Exploiting dominance conditions for computing worst-case time upper bounds in bounded combinatorial optimization problems:application to MIN SET COVERING and MAX CUT

Federico Della Croce, Vangelis Th. Paschos

Exploiting dominance conditions for computing worst-case time upper bounds in bounded combinatorial optimization problems: application to MIN SET COVERING and MAX CUT[1]

Federico Della Croce*, Vangelis Th. Paschos†

Abstract

In the design of branch and bound methods for NP-hard combinatorial optimization problems, dominance conditions have always been applied. In this work we show how the use of dominance conditions within search tree algorithms can lead to non trivial worst-case upper time bounds for the considered algorithms on bounded combinatorial optimization problems. We consider here the MIN 3-SET COVERING problem and the MAX CUT problem in graphs maximum degree three, four, five and six. Combining dominance conditions and intuitive combinatorial arguments, we derive two exact algorithms with worst-case complexity bounded above by $p \cdot O(1.4492^n)$ for the former, and $p_1 \cdot O(1.2920^n)$, $p_2 \cdot O(1.4142^n)$, $p_3 \cdot O(1.6430^n)$ and $p_3 \cdot O(1.6430^n)$, respectively, the latter problems, where $p(\cdot)$ and $p_i(\cdot)$, $i = 1, \ldots, 6$, denote some polynomials and n is the number of subsets for MIN 3-SET COVERING and the number of vertices of the input-graph for MAX CUT.

Key words : Worst-case complexity, Exact algorithm, MIN SET COVERING, MAX CUT

[1]Research performed while the first author was in visit at the LAMSADE on a research position funded by the CNRS

*D.A.I., Politecnico di Torino, Italy, federico.dellacroce@polito.it

†LAMSADE, CNRS UMR 7024 and Université Paris-Dauphine, France, paschos@lamsade.dauphine.fr

1 Introduction

The design of exact methods for NP-hard combinatorial optimization problems has always been a challenging issue. Among the existing exact methods, search tree algorithms and in particular branch and bound approaches have been widely applied. A branch and bound algorithm builds and explores a search tree, thus enumerating the solutions space. In order to reduce the computational burden, several techniques can be applied to prune some branches of the search tree and to avoid the enumeration of known non-optimal solutions. A usual technique is to consider dominance conditions while branching from a node. Dominance conditions can be typically derived by comparing two nodes of the search tree, namely two partial solutions of the given problem where the two nodes share some common features.

In this work we study the application of dominance conditions within search tree algorithms in the context of worst-case analysis of exact algorithms. Let $T(\cdot)$ be a super-polynomial and $p(\cdot)$ be a polynomial, both on integers. In what follows, using notations in [13], for an integer n, we express running-time bounds of the form $p(n).T(n)$ as $O^*(T(n))$, the asterisk meaning that we ignore polynomial factors. We denote by $T(n)$ the worst-case time required to exactly solve the considered combinatorial optimization problem with n variables. We recall (see, for instance, [5]) that, if it is possible to bound above $T(n)$ by a recurrence expression of the type $T(n) \leqslant \sum T(n - r_i) + O(p(n))$, we have $\sum T(n-r_i)+O(p(n)) = O^*(\alpha(r_1, r_2, \ldots)^n)$ where $\alpha(r_1, r_2, \ldots)$ is the largest zero of the function $f(x) = 1 - \sum x^{-r_i}$.

A combinatorial problem will be called *bounded*, if some parameter of its instance (e.g., the maximum degree of the input-graph, when dealing with graph-problems, or the maximum set-cardinality, when dealing with problems on set-systems) is bounded above by a fixed constant. We show that, for two bounded combinatorial optimization problems (MIN 3-SET COVERING where each subset has maximum cardinality three and MAX CUT in graphs with maximum degree three, four, five and six, respectively, denoted by MAX CUT-3, MAX CUT-4, MAX CUT-5 and MAX CUT-6, respectively, in what follows), the combination of dominance conditions and intuitive combinatorial arguments within exact search tree algorithms leads to non trivial upper-time bounds for both algorithms. Indeed, for MIN 3-SET COVERING an upper-time bound $O^*(1.4492^n)$ is derived, while, for MAX CUT-3, MAX CUT-4, MAX CUT-5 and MAX CUT-6, upper-time bounds $O^*(1.2920^n)$, $O^*(1.4142^n)$, $O^*(1.6430^n)$ and $O^*(1.6430^n)$, respectively are derived. For the former problem, the derived bound is, to our knowledge, the best known until now. For the latter problems, even if the yielded bounds are not the best known, they are competitive with respect to the known bounds.

2 The MIN 3-SET COVERING problem

2.1 Preliminaries

In MIN SET COVERING, we are given a universe U of elements and a collection $\mathcal{S}$ of (non-empty) subsets of U. The aim is to determine a minimum cardinality sub-collection $\mathcal{S}' \subseteq \mathcal{S}$ which covers U, i.e., $\cup_{S \in \mathcal{S}'} S = U$ (we assume that $\mathcal{S}$ covers U). The frequency f_i of $u_i \in U$ is the number of subsets $S_j \in \mathcal{S}$ in which u_i is contained. The cardinality d_j of $S_j \in \mathcal{S}$ is the number of elements $u_i \in U$ that S_j contains. We say that S_j *hits* S_k if both S_j and S_k contain an element u_i and that S_j *double-hits* S_k if both S_j and S_k contain at least two element u_i, u_l. Finally, we denote by n the size (cardinality) of $\mathcal{S}$ and by m the size of U. In what follows, we restrict ourselves to MIN SET COVERING-instances such that:

1. no element $u_i \in U$ has frequency $f_i = 1$;
2. no set $S_i \in \mathcal{S}$ is subset of another set $S_j \in \mathcal{S}$.
3. no pair of elements u_i, u_j exists such that every subset $S_i \in \mathcal{S}$ containing u_i contains also u_j.

Indeed, if item 1 is not verified, then the set containing u_i belongs to any feasible cover of U. On the other hand, if item 2 is not verified, then S_i can be replaced by S_j in any solution containing S_i and the resulting cover will not be worse than the one containing S_i. Finally, if item 3 is not verified, then element u_j can be ignored as every sub-collection $\mathcal{S}'$ covering u_i will necessarily cover also u_j. So, for any instance of MIN SET COVERING, a preprocessing of data, obviously performed in polynomial time, leads to instances where all items 1, 2 and 3 are verified.

There exist to our knowledge few results on worst-case complexity of exact algorithms for MIN SET COVERING or for cardinality-constrained versions of it. Let us note that an exhaustive algorithm computes any solution for MIN SET COVERING in $O(2^n)$. For MIN SET COVERING the most recent non-trivial result is the one of [7] (that has improved the result of [10]) deriving a bound (requiring exponential space) of $O^*(1.2301^{(m+n)})$. We consider here, the most notorious cardinality-constrained version of MIN SET COVERING, the MIN 3-SET COVERING, namely, MIN SET COVERING where $d_j \leqslant 3$ for all $S_j \in \mathcal{S}$. It is well known that MIN 3-SET COVERING is **NP**-hard, while MIN 2-SET COVERING (where any set has cardinality at most 2) is polynomially solvable by matching techniques ([2, 8]). Our purpose is to devise an exact (optimal) algorithm with provably improved worst-case complexity for MIN 3-SET COVERING. In what follows, we propose a search tree-based algorithm with running time $O^*(1.4492^n)$ which constitutes, to the best of our knowledge,

the best bound for that problem. (notice, for instance, that the bound of [7] for $f_i = 2$, $u_i \in U$, and $d_j = 3$, for any $S_j \in S$ corresponds to $O^*(1.2301^{(5/2)}) \approx O^*(1.6782^n)$).

Consider, the following algorithm, denoted by `SOLVE-3-SET-COVERING`:

- repeat until possible:
  1. for any unassigned subset S_j test if the preprocessing induced by items 1, 2 and 3 reduces the size of the instance;
  2. select for branching the unassigned subset whose branching induces the minimum worst-case complexity (in case of tie select the subset with smallest index).

2.2 Dominance conditions

The following straightforward lemma holds, inducing some useful domination conditions for the solutions of MIN SET COVERING.

Lemma 1. *There exists at least one optimal solution of* MIN SET COVERING *where*

1. *for any subset S_j with $d_j = 2$ containing elements u_i, u_p, if S_j double-hits S_k, then S_j is excluded from $\mathcal{S}'$ (in case also $d_k = 2$, then it is immaterial to exclude either S_j or S_k);*
2. *for any subset S_j with $d_j = 2$ containing elements u_i, u_p, if S_j is included in $\mathcal{S}'$, then all subsets S_k hitting S_j are excluded from $\mathcal{S}'$;*
3. *for any subset S_j with $d_j = 3$ containing elements u_i, u_p, u_q, where S_j double-hits another subset S_k with $d_k = 3$ on u_i and u_p, if S_j is included in $\mathcal{S}'$ then S_k must be excluded from $\mathcal{S}'$ and viceversa;*
4. *for any subset S_j with $d_j = 3$ containing elements u_i, u_p, u_q, if S_j is included in $\mathcal{S}'$, then either all subsets S_k hitting S_j on element u_i are excluded from $\mathcal{S}'$, or all subsets S_k hitting S_j on elements u_p and u_q are excluded from $\mathcal{S}'$.*

Proof.

1. Notice that the configuration implied by item 1 cannot occur thanks to the first hypothesis (item 1 in Section 2.1) on the form of the MIN SET COVERING-instances dealt.

2. Assume, without loss of generality, that S_j hits S_k on u_i and S_l on u_p. Suppose by contradiction that the optimal solution S' includes S_j and S_k. Then, it cannot include also S_l or else it would not be optimal as a better cover would be obtained by excluding S_j from S'. On the other hand, suppose that S' includes S_j, S_k but does not include S_l. Then, an equivalent optimal solution can be derived by swapping S_j with S_l.

For items 3 and 4, the same kind of analysis as for item 2 holds. ∎

2.3 The worst-case upper-time bound for MIN 3-SET COVERING

The objective of this section is to show the following result.

Proposition 1. *Algorithm* `SOLVE-3-SET-COVERING` *optimally solves* MIN 3-SET COVERING *within time* $O^*(1.4492^n)$.

Proof. The algorithm either detects by means of item 1 a subset S_j to be immediately included in (excluded from) $\mathcal{S}'$ or an element u_i to be ignored (correspondingly reducing the degree of several subsets), or applies a branching on subset S_j, where the following exhaustive relevant branching cases may occur.

1. $d_j = 2$: then no double-hitting occurs to S_j or else, due to Lemma 1, S_j can be excluded from s' without branching. The following subcases occur.

   (a) S_j contains elements u_i, u_k with $f_i = f_k = 2$ where S_j hits S_l on u_i and S_m on u_k. Due to Lemma 1, if S_j is included in $\mathcal{S}'$, then both S_l and S_m must be excluded from $\mathcal{S}'$; alternatively, S_j is excluded from $\mathcal{S}'$ and, correspondingly, both S_l and S_m must be included in $\mathcal{S}'$ to cover elements u_i, u_k. This can be seen as a binary branching where, in both cases, three subsets (S_j, S_l, S_m) are fixed. Then, $T(n) \leqslant 2T(n-3)+O(p(n))$, where the term $T(n-3)$ measures the time for solving the same problem with $n-3$ subsets. Correspondingly, we have $T(n) = O^*(\alpha^n)$, where α is the largest real root of the equation $\alpha^3 = 2$, i.e., $\alpha \approx 1.2599$, implying a time complexity of $O^*(1.2599^n)$.

   (b) S_j contains elements u_i, u_k with $f_i = 2$ and $f_k \geqslant 3$, where S_j hits S_l on u_i and S_m, S_p on u_k. Due to Lemma 1, if S_j is included in $\mathcal{S}'$, then S_l, S_m, S_p must be excluded from $\mathcal{S}'$; alternatively, S_j is excluded from $\mathcal{S}'$ and, correspondingly, S_l must be included in $\mathcal{S}'$ to cover element u_i. This can be seen as a binary branching where either 2 subsets (S_j, S_l), or 4 subsets (S_j, S_l, S_m, S_p) are fixed; hence, $T(n) \leqslant T(n-2) + T(n-4) + O(p(n))$. This results in a time-complexity of $O^*(1.2721^n)$.

(c) S_j contains elements u_i, u_k with $f_i \geqslant 3$ and $f_k \geqslant 3$ where S_j hits S_l, S_m on u_i and S_p, S_q on u_k. Due to Lemma 1, if S_j is included in $\mathcal{S}'$, then S_l, S_m, S_p, S_q must be excluded from $\mathcal{S}'$; alternatively, S_j is excluded from $\mathcal{S}'$. This can be seen as a binary branching where either 1 subset (S_j) is fixed, or 5 subsets (S_j, S_l, S_m, S_p, S_q) are fixed and, hence, $T(n) \leqslant T(n-1) + T(n-5) + O(p(n))$. This results in a time-complexity of $O^*(1.3248^n)$.

2. $d_j = 3$ (that is, there does not exist $S_k \in S$ such that $d_k = 2$) with S_j double-hitting one or more subsets. Notice that if S_j double-hits S_k on elements u_i, u_l, then $f_i \geqslant 3$ and $f_l \geqslant 3$ due to the preprocessing step 1 of the algorithm. The following exhaustive subcases may occur.

   (a) S_j double-hits at least 3 subsets S_k, S_l, S_m. Due to Lemma 1, if S_j is included in $\mathcal{S}'$ then S_k, S_l, S_m must be excluded from $\mathcal{S}'$; alternatively, S_j is excluded from $\mathcal{S}'$. This can be seen as a binary branching where either 1 subset (S_j) is fixed, or 4 subsets (S_j, S_k, S_l, S_m) are fixed and hence, $T(n) \leqslant T(n-1) + T(n-4) + O(p(n))$. This results in a time-complexity of $O^*(1.3803^n)$.

   (b) S_j double-hits 2 subsets S_k, S_l and hits at least one more subset S_m (we assume that S_j hits S_m on element u_i). Due to Lemma 1, if S_j is included in $\mathcal{S}'$, then S_k, S_l must be excluded from $\mathcal{S}'$ and a further branching on subset S_m with $d_m = 2$ can be applied (as u_i is already covered by S_j) where, in the worst case, subcase 1c holds; alternatively, S_j is excluded from $\mathcal{S}'$; this can be seen as a binary branching where either 1 subset (S_j) is fixed, or 3 subsets (S_j, S_k, S_l) are fixed and a branching of type 1c on subset S_m with $n' = n-3$ variables holds. Then $T(n) \leqslant T(n-1) + T(n'-1) + T(n'-5) + O(p(n)) = T(n-1) + T(n-4) + T(n-8) + O(p(n))$. This results in a time-complexity of $O^*(1.4271^n)$.

   (c) S_j contains elements u_i, u_k, u_l and double-hits one subset S_k on elements u_i, u_k. The following exhaustive subcases must be considered.

      i. $f_i \geqslant 3$, $f_k \geqslant 3$, $f_l = 2$ with u_i contained at least by S_j, S_k, S_m, u_k contained at least by S_j, S_k, S_p and u_l contained by S_j, S_q. A composite branching can be devised:

         - either S_j and S_q are included in $\mathcal{S}'$ and, due to Lemma 1, S_k, S_m, S_p must be excluded from $\mathcal{S}'$,
         - or S_j is included in $\mathcal{S}'$ and S_q is excluded from $\mathcal{S}'$ and, correspondingly, S_k must be excluded from $\mathcal{S}'$,
         - or S_j is excluded from $\mathcal{S}'$ and, correspondingly, S_q must be included in $\mathcal{S}'$.

         Then, $T(n) \leqslant T(n-2) + T(n-3) + T(n-5) + O(p(n))$. This results in a time-complexity of $O^*(1.4292^n)$.

ii. $f_i = 3$, $f_j = 3$, $f_l \geqslant 3$ with u_i contained by S_j, S_k, S_m, u_k contained by S_j, S_k, S_p and u_l contained at least by S_j, S_q, S_r. A composite branching can be devised:
   - either S_j and S_k are excluded from $\mathcal{S}'$ and (to cover u_i and u_k) S_m, S_p must be included in $\mathcal{S}'$,
   - or S_j is included in $\mathcal{S}'$ and (due to Lemma 1) either S_k, S_q, S_r are excluded from $\mathcal{S}'$ or S_k, S_m, S_p are excluded from $\mathcal{S}'$,
   - or S_k is included in $\mathcal{S}'$ and (due to Lemma 1) either S_j, S_m, S_p are excluded from $\mathcal{S}'$ or S_j, S_λ, S_μ are excluded from $\mathcal{S}'$, where S_λ, S_μ are the subsets hitting S_k on another element (recall $f_k = 3$) u_v.

   This would induce $T(n) \leqslant 5T(n' - 4) + O(p(n))$, but in all subcases a consequent branching on an unassigned subset (any of those hitting a subset just included in $\mathcal{S}'$) having (therefore) cardinality 2 holds where, in the worst case, subcase 1c holds. Then, $T(n) \leqslant 5T(n-5) + 5T(n-9) + O(p(n))$. This results in a time-complexity of $O^*(1.4389^n)$.

iii. $f_i = 3$, $f_j \geqslant 4$, $f_l \geqslant 3$, with u_i contained by S_j, S_k, S_m, u_k contained at least by S_j, S_k, S_p, S_q and u_l contained at least by S_j, S_r, S_u. A composite branching can be devised:
   - either S_j and S_k are excluded from $\mathcal{S}'$ and (to cover u_i) S_m must be included in $\mathcal{S}'$,
   - or S_j is included in $\mathcal{S}'$ and (due to Lemma 1) either S_k, S_p, S_q are excluded from $\mathcal{S}'$ or S_k, S_m, S_r, S_u are excluded from $\mathcal{S}'$,
   - or S_k is included in $\mathcal{S}'$ and (due to Lemma 1) either S_j, S_p, S_q are excluded from $\mathcal{S}'$ or $S_j, S_m, S_\lambda, S_\mu$ are excluded from $\mathcal{S}'$, where S_λ, S_μ are the subsets hitting S_k on another element (recall $f_k = 3$) u_v.

   This would induce $T(n) \leqslant T(n-3)+2T(n-4)+2T(n-5)+O(p(n))$, but in all subcases a consequent branching on an unassigned subset (any of those hitting a subset just included in $\mathcal{S}'$) having (therefore) cardinality 2 holds where, in the worst case, subcase 1c holds. Then, $T(n) \leqslant T(n-4) + 2T(n-5) + 2T(n-6) + T(n-8) + 2T(n-9) + 2T(n-10) + O(p(n))$. This results in a time-complexity of $O^*(1.4331^n)$.

iv. $f_i \geqslant 4$, $f_j \geqslant 4$, $f_l \geqslant 3$, with u_i contained at least by S_j, S_k, S_m, S_p, u_k contained at least by S_j, S_k, S_q, S_r and u_l contained at least by S_j, S_u, S_v. A composite branching on subset S_j can be devised (due to Lemma 1):
   - S_j is included in $\mathcal{S}'$, S_k, S_m, S_p are excluded from $\mathcal{S}'$ and a further branching on subset S_q can be applied with $d_q = 2$ (as u_k is already covered by S_j),
   - or S_j is included in $\mathcal{S}'$, S_k, S_q, S_r, S_q, S_v are excluded from $\mathcal{S}'$ and a further branching on subset S_m can be applied with $d_m = 2$ (as u_i is already covered by S_j),

- or S_j is excluded from $\mathcal{S}'$.

This can be seen as a composite branch where either 1 or 4 or 6 subsets have been included in or excluded from S' that is $T(n) \leqslant T(n-1) + T(n-4)+T(n-6)+O(p(n))$, where however, in the latter two branches a consequent branching on an unassigned subset having cardinality 2 holds where, in the worst case, subcase 1c holds. Then, $T(n) \leqslant T(n-1) + T(n-5) + T(n-7) + T(n-9) + T(n-11 + O(p(n))$. This results in a time-complexity of $O^*(1.4343^n)$.

3. $d_j = 3$ and no double-hitting occurs to S_j (nor to any other subset) that contains elements u_i, u_k, u_l. The following subcases occur.

   (a) $f_i = f_k = f_l = 2$ with u_i contained by S_j, S_k, u_k contained by S_j, S_l and u_l contained by S_j, S_m. A binary branching on S_j can be devised: either S_j is excluded from $\mathcal{S}'$ and then (to cover u_i, u_k, u_l) S_k, S_l, S_m must be included in $\mathcal{S}'$, or S_j is included in $\mathcal{S}'$. This would induce $T(n) \leqslant T(n-1) + T(n-4) + O(p(n))$, but in all subcases, a consequent branching on an unassigned subset (any of those hitting a subset just included in $\mathcal{S}'$) having (therefore) cardinality 2 holds where, in the worst case, subcase 1c holds. Then, $T(n) \leqslant T(n-2) + T(n-5) + T(n-6) + T(n-9) + O(p(n))$. This results in a time-complexity of $O^*(1.3515^n)$.

   (b) $f_i = f_k = 2$, $f_l \geqslant 3$ with u_i contained by S_j, S_k, u_k contained by S_j, S_l and u_l contained at least by S_j, S_m, S_p. A composite branching on S_j can be devised:

   - either S_j is excluded from $\mathcal{S}'$ and then (to cover u_i, u_k) S_k, S_l must be included in $\mathcal{S}'$,
   - or S_j is included in $\mathcal{S}'$ and S_k, S_l are excluded from $\mathcal{S}'$,
   - or S_j is included in $\mathcal{S}'$ and S_m, S_p are excluded from $\mathcal{S}'$.

   This would induce $T(n) \leqslant 3T(n-3) + O(p(n))$, but in all subcases, a consequent branching on an unassigned subset (any of those hitting a subset just included in $\mathcal{S}'$) having (therefore) cardinality 2 holds where, in the worst case, subcase 1c holds. Then, $T(n) \leqslant 3T(n-4) + 3T(n-8) + O(p(n))$. This results in a time-complexity of $O^*(1.3954^n)$.

   (c) $f_i = 2$, $f_k \geqslant 3$, $f_l \geqslant 3$, with u_i contained by S_j, S_k, u_k contained by S_j, S_l, S_m, and u_l contained at least by S_j, S_p, S_q. A composite branching on S_j can be devised: either S_j is excluded from $\mathcal{S}'$ and then (to cover u_i) S_k must be included in $\mathcal{S}'$, or S_j is included in $\mathcal{S}'$ and S_k, S_l, S_m are excluded from $\mathcal{S}'$, S_j is included in $\mathcal{S}'$ and S_p, S_q are excluded from $\mathcal{S}'$. This would induce $T(n) \leqslant T(n-2) + T(n-3) + T(n-4) + O(p(n))$, but in all subcases, a consequent branching on an unassigned subset (any of those hitting a subset just included in $\mathcal{S}'$) having (therefore) cardinality 2 holds where, in the worst case,

subcase 1c holds. Then, $T(n) \leqslant T(n-3) + T(n-4) + T(n-5) + T(n-7) + T(n-8) + T(n-9) + O(p(n))$. This results in a time-complexity of $O^*(1.4066^n)$.

(d) $f_i = 3$, $f_k \geqslant 3$, $f_l \geqslant 3$ with u_i contained by S_j, S_k, S_l, u_k contained by S_j, S_m, S_p and u_l contained at least by S_j, S_q, S_r. Also, both S_k and S_l have degree 3 and all elements contained by S_k or S_l have frequency at least 3 or else subcase 3c would hold either on subset S_k or on subset S_l. A composite branching can be devised:

- either S_j is included in $\mathcal{S}'$ and then either S_k, S_l are excluded from $\mathcal{S}'$, or S_m, S_p, S_q and S_r are excluded from $\mathcal{S}'$,
- or S_j is excluded from $\mathcal{S}'$, S_k is included in $\mathcal{S}'$ and there are at least 5 other subsets hitting S_k and, hence, either two of these subsets are excluded from $\mathcal{S}'$ or three of these subsets are excluded from $\mathcal{S}'$,
- or S_j, S_k are excluded from $\mathcal{S}'$, S_l is included in $\mathcal{S}'$ (to cover u_i) and there are at least 4 other subsets hitting S_k and, hence, either two of these subsets are excluded from $\mathcal{S}'$, or the other two of these subsets are excluded from $\mathcal{S}'$.

This would induce $T(n) \leqslant T(n-3) + T(n-4) + 4T(n-5) + O(p(n))$, but in all subcases, a consequent branching on an unassigned subset (any of those hitting a subset just included in $\mathcal{S}'$) having (therefore) cardinality 2 holds where, in the worst case, subcase 1c holds. Then, $T(n) \leqslant T(n-4) + T(n-5) + 4T(n-6) + T(n-8) + T(n-9) + 4T(n-10) + O(p(n))$. This results in a time-complexity of $O^*(1.4492^n)$.

(e) $f_i \geqslant 4$, $f_k \geqslant 4$, $f_l \geqslant 4$, u_i is contained by S_j, S_k, S_l, S_m, u_k is contained by S_j, S_p, S_q, S_r and u_l is contained at least by S_j, S_t, S_u, S_v. A composite branching on S_j can be devised:

- either S_j is excluded from $\mathcal{S}'$,
- or S_j is included in $\mathcal{S}'$, S_k, S_l, S_m are excluded from $\mathcal{S}'$ and a further branching on subset S_p can be applied with $d_p = 2$ (as u_k is already covered by S_j),
- or S_j is included in $\mathcal{S}'$, $S_p, S_q, S_r, S_t, S_u, S_w$ are excluded from $\mathcal{S}'$ and a further branching on subset S_m can be applied with $d_m = 2$ (as u_i is already covered by S_j).

This can be seen as a composite branch where either 1 or 4 or 7 subsets have been included in or excluded from S' that is $T(n) \leqslant T(n-1) + T(n-4) + T(n-7) + O(p(n))$, where however, in the latter two branches a consequent branching on an unassigned subset having cardinality 2 holds where, in the worst case, subcase 1c holds. Then, $T(n) \leqslant T(n-1) + T(n-5) + T(n-$

$8) + T(n-9) + T(n-12) + O(p(n))$. This results in a time-complexity of $O^*(1.4176^n)$.

In all, the overall worst-case complexity for MIN 3-SET COVERING is $O^*(1.4492^n)$. ∎

3 The MAX CUT problem

3.1 Preliminaries

In MAX CUT, we are given a graph $G(V,E)$ with $|V| = n$ vertices $v_1, \ldots, v_n$ and $|E| = m$ edges. The goal is to find a partition of V into two subsets V_1 and V_2 that maximizes the number of edges between V_1 and V_2. Here, we consider the restricted case where all vertices have maximum degree $d_{\max} \leqslant 6$.

We denote by d_j the degree of vertex v_j and by $N(v_j)$ the set of vertices (neighborhood) adjacent to v_j (in other words, $d_j = |N(v_j)|$).

The best known and most recent upper time-bound for MAX CUT-3 with bounded degree is, to our knowledge, the one of [12] (see also [11]) where, a worst-case complexity of $O^*(2^{\min((m-n)/2, m/5)})$ is obtained. This bound dominates the recent bounds $O^*(2^{m/4})$ by [6] and $O^*(2^{m/3})$ by [9]. In what follows, we propose a search tree-based algorithm with worst-case time-complexity which is interestingly competitive with respect to the state of the art ([**?**, 11, 6, 9]).

3.2 Dominance conditions

We assume, without loss of generality, that vertex v_1 is assigned to V_1. Also, with respect to the worst-case analysis, we assume without loss of generality, that the input-graph G is connected. The following straightforward lemma holds.

Lemma 2. *There exists at least one optimal solution of* MAX CUT *where, for any vertex $v_j \neq v_1$ assigned to V_1 (resp., V_2), at least $(d_j+1)/2$ vertices $v_i \in N(v_j)$, for d_j odd, and at least $d_j/2$ vertices $v_i \in N(v_j)$, for d_j even, are assigned to V_2 (resp., V_1).*

Proof. For d_j odd, if less than $(d_j+1)/2$ vertices are assigned to V_2 (resp., V_1), then moving j to V_2 (resp., V_1), would improve solution. On the other hand, for d_j even, if less than $d_j/2$ vertices are assigned to V_2 (resp., V_1), then moving j to V_2 (resp., V_1), would again improve (or at least not worsen) the solution. ∎

Remark 1. For d_j even and the vertices $v_i \in N(v_j)$ equally distributed between V_1 and V_2, it is immaterial to assign v_j to V_1 or V_2. ∎

Consider solving MAX CUT by means of a search tree approach. Suppose that at some point a branching is considered related to a vertex v_j and that all (nearly all) of its adjacent vertices v_i ($v_i \in N(v_j)$) have already been assigned. We denote by $S_j(V_k)$ the set of vertices adjacent to vertex v_j and assigned to set V_k, $j = 1, \ldots, n$, $k = 1, 2$; namely, $S_j(V_k) = \{i : v_i \in N(v_j) \cap V_k\}$. Then, using Lemma 2, the following lemma holds.

Lemma 3. *Consider any vertex v_j such that:*

1. *all $v_i \in N(v_j)$ have already been assigned and d_j is odd;*
2. *all $v_i \in N(v_j)$ have already been assigned and d_j is even;*
3. *all but one $v_i \in N(v_j)$ have already been assigned and d_j is even.*

For all of the cases above, there exists at least one optimal solution with the assignment of v_j uniquely determined as follows:

1. *(item 1 holds) if $|S_j(V_1)| \geqslant (d_j + 1)/2$, then v_j is assigned to V_2, else it is assigned to V_1;*
2. *(item 2 holds)*
   - *(a) if $|S_j(V_1)| > d_j/2$, then v_j is assigned to V_2,*
   - *(b) if $|S_j(V_1)| < d_j/2$, then v_j is assigned to V_1,*
   - *(c) if $|S_j(V_1)| = |S_j(V_2)| = d_j/2$, it is immaterial to assign v_j to V_1 or V_2;*
3. *(item 3 holds) if $|S_j(V_1)| \geqslant d_j/2$, then v_j is assigned to V_2, else it is assigned to V_1.*

Proof. Due to Lemma 2 and Remark 1, the proof of items 1 and 2 above, is immediate. For item 3, let $v_k \in N(v_j)$ be the unassigned vertex. If $|S_j(V_1)| \geqslant d_j/2$, whatever the assignment of v_k, either the condition of item 2a, or the condition of item 2c hold and hence there exists at least an optimal solution with v_j assigned to V_2. Analogously, if $|S_j(V_1)| \leqslant d_j/2 - 1$, whatever the assignment of v_k, either the condition of case 2b, or the condition of case 2c hold and hence there exists at least an optimal solution with v_j assigned to V_1. ∎

Remark 2. Since the graph is connected, it is always possible to devise a tree-search algorithm in which we always branch on a vertex v_j that is adjacent to at least another vertex v_i which has already been assigned to one of the sets of the partition (V_1, V_2). ∎

Consider an optimal search tree algorithm for MAX CUT, denoted by `SOLVE-MAX-CUT` where, at any node of the tree, a decision is taken on the assignment of a vertex either to V_1 or to V_2. Algorithm `SOLVE-MAX-CUT` works as follows:

- select arbitrarily vertex v_1 and assign it to V_1;
- apply a search tree algorithm to assign the remaining vertices according to the following rule: select for branching the unassigned vertex whose branching induces the minimum worst-case complexity and, in case of tie, the unassigned vertex with minimum degree (in case of further tie select the vertex with smallest index).

When we consider for branching in SOLVE-MAX-CUT a given vertex v_j and $d_{\max} = 3$, either $d_j \leqslant 2$ and, due to Lemma 3 and Remark 2, v_j can be assigned without branching, or $d_j = 3$. The following dominance condition holds.

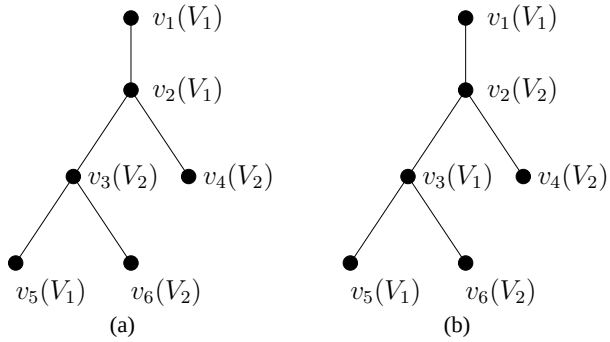

Figure 1: Comparing two configurations with six vertices

Lemma 4. *Consider six vertices $v_1, \ldots, v_6$ connected as in figure 1(a). Then, swapping the assignment of vertices v_2 and v_3 leads to an equivalent solution, i.e., configuration 1(a) can be substituted by configuration 1(b).*

Proof. It is immediate to see that for both configurations exactly three edges belong to the cut. ∎

3.3 The MAX CUT-3 problem

We now prove the following result dealing with MAX CUT-3.

Proposition 2. *Algorithm SOLVE-MAX-CUT optimally solves* MAX CUT-*3 with worst-case time-complexity $O^*(1.2920^n)$.*

Proof. The relevant branching cases for a given vertex v_j in the application of SOLVE-MAX-CUT are those with $d_j = 3$, or else no branching occur. Let v_i, v_k and v_l be the corresponding adjacent vertices. Recall that, from Remark 2, at least one vertex v_i has already been assigned. Also, from Lemma 3, no branching occurs if all adjacent vertices v_i, v_k and v_l have already been assigned. Finally, from Lemma 2, no branching occurs if two of the adjacent vertices have already been assigned to the same set of the partition (V_1, V_2). Then, the following relevant cases may occur.

1. $d_j = 3$, two vertices v_i, v_k adjacent to v_j have already been assigned to different sets of the partition (V_1, V_2), while the third adjacent vertex v_l has not yet been assigned. Then, the following exhaustive subcases may hold.

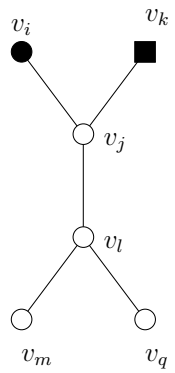

Figure 2: Case 1a of Proposition 2.

   (a) $d_l = 3$, v_l is adjacent to vertex v_j and to other two vertices v_m, v_q both unassigned (see Figure 2, where black circles represent vertices assigned to V_1, black rectangles represent vertices assigned to V_2 and white circles represent yet unassigned vertices). Due to Lemma 2, if at least one of v_m, v_q is assigned to V_1 (resp., V_2), we can arbitrarily assign v_l to V_2 (resp., V_1) and v_j to V_1 (resp., V_2). Alternatively, both vertices v_m, v_q are assigned to V_2 (resp., V_1) and hence we can assign v_l to V_1 (resp., V_2) and v_j to V_2 (resp., V_1). This can be seen as a binary branching where either 2 vertices (v_j, v_l), or 4 vertices (v_j, v_l, v_m, v_q) are assigned. Then, $T(n) \leqslant T(n-2)+T(n-4)+O(p(n))$, where the terms $T(n-2)$ and $T(n-4)$ measure the time for solving the same case with $n-2$ and $n-4$ unassigned vertices, respectively. Correspondingly, we have $T(n) = O^*(1.2721^n)$.

   (b) $d_l = 3$, v_l is adjacent to vertex v_j and to other two vertices v_m, v_q, where at least one of them (say v_m, where v_m may possibly coincide with v_i or v_k) has

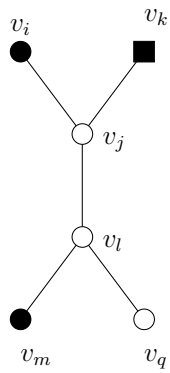

Figure 3: Case 1b of Proposition 2.

already been assigned (Figure 3). Then, whatever will be the assignment of v_q, if v_m is assigned to V_2 (resp., V_1), v_l can be assigned to V_1 (resp., V_2) and v_j to V_2 (resp., V_1) without branching.

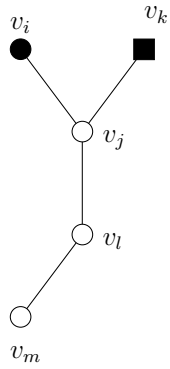

Figure 4: Case 1c of Proposition 2.

(c) $d_l = 2$ (Figure 4). Then, v_l is adjacent to vertex v_j and to another vertex v_m which has not yet been assigned (or else v_l would have already been assigned earlier without branching). If we assign v_m to V_2 (resp., V_1), then v_l must be assigned to V_1 (resp., V_2) and v_j to V_2 (resp., V_1). In other words, we can branching on vertex v_m and, for both branches, correspondingly, fix the assignment of vertices v_j, v_l, v_m all together. This can be seen as a binary branching where, in both cases, three vertices (v_j, v_l, v_m) are assigned and, hence, $T(n) \leqslant 2T(n-3) + O(p(n))$, i.e., $T(n) = O^*(1.2599^n)$.

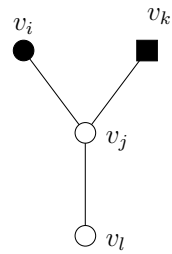

Figure 5: Case 1d of Proposition 2.

(d) $d_l = 1$ (Figure 5). Then, v_j can be assigned to V_1 and v_l to V_2 without branching (actually, it is immaterial to assign v_j to V_1 or V_2, provided that v_l is assigned to the opposite set of the partition).

2. $d_j = 3$, a vertex v_i adjacent to v_j has already been assigned to V_1 (resp., V_2), while the other two adjacent vertices v_k and v_l have not yet been assigned. We assume, without loss of generality, that $d_k \leqslant d_l$. The following subcases must be considered.

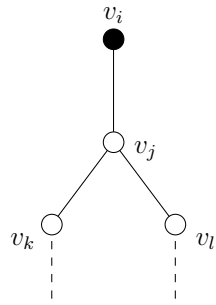

Figure 6: Case 2a of Proposition 2.

(a) $1 \leqslant d_k \leqslant d_l \leqslant 2$ (Figure 6). A branching on vertex v_j can be applied. Either v_j is assigned to V_1 (resp., V_2) and correspondingly vertices v_k and v_l must be assigned, due to Lemma 2, to V_2 (resp., V_1), or v_j is assigned to V_2 (resp., V_1) and, correspondingly, vertices v_k and v_l must be assigned, due to Lemma 2, to V_1 (resp., V_2). This can be seen as a binary branching where, in both cases, 3 vertices (v_j, v_k, v_l) are fixed and the same time-complexity $O^*(1.2599^n)$ of case 1c holds.

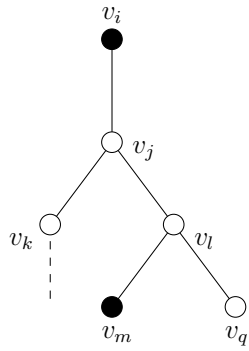

Figure 7: Case 2b of Proposition 2.

(b) $1 \leqslant d_k \leqslant 2$, $d_l = 3$, v_k and v_l are not adjacent and one of the vertices (v_m, v_q) adjacent to vertex v_l has already been assigned while the other has not yet been assigned (Figure 7). We assume without loss of generality that v_m has already been assigned. Notice that, if v_m has been assigned to V_2 (resp., V_1), no branching occurs, due to Lemma 4. We assume then that v_m has already been assigned to V_1 (resp., V_2). A branching on vertex v_j can be applied. If v_j is assigned to V_1 (resp., V_2), vertices v_k, and v_l must be assigned (due to Lemma 2) to V_2 (resp., V_1) while vertex v_q must be assigned (due to Lemma 4) to V_1 (resp., V_2). Else, v_j is assigned to V_2 (resp., V_1) and, correspondingly, v_k can be assigned to V_1. This can be seen as a branching where, either 2 vertices (v_j, v_k), or 4 vertices (v_j, v_k, v_l, v_q) are assigned and the same time-complexity $O^*(1.2721^n)$ of case 1a holds.

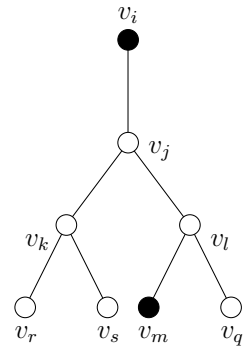

Figure 8: Case 2c of Proposition 2.

(c) $d_k = d_l = 3$, v_k and v_l are not adjacent and one of the vertices (v_m, v_q) adjacent to vertex v_l has already been assigned, while the other has not yet been assigned (Figure 8). Also, the vertices (v_r, v_s) adjacent to vertex v_k have not yet been assigned. We assume without loss of generality that v_m has already been assigned. Notice that, if v_m has been assigned to V_2 (resp., V_1), no branching occurs due to Lemma 4. We assume, then, that v_m has already been assigned to V_1 (resp., V_2). A branching on vertex v_j can be applied. If v_j is assigned to V_1 (resp., V_2), vertices v_k, v_l must be assigned (due to Lemma 2) to V_2 (resp., V_1) and vertices v_q, v_r, v_s must be assigned (due to Lemma 4) to V_2 (resp., V_1). Else, v_j is assigned to V_2 (resp., V_1). This can be seen as a branching where, either 1 vertex (v_j) is assigned, or 6 vertices ($v_j, v_k, v_l, v_q, v_r, v_s$) are assigned. Then, $T(n) \leqslant T(n-1) + T(n-6) + O(p(n))$, i.e., $T(n) = O^*(1.2852^n)$.

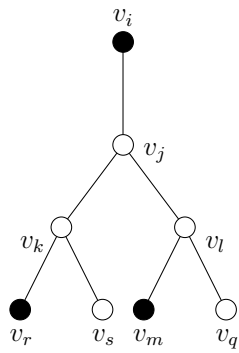

Figure 9: Case 2d of Proposition 2.

(d) $d_k = d_l = 3$, v_k and v_l are not adjacent and one of the vertices (v_m, v_q) adjacent to vertex v_l has already been assigned while the other has not yet been assigned (Figure 9). We assume without loss of generality that v_m has already been assigned. Notice that, if v_m has been assigned to V_2 (resp., V_1), no branching occurs, due to Lemma 4. We assume, then, that v_m has already been assigned to V_1 (resp., V_2). Also, v_k is adjacent (apart from v_j) to two vertices v_r and v_s where vertex v_r has already been assigned while v_s has not yet been assigned. Notice that, if v_r has been assigned to V_2 (resp., V_1), no branching occurs, due to Lemma 4. We assume, then, that v_r has already been assigned to V_1 (resp., V_2). Finally, notice that v_m and v_r may well coincide. A composite branching can be applied. If v_j is assigned to V_1 (resp., V_2), then v_k, v_l must be assigned (due to Lemma 2) to V_2 (resp., V_1) and v_q, v_s must be assigned (due to Lemma 4) to V_1 (resp., V_2). Else, v_j is assigned

to V_2 (resp., V_1): then, either v_q is assigned to V_1 (resp., V_2) and, correspondingly, v_l is assigned to V_2 (resp., V_1), v_k to V_1 (resp., V_2) and v_s to V_2 (resp., V_1); or v_q is assigned to V_2 (resp., V_1) and v_l is assigned to V_1 (resp., V_2): but then for this latter case a branching on vertex k can be applied where two of its adjacent vertices (v_j, v_r) have already been assigned and, in the worst case, subcase 1a holds. Putting things together, this can be seen as a composite branching where, either 5 vertices $(v_j, v_k, v_l, v_q, v_s)$ are assigned, or 5 vertices $(v_j, v_k, v_l, v_s, v_q)$ are assigned, or three vertices (v_j, v_l, v_q) are assigned and a branching of type 1a on vertex v_k with $n' = n - 3$ variables holds. Then, $T(n) \leqslant 2T(n-5)+2T(n'-3)+O(p(n)) = 3T(n-5)+T(n-7)+O(p(n))$. Correspondingly, we have $T(n) = O^*(\alpha^n)$, i.e., $\alpha \approx 1.2886$, implying a time complexity of $O^*(1.2920^n)$.

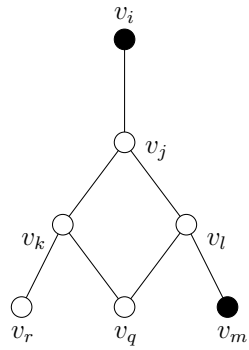

Figure 10: Case 2e of Proposition 2.

(e) $d_k = d_l = 3$, v_k and v_l are not adjacent and one of the vertices (v_m, v_q) adjacent to vertex v_l has already been assigned while the other has not yet been assigned (Figure 10). We assume, without loss of generality, that v_m has already been assigned. Notice that, if v_m has been assigned to V_2 (resp., V_1), no branching occurs, due to Lemma 4. We assume then that v_m has already been assigned to V_1 (resp., V_2). Also, v_k is adjacent (apart from v_j) to v_q and to another unassigned vertex v_r. A composite branch, first on vertex v_j and then on vertex v_q can be applied. If v_j is assigned to V_1 (resp., V_2), vertices v_k, and v_l must be assigned (due to Lemma 2) to V_2 (resp., V_1) and vertices v_q, v_r must be assigned (due to Lemma 4) to V_1 (resp., V_2). Else, v_j is assigned to V_2 (resp., V_1) and, if v_q is assigned to V_2 (resp., V_1), then v_k and v_l must be assigned to V_1 (resp., V_2), else v_q is assigned to V_1 (resp., V_2) and, consequently, v_l must be assigned to V_2 (resp., V_1), v_k to V_1 (resp., V_2) and v_r to V_2 (resp., V_1). This can be seen as a branching with three children nodes

where, either 5 vertices (v_j, v_k, v_l, v_q, v_r), or 4 vertices (v_j, v_k, v_l, v_q), or 5 vertices (v_j, v_k, v_l, v_q, v_r) are assigned. Then, $T(n) \leqslant T(n-4) + 2T(n-5) + O(p(n))$, i.e., $T(n) = O^*(1.2672^n)$.

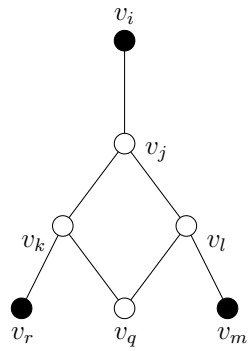

Figure 11: Case 2f of Proposition 2.

(f) $d_k = d_l = 3$, v_k and v_l are not adjacent and one of the vertices (v_m, v_q) adjacent to vertex v_l has already been assigned while the other has not yet been assigned (Figure 11). We assume without loss of generality that v_m has already been assigned. Notice that, if v_m has been assigned to V_2 (resp., V_1), no branching occurs, due to Lemma 4. We assume, then, that v_m has already been assigned to V_1 (resp., V_2). Also, v_k is adjacent (apart from v_j) to v_q and to another vertex v_r that has already been assigned. Notice that, if v_r has been assigned to V_2 (resp., V_1), no branching occurs, due to Lemma 4. We assume, then, that v_r has already been assigned to V_1 (resp., V_2). Finally, notice that v_m and v_r may well coincide. We observe that v_j, v_q must be assigned to the same set of the partition. Indeed, if v_j is assigned to V_1 (resp., V_2), v_q cannot be assigned to V_2 (resp., V_1) due to Lemma 4; on the other hand, if v_q is assigned to V_1 (resp., V_2), then v_k and v_l must be assigned to V_2 (resp., V_1) and, correspondingly, v_j cannot be assigned to V_2 (resp., V_1). Summarizing, either v_j and v_q are assigned to V_1 and, correspondingly, v_k, v_l are assigned to V_2, or v_j and v_q are assigned to V_2 and, correspondingly, v_k, v_l are assigned to V_1. This can be seen as a binary branching where, in both cases, 4 vertices (v_j, v_k, v_l, v_q) are fixed. Then, $T(n) \leqslant 2T(n-4) + O(p(n))$, i.e., $T(n) = O^*(1.1892^n)$.

(g) $1 \leqslant d_k \leqslant d_l = 3$, v_l is adjacent to v_k and to another vertex v_m (that may eventually coincide with v_i) that has already been assigned (Figure 12). If v_m has been assigned to V_2 (resp., V_1), no branching occurs, as v_j must be assigned to V_2 (resp., V_1), or else, v_k, v_l and v_m would all be assigned to V_2 (resp., V_1),

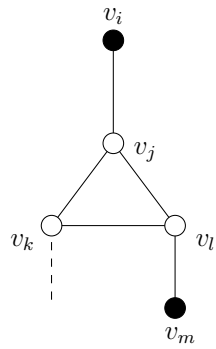

Figure 12: Case 2g of Proposition 2.

violating Lemma 2. We assume, then, that v_m has already been assigned to V_1 (resp., V_2). But then a branching on v_k can be applied. If v_k is assigned to V_1 (resp., V_2), vertices v_j and v_l must be assigned (due to Lemma 2) to V_2 (resp., V_1). If v_k is assigned to V_2 (resp., V_1), then vertices v_j, v_l must be assigned to different sets of the partition (V_1, V_2) but it is immaterial to assign v_j to V_1 and v_l to V_2 or viceversa. Putting things together, this can be seen as a binary branching where, in both cases, 3 vertices (v_j, v_k, v_l) are assigned and the same time-complexity $O^*(1.2599^n)$ of case 1c holds.

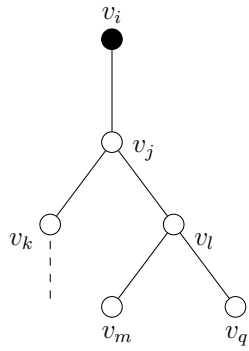

Figure 13: Case 2h of Proposition 2.

(h) $1 \leqslant d_k \leqslant 2$, $d_l = 3$, vertex v_l is not adjacent to v_k but is adjacent to vertices v_m and v_q both unassigned (Figure 13). A branching on vertex v_j can be applied. If v_j is assigned to V_1 (resp., V_2), vertices v_k and v_l must be assigned (due to Lemma 2) to V_2 (resp., V_1), and vertices v_m and v_q must be assigned to V_1

(resp., V_2) as they cannot be assigned to different sets of the partition(due to Lemma 4), nor they can both be assigned to V_2 (resp., V_1) due to Lemma 2 applied to vertex v_l. Else, v_j is assigned to V_2 (resp., V_1) and correspondingly v_k can be assigned to to V_1 (resp., V_2). This can be seen as a branching where, either 2 vertices (v_j, v_k), or 5 vertices (v_j, v_k, v_l, v_m, v_q) are assigned. Then, $T(n) \leqslant T(n-2) + T(n-5) + O(p(n))$, i.e., $T(n) = O^*(1.1939^n)$.

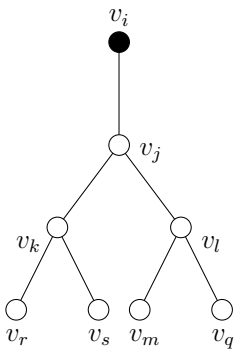

Figure 14: Case 2i of Proposition 2.

(i) $d_k = d_l = 3$, vertex v_l is not adjacent to v_k but is adjacent to vertices v_m and v_q both unassigned (Figure 14). Vertex v_k is adjacent to other two vertices v_r, v_s both unassigned. A branching on vertex v_j can be applied. If v_j is assigned to V_1 (resp., V_2), vertices v_k and v_l must be assigned (due to Lemma 2) to V_2 (resp., V_1), and vertices v_m, v_q, v_r, v_s must be assigned to V_1 (resp., V_2) due to Lemma 4. Else, v_j is assigned to V_2 (resp., V_1). This can be seen as a branching where, either 1 vertex (v_j) is assigned, or 7 vertices ($v_j, v_k, v_l, v_m, v_q, v_r, v_s$) are assigned. Then, $T(n) \leqslant T(n-1) + T(n-7) + O(p(n))$, i.e., $T(n) = O^*(1.2555^n)$.

(j) $d_k = d_l = 3$, vertex v_l is not adjacent to v_k but is adjacent to vertices v_m and v_q both unassigned (Figure 15). Vertex v_k is adjacent to vertex v_q and to another vertex v_r also unassigned. A branching on vertex v_j can be applied. If v_j is assigned to V_1 (resp., V_2), vertices v_k and v_l must be assigned (due to Lemma 2) to V_2 (resp., V_1), and vertices v_m, v_q, v_r must be assigned to V_1 (resp., V_2) due to Lemma 4. Else, v_j is assigned to V_2 (resp., V_1). This can be seen as a branching where, either 1 vertex (v_j) is assigned, or 6 vertices ($v_j, v_k, v_l, v_m, v_q, v_r$) are assigned. Then, $T(n) \leqslant T(n-1) + T(n-6) + O(p(n))$ and the same time-complexity $O^*(1.2852^n)$ of case 2c holds.

(k) $d_k = d_l = 3$, vertex v_l is not adjacent to v_k but is adjacent to vertices v_m and v_q both unassigned (Figure 16). Vertex v_k is also adjacent to vertices v_m and v_q.

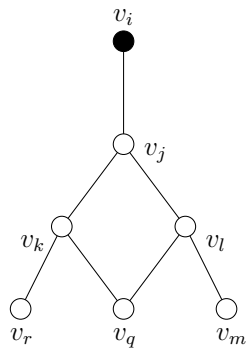

Figure 15: Case 2j of Proposition 2.

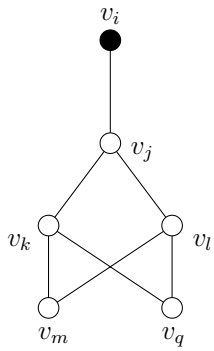

Figure 16: Case 2k of Proposition 2.

Notice that v_j and v_k cannot be assigned to the same set of the partition, or else v_l, v_m, v_q would all be assigned to the other set of the partition violating Lemma 2. Analogously, v_j and v_l cannot be assigned to the same set of the partition. But then, if v_j is assigned to V_1 (resp., V_2), correspondingly, v_k, v_l are assigned to V_2 (resp., V_1) and v_m, v_q are assigned to V_1 (resp., V_2); else, v_j is assigned to V_2 (resp., V_1), v_k, v_l are assigned to V_1 (resp., V_2) and v_m, v_q are assigned to V_2 (resp., V_1). This can be seen as a binary branching where, in both cases, 5 vertices (v_j, v_k, v_l, v_m, v_q) are assigned. Then, $T(n) \leqslant 2T(n-5) + O(p(n))$, i.e., $T(n) = O^*(1.1487^n)$.

(l) $d_k = 2, d_l = 3$, v_l is adjacent to v_k and to another vertex v_m that has not yet been assigned (Figure 17). If v_j is assigned to V_1 (resp., V_2), vertices v_k and v_l must be assigned (due to Lemma 2) to V_2 (resp., V_1), and correspondingly vertex v_m must be assigned to V_1 (resp., V_2). But an equivalent solution is

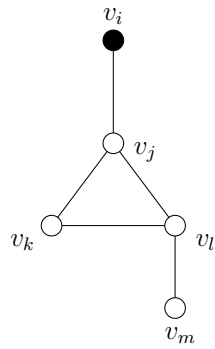

Figure 17: Case 2l of Proposition 2.

obtained by simply swapping the assignment of vertices v_j, v_k. Hence, for this subcase, v_j can be assigned to V_2 (resp., V_1) without branching.

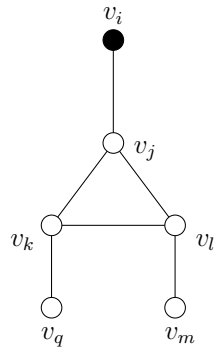

Figure 18: Case 2m of Proposition 2.

(m) $d_k = d_l = 3$, v_k is adjacent to v_l and to an unassigned vertex v_q, v_l is adjacent to v_k and to an unassigned vertex v_m, $v_m \neq v_q$ (Figure 18). If v_j is assigned to V_1 (resp., V_2), vertices v_k and v_l must be assigned (due to Lemma 2) to V_2 (resp., V_1), and correspondingly vertices v_m and v_q must be assigned to V_1 (resp., V_2). But an equivalent solution is obtained by simply swapping the assignment of vertices v_j, v_k. Hence, for this subcase, v_j can be assigned to V_2 (resp., V_1) without branching.

(n) $d_k = d_l = 3$, v_k is adjacent to v_l and both v_k and v_l are adjacent to another unassigned vertex v_m (Figure 19). If v_j is assigned to V_1 (resp., V_2), vertices v_k and v_l must be assigned (due to Lemma 2) to V_2 (resp., V_1), and, correspond-

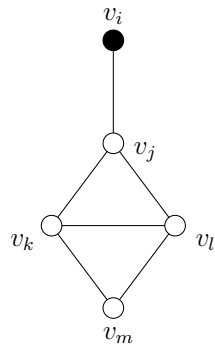

Figure 19: Case 2n of Proposition 2.

ingly, vertex v_m must be assigned to V_1 (resp., V_2). But an equivalent solution is obtained by simply swapping the assignment of vertices v_j, v_k. Hence, for this subcase, v_j can be assigned to V_2 (resp., V_1) without branching.

Putting things together, the global worst-case complexity for MAX CUT-3 with maximum degree three is $O^*(1.2920^n)$. ∎

3.4 The MAX CUT-4 problem

We now deal with graphs of maximum degree four. For this case, the following proposition holds.

Proposition 3. *Algorithm* `SOLVE-MAX-CUT` *optimally solves* MAX CUT*-4 with time-complexity* $O^*(1.4142^n)$.

Proof. For $d_{\max} = 4$, the relevant branching cases for a given vertex v_j in the application of `SOLVE-MAX-CUT` are those mentioned in the proof of Proposition 2 plus all cases related to the presence of vertices with degree four. The following further cases must be taken into account.

1. $d_j = 4$, two vertices v_i, v_k adjacent to v_j have already been assigned to different sets of the partition (V_1, V_2), while the other two adjacent vertices v_l, v_m have not yet been assigned (Figure 20). Then, a branching on vertex v_l can be applied. If v_l is assigned to V_1 (resp., V_2), due to Lemma 2, v_j must be assigned to V_2 (resp., V_1). This can be seen as a binary branching where, in both cases, 2 vertices (v_j, v_l) are assigned. Then, $T(n) \leqslant 2T(n-2) + O(p(n))$, i.e., $T(n) = O^*(1.4142^n)$.

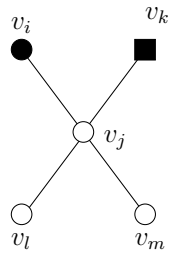

Figure 20: Case 1 of Proposition 3.

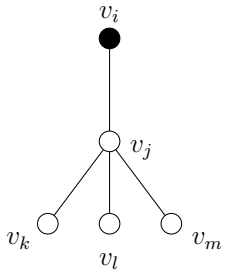

Figure 21: Case 2 of Proposition 3.

2. $d_j = 4$, a vertex v_i adjacent to v_j has already been assigned to V_1 (resp., V_2), while the other three adjacent vertices v_k, v_l, v_m have not yet been assigned (Figure 21). If v_j is assigned to V_1 (resp., V_2), due to Lemma 2, v_k, v_l, v_m must be assigned to V_2 (resp., V_1); else, v_j is assigned to V_2 (resp., V_1). This can be seen as a binary branching where, either a vertex v_j is assigned, or 4 vertices (v_j, v_k, v_l, v_m) are assigned. Then, $T(n) \leqslant T(n-1)+T(n-4)+O(p(n))$, i.e., $T(n) = O^*(1.3803^n)$.

3. $d_j = 3$, two vertices v_i, v_k adjacent to v_j have already been assigned to different sets of the partition (V_1, V_2), while the third adjacent vertex v_l has not yet been assigned and $d_l > 3$ (Figure 22). Then, due to Lemma 2, if v_j is assigned to V_1 (resp., V_2), v_l must be assigned to V_2 (resp., V_1). This can be seen as a binary branching where, in both cases, 2 vertices (v_j, v_l) are assigned and the same time complexity $O^*(1.4142^n)$ of case 1 holds.

4. $d_j = 3$, a vertex v_i adjacent to v_j has already been assigned to V_1 (resp., V_2), while the other two adjacent vertices v_k and v_l have not yet been assigned. We assume, without loss of generality, that $d_k \leqslant d_l$ and that $d_l = 4$ (or else this case has already

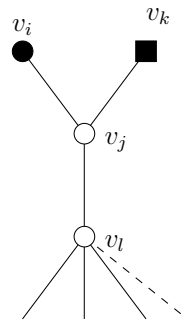

Figure 22: Case 3 of Proposition 3.

been handled). Notice that no vertices adjacent to v_l have already been assigned, or else we get back to cases 1 or 2 applied to vertex v_l. The following subcases must be considered.

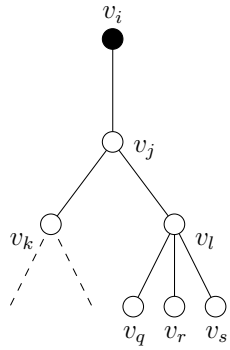

Figure 23: Case 4a of Proposition 3.

(a) v_l is adjacent (apart from v_j) to v_q, v_r, v_s all unassigned (Figure 23). A composite branching can be applied. If v_j is assigned to V_1 (resp., V_2), vertices v_k and v_l must be assigned (due to Lemma 2) to V_2 (resp., V_1). Else, v_j is assigned to V_2 (resp., V_1). Then, if v_l is assigned to V_2 (resp., V_1), v_k, v_q, v_r, v_s must all be assigned to V_1 (resp., V_2) due to Lemma 2, else v_l is assigned to V_1 (resp., V_2). This can be seen as a composite branching where, either 3 vertices (v_j, v_k, v_l) are assigned, or 6 vertices ($v_j, v_k, v_l, v_q, v_r, v_s$) are assigned, or 2 vertices (v_j, v_l) are assigned. Then, $T(n) \leqslant T(n-2) + T(n-3) + T(n-6) + O(p(n))$, i.e., $T(n) = O^*(1.4037^n)$.

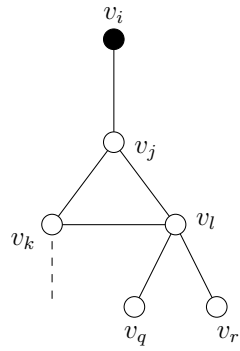

Figure 24: Case 4b of Proposition 3.

(b) v_l is adjacent (apart from v_j) to v_k, v_q, v_r all unassigned (Figure 24). A composite branching can be applied. If v_j is assigned to V_1 (resp., V_2), then, due to Lemma 2, vertices v_k and v_l must be assigned to V_2 (resp., V_1) and vertices v_q and v_r must be assigned to V_1 (resp., V_2). Else, v_j is assigned to V_2 (resp., V_1). This can be seen as a branching where, either 1 vertex (v_j) is assigned, or 5 vertices (v_j, v_k, v_l, v_q, v_r) are assigned. Then, $T(n) \leqslant T(n-1) + T(n-5) + O(p(n))$, i.e., $T(n) = O^*(1.3248^n)$.

Putting things together, the global worst case complexity is $O^*(1.4142^n)$. ∎

3.5 The MAX CUT-5 problem

In this section, we deal with graphs with maximum degree five. The following result holds.

Proposition 4. *Algorithm* `SOLVE-MAX-CUT` *optimally solves* MAX CUT-*5 with time-complexity* $O^*(1.6430^n)$.

Proof. For $d_{\max} = 5$, the relevant branching cases for a given vertex v_j in the application of `SOLVE-MAX-CUT` are those mentioned previously in Propositions 2 and 3, plus all cases related to the presence of vertices with degree five. Notice that, for $d_j = 5$, if at least three vertices adjacent to v_j have been assigned to V_1 (resp., V_2), then v_j can be assigned without branching (due to Lemma 2) to V_2 (resp., V_1). The following further cases must be taken into account.

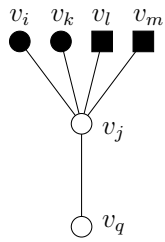

Figure 25: Case 1 of Proposition 4.

1. $d_j = 5$, four vertices v_i, v_k, v_l, v_m adjacent to v_j have already been equally partitioned between V_1 and V_2, while the fifth adjacent vertex v_q has not yet been assigned (Figure 25). Then, a branching on vertex v_j can be applied. If v_j is assigned to V_1 (resp., V_2), due to Lemma 2, v_q must be assigned to V_2 (resp., V_1). This can be seen as a binary branching where, in both cases, 2 vertices (v_j, v_q) are assigned and the same time complexity $O^*(1.4142^n)$ of case 1 in Proposition 3 holds.

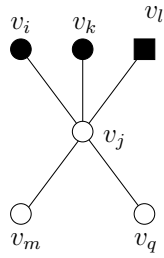

Figure 26: Case 2 of Proposition 4.

2. $d_j = 5$, three vertices v_i, v_k, v_l adjacent to v_j have already been assigned with v_i, v_k assigned to V_1 (resp., V_2) and v_l assigned to V_2 (resp., V_1), while the other two adjacent vertex v_m, v_q have not yet been assigned (Figure 26). Then, a branching on vertex v_j can be applied. If v_j is assigned to V_1 (resp., V_2), due to Lemma 2, v_m, v_q must be assigned to V_2 (resp., V_1), else v_j is assigned to V_2 (resp., V_1). This can be seen as a binary branching where, either a vertex v_j is assigned, or 3 vertices (v_j, v_m, v_q) are assigned. Then, $T(n) \leqslant T(n-1) + T(n-3) + O(p(n))$, i.e., $T(n) = O^*(1.4657^n)$.

3. $d_j = 5$, two vertices v_i, v_k adjacent to v_j have already been assigned to V_1 (resp., V_2), while the other three adjacent vertex v_l, v_m, v_q have not yet been assigned (Figure 27). Then, a branching on vertex v_j can be applied. If v_j is assigned to V_1

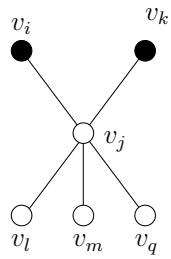

Figure 27: Case 3 of Proposition 4.

(resp., V_2), due to Lemma 2, v_l, v_m, v_q must be assigned to V_2 (resp., V_1), else v_j is assigned to V_2 (resp., V_1). This can be seen as a binary branching where, either a vertex v_j is assigned, or 4 vertices (v_j, v_l, v_m, v_q) are assigned and the same time complexity $O^*(1.3803^n)$ of case 2 in Proposition 3 holds.

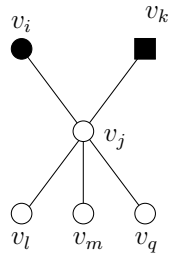

Figure 28: Case 4 of Proposition 4.

4. $d_j = 5$, two vertices v_i, v_k adjacent to v_j have already been assigned to different sets of the partition (V_1, V_2), while the other three adjacent vertex v_l, v_m, v_q have not yet been assigned (Figure 28). A composite branching can be applied. If v_j and v_l are both assigned to the same set of the partition, say V_1 (resp., V_2), then, due to Lemma 2, v_m, v_q must both be assigned to the other set of the partition, say V_2 (resp., V_1); else, if v_j is assigned to V_1 (resp., V_2) and v_l is assigned to V_2 (resp., V_1), then, if v_m is assigned to V_1 (resp., V_2), v_q must be assigned to V_2 (resp., V_1), else v_m is assigned to V_2 (resp., V_1). This can be seen as a composite branching where, four children nodes are generated assigning four vertices (v_j, v_l, v_m, v_q) and two children nodes are generated assigning three vertices (v_j, v_l, v_m). Then, $T(n) \leqslant 2T(n-3) + 4T(n-4) + O(p(n))$, i.e., $T(n) = O^*(1.6430^n)$.

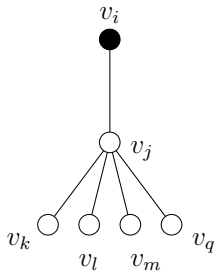

Figure 29: Case 5 of Proposition 4.

5. $d_j = 5$, a vertex v_i adjacent to v_j has already been assigned to V_1 (resp., V_2), while the other four adjacent vertices v_k, v_l, v_m, v_q have not yet been assigned (Figure 29). A composite branching can be applied. If v_j and v_k are both assigned to V_1 (resp., V_2), due to Lemma 2, v_l, v_m, v_q must all be assigned to V_2 (resp., V_1); else, if v_j is assigned to V_1 (resp., V_2) and v_k is assigned to V_2 (resp., V_1), then, if v_l is assigned to V_1 (resp., V_2), v_m and v_q must be assigned to V_2 (resp., V_1); else, if v_j is assigned to V_1 (resp., V_2), v_k is assigned to V_2 (resp., V_1) and v_l is assigned to V_2 (resp., V_1), then, either v_m is assigned to V_1 (resp., V_2), and v_q is assigned to V_2 (resp., V_1) or v_m is assigned to V_2 (resp., V_1) and nothing is derived with respect to v_q; else, v_j is assigned to V_1 (resp., V_2). This can be seen as a composite branching where, three children nodes are generated assigning five vertices (v_j, v_k, v_l, v_m, v_q), a child node is generated assigning four vertices (v_j, v_k, v_l, v_m), and another child node is generated assigning one vertex (v_j). Then, $T(n) \leqslant T(n-1) + T(n-4) + 3T(n-5) + O(p(n))$, i.e., $T(n) = O^*(1.6406^n)$.

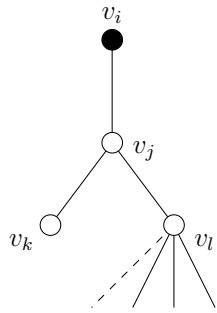

Figure 30: Case 6 of Proposition 4.

6. $d_j = 3$, a vertex v_i adjacent to v_j has already been assigned to V_1 (resp., V_2), while the other two adjacent vertices v_k and v_l have not yet been assigned (Figure 30). We assume, without loss of generality, that $d_k \leqslant d_l$ and that $d_l > 4$ (or else, this case has already been handled). Then, a branching on vertex v_j can be applied. If v_j is assigned to V_1 (resp., V_2), due to Lemma 2, v_k, v_l must be assigned to V_2 (resp., V_1), else v_j is assigned to V_2 (resp., V_1). This can be seen as a binary branching where, either a vertex v_j is assigned, or 3 vertices (v_j, v_k, v_l) are assigned and the same time complexity $O^*(1.4657^n)$ of case 2 holds.

Putting things together, the global worst case complexity is $O^*(1.6430^n)$. ∎

3.6 MAX CUT-6 problem

We finally deal with graphs with maximum degree six. In what follows, we prove the following result.

Proposition 5. *Algorithm* SOLVE-MAX-CUT *optimally solves* MAX CUT-6 *with time-complexity* $O^*(1.6430^n)$.

Proof. For $d_{\max} = 6$, the relevant branching cases for a given vertex v_j in the application of SOLVE-MAX-CUT are those mentioned previously, in Propositions 2 and 3 and 4, plus all cases related to the presence of vertices with degree six. Notice that also for $d_j = 6$, if at least three vertices adjacent to v_j have been assigned to V_1 (resp., V_2), then v_j can be assigned without branching (due to Lemma 2) to V_2 (resp., V_1). The following further cases must be taken into account.

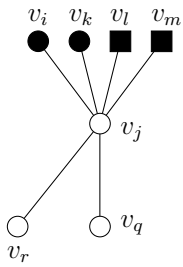

Figure 31: Case 1 of Proposition 5.

1. $d_j = 6$, four vertices v_i, v_k, v_l, v_m adjacent to v_j have already been equally partitioned between V_1 and V_2, while the other two adjacent vertices v_q, v_r have not yet

been assigned (Figure 31). Then, a branching on vertex v_j can be applied. If v_j is assigned to V_1 (resp., V_2), due to Lemma 2, v_q, v_r must be assigned to V_2 (resp., V_1), else v_j is assigned to V_2 (resp., V_1). This can be seen as a binary branching where, either a vertex v_j is assigned, or 3 vertices (v_j, v_q, v_r) are assigned and the same time complexity $O^*(1.4657^n)$ of case 2 in Proposition 4 holds.

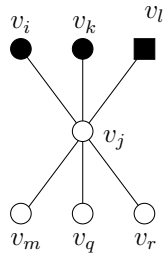

Figure 32: Case 2 of Proposition 5.

2. $d_j = 6$, three vertices v_i, v_k, v_l adjacent to v_j have already been assigned with v_i, v_k assigned to V_1 (resp., V_2) and v_l assigned to V_2 (resp., V_1), while the other three adjacent vertex v_m, v_q, v_r have not yet been assigned (Figure 32). Then, a branching on vertex v_j can be applied. If v_j is assigned to V_1 (resp., V_2), due to Lemma 2, v_m, v_q, v_r must be assigned to V_2 (resp., V_1), else v_j is assigned to V_2 (resp., V_1). This can be seen as a binary branching where, either a vertex v_j is assigned, or 4 vertices (v_j, v_m, v_q, v_r) are assigned and the same time complexity $O^*(1.3803^n)$ of case 2 in Proposition 3 holds.

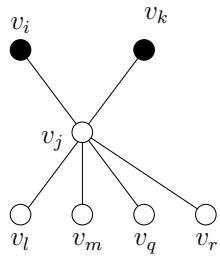

Figure 33: Case 3 of Proposition 5.

3. $d_j = 6$, two vertices v_i, v_k adjacent to v_j have already been assigned to V_1 (resp., V_2), while the other four adjacent vertex v_l, v_m, v_q, v_r have not yet been assigned (Figure 33). Then, a branching on vertex v_j can be applied. If v_j is assigned to V_1

(resp., V_2), due to Lemma 2, v_l, v_m, v_q, v_r must be assigned to V_2 (resp., V_1), else v_j is assigned to V_2 (resp., V_1). This can be seen as a binary branching where, either a vertex v_j is assigned, or 5 vertices $(v_j, v_l, v_m, v_q, v_r)$ are assigned and the same time complexity $O^*(1.3248^n)$ of case 4b in Proposition 3 holds.

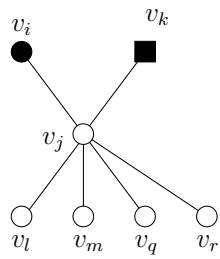

Figure 34: Case 4 of Proposition 5.

4. $d_j = 6$, two vertices v_i, v_k adjacent to v_j have already been assigned to different sets of the partition (V_1, V_2), while the other four adjacent vertices v_l, v_m, v_q, v_r have not yet been assigned (Figure 34). A composite branching can be applied. If v_j and v_l are both assigned to V_1 (resp., V_2), then, due to Lemma 2, v_m, v_q, v_r must all be assigned to V_2 (resp., V_1); else, if v_j, v_m are assigned to V_1 (resp., V_2) and v_l is assigned to V_2 (resp., V_1), v_q, v_r must both be assigned to V_2 (resp., V_1). Else, if v_j, v_q are assigned to V_1 (resp., V_2) and v_l, v_m are assigned to V_2 (resp., V_1), v_r must be assigned to V_2 (resp., V_1); else, either v_j is assigned to V_1 (resp., V_2) and v_l, v_m, v_q are assigned to V_2 (resp., V_1), or v_j is assigned to V_2 (resp., V_1). This can be seen as a composite branching where, three children nodes are generated assigning five vertices $(v_j, v_l, v_m, v_q, v_r)$ a child node is generated assigning four vertices (v_j, v_l, v_m, v_q) and another child node is generated assigning one vertex (v_j). Hence, the same time complexity $O^*(1.6406^n)$ of case 5 in Proposition 4 holds.

5. $d_j = 6$, a vertex v_i adjacent to v_j has already been assigned to V_1 (resp., V_2), while the other five adjacent vertices v_k, v_l, v_m, v_q, v_r have not yet been assigned (Figure 35). A composite branching can be applied. If v_j and v_k are both assigned to V_1 (resp., V_2), then, due to Lemma 2, v_l, v_m, v_q, v_r must all be assigned to V_2 (resp., V_1); else, if v_j, v_l are assigned to V_1 (resp., V_2) and v_k is assigned to V_2 (resp., V_1), v_m, v_q, v_r must all be assigned to V_2 (resp., V_1). Else, if v_j, v_m are assigned to V_1 (resp., V_2) and v_k, v_l are assigned to V_2 (resp., V_1), v_q, v_r must both be assigned to V_2 (resp., V_1); else, if v_j, v_q are assigned to V_1 (resp., V_2) and v_k, v_l, v_m are assigned to V_2 (resp., V_1), v_r must be assigned to V_2 (resp., V_1); else, either v_j is assigned to V_1 (resp., V_2) and v_k, v_l, v_m, v_q are assigned to V_2 (resp., V_1), or v_j is assigned to V_2 (resp., V_1). This can be seen as a composite branching where,

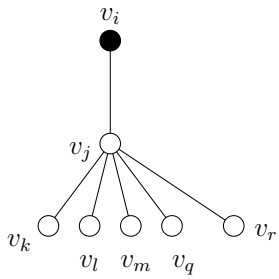

Figure 35: Case 5 of Proposition 5.

four children nodes are generated assigning six vertices ($v_j, v_k, v_l, v_m, v_q, v_r$) a child node is generated assigning five vertices (v_j, v_k, v_l, v_m, v_q) and another child node is generated assigning one vertex (v_j). Then, $T(n) \leqslant T(n-1) + T(n-5) + 4T(n-6) + O(p(n))$, i.e., $T(n) = O^*(1.5751^n)$.

Putting things together, we notice that all cases related to the presence of vertices with degree six are not worse than the worst-case for $d_{\max} = 5$. Hence, also here the global worst case complexity is, as in Proposition 4, $O^*(1.6430^n)$. ∎

4 Conclusion

We have presented an analysis for the worst-case complexity of two search-tree based algorithms for the MIN 3-SET COVERING problem and the MAX CUT-3 problem, strongly based on simple dominance conditions. We point out that these conditions allow not only to prune search-tree nodes corresponding to strictly dominated partial solutions, but also to break ties among equivalent partial solutions (see, for instance, Lemma 4). This second aspect is probably the most rewarding in terms of worst-case complexity analysis. Thanks to the dominance conditions used here, the derived upper-time bounds for both MIN 3-SET COVERING and MAX CUT-3 are competitive with the state of the art available bounds.

This approach seem easily extendable to other bounded combinatorial optimization problems. For instance, we notice that a straightforward (improvable) analysis along the lines of the above ones, leads to an $O^*(1.1679^n)$ time bound for minimum vertex covering in graphs with maximum degree 3. This bound, even though dominated by the ones in [1, 4], $O^*(1.1252^n)$ and $O^*(1.152^n)$, respectively, already dominates the one in [3].

References

[1] R. Beigel. Finding maximum independent sets in sparse and general graphs. In *Proc. Symposium on Discrete Algorithms, SODA*, pages 856–857, 1999.

[2] C. Berge. *Graphs and hypergraphs*. North Holland, Amsterdam, 1973.

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

[4] J. Chen, L Liu, and W. Jia. Improvement on vertex cover for low-degree graphs. *Networks*, 35:253–259, 2000.

[5] D. Eppstein. Improved algorithms for 3-coloring, 3-edge-coloring, and constraint satisfaction. In *Proc. Symposium on Discrete Algorithms, SODA*, pages 329–337, 2001.

[6] S. S. Fedin and A. S. Kulikov. A $2^{|E|/4}$-time algorithm for max-cut. *Journal of Mathematical Sciences*. To appear. Available at http://logic.pdmi.ras.ru/k̃ulikov/maxcut_e.ps.gz

[7] F. V. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: domination – a case study. Reports in Informatics 294, Department of Informatics, University of Bergen, 2005. To appear in the Proceedings of ICALP'05.

[8] M. R. Garey and D. S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness*. W. H. Freeman, San Francisco, 1979.

[9] J. Gramm, E. A. Hirsch, R. Niedermaier, and P. Rossmanith. Worst-case upper bounds for Max2Sat with an application to MaxCut. *Discrete Appl. Math.*, 130:139–155, 2003.

[10] F. Grandoni. A note on the complexity of minimum dominating set. *J. Discr. Algorithms*, 2005. To appear.

[11] A. D. Scott and G. B. Sorkin. Faster algorithms for MAX CUT and MAX CSP, with polynomial expected time for sparse instances. In *Proc. RANDOM'03*, volume 2764 of *Lecture Notes in Computer Science*, pages 382–395. Spinger-Verlag, 2003.

[12] A. D. Scott and G. B. Sorkin. Solving sparse semi-random instances of Max-Cut and Max-CSP in linear expected time. Research Report 23417 (W0411-056), IBM Research division, Thomas J. Watson Researcc Center, 2004.

[13] G. J. Wœginger. Exact algorithms for NP-hard problems: a survey. In M. Juenger, G. Reinelt, and G. Rinaldi, editors, *Combinatorial Optimization - Eureka! You shrink!*, volume 2570 of *Lecture Notes in Computer Science*, pages 185–207. Springer-Verlag, 2003.