l_1} - \overline{l_2} - \ldots - \overline{l_\Delta} - j\right)\right]$$

can be applied. This simple result can be actually improved as the following Lemma holds.

Lemma 1. *Consider vertex j with maximum degree Δ and neighbors $l_1, \ldots, l_\Delta$ where the basic branching scheme of type $[l_1, (\overline{l_1} - l_2), (\overline{l_1} - \overline{l_2} - l_3), \ldots, (\overline{l_1} - \overline{l_2} - \ldots - \overline{l_{\Delta-1}} - l_\Delta), (\overline{l_1} - \overline{l_2} - \ldots - \overline{l_\Delta} - j)]$ can be applied. Then, the last two branches can be substituted by the branch $(\overline{l_1} - \overline{l_2} - \ldots - \overline{l_{\Delta-1}} - j)$.*

Proof. If all neighbors of j but one are not selected, any solution including the last neighbor l_Δ but not including j is not better than the solution that selects j. ∎

From Lemma 1 the following preliminary but interesting result holds for MAX k-VERTEX COVER.

Proposition 1. *The* MAX k-VERTEX COVER *problem can be solved to optimality in $O^*(\Delta^k)$.*

Proof. From Lemma 1 we can see that the basic branching scheme generates Δ nodes. On the other hand, we know that in each branch of the basic branching scheme at least one vertex is selected. As, at most k nodes can be selected, the overall complexity cannot be superior to $O^*(\Delta^k)$. ∎

Note that, according to Proposition 1, in graphs of bounded maximum degree b, MAX k-VERTEX COVER is somehow "better than FPT", since it can be solved in time $O^*(b^k)$ and, in general, $k \leqslant \mathrm{opt}(G)$.

3 An $O^*(2^{\frac{\Delta-1}{\Delta}n})$ algorithm based on the minimum vertex cover problem

Another general approach can be devised if we consider the parameterized solution of the minimum vertex cover problem. Let $V' \subset V$ be a minimum vertex cover of G and let τ be the size of V' that is $\tau = |V'|$. Correspondingly, let $I = G \setminus V'$ be a maximum independent set of G and set $\alpha = |I|$. Notice that V' can be computed, for instance, in $O^*(1.28^\tau)$ time by means of the fixed parameter algorithm of [10].

Let us note that we can assume $k \leqslant \tau$. Otherwise, one could compute a minimum vertex cover V' (that covers all the edges of E) and then arbitrarily add $k - \tau$ vertices without changing the value ($|E|$) of the optimal solution.

Theorem 1. MAX k-VERTEX COVER *can be solved to optimality in $O^*(2^\tau)$ time.*

Proof. Consider all subsets of V' with cardinality $\leqslant k$. Notice that we have at most $\binom{|V'|}{k} \leqslant 2^{|V'|} = 2^{\tau}$ such subsets. The optimal solution K^* of MAX k-VERTEX COVER must necessarily contain one of such subsets (eventually the empty set) or else it would be an independent set. Suppose that $S' \subseteq V'$ is such subset. Then, $K^* = S' \cup I'$, where $I' \subset I$ is obviously an independent set. But, given S' with cardinality $k' = |S'| \leqslant k$, I' can be computed in polynomial time. Indeed, for each vertex i belonging to I we need simply to compute (in linear time) the total number e_i of edges (i, j) for all $j \in V' \setminus S'$. Then, I' is obtained by selecting the $k - k'$ vertices of I with largest e_i value.

By repeating this approach for all subsets of V' cardinality less than, or equal to, k, the best obtained solution is the optimal solution K^* of MAX k-VERTEX COVER. As computing V' requires $O^*(1.28^{\tau})$ time and there are at most $\binom{n}{k'} \leqslant 2^{\tau}$ subsets of V' where for each of these subsets the related vertices of I can be selected in polynomial time, the overall complexity becomes $O^*(1.28^{\tau} + \binom{n}{k'}) \leqslant O^*(2^{\tau})$. ∎

From Theorem 1 we derive the following corollary result.

Proposition 2. MAX k-VERTEX COVER *can be solved to optimality in* $O^*(2^{\frac{\Delta-1}{\Delta}n})$ *time.*

Proof. If a graph G has maximum degree Δ, then for the maximum independent set we have $\alpha \geqslant \frac{n}{\Delta}$. Indeed, we can assume that G is not a clique on $\Delta + 1$ vertices (note that MAX k-VERTEX COVER is polynomial in cliques). In this case, G can be colored with Δ colors [7]. In such a coloring the cardinality of the largest color is greater than $\frac{n}{\Delta}$ and, a fortiori, so is the cardinality of a maximum independent set (since each color is an independent set). Consequently, $\tau \leqslant \frac{\Delta-1}{\Delta}n$. Hence, from Theorem 1, we get $O^*(2^{\tau}) \leqslant O^*(2^{\frac{\Delta-1}{\Delta}n})$. ∎

In Table 1, are indicated the complexity results of bounded degree graphs provided by Proposition 2.

Δ	3	4	5	6	7	8	9	10
Complexity	1.5874^n	1.6817^n	1.7411^n	1.7817^n	1.8114^n	1.834^n	1.8517^n	1.866^n

Table 1: Complexity results for small values of Δ provided by Proposition 2.

Note that MAX k-VERTEX COVER is **NP**-hard even in bipartite graphs. Based upon Theorem 1 and taking into account that, in a bipartite graph of order n, a minimum vertex cover is smaller than, or equal to $n/2$, the following corollary is immediately derived.

Corollary 1. MAX k-VERTEX COVER *is optimally solved in* $O^*(2^{\frac{n}{2}}) = O^*(1.41^n)$ *in bipartite graphs.*

4 A measure and conquer $O^*(2^{\frac{\Delta-1}{\Delta+1}n})$ algorithm

Here we propose a branch and reduce approach based on the measure and conquer paradigm (see for instance [16]). Consider a classical binary branching scheme on some vertex j where j is either selected or discarded. We do not count in the measure the fixed vertices, namely the vertices that have been either selected or discarded at an earlier stage of the search tree and we count with a weight w_h the free vertices h. The vertex j to be selected is the one with largest coefficient $c_j = d_j - \alpha_j$. Let $c_{\max}$ denote such coefficient, hence $c_{\max} \leqslant \Delta$. Then, each free vertex h is assigned a weight $w_h = w_{[i]}$ with $i = c_i = d_h - \alpha_h$ and we impose $w_{[0]} \leqslant w_{[1]} \leqslant w_{[2]} \leqslant w_{[3]} \leqslant \ldots \leqslant w_{[c_{\max}]} = 1$ that is the weights of the vertices are strictly increasing in their c_j coefficients.

We so get recurrences on the time $T(p)$ required to solve instances of size p, where the size of an instance is the sum of the weights of its vertices. Since initially $p = n$, the overall running time is expressed as function of n. This is valid since when $p = 0$, there are only vertices with weight w_0 in the graph and, in this case, the problem is immediately solved by selecting the $k - \gamma$ vertices with largest α_j (if $\gamma < k$ vertices have been selected so far). Correspondingly free vertices j with no adjacent free vertices receive weight $w_0 = 0$.

We claim that MAX k-VERTEX COVER can be solved with running time $O^*(2^{\frac{\Delta-1}{\Delta+1}n})$ by the following algorithm called `MAXKVC`:

Select j such that c_j is maximum and branch according to the following exhaustive cases:

1. if $c_j \geqslant 3$, then branch on j and either select or discard j;

2. else, $c_j \leqslant 2$ and `MAXKVC` is polynomially solvable.

Theorem 2. *Algorithm* ***MAXKVC*** *solves* MAX k-VERTEX COVER *with running time* $O^*(2^{\frac{\Delta-1}{\Delta+1}n})$.

Proof. To prove the above statement, we first show that the branch in step 1 can be solved with complexity $O^*(2^{\frac{\Delta-1}{\Delta+1}n})$ and then we show that step 2 is polynomially solvable. Consider step 1. We always branch on the vertex j with largest $c_j = c_{\max} \leqslant \Delta$ where $c_j \geqslant 3$ and either we select or discard j. If we select j, vertex j is fixed and $c_{\max}$ vertices (the neighbors of j) decrease their degree (and correspondingly their coefficient) by one unit. Similarly, if we discard j, vertex j is fixed and $c_{\max}$ vertices (the neighbors of j) decrease their coefficient as their degree remains unchanged by their α parameter is increased by one unit. Hence, the recurrence becomes:

$$T(p) \leqslant 2T\left(p - w_{[c_{\max}]} - \sum_{h \in N(j)} \left(w_{[c_h]} - w_{[c_h-1]}\right)\right)$$

By constraining the weights to satisfy the inequality $w_{[j]} - w_{[j-1]} \leqslant w_{[j-1]} - w_{[j-2]}$, $\forall j = 2, \ldots, c_{\max}$, the previous recurrence becomes in the worst-case:

$$T(p) \leqslant 2T\left(p - w_{[c_{\max}]} - c_{\max}\left(w_{[c_{\max}]} - w_{[c_{\max}-1]}\right)\right)$$

As $c_{\max} \leqslant \Delta$, where the equality occurs when $\alpha_j = 0$, the above recurrence becomes in the worst-case:

$$T(p) \leqslant 2T\left(p - w_{[\Delta]} - \Delta\left(w_{[\Delta]} - w_{[\Delta-1]}\right)\right)$$

Summarizing, to handle graphs with maximum degree Δ, we need to guarantee that the recurrences $T(p) \leqslant 2T(p - w_{[i]} - i(w_{[i]} - w_{[i-1]}))$, $\forall i \in 3, \ldots, \Delta$ (as $c_j \geqslant 3$), and the constraints:

$$\begin{array}{rcl} w_{[i]} - w_{[i-1]} & \leqslant & w_{[i-1]} - w_{[i-2]} \ \ \forall i = 2, \ldots, \Delta \\ 0 = w_{[0]} \leqslant w_{[1]} \leqslant w_{[2]} \leqslant w_{[3]} \leqslant \ldots & \leqslant & w_{[\Delta-1]} \leqslant w_{[\Delta]} = 1 \end{array}$$

are satisfied contemporaneously. This corresponds to a non linear optimization problem of the form:

$$\begin{array}{ll} \min & \alpha \\ & \alpha^{(w_{[i]} + i(w_{[i]} - w_{[i-1]}))} \geqslant 2 \ \ \forall i = 3, \ldots, \Delta \quad (1) \\ & w_{[i]} - w_{[i-1]} \leqslant w_{[i-1]} - w_{[i-2]} \ \ \forall i = 2, \ldots, \Delta \quad (2) \\ & 0 = w_{[0]} \leqslant w_{[1]} \leqslant w_{[2]} \leqslant w_{[3]} \leqslant \ldots \leqslant w_{[\Delta-1]} \leqslant w_{[\Delta]} = 1 \quad (3) \end{array}$$

By means of a non linear solver [1], we get Table 2 presenting the corresponding performances of the algorithm for small values of Δ.

Δ	3	4	5	6	7	8	9	10
Complexity	1.4142^n	1.5157^n	1.5874^n	1.6407^n	1.6818^n	1.7145^n	1.7411^n	1.7632^n

Table 2: Complexity results for small values of Δ with the measure and conquer approach.

Interestingly enough, for all these values of Δ, the complexity corresponds to $O^*(2^{\frac{\Delta-1}{\Delta+1}n})$. Indeed, this is not accidental. By setting:

$$\begin{array}{rcll} w_{[i]} & = & \dfrac{(i-1)(\Delta+1)}{(i+1)(\Delta-1)} \ \ \forall i = 2, \ldots, \Delta & (4) \\ w_{[1]} & = & \dfrac{1}{2} w_{[2]} & (5) \\ w_{[0]} & = & 0 & (6) \end{array}$$

we can see that constraints (2) and (3) are satisfied. To see that inequalities (2) are satisfied, notice that:

$$\begin{aligned} w_{[3]} - w_{[2]} &= w_{[2]} - w_{[1]} = \frac{1}{3}w_3 \\ w_{[2]} - w_{[1]} &= w_{[1]} - w_{[0]} = w_1 \end{aligned}$$

For the general recursion with $i \geqslant 4$, we have to show that $w_{[i]} - w_{[i-1]} \leqslant w_{[i-1]} - w_{[i-2]}$, i.e., that $w_{[i]} - 2w_{[i-1]} + w_{[i-2]} \leqslant 0$. This corresponds to:

$$\begin{aligned} & \left(\frac{i-1}{i+1} - 2\frac{i-2}{i} + \frac{i-3}{i-1}\right)\left(\frac{\Delta+1}{\Delta-1}\right) \leqslant 0 \implies \frac{i-1}{i+1} - 2\frac{i-2}{i} + \frac{i-3}{i-1} \leqslant 0 \\ \iff & i(i-1)^2 - 2(i-2)\left(i^2-1\right) + i(i-3)i + 1 \leqslant 0 \\ \iff & i^3 - 2i^2 + i - 2i^3 + 4i^2 + 2i - 4 + i^3 - 2i^2 - 3i = -4 \leqslant 0, \quad \forall i \end{aligned}$$

Also, to see that inequalities (3) are satisfied, notice that equations (4) imply:

$$\begin{aligned} w_{[\Delta]} &= 1 \\ w_{[i]} &> 0 \ \forall i = 2, \ldots, \Delta \\ w_{[i]} &> w_{[i-1]} \ \forall i = 3, \ldots, \Delta \end{aligned}$$

while equations (5) and (6) imply $w_{[2]} > w_{[1]} > w_{[0]} = 0$.

Finally, notice that such values of $w_{[j]}s$ satisfy constraints (1) that now correspond to $\Delta - 2$ copies of the inequality $\alpha^{\frac{\Delta+1}{\Delta-1}} \geqslant 2$ where the minimum value of α is obviously given by $2^{\frac{\Delta-1}{\Delta+1}n}$. Correspondingly, the overall complexity of step 1 is $O^*(2^{\frac{\Delta-1}{\Delta+1}n})$.

We consider now step 2. For $c_j = c_{\max} \leqslant 2$, MAX k-VERTEX COVER can be seen as a maximum weighted k-vertex cover problem in an undirected graph G where each vertex j has a weight α_j and a degree $d_j = c_j$ and we search for k vertices such that the total number of edges covered by these vertices plus the sum of the α_js related to the selected vertices is maximized. Let denote by MAX WEIGHTED k-VERTEX COVER such problem. As $d_j = c_j \leqslant c_{\max} \leqslant 2$, each vertex has at most two edges and the graph is then a collection of paths or cycles. But then this problem can be solved to optimality by means of dynamic programming approach. We first show that the approach works on a single path and then extend the result to the more general collection of paths and cycles.

Consider a path $P = \{1, \ldots, n\}$. Let $P(i)$ denote the path composed by vertices $1, \ldots, i$ (notice that if $P(i)$ is considered with $i \leqslant n-1$, then the edge $(i, i+1)$ is not considered). Let denote by $\mathrm{WKVC}(h, i, 0)$ the optimal solution of MAX WEIGHTED k-VERTEX COVER on $P(i)$ for $k = h$ when i is discarded. Correspondingly, let denote by $\mathrm{WKVC}(h, i, 1)$ the optimal solution of MAX WEIGHTED k-VERTEX COVER on $P(i)$ for $k = h$ when i is selected. Finally, let denote by $\mathrm{WKVC}_P(h)$ the optimal solution of MAX WEIGHTED k-VERTEX COVER on P for any given h. We can formulate the following recursive equations, $\forall i = 2, \ldots, n-1,\ h = 1, \ldots, k$:

$$\mathrm{WKVC}(h, i, 0) = \max\{\mathrm{WKVC}(h, i-1, 0), \mathrm{WKVC}(h, i-1, 1) + 1\} \tag{7}$$

where in the second term of (7), if i is discarded and $i-1$ is selected, then the edge $(i, i+1)$ is covered and $\mathrm{WKVC}(h, i, 1) = \max\{\mathrm{WKVC}(h-1, i-1, 0) + \alpha_i + 1, \mathrm{WKVC}(h-1, i-1, 1) + \alpha_i + 1\}$.

Finally, for $i = n$ we have:

$$\begin{aligned} \mathrm{WKVC}(h, n, 0) &= \max\{\mathrm{WKVC}(h, n-1, 0), \mathrm{WKVC}(h, n-1, 1) + 1\} \\ \mathrm{WKVC}(h, n, 1) &= \max\{\mathrm{WKVC}(h-1, n-1, 0) + \alpha_n + 1, \mathrm{WKVC}(h-1, n-1, 1) + \alpha_n + 1\} \end{aligned}$$

Then, the optimal solution $\mathrm{KVC}(k)$ of MAX WEIGHTED k-VERTEX COVER on P is given by:

$$\mathrm{KVC}(k) = \max\{\mathrm{WKVC}(k, n, 0), \mathrm{WKVC}(k, n, 1)\}$$

The initial conditions are:

$$\begin{aligned} \mathrm{WKVC}(h, 1, 0) &= -\infty \ \forall h \in 1, \ldots, k \\ \mathrm{WKVC}(h, 1, 1) &= -\infty \ \forall h \in 2, \ldots, k \\ \mathrm{WKVC}(1, 1, 1) &= \alpha_1 \end{aligned}$$

But then, as there are $O^*(nk)$ of such $WKVC(\cdot,\cdot,\cdot)$ solutions, and all solutions can be computed in constant time (either from the initial conditions or from the recursive equations), then MAX WEIGHTED k-VERTEX COVER on a path is solvable in polynomial time. Notice, that as a byproduct the procedure provides $\mathrm{KVC}(h)$, $\forall h \in 1,\ldots,k$ namely the optimal solution of our problem for all values of $h \in 1,\ldots,k$ where h denotes the cardinality of the set of selected vertices.

Similarly, if we have a cycle $1,\ldots,n$ instead of a path, then it is sufficient to solve twice (then, still in polynomial time) the problem on path $2,\ldots,n$ where vertex 1 is either selected or discarded and take the best among the two solutions. Finally, if a collections of independent paths and cycles is present, then it is sufficient to merge one at a time the solutions of a couple of such paths or cycles for all values of $h = 1,\ldots,k$. Consider, for instance, paths P_1 and P_2. This corresponds to compare two vectors of size k, $\mathrm{KVC}_{P_1}(h)$ and $\mathrm{KVC}_{P_2}(h)$, $\forall h \in 1,\ldots,k$, and take in time $O(k^2)$ the best pairs $(\mathrm{KVC}_{P_1}(i), \mathrm{KVC}_{P_2}(j))$ such that $i + j = h$, $\forall h = 1,\ldots,k$. As the number of independent paths and cycles is bounded by the number of vertices in the graph, the complexity remains polynomial. ∎

We now show how the result of Theorem 2 can be improved by *trading space for time*. For this, we use the principle of *memorization* (see [15]) works as follows. Before running the algorithm on the main graph, we run it on every induced subgraph of size at most αn and store the results in a table. In absence of weights, as fixed vertices never change their status, the number of subproblems of size αn to consider is $\binom{n}{\alpha n}$. Besides, thanks to the recurrence defined above, we only need to call a finite number of subproblems of size $k-1$ or less in order to compute a given subproblem of size k. Then, using a classical bottom up technique, the total computation time (and space) for all the subproblems of size k, say $S(k,G)$, is at most $O^*(\binom{n}{k} + \sum_{i \leqslant k-1} S(i,G))$, from what we get (for $\alpha \leqslant 1/2$): $\sum_{k \leqslant \alpha n} S(k,G) \in O^*(\binom{n}{\alpha n})$.

Then, if the polynomial space algorithm has complexity $O^*(\beta)^n$, we run the main algorithm until the remaining graph has size αn or less; then a polynomial-time query in the storage table allows us to conclude. The total running time and space is then $O^*(\max\{\binom{n}{\alpha n}, \beta^{(1-\alpha)n}\})$. This value is then minimal for an α solution of the equation:

$$\beta^{1-\alpha} = \frac{1}{\alpha^\alpha (1-\alpha)^{1-\alpha}} \tag{8}$$

In our case, due to the presence of weights, we know that to reach a subgraph of size at most αn we need to run the main algorithm until the remaining graph has weight $w_1 \alpha n$ (as in the worst-case all vertices but a constant number may have the smallest weight w_1). Correspondingly, (8) becomes:

$$\beta^{1-w_1\alpha} = \frac{1}{\alpha^\alpha (1-\alpha)^{1-\alpha}} \tag{9}$$

and, as $\beta = 2^{\frac{\Delta-1}{\Delta+1}}$ and $w_1 = \frac{\Delta+1}{6(\Delta-1)}$, we get $2^{(\frac{\Delta-1}{\Delta+1} - \frac{\alpha}{6})} = \frac{1}{\alpha^\alpha(1-\alpha)^{1-\alpha}}$.

In Table 3, are indicated the complexity results on small values of Δ when the memorization approach is applied.

Δ	3	4	5	6	7	8	9	10
Complexity	1.3973	1.4919	1.5579	1.6066	1.6438	1.6733	1.6971	1.7168

Table 3: Complexity results for small values of Δ with the memorization approach.

5 Tailoring measure and conquer to graphs with maximum degree 3

We consider now the case where $\Delta = 3$. In order to tailor the measure and conquer approach to graphs with $\Delta = 3$, The following remark holds.

Remark 1. The graph can be cubic just once. When branching on a vertex j of maximum degree 3, we can always assume that it is adjacent to at least one vertex k that has already been selected or discarded. That is, either $d_k \leqslant 2$, or $\alpha_k \geqslant 1$, that is $c_k \leqslant 2$. Indeed, the situation where the graph is 3-regular occurs at most once (even in case of disconnection). Thus, we make only one "bad" branching (where every free vertex of maximum degree 3 is adjacent only to free vertices of degree 3). Such branching may increase the global running time only by a constant factor. ∎

The following Lemmata also hold.

Lemma 2. *Any vertex i with $d_i \leqslant 1$ and $\alpha_i = 0$ can be discarded wlog.*

Proof. If $d_i = \alpha_i = 0$, then i can be obviously discarded. If $d_i = 1$ and $\alpha_i = 0$, then i is adjacent to another free vertex k. But then, if k is selected, i becomes of degree 0 and can be discarded. Alternatively, k is discarded, but then any solution with i but not k is dominated by the one including k instead of i. ∎

Lemma 3. *Any vertex i with $\alpha_i \geqslant 2$ and $d_i = 3$ can be selected wlog.*

Proof. If $\alpha_i = 3$, then i can be obviously selected. If $d_i = 3$ and $\alpha_i = 2$, then i is adjacent to another free vertex k. But then, if k is discarded, we have $\alpha_i = 3$ and i can be selected. Alternatively k is selected, but then any solution with k but not i is dominated by the one including i instead of k. ∎

Then, we claim that we can solve MAX k-VERTEX COVER on graphs with $\Delta = 3$ with running time $O^*(1.3339^n)$ by the following algorithm called `MAXKVC-3`:

Select j such that c_j is maximum and branch according to the following exhaustive cases:

1. if $c_j = 3$, where wlog j is adjacent to i, l, m free vertices with $c_i \leqslant 2$ (due to Remark 1) and $c_i \leqslant c_l \leqslant c_m$, then branch on j according to the following exhaustive subcases.
   (a) $c_i = c_l = c_m = 1$
   (b) $c_i = c_l = 1$, $c_m = 2$
   (c) $c_i = c_l = 1$, $c_m = 3$
   (d) $c_i = 1$, $c_l = c_m = 2$ with l, m adjacent
   (e) $c_i = 1$, $c_l = c_m = 2$ with l, m non adjacent
   (f) $c_i = 1$, $c_l = 2$, $c_m = 3$
   (g) $c_i = c_l = 2$, $c_m = 3$ with i, l adjacent
   (h) $c_i = c_l = 2$, $c_m = 3$ with i, l non adjacent
   (i) $c_i = 2$, $c_l = c_m = 3$
2. else $c_j \leqslant 2$ and `MAXKVC-3` is polynomially solvable.

Theorem 3. *Algorithm `MAXKVC-3` solves* MAX k-VERTEX COVER *on graphs with maximum degree* 3 *with running time* $O^*(1.3339^n)$.

Proof. Similarly to the general case, we count with a weight w_h the free vertices h, where each free vertex h is assigned a weight $w_h = w_{[i]}$ with $i = c_h = d_h - \alpha_h$. Also, we impose $w_{[0]} = 0$, $w_{[1]} = 0.2909$, $w_{[2]} = 0.5818$ and $w_{[3]} = 1$, hence $w_j = 1$. Notice that with these values we have $w_{[1]} = w_{[2]} - w_{[1]} = 0.2909 < w_{[3]} - w_{[2]} = 0.4182$. The overall complexity is then due to the worst-case among subcases 1a, 1b, 1c, 1d, 1e, 1f, 1g, 1h and 1i.

Subcase 1a. Vertices i, j, l, m form an independent subgraph disjoint from the rest of the graph. Then, as in the case of independent cycles and paths described in the last paragraph of Section 4, the optimal solution of this subgraph for any value of k can be computed with constant complexity and taken aside to be merged to the solution of the rest of graph. Then, with respect to the remaining graph, i, j, l, m are fixed without branching.

Subcase 1b. We branch on vertex m and either we select or discard m. In both cases, i, j, l form a disconnected subgraph and can be fixed similarly to subcase 1a, while the other neighbor q of m decreases its coefficient by one unit where the worst-case occurs when $c_q = 2$ as, for $c_q = 1$, vertices i, j, l, m, q would all together form an independent subgraph and no branching would occur. Hence, the recursion is $T(p) \leqslant 2T(p - w_{[3]} - w_{[2]} - 2w_{[1]} - (w_{[2]} - w_{[1]})) \approx 2T(p - 2.4545)$ with complexity $O^*(1.3263^n)$.

Subcase 1c. We branch on vertex m and either we select or discard m. In both cases, i, j, l form a disconnected subgraph and can be fixed similarly to subcase 1a, while the other neighbors q and r of m decrease their coefficients by one unit where the worst-case occurs when $c_q = 1$ and $c_r = 2$ as for $c_q = c_r = 1$ vertices i, j, l, m, q, r would all together form an independent subgraph and no branching would occur. Hence, the recursion is $T(p) \leqslant 2T(p - 2w_{[3]} - 3w_{[1]} - (w_{[2]} - w_{[1]})) \approx 2T(p - 3.1636)$ with complexity $O^*(1.245^n)$.

Subcase 1d. Vertices i, j, l, m form an independent subgraph disjoint from the rest of the graph and the same consideration of subcase 1a holds.

Subcase 1e. We branch on vertex j and either we select or discard j. Notice that, as $c_l = 2$, either $d_l = 3$ and $\alpha_l = 1$, or $d_l = 2$ and $\alpha_l = 0$. Now, for $d_l = 2$, if we select j, we have $d_l = 1$ and $\alpha_l = 0$ and, due to Lemma 2, i can be discarded and therefore fixed. Similarly, for $d_l = 3$, if we discard j, we have $d_l = 3$ and $\alpha_l = 2$ and, due to Lemma 3, l can be selected and therefore fixed. Overall, l will necessarily be fixed in one of the branches and correspondingly its neighbor decreases the coefficient by one unit. Similar consideration occurs for m as also $c_m = 2$. But then, the worst-case occurs when l and m are fixed in the same branch. Correspondingly, the recursion is $T(p) \leqslant T(p-w_{[3]}-2w_{[2]}-2w_{[1]}-w_{[1]})+T(p-w_{[3]}-2(w_{[2]}-w_{[1]})-w_{[1]}) \approx T(p-3.0363)+T(p-1.8727)$ with complexity $O^*(1.3339^n)$.

Subcase 1f. We branch on vertex m and either we select or discard m. Notice that, as $c_i = 1$, we have $d_i = 2$ and $\alpha_i = 1$ or else i would be fixed without branching either due to Lemma 2 or to Lemma 3. But then, when m is selected, we have $d_j = 2$ with $alpha_j = 0$ with i and j adjacents: this implies that j is dominated by i and that i and j cannot be selected together, that is j can be discarded. Notice also, that when m is fixed the other two neighbors q and r decrease their coefficient by one unit where the worst case occurs for $c_q = c_r = 1$. Correspondingly, the recursion is $T(p) \leqslant T(p-2w_{[3]}-(w_{[2]}-w_{[1]})-3w_{[1]})+T(p-w_{[3]}-(w_{[3]}-w_{[2]})-2w_{[1]}) \approx T(p-3.1636)+T(p-2)$ with complexity $O^*(1.3144^n)$.

Subcase 1g. We branch on vertex m and either we select or discard m. In both cases, i, j, l form a disconnected subgraph and can be fixed, while the other neighbors q and r of m decrease their coefficients by one unit where the worst-case occurs when $c_q = 1$ and $c_r = 2$ as for $c_q = c_r = 1$ vertices i, j, l, m, q, r would all together form an independent subgraph and no branching would occur. Hence, the recursion is $T(p) \leqslant 2T(p - 2w_{[3]} - 2w_{[2]} - w_{[1]} - (w_{[2]} - w_{[1]})) \approx 2T(p - 3.7454)$ with complexity $O^*(1.2033^n)$.

Subcase 1h. We branch on vertex j and either we select or discard j. Notice that, as in subcase 1e, overall, l will necessarily be fixed in one of the branches and correspondingly its neighbor will decrease the coefficient by one unit. Similar consideration occurs for i as also $c_i = 2$. But then, the worst-case occurs when i and l are fixed in the same branch. Correspondingly, the recursion is $T(p) \leqslant T(p - w_{[3]} - 2w_{[2]} - 2w_{[1]} - (w_{[3]} - w_{[2]})) + T(p - w_{[3]} - 2(w_{[2]} - w_{[1]}) - (w_{[3]} - w_{[2]})) \approx T(p - 3.1636) + T(p - 2)$ with complexity $O^*(1.3144^n)$.

Subcase 1i. We branch on vertex j and either we select or discard j. Notice that, as in subcase 1e, overall, i will necessarily be fixed in one of the branches and correspondingly its neighbor will decrease the coefficient by one unit. Correspondingly, the recursion is $T(p) \leqslant T(p - w_{[3]} - w_{[2]} - w_{[1]} - 2(w_{[3]} - w_{[2]})) + T(p - w_{[3]} - 2(w_{[3]} - w_{[2]}) - (w_{[2]} - w_{[1]})) \approx T(p - 2.7091) + T(p - 2.1273)$ with complexity $O^*(1.3339^n)$.

The worst-case is then given by subcases 1e and 1i with complexity $O^*(1.3339^n)$. ∎

By solving (9) for $\beta = 1.3339$ and $w_1 = 0.2909$, we get $\alpha = 0.081$ and, correspondingly, the following proposition holds and concludes the section.

Proposition 3. *By applying the memorization approach of section 4,* ***MAXKVC-3*** *solves* MAX k-VERTEX COVER *on graphs with maximum degree* 3 *with running time* $O^*(1.3249^n)$.

6 Approximating MAX k-VERTEX COVER by moderately exponential algorithms

We now show how one can get approximation ratios non-achievable in polynomial time using moderately exponential algorithms with worst-case running times better than those required for an exact computation (see [3, 4] for more about this issue). Denote by $\mathrm{opt}(G)$ the cardinality of an optimal solution for MAX k-VERTEX COVER in G and by $m(G)$, the cardinality of an approximate solution. Our goal is to study the approximation ratio $m(G)/opt(G)$.

In what follows, we denote, as previously, by K^* the optimal solution for MAX k-VERTEX COVER. Given a set K of vertices, we denote by $C(K)$, the set of edges covered by K (in other words, the value of a solution K for MAX k-VERTEX COVER is $|C(K)|$; also, according to our previous notation, $\mathrm{opt}(G) = |C(K^*)|$). We first prove the following easy lemma that will be used later.

Lemma 4. *For any $\lambda \in [0,1]$, the subset H^* of λk vertices of K^* covering the most of edges covered by K^*, covers more than $\lambda\,\mathrm{opt}(G)$ edges.*

Proof. Indeed, if the λk "best" vertices of K^* cover less than $\lambda\,\mathrm{opt}(G)$ edges, then any disjoint union of k/λ subsets of K^*, each of cardinality λk covers less than $\mathrm{opt}(G)$ edges, a contradiction. ∎

Now, for some $\lambda \in [0,1]$, run the following algorithm, denoted by `APMAXKVC1`:

1. take all the combinations $C_n^{\lambda k}$ of λk vertices among n;
2. for any of the sets K' produced in step 1, remove K' and $C(K')$ from G and run a ρ-approximation algorithm in the surviving graph in order to compute a solution K'' for the MAX $(1-\lambda)k$-VERTEX COVER problem;
3. output the best $\hat{K} = \hat{K}' \cup \hat{K}''$, i.e., the one that covers the most of edges, produced in step 2.

It is easy to see that the complexity of `APMAXKVC1` is bounded above by $O^*(n^{\lambda k})$. Furthermore, it can be implemented to run with polynomial space by simply storing the best current K.

Proposition 4. MAX k-VERTEX COVER *can be solved within approximation ratio $1-\epsilon$, for any $\epsilon > 0$, in time $O^*(n^{(1-4\epsilon)k})$ and with polynomial space.*

Proof. Fix an optimal solution K^* for MAX k-VERTEX COVER ($|K^*| = k$) in G. Denote by K_1^* the λk vertices of K^* that cover the most of edges of $C(K^*)$. Since K_1^* has been produced by algorithm `APMAXKVC1` in step 1, if $C(K_2^*)$ is the value of the solution for MAX $(1-\lambda)k$-VERTEX COVER, associated with K_1* produced in step 2, the following holds:

$$\left|C\left(\hat{K}' \cup \hat{K}''\right)\right| = \left|C\left(\hat{K}'\right)\right| + \left|C\left(\hat{K}''\right)\right| \geqslant |C\left(K_1^* \cup K_2^*\right)| = |C\left(K_1^*\right)| + |C\left(K_2^*\right)| \tag{10}$$

Consider the subgraph $G_1 = G[V \setminus K_1^*]$ of G induced by $V \setminus K_1^*$. There, the set $K'^* = K^* \setminus K_1^*$ is a feasible solution for MAX $(1-\lambda)k$-VERTEX COVER. Denoting by $\mathrm{opt}_{(1-\lambda)k}(G_1)$ the value of an optimal solution for MAX $(1-\lambda)k$-VERTEX COVER in G_1, we have:

$$|C\left(K'^*\right)| = \mathrm{opt}(G) - |C\left(K_1^*\right)| \leqslant \mathrm{opt}_{(1-\lambda)k}\left(G_1\right) \tag{11}$$

where, as previously, $\mathrm{opt}(G) = |C(K^*)|$, is the optimal value for MAX k-VERTEX COVER in G.

Recall also that, by step 2, $C(K_2^*)$ satisfies:

$$|C\left(K_2^*\right)| \geqslant \rho \underset{(1-\lambda)k}{\mathrm{opt}}\left(G_1\right) \tag{12}$$

Putting together (11) and (12) we get:

$$\begin{aligned} \frac{|C\left(K_1^*\right)| + |C\left(K_2^*\right)|}{\mathrm{opt}(G)} &\geqslant \frac{|C\left(K_1^*\right)| + \rho\,\mathrm{opt}_{(1-\lambda)k}\left(G_1\right)}{\mathrm{opt}(G)} \geqslant \frac{|C\left(K_1^*\right)| + \rho\left(\mathrm{opt}(G) - |C\left(K_1^*\right)|\right)}{\mathrm{opt}(G)} \\ &= \frac{\rho\,\mathrm{opt}(G) + (1-\rho)\,|C\left(K_1^*\right)|}{\mathrm{opt}(G)} \end{aligned} \tag{13}$$

By Lemma 4, $|C(K_1^*)| \geqslant \lambda\,\mathrm{opt}(G)$ and combining it with (10) and (13), we get:

$$\frac{m(G)}{\mathrm{opt}(G)} \geqslant \frac{\left|C\left(\hat{K}' \cup \hat{K}''\right)\right|}{\mathrm{opt}(G)} \geqslant \frac{|C\left(K_1^*\right)| + |C\left(K_2^*\right)|}{\mathrm{opt}(G)} \geqslant \rho + \lambda(1-\rho) \tag{14}$$

Taking $\rho = \frac{3}{4}$, the best known polynomially achieved ratio for MAX k-VERTEX COVER ([18]), in order to get an approximation ratio $1-\epsilon$, for some $\epsilon > 0$, one has to choose $\lambda = 1 - 4\epsilon$ and the proposition follows. ∎

We now improve the result of Proposition 4 by the following algorithm, called `APMAXKVC2` in what follows:

1. fix some $\lambda \in [0,1]$ and optimally solve MAX λk-VERTEX COVER in G (as previously, let H^* be the optimal solution built and $C(H^*)$ be the edge-set covered by H^*);
2. remove H^* and $C(H^*)$ from G and approximately solve MAX $(1-\lambda)k$-VERTEX COVER in the surviving graph (by some approximation algorithm); let K' be the obtained solution;
3. output $K = H^* \cup K'$.

It is easy to see that if $T(p,k)$ is the running time of an optimal algorithm for MAX k-VERTEX COVER, where p is some parameter of the input-graph G (for instance, n, or τ), then the complexity of `APMAXKVC2` is $T(p,\lambda k)$. Furthermore, `APMAXKVC2` requires polynomial space.

Theorem 4. *If $T(p,k)$ is the running time of an optimal algorithm for* MAX k-VERTEX COVER*, then, for any $\epsilon > 0$,* MAX k-VERTEX COVER *can be approximated within ratio $1-\epsilon$ and with worst-case running time $T(p,(1+2\sqrt{1-3\epsilon})k/3)$.*

Proof. Denote by K^* an optimal solution of MAX k-VERTEX COVER in G, by G_2 the induced subgraph $G[V \setminus H^*]$ of G, by $\mathrm{opt}_{(1-\lambda)}(G_2)$, the value of an optimal for MAX $(1-\lambda)k$-VERTEX COVER in G_2. Suppose that E' edges are common between $C(H^*)$ and $C(K^*)$. This means that $C(K^*) \setminus E'$ edges of $C(K^*)$ are in G_2 and are exclusively covered by the vertex-set $L^* = K^* \setminus H^*$ that belongs to G_2. Set $\ell^* = |L^*|$ and note that $\ell^* \leqslant k$ and $\ell^* \geqslant (1-\lambda)k$.

According to Lemma 4, the $(1-\lambda)k$ "best" vertices of L^* cover more than $(1-\lambda)|C(K^*) \setminus E'| = (1-\lambda)(\mathrm{opt}(G)-|E'|)$ edges in G_2 and these vertices constitute a feasible solution for MAX $(1-\lambda)k$-VERTEX COVER in G_2. Hence:

$$\mathrm{opt}_{(1-\lambda)}(G_2) \geqslant (1-\lambda)(\mathrm{opt}(G) - |E'|) \tag{15}$$

Taking into account (15), the fact that K' in step 2 of `APMAXKVC2` has been computed by, say, a ρ-approximation algorithm and the fact that $|E'| \leqslant |C(H^*)|$, we get:

$$\begin{aligned} m(G) &= C(H^*) + C(K') \geqslant C(H^*) + \rho(1-\lambda)\mathrm{opt}_{(1-\lambda)}(G_2) \\ &\geqslant C(H^*) + \rho(1-\lambda)(\mathrm{opt}(G) - |E'|)\, C(H^*) + \rho(1-\lambda)(\mathrm{opt}(G) - C(H^*)) \\ &\geqslant (1-\rho(1-\lambda))C(H^*) + \rho(1-\lambda)\,\mathrm{opt}(G) \end{aligned} \tag{16}$$

Using once more Lemma 4, $|C(H^*)| \geqslant \lambda\,\mathrm{opt}(G)$, and putting it together with (16), we get:

$$\frac{m(G)}{\mathrm{opt}(G)} \geqslant \rho(1-\lambda) + \lambda(1-\rho(1-\lambda)) \tag{17}$$

Setting $\rho = \frac{3}{4}$ in (17), in order to achieve an approximation ratio $m(G)/\mathrm{opt}(G) = 1-\epsilon$, for some $\epsilon > 0$, we have to choose an λ satisfying $\lambda = (1+2\sqrt{1-3\epsilon})/3$, that completes the proof of the theorem. ∎

Note that Proposition 4 and Theorem 4 reach the same complexity $O^(n^{0.737k})$ for $\epsilon = 0.0657$. In Table 4, a comparative study of the running times claimed by Proposition 4 and Theorem 4 with respect to some ratio values is given.

ϵ	0.2	0.15	0.1	0.08	0.0657	0.05	0.02	0.01	0.001
Proposition 4	$n^{0.2k}$	$n^{0.4k}$	$n^{0.6k}$	$n^{0.68k}$	$n^{0.737k}$	$n^{0.8k}$	$n^{0.92k}$	$n^{0.96k}$	$n^{0.996k}$
Theorem 4	$n^{0.598k}$	$n^{0.656k}$	$n^{0.706k}$	$n^{0.724k}$	$n^{0.737k}$	$n^{0.751k}$	$n^{0.776k}$	$n^{0.784k}$	$n^{0.791k}$

Table 4: Complexity results for small values of ϵ provided by Proposition 4 and Theorem 4.

It can be seen from Table 4 that Proposition 4 dominates Theorem 4 for large ϵ's and is dominated by Theorem 4 for small ϵ's (the break being on $\epsilon = 0.0657$ inducing complexity $O^*(n^{0.737k})$. Notice, also, that Proposition 4 makes sense for $\epsilon \geqslant 0.05$ only, as for lower values of ϵ the computational cost claimed by Proposition 4, becomes superior to that of the exact approach proposed in [8].

Corollary 2. MAX k-VERTEX COVER *can be approximated within ratio $1-\epsilon$ and with running time:*

$$\min\left\{O^*\left(n^{\left(1+2\sqrt{1-3\epsilon}\right)(\omega k)/9}\right), O^*\left(\left(\frac{\tau}{\left(1+2\sqrt{1-3\epsilon}\right)k/3}\right)\right), O^*(n^{(1-4\epsilon)k})\right\}$$

For Corollary 2, just observe that the running-times claimed for the first two entries are those needed to optimally solve MAX λk-VERTEX COVER (the former due to [8] and the latter due to Theorem 1), while the last running time corresponds to the complexity of the approach proposed in Proposition 4.

References

[1] Lingo 2.0 - optimization modeling software for linear, nonlinear, and integer programming. http://www.lindo.com/index.php, 2010.

[2] A. Björklund, T. Husfeldt, and M. Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.* To appear in the special issue dedicated to selected papers from FOCS'06.

[3] N. Bourgeois, B. Escoffier, and V. Th. Paschos. Efficient approximation by "low-complexity" exponential algorithms. Cahier du LAMSADE 271, LAMSADE, Université Paris-Dauphine, December 2007. Available at `http://www.lamsade.dauphine.fr/cahiers/PDF/cahierLamsade271.pdf`.

[4] N. Bourgeois, B. Escoffier, and V. Th. Paschos. Efficient approximation of combinatorial problems by moderately exponential algorithms. In F. Dehne, M. Gavrilova, J.-R. Sack, and C. D. Tóth, editors, *Proc. Algorithms and Data Structures Symposium, WADS'09*, volume 5664 of *Lecture Notes in Computer Science*, pages 507–518. Springer-Verlag, 2009.

[5] N. Bourgeois, B. Escoffier, and V. Th. Paschos. Efficient approximation of MIN COLORING by moderately exponential algorithms. *Inform. Process. Lett.*, 109(16):950–954, 2009.

[6] N. Bourgeois, B. Escoffier, and V. Th. Paschos. Efficient approximation of MIN SET COVER by moderately exponential algorithms. *Theoret. Comput. Sci.*, 410(21-23):2184–2195, 2009.

[7] R. L. Brooks. On coloring the nodes of a network. *Math. Proc. Cambridge Philos. Soc.*, 37:194–197, 1941.

[8] L. Cai. Parameter complexity of cardinality constrained optimization problems. *The Computer Journal*, 51:102–121, 2008.

[9] L. Cai and X. Huang. Fixed-parameter approximation: conceptual framework and approximability results. In H. L. Bodlaender and M. A. Langston, editors, *Proc. International Workshop on Parameterized and Exact Computation, IWPEC'06*, volume 4169 of *Lecture Notes in Computer Science*, pages 96–108. Springer-Verlag, 2006.

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

[11] M. Cygan, L. Kowalik, and M. Wykurz. Exponential-time approximation of weighted set cover. *Inform. Process. Lett.*, 109(16):957–961, 2009.

[12] M. Cygan and M. Pilipczuk. Exact and approximate bandwidth. *Theoret. Comput. Sci.*, 411(40–42):3701–3713, 2010.

[13] R. G. Downey, M. R. Fellows, and C. McCartin. Parameterized approximation problems. In H. L. Bodlaender and M. A. Langston, editors, *Proc. International Workshop on Parameterized and Exact Computation, IWPEC'06*, volume 4169 of *Lecture Notes in Computer Science*, pages 121–129. Springer-Verlag, 2006.

[14] U. Feige and M. Langberg. Approximation algorithms for maximization problems arising in graph partitioning. 41:174–211, 2001.

[15] F. V. Fomin, F. Grandoni, and D. Kratsch. Some new techniques in design and analysis of exact (exponential) algorithms. Bulletin of the European Association for Theoretical Computer Science 87, pp. 47–77, 2005.

[16] F. V. Fomin, F. Grandoni, and D. Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. Assoc. Comput. Mach.*, 56(5):1–32, 2009.

[17] M. Fürer, S. Gaspers, and S. P. Kasiviswanathan. An exponential time 2-approximation algorithm for bandwidth. In *Proc. International Workshop on Parameterized and Exact Computation, IWPEC'09*, volume 5917 of *Lecture Notes in Computer Science*, pages 173–184. Springer, 2009.

[18] G. Jager and A. Srivastav. Improved approximation algorithms for maximum graph partitioning problems. In *Proc. Foundations of Software Technology and Theoretical Computer Science, FST&TCS'04*, volume 3328 of *Lecture Notes in Computer Science*, pages 348–359. Springer-Verlag, 2004.

[19] J. Nešetřil and S. Poljak. On the complexity of the subgraph problem. *Comment. Math. Univ. Carolinae*, pages 415–419, 1985.