Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

CliSAT: A new exact algorithm for hard maximum clique problems

Pablo San Segundo[a,*], Fabio Furini[b], David Álvarez[a], Panos M. Pardalos[c]

[a] Universidad Politécnica de Madrid (UPM), Centre for Automation and Robotics (CAR), Ronda de Valencia 3, Madrid 28012, Spain
[b] Department of Computer, Control and Management Engineering Antonio Ruberti, Sapienza University of Rome, Via Ariosto 25, Rome, RM 00185, Italy
[c] Center for Applied Optimization, University of Florida (UF), 358 Little Hall Gainesville, Florida, FL 32611-6595, USA

ARTICLE INFO

Article history:
Received 20 April 2022
Accepted 14 October 2022
Available online 20 October 2022

Keywords:
Combinatorial optimization
Exact algorithm
Branch-and-bound algorithm
Maximum clique problem

ABSTRACT

Given a graph, the maximum clique problem (MCP) asks for determining a complete subgraph with the largest possible number of vertices. We propose a new exact algorithm, called `CliSAT`, to solve the MCP to proven optimality. This problem is of fundamental importance in graph theory and combinatorial optimization due to its practical relevance for a wide range of applications. The newly developed exact approach is a combinatorial branch-and-bound algorithm that exploits the state-of-the-art branching scheme enhanced by two new bounding techniques with the goal of reducing the branching tree. The first one is based on graph colouring procedures and partial maximum satisfiability problems arising in the branching scheme. The second one is a filtering phase based on constraint programming and domain propagation techniques. `CliSAT` is designed for structured MCP instances which are computationally difficult to solve since they are dense and contain many interconnected large cliques. Extensive experiments on hard benchmark instances, as well as new hard instances arising from different applications, show that `CliSAT` outperforms the state-of-the-art MCP algorithms, in some cases by several orders of magnitude.

1. Introduction

Let G be a simple undirected graph, we denote $V(G)$ its set of n vertices and $E(G)$ its set of m edges. Two vertices $u, v \in V(G)$ are called *adjacent* or *neighbours* if there is an edge $\{u, v\} \in E(G)$. A *clique* is a subset of pairwise adjacent vertices or, equivalently, a subset of vertices inducing a complete graph. The *maximum clique problem* (MCP) calls for determining a clique of G with the largest possible number of vertices, the size of which is known as the *clique number* $\omega(G)$. Fig. 1 provides an example graph G with $n = 8$ vertices and $m = 22$ edges where $\omega(G) = 4$. A maximum clique is $K = \{v_1, v_2, v_3, v_4\}$ (the red vertices of the figure), the edges of the complete graph induced by these vertices are depicted with red lines. This graph contains multiple maximum cliques, another maximum clique is, e.g., the set $\{v_3, v_4, v_5, v_6\}$.

The MCP is one of the most studied combinatorial optimization problems in graph theory. It is known to be strongly $\mathcal{NP}$-hard and also hard to approximate within any polynomial factor unless $\mathcal{P} = \mathcal{ZPP}$ (Håstad, 1999). The MCP finds numerous applications which span disciplines such as computer vision (San Segundo & Artieda, 2015; San Segundo, Rodriguez-Losada, Matia, & Galan, 2010; Stentiford, 2019), robotics (San Segundo & Rodriguez-Losada, 2013), coding theory, network analysis and bioinformatics, see, e.g., Tomita, Akutsu, & Matsunaga (2011).

In this work, we describe a new exact branch-and-bound (BnB) algorithm for the MCP that we call `CliSAT`. This algorithm is designed for hard MCP instances with up to several tenths of thousands of vertices. Hard MCP instances are those with many large interconnected cliques and they are typically dense. For these instances, the state-of-the-art techniques are combinatorial BnB algorithms (see, e.g., Wu & Hao, 2015) that employ bounding procedures based on graph colouring and partial maximum satisfiability (SAT) problems arising in the branching scheme. Our new exact algorithm is an enhancement of this class of algorithms that introduces new bounding procedures. These procedures, combined with the state-of-the-art branching scheme, are very effective in solving hard MCP instances as shown in the computational section. On the classical `DIMACS` set of instances, `CliSAT` compares favorably with previous state-of-the-art exact algorithms; moreover, on several new classes of hard instances `CliSAT` is the best performing algorithm, in some cases by several orders of magnitude.

It is important to mention that solving the MCP on very sparse massive graphs is, in practice, much easier than solving it for struc-

* Corresponding author.
E-mail addresses: pablo.sansegundo@upm.es (P. San Segundo), fabio.furini@uniroma1.it (F. Furini), david.asanchez@upm.es (D. Álvarez), pardalos@ise.ufl.edu (P.M. Pardalos).

https://doi.org/10.1016/j.ejor.2022.10.028
0377-2217/© 2022 Elsevier B.V. All rights reserved.

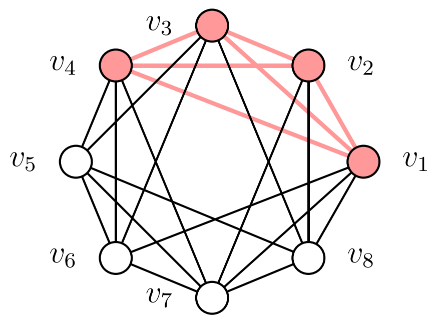

Fig. 1. An example graph G with $\omega(G) = 4$. In red, a maximum clique $K = \{v_1, v_2, v_3, v_4\}$. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

tured dense graphs. A different class of specialized algorithms has been specifically proposed in the literature for the former setting. This family of algorithms is based on tailored graph reduction techniques that are only effective for sparse instances (see e.g., Jiang, Li, & Manya, 2016; San Segundo, Lopez, & Pardalos, 2016b; Walteros & Buchanan, 2020). The proposed algorithm CliSAT, even if it is not designed for this type of instances, is competitive with the state-of-the-art algorithms also for sparse graphs with up to 150,000 vertices.

1.1. Basic notation and definitions

Given a simple graph G and a subset of its vertices $W \subseteq V(G)$, we denote $G[W]$ the *induced graph* by W, i.e., the graph with vertex set $V(G[W])$ equal to W, and edge set $E(G[W])$ containing the subset of edges of $E(G)$ with both endpoints in W. The *complement graph*, denoted $\overline{G}$, has the same vertex set of G and edge set $\overline{E}(G) = \{u, v \in V(G) : \{u, v\} \notin E(G), u \neq v\}$. A subset $I \subseteq V(G)$ of pairwise non-adjacent vertices is called an *independent set* and it corresponds to a clique in $\overline{G}$. Moreover, let $N(u) = \big\{v \in V(G) : \{u, v\} \in E(G)\big\}$ denote the *neighbourhood* of a vertex $u \in V(G)$. A *vertex colouring* of a graph G is a partition of its vertex set into independent sets, also referred to as *colour-classes*. The *vertex colouring problem* (VCP) calls for determining the minimum number of colour-classes in any feasible vertex colouring, i.e., to determine the *chromatic number* $\chi(G)$ of the graph. We refer the interested reader to Malaguti & Toth (2010) for further details on the VCP. A (vertex) *k-colouring* of a graph G, which we denote $C_k(G)$, is a partition of $V(G)$ into k independent sets; precisely: $C_k(G) = \big\{I_1, I_2, \dots, I_k\big\}$. Clearly, $\chi(G)$ provides an upper bound on the clique number $\omega(G)$, see, e.g., Balas & Yu (1986), and, consequently, so is the value k of any k-colouring of the graph, i.e., $\omega(G) \leq \chi(G) \leq |C_k(G)| = k$. Given a subset of vertices $W \subseteq V(G)$ and a partition of W into k independent sets, the k-colouring $C_k(G[W])$ is a called a *partial vertex colouring* of the graph G, i.e., a vertex colouring in which only the vertices of W are coloured.

1.2. Reduction of the MCP to a partial maximum satisfiability problem

Given a graph G together with a k-colouring $C_k(G)$, we describe in this section a reduction, first proposed in Li & Quan (2010b), of the MCP to a *partial maximum satisfiability problem* (PMAX-SAT-P). A boolean variable $x \in \{0, 1\}$ is associated to two literals, a *positive literal*, denoted y and a *negative literal* denoted $\bar{y}$. The positive literal is true if $x = 1$ and the negative literal is true if $x = 0$. A *clause* is a finite collection of literals linked by logical operators (e.g., $\vee$ and $\wedge$). A *unit clause* refers to a clause with a single literal. Boolean formulas comprise clauses linked by logical operators. A Boolean formula in *conjunctive normal form* (CNF) is a conjunction of clauses, where a clause is a disjunction of literals.

The PMAX-SAT-P, associated to a MCP and a k-colouring, comprises two types of clauses denoted *hard clauses* and *soft clauses*. It calls for satisfying the maximum number of soft clauses, while satisfying all the hard ones. This PMAX-SAT-P features a vector of boolean variables $x \in \{0, 1\}^{|V(G)|}$, where each variable x_v represents a vertex $v \in V(G)$. Its $|\overline{E}(G)|$ hard clauses are associated to the non-edges of G. They contain only negative literals and encode the fact that at most one vertex from each pair of non-adjacent vertices of G can be part of a clique:

$$(\ \bar{y}_u\ \vee\ \bar{y}_v\ ), \quad \forall \{u, v\} \in \overline{E}(G). \tag{1}$$

The hard clauses form a CNF boolean formula modelling the feasibility part of the MCP. The k soft clauses are associated to the independent sets of $C_k(G)$. They contain only positive literals and encode the fact that only one vertex from each independent set can be part of a clique:

$$\big(\ y_{v(I,1)}\ \vee\ y_{v(I,2)}\ \vee\ \dots\ \vee\ y_{v(I,t)}\ \big), \quad \forall I \in C_k(G). \tag{2}$$

For each independent set $I \in C_k(G)$, the function $v(I, s)$ returns the vertex $v \in V(G)$ associated to the sth vertex of I, and $t = |I|$. We denote $\mathscr{I}$ the collection of all the soft clauses (2), which form a CNF boolean formula modelling the objective function of the MCP, i.e., each satisfied clause corresponds to inserting the vertex of its true positive literal in a MCP solution. We denote $\mathrm{PSAT}(G, C_k(G))$ the PMAX-SAT-P associated to the graph G together with the k-colouring $C_k(G)$.

1.3. PMAX-SAT-P based upper bounds on the clique number

For a given graph G together with a colouring $C_k(G)$, upper bounds on the clique number $\omega(G)$ can be derived by reasoning and propagating the information of the hard clauses (1) and soft clauses (2) of the associated $\mathrm{PSAT}(G, C_k(G))$. It is straightforward to see that the existence of a clique of size k in G requires that all the k soft clauses (2) are satisfied. A subset $\mathscr{C} \subseteq \mathscr{I}$ of soft clauses (2) where at most $|\mathscr{C}| - 1$ of them can be satisfied, is called a *conflict*. A conflict-detection procedure determines a conflict by setting to false literals, i.e., removing them from the hard and soft clauses, while preserving logical entailment, until a clause becomes empty. A conflict logically entails an *empty clause*, i.e., a clause that contains no literals and, by definition, evaluates to false. If a conflict is found in $\mathrm{PSAT}(G, C_k(G))$, a clique of size k cannot exist in G; accordingly, $k - 1$ is in this case an upper bound on $\omega(G)$.

Unit Propagation (UP) is one of the main conflict-detection procedures, see Davis & Putnam (1960). It exploits the fact that a unit clause can only be satisfied by fixing its literal to true and, consequently, removing the negated literal from the remaining clauses. UP is applied iteratively after each removal until either *i*) there are no more unit clauses, or (*ii*) an empty clause is found. In the latter case, the soft clauses (2) in which a positive literal is set to true, together with the soft clause that becomes empty, determine a conflict.

Strong upper bounds on $\omega(G)$ can be obtained if more than one conflict is determined, see, e.g., Li, Fang, Jiang, & Xu (2018a); Li, Jiang, & Manyà (2017). A collection of conflicts $\mathscr{P} = \{\mathscr{C}_1, \mathscr{C}_2, \dots, \mathscr{C}_{|\mathscr{P}|}\}$ is denoted a *proper set of conflicts* if for each pair of conflicts $(\mathscr{C}_a, \mathscr{C}_b)$ in $\mathscr{P}$, the set of soft clauses in $(\mathscr{C}_a \cup \mathscr{C}_b) \setminus (\mathscr{C}_a \cap \mathscr{C}_b)$ is also a conflict. In other words, the soft clauses that belong exactly to only one of the two conflicts also contain a conflict. If a proper set of conflicts is found, then $\omega(G) \leq k - |\mathscr{P}|$. Determining a proper set of conflicts can be done iteratively, one conflict at a time, see, e.g., Li et al. (2018a, 2017). We recall, in what follows, the overarching idea of such procedures. For each conflict $\mathscr{C}$ found, the $\mathrm{PSAT}(G, C_k(G))$ is modified in such a way that the set of clauses in $\mathscr{C}$ are satisfiable. Precisely, the graph G is enlarged with $|\mathscr{C}|$ additional vertices, by inserting a new vertex per independent set associated to the clauses of $\mathscr{C}$. Each new vertex is connected to every vertex $V(G)$ in the graph, except to those vertices associated to the literals of its clause. In this way, we obtain a new graph called the *transformed graph of a conflict*, which we denote $G(\mathscr{C})$. The new $C_k(G(\mathscr{C}))$ is obtained from the original $C_k(G)$ by colouring each new vertex with the colour class of its associated clause. In addition, a new $\text{PSAT}\big(G(\mathscr{C}), C_k(G(\mathscr{C}))\big)$ can be defined in which the relaxed clauses of $\mathscr{C}$ are satisfiable. This problem is used to determine additional conflicts. A set of conflicts iteratively determined in this way is, by nature, a proper set of conflicts.

In addition to UP, and when no unit clauses are available, the *failed literal* conflict-detection procedure (FL), another well-established inference procedure used by SAT solvers, can be used in this context to determine conflicts. A positive literal of a soft clause is denoted *failed* if an empty clause is determined when it is set to true. If every literal in a clause is proven failed by successive calls to FL, a conflict has been found. The soft clauses of this conflict are those in which a positive clause is fixed to true by the different calls to FL together with the corresponding empty clauses.

1.4. Literature review on exact MCP algorithms

A large amount of effort has been devoted to solving the MCP to proven optimality. We refer the reader to Wu & Hao (2015) for a detailed survey on this topic. A complete overview of exact algorithms is out of the scope of this work. In what follows, we describe what we consider the most relevant ones together with their corresponding breakthroughs. One of the first successful BnB algorithms is described in Carraghan & Pardalos (1990), where a tailored n-ary branching scheme for the MCP is proposed. A bounding technique based on vertex colouring is described in Fahle (2002), an idea almost universally employed by modern exact MCP algorithms, see, e.g., San Segundo, Matia, Rodriguez-Losada, & Hernando (2013); San Segundo, Rodríguez-Losada, & Jiménez (2011); San Segundo & Tapia (2014); Tomita, Sutani, Higashi, Takahashi, & Wakatsuki (2010). One of the major breakthroughs of the last decade is the bounding technique proposed in Li & Quan (2010a,b). This family of upper bounds is based on partial maximum satisfiability problems arising in the branching scheme. Thanks to this new idea, the exact MCP algorithms have substantially improved their performance. Some of the state-of-art algorithms of this type are, e.g., Li et al. (2018a, 2017); San Segundo, Nikolaev, & Batsyn (2015); San Segundo, Nikolaev, Batsyn, & Pardalos (2016c). Finally, bitstring optimizations are known to be an additional source of efficiency, see, e.g., San Segundo et al. (2013, 2015); San Segundo et al. (2016c, 2011); San Segundo & Tapia (2014).

To the best of our knowledge, the most successful exact algorithm for hard dense MCP instances is MoMC, which is described in Li et al. (2017). In the computational section, we compare the performance of our new algorithm CliSAT against MoMC, as well as several other efficient exact algorithms and integer linear programming (ILP) models solved by a state-of-the-art commercial solver.

Another recent stream of research aims at determining the clique number of real and very sparse massive graphs, such as those associated with social networks. Specialized algorithms exploit the scale-free nature of such graphs, i.e., graphs whose degree distribution follow a power law. These algorithms are able to solve the MCP to proven optimality in networks with millions of vertices, see, e.g., Hespe, Lamm, Schulz, & Strash (2020); San Segundo et al. (2016b); Walteros & Buchanan (2020). The techniques employed to determine a maximum clique for these instances are typically not effective for hard and dense MCP instances. For sparse massive instances, the most successful exact algorithms are dOmega, proposed in Walteros & Buchanan (2020), BBMCSP, proposed in San Segundo et al. (2016b) and LMC proposed in Jiang et al. (2016). These two algorithms are compared against CliSAT in the computational section.

It is also worth mentioning that exact algorithms have been developed in recent years for variants and generalization of the MCP. Efficient exact algorithms for the maximum vertex weighted clique problem are described in Jiang, Li, Liu, & Manya (2018); San Segundo, Furini, & Artieda (2019b), while exact algorithms for the edge-weighted case are described in San Segundo, Coniglio, Furini, & Ljubić (2019a); Shimizu, Yamaguchi, & Masuda (2018). In addition, exact algorithms for vertex and edge interdiction variants of the MCP have been described in Furini, Ljubic, Segundo, & Zhao (2021) and Furini, Ljubic, Martin, & Segundo (2019). Finally, a recent exact algorithm for the knapsack problem with conflicts is described in Coniglio, Furini, & Segundo (2021); this problem corresponds to the MCP with an additional knapsack constraint.

1.5. Methodological contributions and outline of the article

The main contribution of this paper is the development and the extensive testing of a new exact algorithm for the maximum clique problem. The algorithm, called CliSAT, is designed for hard MCP instances and is built upon the state-of-the-art procedures of the best-performing MCP algorithms in the literature. CliSAT integrates modern branching schemes with effective bounding techniques to reduce the size of the branching tree. The two state-of-art bounding mechanisms are based on graph colouring procedures and partial maximum satisfiability problems arising in the branching scheme. Starting from these cutting-edge techniques, CliSAT exploits new routines which are crucial for improving its performance to solve hard MCP instances.

Section 2 is entirely devoted to the presentation of the new algorithm. The first Section 2.1 presents the state-of-the-art incremental branching scheme of CliSAT. This n-ary scheme employs the notions of branching and pruned sets of vertices and is described in Section 2.2. In Section 2.2.1 we present the most effective state-of-the-art techniques employed by MCP algorithms to enlarge the pruned set, which are based on PMAX-SAT-P-based upper bounds. In this context, the new SATCOL procedure presented in Section 2.2.2 is the first methodological improvement of CliSAT. Its goal is to further enlarge the pruned set by combining colouring-based and PMAX-SAT-P-based upper bounds. A second important methodological contribution is the filtering phase of CliSAT described in Section 2.3. To the best of our knowledge, CliSAT is the first exact MCP algorithm to employ constraint programming and domain propagation techniques to filter vertices from the branching set, i.e., to completely remove them from branching subtrees. To this end, two *ad hoc* procedures are designed: the first one, denoted FiltCOL, is presented in Section 2.3.1; the second one, denoted FiltSAT, is presented in Section 2.3.2. After explaining the incremental upper bounds also employed by CliSAT in Section 2.4, the pseudocode for the algorithm is discussed in Section 2.5. Extensive experiments on hard benchmark MCP instances, as well as new hard instances arising from different applications, are presented in Section 3. Our computational campaign demonstrates the effectiveness of CliSAT on solving hard MCP instances and demonstrates that CliSAT outperforms the state-of-the-art MCP algorithms, for some classes of instances, by orders of magnitude. Section 4 concludes the paper summarizing the principal algorithmic improvements of CliSAT and outlines several promising lines of future research.

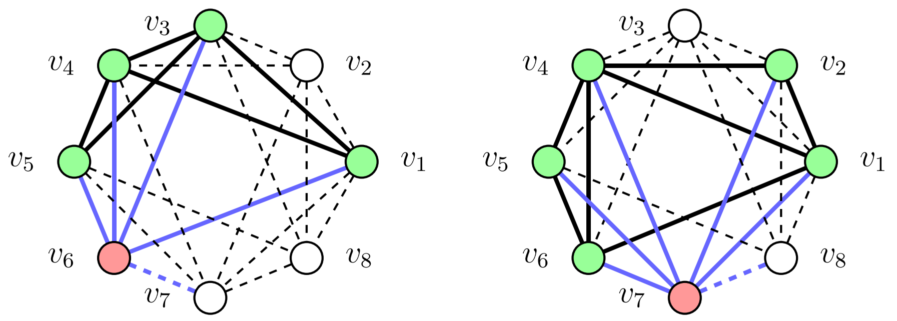

Fig. 2. The graphs $G(v_6)$ (left part) and $G(v_7)$ (right part) associated to the graph G of Fig. 1.

2. The new exact BnB algorithm: CliSAT

In this section, we describe the new BnB exact algorithm CliSAT for the MCP. CliSAT employs an n-ary branching scheme of a constructive type that iteratively builds a clique by adding one vertex at a time in a recursive fashion. We denote $\hat{K} \subseteq V(G)$ the *subproblem clique* associated to a branching node. Precisely, $\hat{K}$ contains the vertices fixed during branching and added to the subproblem clique in the nodes preceding the current one. Moreover, each branching node is associated to a *subproblem graph*, denoted $\hat{G}$. This graph contains the vertices which can be added singularly to $\hat{K}$, see Section 2.1. During its execution, CliSAT keeps track of the *incumbent solution*, denoted K_{inc}. The size $|K_{inc}|$ of the incumbent solution is denoted lb (a lower bound on $\omega(G)$). Moreover, if a larger clique is found during the branching, i.e., if the condition $|\hat{K}| > lb$ holds, both K_{inc} and lb are updated accordingly. After the execution of CliSAT, K_{inc} corresponds to a maximum clique of G and, accordingly, $lb = \omega(G)$.

The main idea of the branching scheme is to partition the set of vertices of the subproblem graph $\hat{G}$ into two subsets: *i*) the *branching set* B and *ii*) the *pruned set* P (see Section 2.2). This idea has been used in the state-of-the-art combinatorial BnB algorithms for the MCP and their variants, see, e.g., Li et al. (2018a); Li et al. (2017); Li, Liu, Jiang, Manya, & Li (2018b); San Segundo et al. (2019a, 2019b); San Segundo et al. (2013, 2015); San Segundo et al. (2016c, 2011); San Segundo & Tapia (2014). By definition of P, at least one vertex from $B = V(\hat{G}) \setminus P$ is necessary to improve the incumbent solution K_{inc}. Accordingly, branching on any of the vertices in P is not necessary in a given branching node, and the algorithm backtracks when the set B is empty. After the pruned and branching sets are determined, CliSAT carries out a $|B|$-ary branching operation, creating a branching node for every vertex in B by adding it to the current subproblem clique $\hat{K}$ (see Section 2.1).

We consider the vertex set $V(G)$ of the input graph G sorted according to a given *initial ordering* $(v_1, v_2, \dots, v_n)$, see Section 2.5 for further details on this topic. We denote $V_i(G) \subseteq V(G)$ the subset of vertices that comprises the first $i \le n$ vertices of $V(G)$; precisely: $V_i(G) = \{v_1, \dots, v_i\}$ with $i = 2, \dots, n$, and $V_1(G) = \{v_1\}$. Moreover, we denote $V(v_i, G) \subseteq V(G)$ the subset of vertices that comprises the first i vertices of $V(G)$ intersected with the neighbourhood of the vertex v_i; precisely: $V(v_i, G) = V_i(G) \cap N(v_i)$, $i = 1, \dots, n$. We then define $|V(G)|$ graphs $G(v_i)$ as the ones induced by the (non-empty) sets of vertices $V(v_i, G)$; precisely:

$$G(v_i) = G[V(v_i, G)], \qquad i = 1, \dots, |V(G)|. \tag{3}$$

Fig. 2 depicts the graphs $G(v_6)$ and $G(v_7)$ associated to the graph G of Fig. 1. The vertices v_6 and v_7 appear in red, $V(G(v_6))$ and $V(G(v_7))$ in green. The edges of both graphs are drawn as thick black straight lines. The edges connecting vertex v_7 to the vertices preceding it according to the initial ordering are coloured in blue. The edge that connects v_7 to v_8 appears as a dashed blue line, indicating that v_8 does not belong to $E(G(v_7))$. The same information is shown for $G(v_6)$. All the remaining edges in $E(G)$ are shown as dashed black lines.

2.1. The incremental branching scheme of CliSAT

The input of the branching scheme of CliSAT corresponds to the family of graphs $G(v_i)$, $i = 2, \dots, n$. CliSAT executes a BnB procedure for each one of these graphs, examining them in order. We recall that $\hat{G}$ is the subproblem graph and $\hat{K}$ is the subproblem clique associated to a branching node. In the first level of the branching tree, $\hat{G}$ corresponds to one of the graphs $G(v_i)$ and $\hat{K}$ is the singleton $\{v_i\}$. In order to determine the subproblem graphs $\hat{G}$ for subsequent child nodes, CliSAT first partitions the vertex set $V(\hat{G})$ into the pruned and branching sets P and B, i.e., $V(\hat{G}) = P \cup B$ and $B \cap P = \emptyset$.

The pruned set P is a subset of vertices of $V(\hat{G})$ respecting the following condition:

$$|\hat{K}| + \overline{\omega}\left(\hat{G}[P]\right) \le lb, \tag{4}$$

where $\overline{\omega}\left(\hat{G}[P]\right)$ is any upper bound on the clique number of $\hat{G}[P]$. The entire left hand side of (4) corresponds to an upper bound on the clique number of the graph $G[\hat{K} \cup P]$. In other words, the condition states that the graph induced by the vertices in $\hat{K} \cup P$ does not contain a clique of size larger than $lb = |K_{inc}|$. Precisely, if a set P that respects the condition (4) is found, it means that, in order to improve the incumbent solution K_{inc}, it is necessary to add to $\hat{K}$ at least one of the vertices in $V(\hat{G})$ which is not in P. Consequently, we define the branching set B as $V(\hat{G}) \setminus P$. The specific way in which the P set is constructed by CliSAT, as well as the specific upper bounds on the clique number it employs, are presented in Section 2.2.

Once the sets P and B are created, the vertices of these sets are ordered according to the initial ordering $(v_1, v_2, \dots, v_n)$ and relabelled as follows:

$$P = \left\{p_1, p_2, \dots, p_{|P|}\right\} \quad \text{and} \quad B = \left\{b_1, b_2, \dots, b_{|B|}\right\}. \tag{5}$$

CliSAT keeps track of the initial labels of the vertices $v \in V(G)$ by establishing a mapping between the vertices $p \in P$ and $b \in B$ and the corresponding vertices in $V(G)$. This is done efficiently with the help of its bitstring encoding of vertex sets in memory.

An example of the P and B sets is presented in the left part of Fig. 3. Precisely, it shows the partition of the vertex set $\{v_1, v_2, v_4, v_5, v_6\}$ of the subproblem graph $G(v_7)$ of Fig. 2 (the original graph G, we recall, is shown in Fig. 1), into the sets $P = \{v_2, v_6\}$ (grey) and $B = \{v_1, v_4, v_5\}$ (black). The edges of $G(v_7)$ are depicted as thick black lines. In this example we assume G to be the input graph, so $\hat{K} = \{v_7\}$ (v_7 is shown in red). Since $\omega(\hat{G}[\{p_1, p_2\}]) = 1$, it follows that the size lb of the incumbent solution must be equal to 2 for the condition (4) to hold. For the sake of clarity, the vertices of the sets B and P are shown according to the relabelling established by Eq. (5), i.e., $P = \{p_1, p_2\}$ and $B = \{b_1, b_2, b_3\}$. In blue,

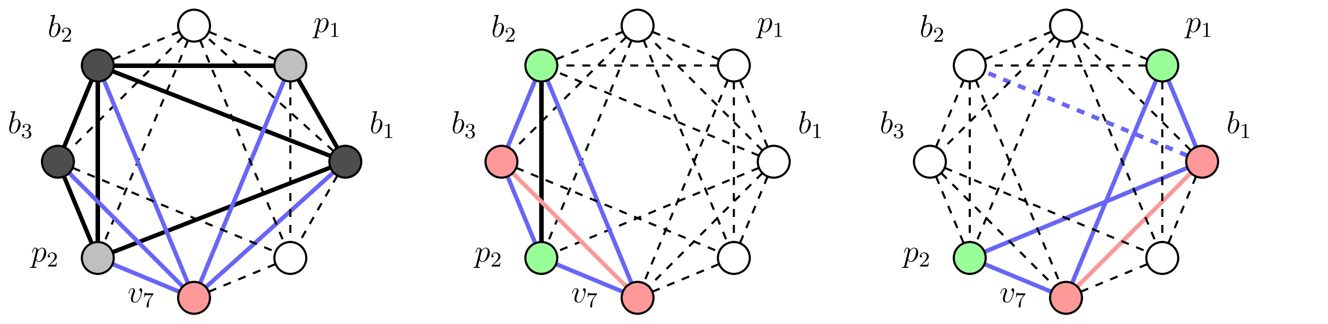

Fig. 3. Left: the pruned and branching sets P and B for the subproblem graph $G(v_7)$ of Fig. 2. Middle: the subproblem graph $\tilde{G}(b_3)$. Right: the subproblem graph $\tilde{G}(b_1)$.

the edges incident to v_7 which are involved in the branching. The vertices v_3 and v_8 are shown in white (without a label) since they do not belong to $G(v_7)$, i.e., v_3 is not adjacent to v_7 and v_8 comes after v_7 in the initial ordering. Finally, the incident edges to v_3 and v_8 are drawn as dashed lines.

To create the subproblem graphs $\hat{G}$ of the child nodes associated to branching on the vertices of the set B, we define a new family of graphs, called $\tilde{G}$. We denote $B_j(\hat{G}) \subseteq B$, the subset of vertices that comprises the first $j \le |B|$ vertices; precisely: $B_j(\hat{G}) = \{b_1, \dots, b_j\}$, with $j = 2, \dots, |B|$, and $B_1(\hat{G}) = \{b_1\}$. In addition, we denote $\hat{V}(b_j, \hat{G}) \subseteq \{P \cup B\}$ the subset of vertices that comprises the intersection between the set P, together with the first j vertices of B, with the neighbourhood of the vertex b_j; precisely: $\hat{V}(b_j, \hat{G}) = \{P \cup B_j(\hat{G})\} \cap N(b_j)$, $j = 1, \dots, |B|$. We then define $|B|$ graphs $\tilde{G}(b_j)$ as the graphs induced by the (non-empty) sets of vertices $\hat{V}(b_j, \hat{G})$; precisely:

$$\tilde{G}(b_j) = \hat{G}\big[\hat{V}(b_j, \hat{G})\big], \qquad j = 1, \dots, |B|. \tag{6}$$

The graphs $\tilde{G}(b_j)$ become the subproblem graphs $\hat{G}$ in subsequent child nodes, and $\hat{K} \cup \{b_j\}$ the associated subproblem cliques. By construction, the vertices of $\tilde{G}(b_j)$ are connected to all the vertices of $\hat{K}$. CliSAT proceeds recursively until either all the vertices in B have been explored, or B becomes the empty set.

Fig. 3 shows the graphs $\tilde{G}(b_3)$ (middle part) and $\tilde{G}(b_1)$ (right part) associated to the branching set B. As in previous figures, the set of vertices of both graphs, i.e., $\{p_2, b_2\}$ and $\{p_1, p_2\}$ respectively, are coloured in green. By branching on the vertex b_3 (resp. b_1), $\hat{K}$ becomes $\{v_7, b_3\}$ (resp. $\{v_7, b_1\}$) and its unique edge, i.e., $\{v_7, b_3\}$ (resp. $\{v_7, b_1\}$), is shown as a red line. In blue, the edges that connect the vertices of $\tilde{G}(b_3)$ and $\tilde{G}(b_1)$ to the associated $\hat{K}$. The edge set of $\tilde{G}(b_1)$ is empty, while the edge set of $\tilde{G}(b_3)$ is the singleton $\{p_2, b_2\}$ (drawn as a black line). The edge $\{b_1, b_2\}$ is drawn as a dashed blue line in the right part of the figure to indicate that b_2 does not belong to $V(\tilde{G}(b_1))$, since b_2 comes after b_1 in the new labelling (see Eq. (5)). All the remaining edges of $E(G)$ are shown as dashed black lines.

We denote this way of branching *incremental* hereafter, as opposed to the more traditional branching scheme that considers the child subproblems derived from the full neighbourhood of the vertices selected for branching, see, e.g., San Segundo et al. (2015, 2016c); San Segundo & Rodriguez-Losada (2013). Incremental branching has been employed by the recent efficient algorithms MoMC (Li et al., 2017) and IncMC2 (Li et al., 2018a), and we have adopted this strategy for our algorithm CliSAT.

2.2. Determining the pruned and branching sets

In this section we explain the techniques used by CliSAT to determine the branching and pruned sets. We recall that the branching operations of CliSAT require determining a pruned set $P \subseteq V(\hat{G})$ respecting the condition (4). One such type of pruned set, which we denote P_C, is determined by a partial κ-colouring:

$$C_\kappa(\hat{G}[P_C]) = \{I_1, I_2, \dots, I_\kappa\}, \qquad \text{where} \quad \kappa = lb - |\hat{K}|. \tag{7}$$

The value κ corresponds to an upper bound $\overline{\omega}(\hat{G}[P_C])$, and the κ-colouring is a collection of κ independent sets (Section 1.1). CliSAT employs the *greedy independent-set sequential colouring procedure* to compute a κ-colouring. This procedure is referred to as ISEQ, and was first proposed (in connection with the MCP) in San Segundo et al. (2013, 2011). We outline, in what follows, the main operations of ISEQ, and refer the reader to the aforementioned papers for further details. Given a vertex ordering $(v_1, v_2, \dots, v_{|V(\hat{G})|})$ of $V(\hat{G})$, each iteration of ISEQ builds an independent set processing the vertices in order. At each step within an iteration, and starting from the empty set, a vertex is added to the independent set under construction if it is not linked to any of its vertices. ISEQ continues iterating until κ independent sets are determined. It is worth mentioning that CliSAT implements ISEQ efficiently using a bitstring encoding of the vertex sets in memory, see (San Segundo et al., 2011).

We show the operations of ISEQ considering the subproblem graph $\hat{G}$ depicted in the left part of Fig. 4. This graph is chosen since it features a gap of one unit between its clique number and its chromatic number, i.e., $\omega(\hat{G}) = 4$ and $\chi(\hat{G}) = 5$. In addition, $\hat{G}$ is considered associated to a branching node with $|\hat{K}| = 1$ and $lb = 5$, so $\kappa = 4$ according to Eq. (7). Given the ordering $(v_1, v_2, \dots, v_7)$ of the vertex set $V(\hat{G})$, ISEQ determines the following 4 independent sets in order: $I_1 = \{v_1\}$, $I_2 = \{v_2\}$, $I_3 = \{v_3, v_4\}$ and $I_4 = \{v_5, v_6\}$. Each independent set is depicted with a different colour. The grey vertices in the right part of the figure correspond to the pruned set $P_C = \{p_1, p_2, \dots, p_6\}$, shown after the relabelling according to Eq. (5). The remaining vertex b_1 (depicted in black) becomes the branching set B. The edges incident to b_1 are represented as dashed lines.

In this example, the ISEQ procedure is not able to construct a pruned set $P_C = V(\hat{G})$, so branching is necessary. The example illustrates the limits of using a (heuristic) vertex colouring procedure to create the set P. Since $\chi(\hat{G}) = 5$, the branching node cannot be pruned even with an optimal colouring.

2.2.1. Enlarging the pruned set with PMAX-SAT-P-based upper bounds

Given a κ-colouring $C_\kappa(\hat{G}[P_C])$, defined in the previous Section 2.2, the set P_C can be enlarged by adding vertices from $B = V(\hat{G}) \setminus P_C$, one at a time, using the PMAX-SAT-P upper bound presented in Section 1.3. We describe in what follows the state-of-the-art procedure of this type employed by, e.g., Li et al. (2017). Hereafter, we denote for short $P_C \cup \{b\}$ as P_b. A vertex $b \in B$ can be added to P_C if an upper bound $\overline{\omega}(\hat{G}[P_b]) \le lb - |\hat{K}|$ can be determined. To this end, a partition of $V(\hat{G}[P_b])$ into $\kappa + 1$ colour classes is created by assigning the vertex b to a new colour class. Precisely, $C_{\kappa+1}(\hat{G}[P_b]) = C_\kappa(\hat{G}[P_C]) \cup \{b\}$ and the associated $\mathrm{PSAT}(\hat{G}[P_b], C_{\kappa+1}(\hat{G}[P_b]))$ can be used to prove that $\overline{\omega}(\hat{G}[P_b]) = \kappa$

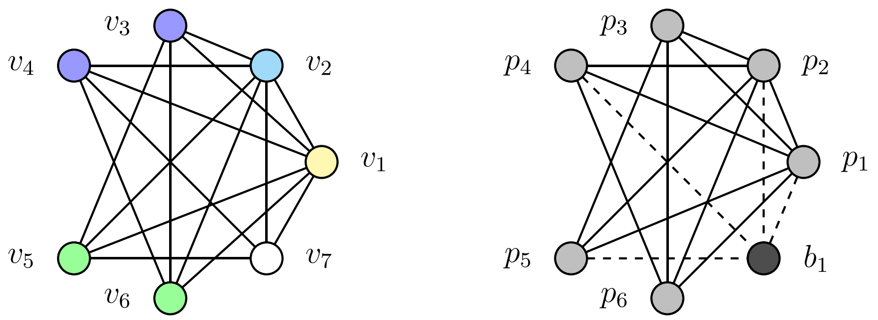

Fig. 4. To the left, a subproblem graph $\hat{G}$ associated to a branching node of CliSAT. The coloured vertices correspond to the independent sets determined by ISEQ. To the right, the corresponding pruned set $P_C = \{p_1, p_2, \ldots, p_6\}$ and the branching set $B = \{b_1\}$.

if the UP procedure determines a conflict $\mathscr{C}$ (starting from the unit clause of $\{b\}$). If a conflict $\mathscr{C}$ is found, P_b becomes the new pruned set and b is removed from the branching set B. In order to add more than one vertex from B to P_C, it is necessary to find a proper set of conflicts by iteratively building the transformed-graphs as explained in Section 1.2. The effect on the branching tree is twofold: *i*) a node is fathomed if the branching set B becomes empty; *ii*) the number of child nodes is reduced if the set P_C is enlarged, see Section 2.1.

We illustrate this technique by referring again to the subproblem graph $\hat{G}$ in Fig. 4 and $C_4(\hat{G}[P_C])$. We recall that the branching set is $B = \{b_1\}$ and the pruned set is $P_C = \{p_1, p_2, \ldots, p_6\}$, and show how to obtain $\overline{\omega}(\hat{G}[P_b]) = 4$ after determining $C_5(\hat{G}[P_b])$ as explained previously. Precisely, the associated $\text{PSAT}(\hat{G}[P_b]), C_5(\hat{G}[P_b])$ contains the 5 soft clauses: $(y_{p_1}), (y_{p_2}), (y_{p_3} \vee y_{p_4}), (y_{p_5} \vee y_{p_6}), (y_{b_1})$, and it is possible to determine a conflict by executing UP on the unit clause (y_{b_1}). The reasoning is as follows: setting to true the literal y_{b_1} removes the literals y_{p_3} and y_{p_6} (according to the hard clauses $(\bar{y}_{b_1} \vee, \bar{y}_{p_3})$ and $(\bar{y}_{b_1} \vee, \bar{y}_{p_6})$) so that both clauses $(y_{p_3} \vee y_{p_4})$ and $(y_{p_5} \vee y_{p_6})$ become unit. Finally, setting to true the literal y_{p_4} empties the other unit clause, resulting in the conflict $\{(y_{b_1}), (y_{p_3} \vee y_{p_4}), (y_{p_5} \vee y_{p_6})\}$. Consequently, $P = V(\hat{G})$, $B = \emptyset$ and the branching node is fathomed. As can be seen, the PMAX-SAT-P-based upper bounds can be stronger than the chromatic number.

2.2.2. *Enlarging the pruned set with the* SATCOL *procedure*

In what follows, we describe a new procedure, denoted SATCOL, that is employed by CliSAT to (potentially) enlarge the pruned set P_C by adding one independent set $I \subseteq B$ at a time. Each independent set is computed by one iteration of ISEQ on the vertices in B. We denote for short $P_C \cup I$ as P_I. A larger pruned set P_I is determined if a conflict $\mathscr{C}$ is found in $\text{PSAT}(\hat{G}[P_I], C_{\kappa+1}(\hat{G}[P_I]))$, where $C_{\kappa+1}(\hat{G}[P_I])$ corresponds to the κ-colouring $C_\kappa(\hat{G}[P_C])$ together with the independent set I. In such a case, $\bar{\omega}(\hat{G}[P_I]) = \kappa$, the new pruned set becomes P_I, I is removed from B, and the transformed-graph $\hat{G}[P_I](\mathscr{C})$ is computed. To find a conflict $\mathscr{C}$, SATCOL executes the procedure FL on each of the literals associated to the vertices of I attempting to prove them failed, see Section 1.3. It follows that, if $\mathscr{C}$ is found, the soft clause associated to I must be part of $\mathscr{C}$. SATCOL continues examining independent sets in B until it either fails to find a conflict, or the set $B = \varnothing$ and the branching node is fathomed. When the procedure stops, the pruned set determined in this way is denoted P_S. The transformed-graphs are necessary to ensure that the set of conflicts determined iteratively by SATCOL is a proper set of conflicts, see Section 1.2.

SATCOL presents a number of advantages with respect to prior state-of-the-art procedures that examine vertices in B individually. In the first place, SATCOL, creates a single soft clause per independent set I (if it is part of a conflict). An equivalent procedure that executes UP to find a conflict for each of the vertices in I, generates $|I|$ soft clauses and $|I|$ transformed-graphs during the reasoning. In addition, each transformation relaxes the clauses of the corresponding conflict with an additional literal, see Section 1.3, so emptying these clauses becomes more difficult in subsequent iterations. Keeping the number of soft clauses low (and of small size) is crucial for the overall efficiency of SATCOL. In the second place, SATCOL can also determine larger pruned sets, since it typically examines the vertices in B in a "better" order (according to independent sets) than the initial order. We illustrate this behaviour by means of the following example.

The left part of Fig. 5 shows a new subproblem graph $\hat{G}$ associated to a branching node, with $|\hat{K}| = 1$ and $lb = 4$, so $\kappa = 3$ according to Eq. (7). The figure also shows the 3-colouring $C_3(G[P_C])$ determined by ISEQ. The right part of the figure depicts the relabelled vertices of the pruned set $P_C = \{p_1, p_2, \ldots, p_6\}$ (grey) and the branching set $B = \{b_1, b_2, b_3\}$ (black). The edges with an endpoint in B appear dashed. SATCOL first examines the independent set $I = \{b_1, b_2\}$ from B, and the procedure FL determines a first conflict $\mathscr{C}_1 = \{(y_{p_3} \vee y_{p_4}), (y_{p_5} \vee y_{p_6}), (y_{b_1} \vee y_{b_2})\}$ in the associated $\text{PSAT}\big(G[P_I], C_4(\hat{G}[P_I])\big)$, where, we recall, $P_I = P_C \cup I$. Consequently, P_I becomes the enlarged pruned set, and I is removed from B. In the next and final iteration, SATCOL considers the remaining vertex b_3 in B and finds a second conflict $\mathscr{C}_2 = \{(y_{p_1} \vee y_{p_2}), (y_{p_3} \vee y_{p_4}, z_1), (y_{p_5} \vee y_{p_6} \vee z_2), (y_{b_3})\}$ in the associated $\text{PSAT}\big(\hat{G}(\mathscr{C}_1), C_5(\hat{G}(\mathscr{C}_1))\big)$. For completeness we provide its 5 (unsatisfiable) soft clauses: $(y_{p_1} \vee y_{p_2})$, $(y_{p_3} \vee y_{p_4} \vee z_1)$, $(y_{p_5} \vee y_{p_6} \vee z_2)$, $(y_{b_1} \vee y_{b_2} \vee z_3)$ and (y_{b_3}). The added z literals correspond to the transformed-graph $\hat{G}(\mathscr{C}_1)$. As can be seen from the conflict $\mathscr{C}_2$, the reasoning involves (besides the unit clause of b_3) the soft clauses associated to the yellow, blue and cyan colour classes in the figure.

Alternatively, we now consider the operations of the UP procedure on the vertices in B following the initial order, i.e., v_7, v_8 and v_9 (also b_1, b_3 and b_2). The first conflict determined by UP when setting y_{b_1} to true is $\mathscr{C}_1 = \{(y_{p_3} \vee y_{p_4}), (y_{p_5} \vee y_{p_6}), (y_{b_1})\}$, and a second conflict, when setting y_{b_3} to true, is $\mathscr{C}_2 = \{(y_{p_1} \vee y_{p_2})(y_{p_3} \vee y_{p_4} \vee z_1), (y_{b_1} \vee z_3), (y_{b_3})\}$. At this point, UP is unable to find a third conflict in the associated $\text{PSAT}\big((\hat{G}(\mathscr{C}_1))(\mathscr{C}_2), C_6((\hat{G}(\mathscr{C}_1))(\mathscr{C}_2))\big)$. Its 6 soft clauses are: $(y_{p_1} \vee y_{p_2} \vee z'_1)$, $(y_{p_3} \vee y_{p_4} \vee z_1 \vee z'_2)$, $(y_{p_5} \vee y_{p_6} \vee z_2)$, $(y_{b_1} \vee z_3 \vee z'_3)$, $(y_{b_3} \vee z'_4)$ and (y_{b_2}), where z and z' are the added literals corresponding to the conflicts $\mathscr{C}_1$ and $\mathscr{C}_2$ respectively. It is not difficult to see that setting y_{b_2} to true is unable to turn into unit any of the remaining (relaxed) clauses.

2.3. *The filtering phase of* CliSAT

We now describe one of the main algorithmic contributions of CliSAT. After the ISEQ procedure terminates and computes a

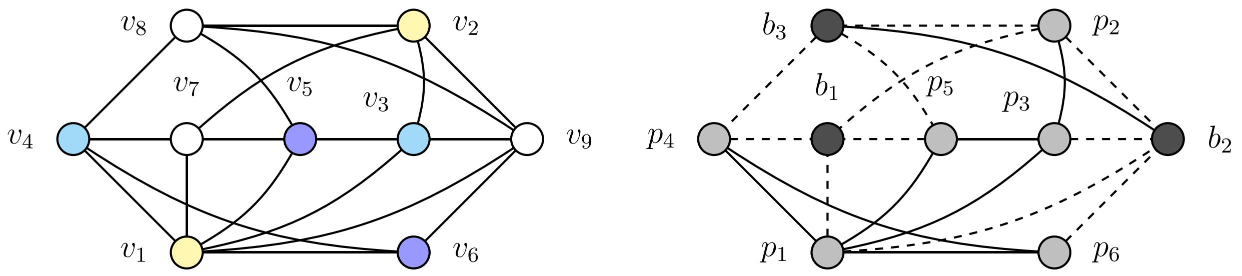

Fig. 5. To the left, a subproblem graph $\hat{G}$ associated to a branching node of CliSAT. The coloured vertices correspond to the independent sets determined by ISEQ. To the right, the corresponding pruned set $P_C = \{p_1, p_2, \ldots, p_6\}$ and branching set $B = \{b_1, b_2, b_3\}$.

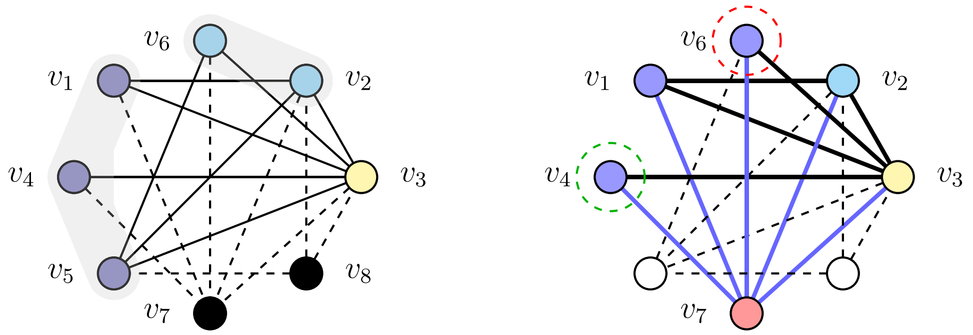

Fig. 6. On the left, a $(\kappa+1)$-partite subproblem graph $\hat{G}$ of a $(\kappa+1)$-partite branching node with $\kappa = 3$, together with a 4-colouring. The independend sets $I_1 = \{v_1, v_4, v_5\}$, $I_2 = \{v_2, v_6\}$ and $I_3 = \{v_3\}$ are the first 3 colours; the branching set $B = \{v_7, v_8\}$, in black, is the 4th colour. On the right part, the $(\kappa+1)$-partite subproblem graph $\hat{G}$, with $\kappa = 2$, resulting from branching on the vertex v_7 in the reference node shown in the left part. Encircled vertices are filtered: v_6 (red) by FiltCOL and v_4 (green) by FiltSAT. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

partial κ-colouring of a subproblem graph $\hat{G}$, i.e., determines the pruned set P_C, CliSAT attempts to find a $(\kappa+1)$-colouring of $\hat{G}$, $C_{\kappa+1}(\hat{G})$, by checking if the branching set B is an independent set. If this is the case, clearly B is the last colour class of $C_{\kappa+1}(\hat{G})$. CliSAT exploits such a colouring to further reduce the branching tree.

A branching node where CliSAT is able to determine a $(\kappa+1)$-colouring of $\hat{G}$, i.e., $\hat{G}$ is $(\kappa+1)$-partite, is called a *$(\kappa+1)$-partite branching node*. In these "special" branching nodes it is necessary to add to $\hat{K}$ exactly one vertex from each of the $\kappa+1$ colour classes in order to improve the lower bound *lb*. This is true since, in $(\kappa+1)$-partite graphs, only one vertex from each of the $\kappa+1$ colour classes can make part of a clique.

The left part of Fig. 6 shows a $(\kappa+1)$-partite subproblem graph $\hat{G}$ (associated to a $(\kappa+1)$-partite branching node) with $\kappa = 3$ (assuming $|\hat{K}| = 1$ and $lb = 4$). The ISEQ procedure determines the 3 independent sets: $I_1 = \{v_1, v_4, v_5\}$, $I_2 = \{v_2, v_6\}$ and $I_3 = \{v_3\}$ (shown with different colours in the figure). Moreover, CliSAT is able to determine a 4-colouring of $\hat{G}$, since the branching set $B = \{v_7, v_8\}$ forms an independent set. In the example, the 4-clique $\{v_1, v_2, v_3, v_7\}$ improves the *lb* value and, as can be seen, each of its vertices belongs to one of the colour classes of the 4-colouring.

CliSAT exploits a $(\kappa+1)$-partite subproblem graph $\hat{G}$ by discarding some vertices from $V(\hat{G})$ that cannot improve the incumbent solution. We call these operations of CliSAT the *filtering* phase of the algorithm. To the best of our knowledge, no state-of-the-art MCP algorithm employs filtering techniques, which are however extensively used for solving (Binary) Constraint Satisfaction Problems, see, e.g., San Segundo, Furini, & León (2022); Zhou, Kjellerstrand, & Fruhman (2015). Moreover, filtering is a core technique in state-of-the-art Constraint Programming solvers, see e.g., Rossi, Van, & Walsh (2006). Filtering vertices from $V(\hat{G})$ can have a substantial impact on the size of the branching tree, since, once a vertex is filtered, it is discarded from the entire branching subtree rooted in a $(\kappa+1)$-partite branching node. In contrast, the vertices in the pruned set P can still make part of a solution in subsequent child nodes, i.e., they can belong to future branching sets in subsequent child nodes.

The general condition to filter a vertex of a $(\kappa+1)$-subproblem graph $\hat{G}$ is to prove that it cannot make part of any clique of size $(\kappa+1)$ contained in $\hat{G}$. In practice, a necessary condition which is easier to check is that the vertex is not linked to any of the vertices from another colour class, given a $(\kappa+1)$-colouring of $\hat{G}$. To evaluate this condition efficiently, CliSAT employs the procedure FiltCOL, which is described in Section 2.3.1. An alternative sufficient condition is that the corresponding literal of the vertex in the associated PSAT$(\hat{G}, C_{\kappa+1}(\hat{G}))$ is a failed literal. This is evaluated by a second procedure FiltSAT presented in Section 2.3.2.

2.3.1. *The* FiltCOL *filtering procedure*

FiltCOL is the efficient colour-based procedure employed by CliSAT to filter vertices. To better explain the operations of FiltCOL, we first introduce some definitions and notation. We call *reference node* the $(\kappa+1)$-partite root node of a subtree, and denote $\hat{G}_R$ its associated subroblem graph. We call *reference (vertex) colouring*, $C_{\kappa+1}(\hat{G}_R)$, the κ-colouring computed by ISEQ, see Section 2.2, together with the colour class determined by the branching set B. The reference colouring $C_{\kappa+1}(\hat{G}_R)$ *induces* a colouring $C^r_\alpha(\hat{G})$, $\alpha < (\kappa+1)$, in any α-partite subproblem graph $\hat{G}$ of the subtree rooted in the reference node. Precisely, $C^r_\alpha(\hat{G})$ is obtained when the vertices of $\hat{G}$ preserve the colour class of $C_{\kappa+1}(\hat{G}_R)$. FiltCOL exploits the fact that $C^r_\alpha(\hat{G})$ differs from $C_\alpha(\hat{G})$ to filter vertices of $\hat{G}$.

We illustrate the above notions by again referring to the example of Fig. 6. Precisely, we consider its $(\kappa+1)$-partite subproblem graph, with $\kappa = 3$, to be the reference branching node $\hat{G}_R$. The coloured vertices show the reference colouring $C_4(\hat{G}_R)$: $I_1(\hat{G}_R) = \{v_1, v_4, v_5\}$, $I_2(\hat{G}_R) = \{v_2, v_6\}$, $I_3(\hat{G}_R) = \{v_3\}$ and $I_4(\hat{G}_R) = \{v_7, v_8\}$ (left part). The right part of the figure depicts the $(\kappa+1)$-partite child subproblem graph $\hat{G} = \hat{G}_R(v_7)$, with $\kappa = 2$, which results from branching on the vertex $v_7 \in G_R$ (pink). The edges of $\hat{G}$ appear in black; in blue the edges with an endpoint in v_7. The induced colouring $C^r_\alpha(\hat{G})$, $\alpha = 3$, is $I^r_1 = \{v_1, v_4\}$, $I^r_2 = \{v_2, v_6\}$ and $I^r_3 = \{v_3\}$. The coloured vertices correspond to $C_3(\hat{G})$, i.e. $I_1 = \{v_1, v_4, v_6\}$ and $I_2 = \{v_2\}$, together with the branching set $B = \{v_3\} = I_3$. As can be seen, $C^r_3(\hat{G}) \neq C_3(\hat{G})$, since the vertex v_6 (encircled in red) does not belong to the same colour class.

In a nutshell, given a reference colouring $C_{\kappa+1}(\hat{G}_R)$ and an α-partite subproblem graph $\hat{G}$, $\alpha < \kappa + 1$, FiltCOL computes the induced colouring $C^r_\alpha(\hat{G})$ while, at the same time, attempts to filter vertices of $\hat{G}$ that do not belong to its associated colour class in $C_\alpha(\hat{G})$. In detail, the operations of FiltCOL are as follows. FiltCOL processes the vertices of $\hat{G}$ according to the initial order. At the beginning of each iteration, FiltCOL starts with an empty independent set I. The first time a vertex $v \in V(\hat{G})$ is added to I, the procedure determines a correspondence between I and the independent set $I(\hat{G}_R) \in C_{\kappa+1}(\hat{G}_R)$ to which v belonged in the reference colouring, i.e., $v \in I(\hat{G}_R)$. Then, for each additional vertex $w \in V(\hat{G})$ that can enlarge I, i.e., $I \cup \{w\}$ is an independent set, FiltCOL checks if the correspondence with $I(\hat{G}_R)$ is preserved, i.e., if $w \in I(\hat{G}_R)$. If this is the case, w is added to I. Alternatively, there are two possibilities: (a) the vertex w comes after the last vertex of $I(\hat{G}_R)$ according to the initial order, in which case it is filtered from $\hat{G}$. This is possible because w is not a member of $I(\hat{G}_R)$ and is non-adjacent to all its vertices. (b) the vertex w precedes the last vertex of $I(\hat{G}_R)$, in which case w is skipped for future iterations. In this case w cannot be filtered, since it could still be linked to other vertices of $I(\hat{G}_R)$ that are also in $\hat{G}$ and which have not yet been examined. The iteration ends when all the vertices in $V(\hat{G})$ have been considered. FiltCOL continues building independent sets until the induced colouring $C^r_\alpha(\tilde{G})$ is determined for the resulting reduced graph $\tilde{G}$.

Considering the suproblem graph $\hat{G}$ of Fig. 6, FiltCOL is able to filter the vertex v_6 (encircled in red) in its first iteration with the following operations. Initially, I_1 is the empty set and vertex v_1 is added to I_1, establishing a correspondence with the independent set $I_1(\hat{G}_R) = \{v_1, v_4, v_5\}$ of the reference colouring $C_4(\hat{G}_R)$. Next, FiltCOL adds vertex v_4 to I_1 successfully, since $v_4 \in I_1(\hat{G}_R)$. Finally, v_6 is selected to enlarge I_1; however, since $v_6 \notin I_1(\hat{G}_R)$ and it has a higher index than the last vertex of $I_1(\hat{G}_R)$, i.e., v_5, it is filtered (removed) from the graph. In the remaining 2 iterations, the independent sets $I_2 = \{v_2\}$ and $I_3 = \{v_3\}$ are determined. The vertices of the reduced graph are $V(\tilde{G}) = \{v_1, v_2, v_3, v_4\}$.

Finally, we mention an important optimization related to FiltCOL. Once CliSAT executes both filtering procedures (FiltCOL and FiltSAT), and before branching, it keeps track of the vertices with the highest index from each of the α colour classes of $C^r_\alpha(\hat{G})$. These α vertices, and not the ones from the reference colouring, are used to determine if a vertex is skipped or filtered during the execution of FiltCOL in the child nodes of $\tilde{G}$. In the example, and considering only the execution of FiltCOL, the vertices stored would be v_4, v_2 and v_3, for the independent sets I_1, I_2 and I_3 respectively.

2.3.2. The FiltSAT filtering procedure

Upon termination of FiltCOL, CliSAT executes the second filtering procedure FiltSAT on the reduced subproblem $(\kappa + 1)$-partite graph $\tilde{G}$, with $\kappa + 1 = \alpha$, attempting to filter additional vertices and, ultimately, fathom the node.

FiltSAT exploits the following observations concerning the associated PSAT($\tilde{G}$, $C^r_\alpha(\tilde{G})$): i) if a failed literal is found, its associated vertex cannot be part of an α-clique in $\tilde{G}$ and the corresponding vertex can be filtered, i.e., removed from $\tilde{G}$; ii) if a conflict is found, an α-clique cannot exist in $\tilde{G}$ and, therefore, the node can be fathomed. The latter is true since, as explained in Section 1.3, a conflict found in PSAT($\tilde{G}$, $C^r_\alpha(\tilde{G})$) reduces the colour-based upper bound $\overline{w}(\tilde{G}) = \alpha$ by one unit. The vertex associated to a failed literal can be filtered for similar reasons.

FiltSAT attempts to filter every vertex in $V(\tilde{G})$ by executing the procedure FL on the associated literals in PSAT($\tilde{G}$, $C^r_\alpha(\tilde{G})$), starting from the vertices of the branching set B. Any literal proven failed by FL is filtered from $V(\tilde{G})$. The procedure ends when all the vertices in $V(\tilde{G})$ have been examined or any one of the PSAT($\tilde{G}$, $C^r_\alpha(\tilde{G})$) α clauses becomes empty, in which case the node is fathomed.

We illustrate the operations of FiltSAT by referring again to the example from Fig. 6. Precisely, we consider the reduced subproblem graph $\tilde{G}$ that results from the execution of FiltCOL, where, we recall, $V(\tilde{G}) = \{v_1, v_2, v_3, v_4\}$. FiltSAT executes FL on the literals of PSAT($\tilde{G}$, $R_3(\tilde{G})$), starting with the literal associated to the branching set y_{v_3}. In this case y_{v_3} cannot be filtered, since it is part of the solution $\{v_1, v_2, v_3\}$, but y_{v_4} is found to be a failed literal (v_4 is non-adjacent to the singleton vertex v_2 of I_2). Consequently, v_4 (encircled in green in the figure) is removed from $\tilde{G}$. The resulting graph $\tilde{G}[\{v_1, v_2, v_3\}]$ is a 3-clique, so the filtering is optimal.

Finally, it is worth mentioning that for the subproblem graph $\hat{G}_R$ of the reference node, FiltCOL is not executed since there is no reference colouring available. In this case only FiltSAT is run on PSAT($\hat{G}_R$, $C_{\kappa+1}(\hat{G}_R)$).

2.4. Incremental upper bounds

One of the advantages of the incremental branching scheme is that upper bounds on the large subproblems can be efficiently computed based on upper bounds of previously examined smaller subproblems. Such upper bounds, denoted *incremental* in Li, Fang, & Xu (2013), have been employed in recent SAT-based algorithms for the MCP, see, e.g., Li et al. (2018a, 2017), and are also employed by the new algorithm CliSAT. We briefly describe the incremental bound employed by CliSAT in what follows.

Let $\hat{G} = (\hat{V}, \hat{E})$ be a subproblem graph whose vertices are sorted according to the ordering $(\hat{v}_1, \hat{v}_2, \dots \hat{v}_{|\hat{V}|})$. We define $\mu(\hat{G}) = (\mu[\hat{v}_1], \mu[\hat{v}_2], \dots, \mu[\hat{v}_{|\hat{V}|}])$ as an ordered collection of $|\hat{V}|$ values associated to $\hat{V}$, such that each value $\mu[\hat{v}_i]$ is a valid upper bound on the clique number of the graph induced by $\hat{v}_i$ together with the set of adjacent vertices to $\hat{v}_i$ that precede it in the ordering. Precisely, this induced graph corresponds with a branching subproblem of CliSAT's incremental branching scheme. Furthermore, and owing to the hereditary nature of cliques, a valid value (upper bound) $\mu[\hat{v}_i]$, $2 < i \leq |\hat{V}|$, can always be computed in $O(|\hat{V}|)$, given the values of μ associated to the vertices in $\hat{V}$ preceding $\hat{v}_i$ ($\mu[\hat{v}_1] = 1$), as follows:

$$\mu[\hat{v}_i] = 1 + \max\left\{\mu[u] : u \in \hat{V}_{i-1}(\hat{G}), (u, \hat{v}_i) \in \hat{E}\right\}, \quad i = 2, \dots |\hat{V}|, \tag{8}$$

where $\hat{V}_{i-1}(\hat{G})$ is the set of vertices that precede $\hat{v}_i$ in $\hat{V}$.

The values of $\mu(\hat{G})$ are dynamically updated during the execution of CliSAT according to Eq. (8), taking into account as well the size of the incumbent solution obtained after examining the corresponding subproblem. They provide a computationally cheap upper bounding condition for reducing the number of branching child nodes for a given a branching set B. This condition is evaluated just after the child subproblem is determined, and before the bounding techniques described in the previous sections are executed. In practice, CliSAT considers the vertex ordering (5), i.e., vertices in the pruned set P first, followed by the vertices in the branching set B, to determine the values of μ in every node, as in Li et al. (2017). The specific details concerning how the μ values are employed by CliSAT to prune the branching tree are described in Algorithm 1.

Algorithm 1: CliSAT algorithm for the maximum clique problem.

```
Input: A simple graph $G = (V, E)$
Output: A maximum clique $K$ in $G$ ($lb = |K| = \omega(G)$)
1  $(v_1, v_2, \dots, v_n) \leftarrow$ Sort($V$)
2  $K \leftarrow$ FindClique($V$), $lb \leftarrow |K|$
3  Initialize $\mu(G)$
4  for $i \leftarrow |K| + 1$ to $n$ do
5      $\hat{V} \leftarrow \{v \in V_{i-1}(G) : \{v, v_i\} \in E\}$        ▷ child subproblem
6      $P \leftarrow \{\hat{v}_1, \hat{v}_2, \dots, \hat{v}_{|K|}\}$
7      FindMaxClique($G[\hat{V}]$, $\{v_i\}$, $P$, $\mu(G)$)
8      $\mu[v_i] \leftarrow lb$

9  FindMaxClique($\hat{G}$, $\hat{K}$, $P$, $\mu$)
10 $\hat{\mu} \leftarrow \{\mu[v] : v \in P\}$
11 $B = \{b_1, \dots, b_{|B|}\} \leftarrow \hat{V} \setminus P$
12 for $l \leftarrow 1$ to $|B|$ do
13     Compute $\hat{\mu}[b_l]$                          ▷ see Equation (8)
14     if $\hat{\mu}[b_l] + |\hat{K}| \leq lb$           ▷ skip the $l$-th subproblem
15     then
16         $P \leftarrow P \cup \{b_l\}$ and $B \leftarrow B \setminus \{b_l\}$
17     else
18         $\tilde{V} \leftarrow \{P \cap N(b_l)\} \cup \{b_j \in B : j < l, \{b_j, b_l\} \in \hat{E}\}$
           ▷ child subproblem
19         if $\tilde{V} = \emptyset$ then
20             if $|\hat{K}| > lb$ then $lb \leftarrow |\hat{K}|$ and $K \leftarrow \hat{K}$
21             return
22         if the current branching node is $(\kappa + 1)$-partite
           ▷ Section 2.3
23         then
24             $(\tilde{P}, \tilde{B}) \leftarrow$ FiltCOL ($\tilde{V}$)         ▷ Section 2.3.1
25             $(\tilde{P}, \tilde{B}) \leftarrow$ FiltSAT ($\tilde{P}, \tilde{B}$)   ▷ Section 2.3.2
26         else
27             $(\tilde{P}, \tilde{B}) \leftarrow$ SATCOL ($\tilde{V}$);        ▷ Section 2.2.2
28         if $\tilde{B} \neq \emptyset$ then
29             FindMaxClique($\hat{G}[\tilde{V}]$, $\hat{K} \cup \{b_l\}$, $\tilde{P}$, $\hat{\mu}$)
30     $\hat{\mu}[b_l] \leftarrow \min\{\hat{\mu}[b_l], lb - |\hat{K}|\}$
```

2.5. The algorithm CliSAT

The algorithm CliSAT produces a branching tree that interleaves the bounding procedures SATCOL, FiltCOL and FiltSAT presented in the previous sections with the general branching scheme described in Section 2.1. Pseudocode for CliSAT is presented in Algorithm 1. In the pseudocode, the steps (1–3) correspond to the initial preprocessing phase of CliSAT, which is covered at the end of this section. Branching takes place in the recursive call to FindMaxClique (step 7), and is described in what follows.

At the end of its preprocessing phase, CliSAT branches on the vertices in V (according to the initial order established in step 1) starting from the $|K|$-th + 1 vertex (the first $lb = |K|$ vertices are skipped, since they cannot improve the initial clique by themselves). Then, for each vertex $v_i \in V$, $i = lb + 1, \dots, n$, selected for branching, CliSAT determines the set of vertices $\hat{V}$ of the child subproblem, i.e., the adjacent vertices to v_i that precede it in V (step 5), computes a trivial Pruned Set P that comprises the first lb vertices in $\hat{V}$ (step 6) and calls the recursive procedure FindMaxClique to explore $G(\hat{V})$ (step 7). On backtracking, the value of μ corresponding to the branched vertex v_i is updated with lb (step 8).

Inside a branching node, the sets P and B always store the vertices according to their index number, the predetermined initial order of the vertices in G. This operation is done efficiently with the help of bitsets. The first task executed by FindMaxClique is to compute the values of $\hat{\mu}$ for the vertices in P. Since these vertices will not be branched on, preliminary tests established that the best compromise between efficiency and pruning ability was to give them the corresponding values in the father node (step 10). This efficient *inheritance* (originally described in Li et al. (2017), to the best of our knowledge, in combination with incremental branching) is possible because, as stated previously, the order of the vertices in P is preserved in every node. Since child subproblems are always subsets of father subproblems, the upper bound values concerning the latter are also valid for the former. In contrast, the values of $\hat{\mu}$ for each branching vertex in B are computed in step 13 according to Eq. (8).

Pruning with the (upper bound) values of $\hat{\mu}$ occurs prior to the computation of each new child subproblem in step 14. If the pruning is successful, the vertex $b_l \in B$, $l = 1 \dots |B|$ is added to P and removed from B (and the corresponding subproblem is not explored); otherwise, the child subproblem is determined in step 18. If the latter corresponds to a leaf node that improves the current solution, the incumbent clique is updated in step 20; else the child node is either processed according to the procedure SATCOL, or, in case the node is $(\kappa + 1)$-partite, according to the filtering procedures FiltCOL and FiltSAT (steps 22–27), see the Sections 2.2.2, 2.3.1 and 2.3.2 respectively. Finally, if at this point the child node has not been fathomed, CliSAT branches to the child subproblem in a recursive fashion (step 29). Worst-case complexity analysis of the filtering and SATCOL phases of CliSAT, which are complex procedures, could be the object of future research.

We conclude this section by presenting the initial preprocessing phase of CliSAT. This phase comprises the following 3 operations executed in the first 3 steps of the algorithm: (i) an initial ordering of the vertices (step 1); (ii) a clique is computed heuristically (step 2); (iii) the collection of upper bound values μ is initialized (step 3). We describe the three operations in the following.

It is well established in the literature that the initial ordering of vertices plays an important role in BnB algorithms for the MCP, see, e.g. Maslov, Batsyn, & Pardalos (2014). More precisely, state-of-the-art exact MCP algorithms employ two different orderings: (i) *degenerate* degree-based (DEG-SORT) and (ii) colour-based (COLOR-SORT). The term *degenerate* in (i) refers to the fact that the sorting criterium (vertex degree) is dynamic, i.e., it is recomputed on the remaining unsorted vertices each time a vertex is selected. These two orderings are briefly presented in what follows; for a more in-depth analysis we refer the interested reader to San Segundo, Lopez, Batsyn, Nikolaev, & Pardalos (2016a).

The most frequently employed ordering is DEG-SORT, which can be traced back to Carraghan & Pardalos (1990). In its basic form, the degree-based ordering $(v_1, v_2, \dots, v_n)$ is such that v_n is a vertex with smallest degree in G, v_{n-1} is a vertex with smallest degree in the induced graph $G[V_{n-1}(G)]$, and so on. A successful colour-based ordering for the MCP was first described in Li et al. (2013) to the best of our knowledge.

COLOR-SORT partitions V into k independent sets $\{I_1, I_2, \dots, I_k\}$, such that I_1 is a maximum independent set in G, I_2 is a maximum independent set in the induced subgraph $G[V \setminus I_1]$, and so on. Moreover, CliSAT considers the following order within each independent set I: for any pair of vertices

$(v_i, v_j) \in I$ such that $1 \le i < j \le n$ (v_i precedes v_j), the degree of v_i is greater or equal to the degree of v_j. It is worth noting that finding partitions of maximum independent sets is as computationally hard as the original problem. However, hard MCP instances are normally dense or very dense, and, therefore, determining maximum independents sets is expected to be easy. In practice, to determine COLOR-SORT we execute CliSAT on the complement graph and search for maximum cliques with a fixed time limit.

Depending on the actual instance, CliSAT employs either DEG-SORT or COLOR-SORT. Extensive preliminary tests carried out showed that, in the general case, COLOR-SORT improves the efficiency of CliSAT when the size k of the independent set partition provides a tight upper bound on $\omega(G)$. If this is not the case, DEG-SORT is to be preferred. This is consistent with the results found in the literature, see, e.g., San Segundo et al. (2016a). The procedure referred to as Sort(V) in step 1 of the pseudocode, selects the concrete ordering and is adapted from San Segundo et al. (2016a); we refer the reader to the latter for further details. When Sort(V) terminates, the adjacency matrix of G is processed so that the vertex order becomes the index order of the vertices in G, i.e., we compute an isomorphic graph to G which becomes the new input graph to CliSAT. This optimization was originally described in the bitstring algorithm (San Segundo et al., 2011) to the best of our knowledge.

To compute an initial clique K_{inc} (step 2 of the initial preprocessing phase), CliSAT executes the multi-start tabu search heuristic AMTS (Wu & Hao, 2013) with a reduced time limit (see the computational section Section 3), and sets lb accordingly, i.e., $lb = |K_{inc}|$. Finally, $\mu(G)$ is initialized in step 3 according to Eq. (8) ($\mu[v_1] = 1$). In addition, the first $|K_{inc}|$ values of μ are bounded by lb, and its remaining values are bounded by the size k of the independent set partition determined by COLOR-SORT.

3. Computationals

In this section we assess the computational performance of the new BnB algorithm CliSAT presented in this work. The goal of this computational study is twofold: *i*) to evaluate the performance of CliSAT with respect to its main components, covered in Section 3.2; *ii*) to compare CliSAT against the state-of-art algorithms in the literature, covered in Sections 3.3 and 3.4.

3.1. Experimental setting and testbed of instances

All the experiments have been carried out on a 20-core Intel(R) Xeon(R) CPU E5-2690 v2@3.00 gigahertz, disposing of 128 gigabyte of main memory and running a 64 bit Linux operating system. The source code was compiled with gcc 5.4.0 and the -O3 optimization flag. The configuration parameters of CliSAT are as follows. In all the runs, during CliSAT's initial preprocessing phase the heuristic AMTS is executed with a time limit of 0.05 seconds to determine an initial large clique. Since our testbed contains a wide variety of instances whose solution time varies from seconds to days, we employ this very short and fixed time limit so as not to degrade the performance of CliSAT on the easy instances. Moreover, the CliSAT algorithm is typically able to find good quality heuristic solutions at the early stages of the search, thus limiting the impact of initial solutions. According to our extensive preliminary results, running AMTS for 0.05 seconds during the initial preprocessing phase was significant for a limited number of instances of the DIMACS dataset, such as, for example, the subset *brock800* or the instance *keller5*. In the case of the BHOSLIB dataset, AMTS only proved useful in those few cases where it found a maximum clique, whereas CliSAT by itself was unable to do so under 0.05 seconds. Finally, fixing this very short time limit allows us to focus on the impact of the new components of CliSAT. A systematic analysis of the impact of the initial solutions on the overall performance of CliSAT could be the object of future research.

The time limit to determine each maximum independent set required by COLOR-SORT is also fixed to 0.05 seconds (see the description of the initial preprocessing phase of CliSAT at the end of the previous Section 2.5 for an explanation of this threshold).

For the tests, we have considered a testbed of 791 instances which comprises 501 structured instances (see Table 1) and 290 uniform random instances (see Table 4). The choice of orders and densities of the 290 random instances is in accordance with similar tests that can be found in the literature for exact MCP algorithms, see, e.g., Table 2 of (San Segundo et al., 2016c). The 501 structured instances can be divided into the following 4 categories (datasets): (*i*) the 86 instances from the 2nd DIMACS Challenge (http://dimacs.rutgers.edu/programs/challenge/); (*ii*) the 41 instances from the BHOSLIB dataset; (*iii*) 223 representative instances derived from binary constraint satisfaction problems (BCSPs), which we denote the CSPLIB dataset and (*iv*) 151 hard MCP instances taken from different sources, hereafter the miscellaneous dataset MISCLIB. The 501 structured instances are publicly available in the github repository https://github.com/psanse/CliSAT. We consider this extended dataset, wrt typical clique benchmarks employed elsewhere, an additional contribution of this work, and hope it will stimulate further research in this field. Moreover, the repository also contains a linux release of CliSAT and additional comparison performance results to those reported in this section.

The DIMACS and BHOSLIB datasets are consistently employed in the literature to test exact MCP algorithms. The instances of our CSPLIB dataset are obtained as follows: vertices represent specific values of variable domains, and there is an edge between two vertices if the corresponding 2 values are compatible according to the constraints imposed on the original BCSP. It is worth mentioning that all the instances from the BHOSLIB dataset also derive from BCSPs, which has motivated the choice of the CSPLIB. Last of all, the miscellaneous dataset MISCLIB comprises 4 families: (*i*) the 20 instances of the recent evil dataset (Szabó & Zaválnij, 2019), claimed to be harder than the BHOSLIB dataset ; (*ii*) 3 instances derived from monotone matrices (mon) (Szabó, 2013); (*iii*) 78 instances (denoted vc) derived from the 200 vertex cover problems from the PACE Challenge (Track 1a) (https://pacechallenge.org/2019/vc/). Precisely, we have included those instances from the PACE Challenge with less than 8,000 vertices; (*iv*) the first 50 (out of more than 49,150) instances of the Gordon Royle's 17 -clue Sudoku collection (https://github.com/t-dillon/tdoku/blob/master/data.zip), and referred to as sud in the following. In the sud instances, vertices represent a specific number and square of the 9×9 Sudoku grid, and there is an edge between two vertices if the corresponding (number, square) pairs are compatible according to the rules of the game. All the instances of sud have 729 vertices and a unique maximum clique of order 81.

Table 1 reports information related to the number of instances (#inst.), order ($|V|$) and density ($d(G)$) of our 501 dataset of structured instances classified by categories (datasets) and aggregated by families. As can be seen from the table, the average density of the different families of instances is high, i.e. with the exception of the family c-fat, the smallest average density is 0.49; moreover, in 15 out of the 24 families, the average density is greater than 0.75. It is also worth noting that some of the instances were not solved by any of the algorithms tested and remain open.

3.2. Empirical analysis of the main components of CliSAT

In this section, we evaluate the impact on the performance of CliSAT of its main components over the entire 501 structured instance dataset, considering a time limit of 1800 seconds.

Table 1
Information on the dataset of 501 structured and dense MCP instances considered in this work.

			Number of vertices $\|V\|$			Edge density $d(G)$		
Category	Family	# instances	min	avg	max	min	avg	max
DIMACS	brock	12	200	466.7	800	0.50	0.67	0.75
	C	7	125	1,410.7	4,000	0.50	0.79	0.90
	c-fat	7	200	371.4	500	0.04	0.19	0.43
	dsjc	7	250	678.6	1,000	0.10	0.50	0.90
	gen	5	200	320.0	400	0.90	0.90	0.90
	ham	6	64	448.0	1,024	0.35	0.78	0.99
	john	5	28	208.8	496	0.56	0.78	0.91
	keller	3	171	1,436.0	3,361	0.65	0.74	0.82
	MANN	4	45	1,194.8	3,321	0.93	0.98	0.999
	p_hat	15	300	800.0	1,500	0.24	0.49	0.75
	san	15	200	346.7	1,000	0.50	0.73	0.90
		86						
CSPLIB	aim	48	472	909.8	2,016	0.91	0.93	0.96
	B	25	529	627.0	729	0.72	0.74	0.75
	comp	25	330	616.4	1,050	0.88	0.93	0.96
	D	25	320	1,824.0	7,200	0.86	0.87	0.89
	ehi	25	2,079	2,144.5	2,205	0.95	0.95	0.95
	geom	25	1,000	1,000.0	1,000	0.88	0.90	0.91
	lat	25	613	3,023.0	6,961	0.97	0.98	0.99
	RB2	25	450	773.2	1,150	0.82	0.85	0.88
		223						
MISCLIB	evil	20	120	182.6	253	0.87	0.94	0.98
	mon	3	343	528.0	729	0.79	0.81	0.84
	vc	78	153	1,501.8	7,400	0.82	0.96	0.9995
	sud	50	729	729.0	729	0.63	0.63	0.63
		151						
BHOSLIB	frb	41	450	1,086.1	4,000	0.82	0.87	0.93

Table 2
Analysis of the main components of the algorithm CliSAT over the entire dataset of 501 instances. The time limit is set to 1800 seconds.

				CliSAT					
		CliSAT		No SATCOL		No FiltCOL\FiltSAT		No both	
Categ.	#inst.	#opt	time [seconds]	#opt	time [seconds]	#opt	time [seconds]	#opt	time [seconds]
DIMACS	86	**72**	59.8	67	63.3	68	50.8	67	58.0
CSPLIB	223	**213**	72.3	175	82.0	147	86.7	144	85.6
MISCLIB	151	**138**	52.5	130	61.0	77	117.0	74	84.2
BHOSLIB	41	**30**	209.2	**30**	227.2	20	76.9	20	76.6
Total	501	**453**	73.3	402	82.9	312	85.8	305	78.6

Table 2 summarizes the results obtained. The table shows the number of instances solved to proven optimality (#opt) grouped by categories, and the average time (in seconds) spent by the different algorithmic variants to prove optimality, i.e., those instances in which the time limit was reached are not included. Specifically, we report performance results for the following procedures: (*i*) the algorithm CliSAT; (*ii*) CliSAT without the SATCOL procedure (described in Section 2.2.2); (*iii*) CliSAT without the filtering procedures FiltCOL (Section 2.3.1) and FiltSAT (Section 2.3.2) for $(\kappa+1)$-partite branching nodes, and (*iv*) CliSAT without both components.

As shown in Table 2, CliSAT solves to proven optimality 453 instances of the 501 dataset within the time limit, and removing one or both of the components leads to a degradation in its performance. Specifically, if the filtering component, i.e., the procedures FiltCOL and FiltSAT, is removed, 312 instances are solved, whereas if the component SATCOL is removed, 407 instances are solved. This indicates that the filtering component has more impact on the overall performance of CliSAT than its counterpart SATCOL. In addition, the algorithm performs the worst when both components are removed, solving only 305 instances. This provides clear empirical evidence that the main source of efficiency of CliSAT is the combined effect of both components.

It is worth noting that, according to the reported results, the impact of the filtering component of CliSAT is smaller on the DIMACS dataset than on the other 3 datasets. A possible explanation for this fact is that in the DIMACS dataset, the average gap between the clique number and the colour-based bound is larger than in the other 3 datasets. Consequently, the probability of finding $(\kappa+1)$-partite nodes in the shallow levels of the branching tree is lower. In extensive preliminary tests we have observed that the incremental nature of CliSAT's branching scheme favours the appearance of $(\kappa+1)$-partite nodes, and might be one explanation for the "good" overall performance of CliSAT when combined with the FiltCOL\FiltSAT component.

We also report the performance profile, see Dolan & Moré (2002). The performance profile is constructed in the following way. We compute the normalized time τ as the ratio of the computing time of each algorithm (which is ∞ if the instance is not solved to optimality within the time limit, here set to 1800 seconds) over the minimum computing time spent the tested algorithms. For each value of τ on the horizontal axis, the vertical axis reports the percentage of instances for which the corresponding algorithm spent at most τ times the computing time of the fastest algorithm (see Dolan & Moré, 2002 for further details). The chart interpretation at both ends of the horizontal axis is as follows. At

Table 3
Performance comparison of the algorithms CliSAT and MoMC for the entire strucured dataset of 501 instances: *i*) 86 DIMACS instances (time limit 15 days); *ii*) 223 CSPLIB instances (time limit 1800 seconds); *iii*) 151 MISCLIB instances (time limit 15 days for the evil, mon and vc families, 1800 seconds for the sud family) ; *iv*) 41 BHOSLIB instances (time limit 15 days).

			CliSAT			MoMC		
				Time [seconds]			Time [seconds]	
Categ.	Family	#inst.	#opt	avg.	max.	#opt	avg.	max.
DIMACS	brock	12	12	827.7	3,652.2	12	**500.7**	1,867.1
	C	7	3	**9,263.2**	27,660.5	3	11,689.1	34,936.7
	c-fat	7	7	0.05	0.1	7	**0.03**	0.1
	dsjc	7	5	**17.5**	86.2	5	22.9	112.3
	gen	5	5	**0.1**	0.1	5	0.4	1.1
	ham	6	5	**0.1**	0.1	5	5.0	24.5
	john	5	3	0.10	0.2	3	**0.03**	0.1
	keller	3	2	**11.6**	23.1	2	78.9	157.7
	MANN	4	4	**90,361.5**	361,440.5	4	242,661.6	970,637.5
	p_hat	15	14	1,215.8	16,289.6	14	**1,082.2**	14,453.0
	san	15	15	**2.8**	39.1	15	3.7	51.5
	86	75			75			
CSPLIB	aim	48	**47**	133.5	1,459.7	20	215.8	1,620.4
	B	25	25	**33.3**	146.5	25	82.1	349.0
	comp	25	**25**	0.1	0.2	22	0.5	1.1
	D	25	**25**	51.7	858.4	22	18.3	191.5
	ehi	25	**25**	18.9	139.2	24	171.3	273.5
	geom	25	25	**0.4**	4.9	25	2.7	12.0
	lat	25	**16**	65.8	528.1	8	49.3	158.2
	RB2	25	**24**	153.0	1,514.6	22	64.4	662.5
	223	212			168			
MISCLIB	evil	20	**20**	4,176.4	54,828.4	17	13,721.2	165,792.0
	mon	3	**3**	22,722.8	68,019.1	2	209.3	415.9
	vc	78	**76**	7,262.6	406,912.1	63	62.9	1,463.0
	sud	50	**50**	1.5	16.9	1	1.6	1.6
	151	149			83			
BHOSLIB	frb30-15	5	5	**0.1**	0.1	5	0.3	0.4
	frb35-17	5	5	**0.2**	0.4	5	1.0	1.5
	frb40-19	5	5	**1.0**	2.9	5	3.3	5.8
	frb45-21	5	5	**31.1**	100.7	5	76.8	168.1
	frb50-23	5	5	**612.8**	2,534.4	5	1,400.6	5,932.0
	frb53-24	5	5	**861.6**	1,557.3	5	1,758.0	3,415.1
	frb56-25	5	5	**19,907.5**	53,642.3	5	45,209.2	121,379.9
	frb59-26	5	5	**73,261.5**	108,058.4	5	146,109.4	257,586.4
	frb100-40	1	0	tl	–	0	tl	–
		41	40			40		
Total	501	476			366			

$\tau = 1$, the value of the curves is equal to the percentage of instances which the corresponding algorithm solves to optimality in less time. At the right-end, i.e., the largest value of τ, each curve corresponds to the percentage of instances solved to optimality by the specific algorithm. Consequently, in the performance profile the best performance is achieved by those algorithms whose curves appear highest in the chart, "wrapping" the other curves.

According to Fig. 7, the best performing algorithm is, clearly, CliSAT, which is the fastest in slightly less than 90% of the instances (left-end of the figure), and also solves the largest amount, i.e., slightly over 90% (as shown by the intersection of its curve in the right-end) within the time limit. The performances of the other algorithmic variants are consistent with the results reported in Table 2. Precisely, the filtering variant (no SATCOL) performs second best, solving to optimality slightly over 80% of the instances, followed by the variant without filtering, which solves slightly over 62% of the instances. Finally, the performance profile of the variant without both components is dominated by the other 3 curves, which appear on top in the chart.

We end the section reporting the impact of the components of CliSAT on the number of the recursive calls (steps) made by the algorithm during branching. The reduction of the number of steps is significant, and can go up to an order of magnitude. The average number of steps of CliSAT for this subset of instances is approximately 1.7×10^5. On the other hand, the variant "no FiltCOL\FiltSAT" explores 2.0×10^6 steps, the variant "no SATCOL" explores 9.4×10^6 steps and, finally, the variant "no both" explores 1.8×10^7 steps. To avoid any distortion in the data, due to time limits, we consider in the above reported averages only those instances that are solved to proven optimality by all the variants of CliSAT.

3.3. Comparison between CliSAT and MoMC over structured instances

We compare in detail the performance of CliSAT against the exact combinatorial BnB algorithm MoMC (Li et al., 2017). MoMC is the most recent and successful SAT-based algorithm for the MCP to the best of our knowledge. In this section, we consider for comparison purposes the 501 structured instance dataset described in Section 3.1. The results obtained are reported in the Table 3. The tables show aggregated results by families for the categories (datasets) DIMACS, CSPLIB, MISCLIB and BHOSLIB respectively, reporting the number of instances solved (#opt) and the average and maximum times in seconds spent by the 2 algorithms to solve the instances to proven optimality.

We fixed the time limit to 15 days for families of instances which either had been consistently employed in the recent literature for similar purposes, i.e., the DIMACS and BHOSLIB datasets, or we considered relevant, i.e., evil, mon and vc. For

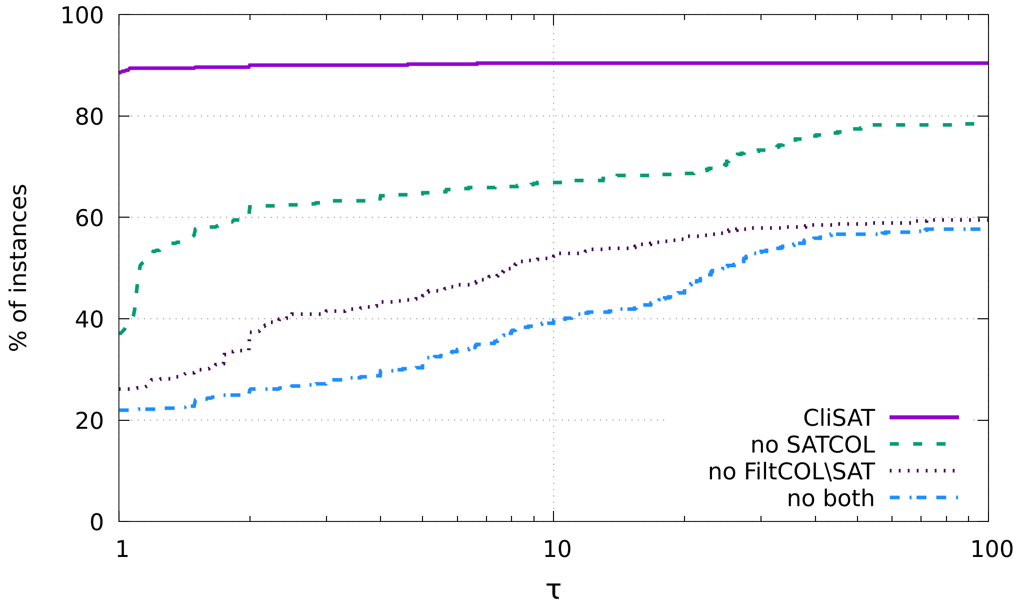

Fig. 7. Performance profile of the main components of the algorithm CliSAT over the entire dataset of 501 instances. The time limit was fixed at 1800 seconds.

Table 4
Comparison between the algorithms CliSAT, LMC and MoMC over 290 uniform random graphs. In all the instances with more than 10,000 vertices, MoMC reported a memory problem (indicated by "_").

			Clique number $\omega(G)$			Average time [seconds]		
$\|V\|$	$d(G)$	#inst.	min.	av.	max.	CliSAT	LMC	MoMC
150	0.7	10	16	16.5	17	0.06	**0.02**	0.05
150	0.8	10	22	22.9	24	0.08	**0.05**	0.07
150	0.9	10	35	37.1	41	0.10	**0.09**	0.10
150	0.95	10	53	54.5	57	0.05	**0.01**	**0.01**
200	0.7	10	18	18.2	19	**0.12**	0.13	0.17
200	0.8	10	24	25.1	26	**0.63**	0.88	1.11
200	0.9	10	39	40.7	42	**3.56**	6.11	3.73
200	0.95	10	60	62.3	64	**0.40**	0.79	0.49
200	0.98	10	91	94.7	98	0.05	**0.02**	**0.02**
300	0.6	10	15	15.4	16	**0.25**	0.44	0.37
300	0.7	10	20	20.2	21	**2.02**	3.52	5.03
300	0.8	10	28	28.5	30	**41.32**	58.44	98.18
500	0.4	10	10	10.7	11	**0.13**	0.19	0.23
500	0.5	10	13	13.3	14	**0.67**	1.23	1.36
500	0.6	10	17	17.0	17	**9.09**	17.20	12.86
500	0.7	10	22	22.4	23	**335.06**	448.65	728.74
500	0.99	10	261	266.2	276	**0.06**	0.68	0.65
1000	0.2	10	7	7.5	8	**0.09**	0.10	0.17
1000	0.3	10	9	9.2	10	**0.40**	0.61	0.78
1000	0.4	10	12	12.0	12	**3.54**	6.27	5.15
1000	0.5	10	15	15.0	15	**80.13**	148.89	113.47
3000	0.1	10	6	6.4	7	**0.31**	0.42	0.98
3000	0.2	10	9	9.0	9	**4.41**	4.90	5.19
5000	0.1	10	7	7.0	7	**1.31**	2.11	3.18
5000	0.2	10	9	9.1	10	62.87	**51.18**	59.90
10,000	0.1	10	7	7.4	8	21.62	**19.39**	21.00
15,000	0.1	10	8	8.0	8	126.10	**74.81**	–
20,000	0.1	10	8	8.0	8	549.11	**221.58**	–
30,000	0.1	10	8	8.0	8	5,565.21	**1313.10**	–

example, the evil family is interesting because, as mentioned in Section 3.1, the creators claim it to be harder than BHOSLIB. For the CSPLIB dataset and the Sudoku family (sud), the time limit was reduced to 1800 seconds for practical purposes.

According to Table 3, CliSAT consistently outperforms MoMC solving more instances than its counterpart or spending less time, on average, when both algorithms solve the same number of instances to proven optimality. It is worth mentioning that there is no family in which MoMC solves more instances than CliSAT. In detail, CliSAT solves within the time limit 75 DIMACS instances out of a possible 86, 212 CSPLIB instances out of a possible 223, 149 MISCLIB instances out of a possible 151 and 40 out of the 41 instances of the BHOSLIB dataset. Overall, it is able to solve to proven optimality 476 instances out of a possible 501. In contrast, MoMC solves the same number of instances as CliSAT from the DIMACS and BHOSLIB datasets, but its performance drops to 168 instances from MISCLIB, and 83 from CSPLIB. Altogether, MoMC manages to solve within the time limit

366 instances out of the possible 501, i.e., 110 instances less than CliSAT.

One possible explanation for the difference in the number of instances solved from the CSPLIB dataset, and also the evil and sud families (MISCLIB), is the pruning ability of the new filtering component of CliSAT. This is because, in these instances, there is more probability of finding $(\kappa + 1)$-partite branching nodes in the shallow levels of the tree.

Also from the reported results in the Table, and comparing instances of similar size, the evil family is much harder to solve than the BHOSLIB dataset for MoMC, in accordance with what is claimed in the literature, see Szabó & Zaválnij (2019). However this is not the case for CliSAT, which manages to solve the 20 evil instances within the time limit. CliSAT also outperforms MoMC in the BHOSLIB dataset, e.g., it is more than twice as fast in the instances from the family frb59-26. With respect to the 50 Sudoku instances, the difference in performance in favour of CliSAT is even more acute, solving all the instances in an average time of 1.5 seconds, whereas MoMC is able to solve just one instance. In addition, Table 2 clearly shows that the filtering component of CliSAT is the major cause of its good performance on the sud family.

To end the section, we highlight that even though CliSAT outperforms MoMC in most of the families, there are exceptions. Specifically, in the (hard) families brock and p_hat from the DIMACS dataset, MoMC significantly outperforms CliSAT. The brock family is very sensitive to initial pre-processing, so it is difficult to relate the poor performance of CliSAT on this family with its algorithmic components. In the case of p_hat, the computing performance of CliSAT is reasonably close to MoMC. The other 2 cases in which MoMC outperforms CliSAT are the family c-fat and 3 instances of the family john. These are easy instances solved by both algorithms in less than 1 second, and therefore not representative enough, in our opinion, to draw any conclusion.

3.4. Comparison between CliSAT, MoMC and LMC over uniform random instances

We also compare the algorithms CliSAT and MoMC, together with the algorithm LMC (Jiang et al., 2016) designed for sparse graphs, over a set of 290 Erdös-Rényi random $G(n, p)$ graphs of different sizes ($n = |V|$ ranging from 150 up to 30,000) and edge densities (see Table 4 for the specific density values tested). These uniform random graphs are created according to a given probability (equal to the desired edge density value) of existence of an edge between any pair of vertices, and are commonly used for testing clique algorithms; precisely, the testbed employed is the same as the one used in San Segundo et al. (2016c), extended with the families $G(20,000, 0.1)$ and $G(30,000, 0.1)$. For each of the 29 different classes of random graphs considered, we run 10 instances with similar features. All instances were solved to optimality by both algorithms within the time limit, with the exception of the instances from the families $n \geq 15,000$, for which MoMC reported a memory problem in all 10 cases.

According to Table 4, CliSAT also outperforms MoMC in this testbed (not taking into account families where a memory problem is reported). In detail, CliSAT is faster than MoMC (on average) in 23 families out of a possible 29. The bigger differences in favour of CliSAT occur in the dense graphs of order 300 and 500. For example, CliSAT is more than twice as fast in the classes $G(300, 0.8)$ and $G(500, 0.7)$ than its counterpart. From the table, it can also be observed that the difference in performance between both algorithms becomes less acute as the order of the graphs increase for those families with $n \geq 1000$ (densities ≤ 0.5). On the other hand, LMC outperforms CliSAT (and MoMC) in the large graphs with 0.1 density, i.e., $n \geq 15,000$, also scaling much better as the size of the graphs increase. This is consistent with the fact that LMC is designed precisely for large and massive graphs with small densities. LMC is also the fastest in some small easy instances, possibly because of CliSAT's initial pre-processing phase (see Section 2.5).

3.5. Comparison with algorithms designed for sparse real-world graphs

As mentioned in the introductory section, the algorithm CliSAT is tailored to solve hard dense graphs of small and medium order, i.e. $|V| \leq 25,000$. Existing algorithms for sparse large and massive real-world graphs, such as, e.g., Hespe et al. (2020); Jiang et al. (2016); San Segundo et al. (2016b); Walteros & Buchanan (2020) (see also the introductory section), exploit the specific topology of such networks, e.g. the fact that the clique number is usually "close" to the graph's *degeneracy* $\gamma(G)$. We recall that the degeneracy $\gamma(G)$ of a graph G (also known as the graph's *k-core*) is the maximum integer such that a subgraph G' of G exists with minimum degree $\delta(G')$ greater or equal than $\gamma(G)$. It follows that $\gamma(G) + 1$ is an upper bound on the clique number $\omega(G)$ of the graph. Such algorithms rely heavily on *kernelization*, i.e., a pre-processing stage (typically) in which the original input network is replaced by a smaller network called a *kernel*, and other reduction techniques inspired in the vertex cover problem, see, e.g., Hespe et al. (2020), that are employed in the nodes of a combinatorial *branch-and-reduce* tree.

The aim of this section is to establish an approximate (not exhaustive) performance comparison between CliSAT and the state-of-the-art algorithms for real-world graphs. For this purpose, we have selected the algorithms LMC (Jiang et al., 2016), dOmega (Walteros & Buchanan, 2020) and BBMCSP (San Segundo et al., 2016b). The 3 algorithms employ some form of kernelization, but while dOmega is of the type branch-and-reduce, BBMCSP and LMC are branch-and-bound algorithms that employ kernelization during pre-processing.

Table 5 reports results over 27 real-world networks with less than 150,000 vertices: 22 networks are taken from the DIMACS10, SNAP and Social Networks collections and 5 from other sources (all the instances are available at the network repository http://www.networkrepository.com/). Specifically, the dataset contains all the instances with less than 150,000 vertices reported in Jiang et al. (2016) and a subset of those reported in San Segundo et al. (2016b); Walteros & Buchanan (2020). In the latter case, the real-world graphs with less than 150,000 vertices are all relatively easy. The table shows the number of vertices and edges, the degeneracy ($\gamma(G)$), the clique number($\omega(G)$) and the time spent by the 4 algorithms for the 27 instances reported. In all the tests, the time limit was fixed at 7200 seconds. The $|V| < 150,000$ constraint is motivated by the memory requirements of CliSAT, which are too large for massive graphs since it stores the full adjacency matrix in memory to operate efficiently with vertex neighbourhoods using bitmasks.

According to Table 5, CliSAT is outperformed by at least one of the other 3 algorithms in 25 out of the 27 instances, being the fastest in the two smallest instances bio-human-gene1 and bio-human-gene2. Nevertheless, CliSAT solves all the instances within the time limit, whereas BBMCSP and dOmega are unable to solve one instance each. Besides the two smallest instances, CliSAT also performs similarly to the best algorithm LMC in the instance bio-mouse-gene, performs better than BBMCSP in 2 instances and better than dOmega in 5. The reported results also show that LMC is the best algorithm for this dataset, and that kernelization is an effective technique specially

Table 5
Performance comparison of the algorithm `CliSAT` with 3 state-of-the-art algorithms designed for real-world networks, over a set of 27 instances with $|V| < 150,000$. The symbol *tl* indicates that the time limit of 7200 seconds was reached.

						`LMC`	`BBMCSP`	`dOmega`	`CliSAT`
Source	Name	$\lvert V\rvert$	$\lvert E\rvert$	$\gamma(G)$	$\omega(G)$	Time [seconds]	Time [seconds]	Time [seconds]	Time [seconds]
Misc.	bio-human-gene2	14,340	9,027,024	1902	1300	87.5	2524.4	102.5	**5.2**
Misc.	bio-human-gene1	22,283	12,323,680	2047	1335	224.9	tl	210.0	**40.8**
SNAP	p2p-Gnutella24	26,518	65,369	5	4	**0.0**	**0.0**	**0.0**	0.1
SNAP	Cit-HepTh	27,769	352,285	37	23	**0.0**	0.1	0.3	0.2
DIMACS10	delaunay_n15	32,768	98,274	4	4	**0.0**	**0.0**	**0.0**	0.2
SNAP	Cit-HepPh	34,546	420,877	30	19	**0.0**	0.2	0.2	0.2
Misc.	sc-TSOPF-RS-b2383-c1	38,120	16,115,324	655	7	6.3	**3.7**	106.8	6.0
DIMACS10	cond-mat-2005	40,421	175,691	29	30	**0.0**	**0.0**	**0.0**	0.2
DIMACS10	fe-body	45,087	163,734	6	6	**0.0**	**0.0**	**0.0**	0.3
Biolog.	bio-mouse-gene	45,101	14,461,095	1045	561	**87.8**	4141.4	2375.6	94.9
DIMACS10	t60k	60,005	89,440	2	2	**0.0**	**0.0**	0.1	0.5
DIMACS10	wing	62,032	121,544	3	3	**0.0**	0.1	**0.0**	0.5
DIMACS10	delaunay_n16	65,536	196,575	4	4	**0.0**	0.1	0.1	0.6
Misc.	rec-movielens	71,567	9,991,339	531	29	**15.0**	75.0	tl	70.2
SNAP	soc-Epinions1	75,879	405,740	67	23	**0.1**	0.2	0.6	0.9
DIMACS10	fe-tooth	78,136	452,591	7	5	**0.0**	0.2	0.3	1.1
Social	soc-Slashdot0902	82,168	504,230	55	27	**0.0**	0.2	0.4	1.1
Misc.	ia-enron-email-dynamic	87,273	297,456	53	33	**0.0**	0.1	0.2	26.1
Social	soc-BlogCatalog	88,784	2,093,195	221	45	**1.5**	3.9	186.5	236.8
DIMACS10	fe_rotor	99,617	662,431	8	5	**0.1**	0.3	0.4	2.5
Social	soc-buzznet	101,163	2,763,066	153	31	**1.1**	3.2	43.0	10.1
Social	soc-LiveMocha	104,103	2,193,083	92	15	**0.4**	2.1	4.0	4.3
DIMACS10	598a	110,971	741,934	8	7	**0.1**	0.5	0.3	2.3
Social	soc-wiki-conflict	118,100	2,027,871	145	25	**0.5**	1.3	7.4	47.1
DIMACS10	delaunay_n17	131,072	393,176	4	4	**0.0**	0.2	0.1	2.4
DIMACS10	fe-ocean	143,437	409,593	4	2	**0.0**	0.2	0.2	2.9
DIMACS10	144	144,649	1,074,393	9	7	**0.1**	0.7	0.6	4.6

when applied to the instances with millions of edges such as, e.g., `soc-BlogCatalog`, `rev-movielens` and `soc-buzznet`.

The reduction techniques employed by `dOmega` in the search tree are less effective when the gap between the graph's degeneracy and its clique number is large (with the exception of the two smallest graphs). In contrast, the algorithms BBMCSP and LMC are less affected by this gap, possibly because they rely more on typical maximum clique techniques employed for small and medium dense graphs.

3.6. Comparison with additional MCP exact approaches

In order to provide a broader picture of the performance of `CliSAT`, we provide a comparison against integer linear programming (ILP) formulations, solved by a general purpose ILP solver, and 3 additional effective combinatorial branch-and-bound algorithms from the literature.

Let x_v be a binary variable taking value 1 if and only if vertex $v \in V(G)$ belongs to the maximum clique. The natural ILP formulation for the MCP reads as follows:

$$\omega(G) = \max \sum_{u \in V} x_u \tag{9a}$$

$$x_u + x_v \le 1, \quad \forall (u, v) \in \overline{E}(G), \tag{9b}$$

$$x_u \in \{0, 1\}, \quad \forall u \in V(G). \tag{9c}$$

The objective function (9a) corresponds to the total number of vertices of the maximum clique. Constraints (9b) impose that at most one vertex from each pair of non-adjacent vertices is selected. It is well known that the linear programming (LP) relaxation of this formulation provides a very weak upper bound $\ge |V|/2$. In line with what is typically done in the literature to strengthen Constraints (9b), we consider a collection $\mathscr{C}$ of independent sets of the graph G, covering all the pairs of non-adjacent vertices $\{u, v\} \in \overline{E}(G)$. We therefore can replace Constraints (9b) by:

$$\sum_{u \in I} x_u \le 1, \quad \forall I \in \mathscr{C}. \tag{10}$$

Constraints (10) impose that no more than a single vertex is selected from each independent set $I \in \mathscr{C}$. Different heuristic procedures can be used for creating $\mathscr{C}$; we employ the one proposed in Bettinelli, Cacchiani, & Malaguti (2017).

We use the IBM CPLEX Optimizer version 12.8, one of the state-of-the-art commercial solvers, to tackle the ILP model (9) enhanced by Constraints (10). According to extensive preliminary experiments, these constraints have a positive impact on the performance of the solver. The solver also generates several additional valid inequalities of type (10) (as well as several other families of general purpose valid inequalities) during the execution of its branch-and-cut scheme to further strengthen the LP relaxation of the formulation. For a fair comparison against the branch-and-bound algorithms, the solver is run in single-thread mode (with default parameters). We denote this methodology to solve the MCP based on an ILP formulation as CPLEX in the remainder of this section.

We now briefly introduce 3 additional combinatorial branch-and-bound algorithms for the MCP from the literature that are tested in this work:

- `IncMC2` (Li et al., 2018a): An incremental SAT-based solver in the lines of `MoMC`, but which was developed some time earlier.
- `BBMCX` (San Segundo et al., 2015): An incremental SAT-based solver, denoted *infrachromatic* (see also Section 1.2), whose reasoning scheme is restricted to determining (a subset of) conflicting independent sets of cardinality 3. The algorithm also employs a similar bit-encoding as `CliSAT` to represent the graph and sets of vertices in memory.

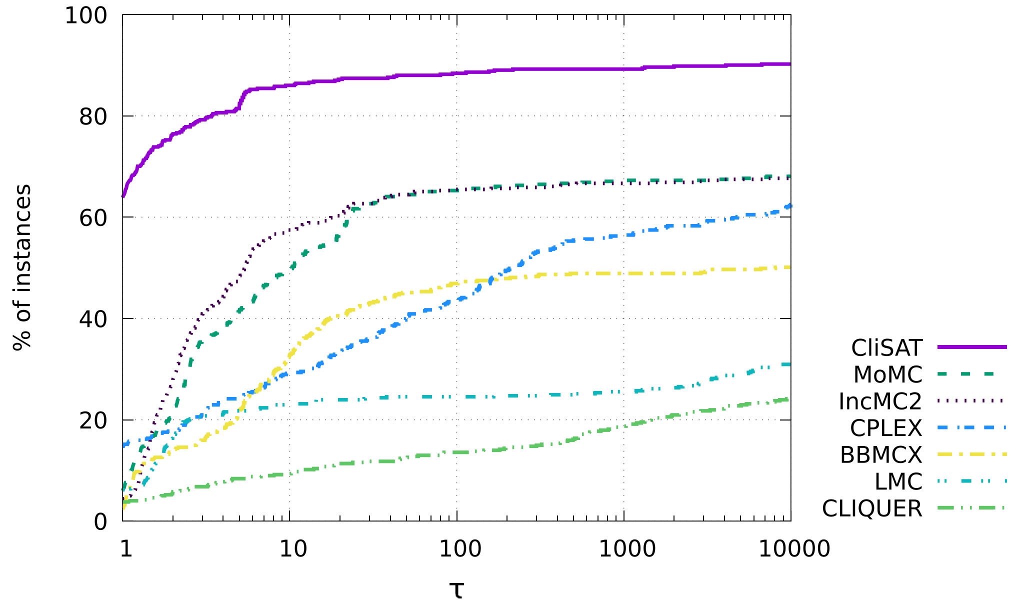

Fig. 8. Performance profile of the algorithm CliSAT and other 6 state-of-the-art algorithms over the entire dataset of 501 structured instances. The time limit is fixed at 1800 seconds.

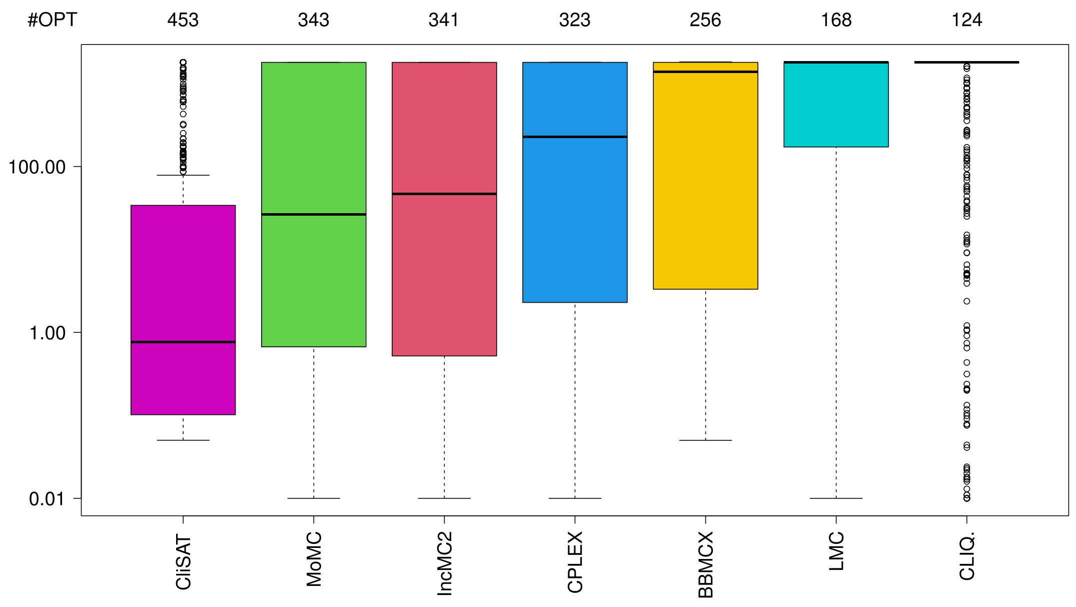

Fig. 9. Box plots of the performances of CliSAT and other 6 state-of-the-art algorithms over the entire dataset of 501 structured instances. The time limit is fixed at 1800 seconds.

- CLIQUER (Östergård, 2002): To the best of our knowledge, the first successful exact MCP solver that employs the *Russian Doll Search* (RDS) branching scheme.

Fig. 8 shows the performance profile (see Dolan & Moré, 2002) of the 3 algorithms described above plus CPLEX, together with CliSAT, MoMC and LMC, over our 501 structured instance dataset. According to Fig. 8, the best performing algorithm is, clearly, CliSAT, which is the fastest in more than 63% of the instances (left-end of the figure), and also solves the largest amount, i.e., slightly over 90% (as shown by the intersection of its curve in the right-end). The algorithms MoMC and IncMC2 are the second best performers according to the figure, MoMC solving to optimality 2 more instances (343) than its counterpart IncMC2 (341); this represents slightly more than 63% of the 501 dataset in both cases. The fourth performer is CPLEX, which initially solves around 18% of the instances, and shows the best slope as τ increases, solving more than 64% of the instances to optimality. The worst performing solvers according to the figure are BBMCX, which solves more than 51% of the instances and, finally, LMC and CLIQUER, which managed to solve slightly over 33% and 24% of the instances respectively.

We end the section by showing in Fig. 9 the computing time boxplots of the 7 algorithms. The figure plots the time (in seconds and logarithmic scale) spent by each algorithm through their quartiles. Precisely, each box represents the first (lower) and third (upper) quartiles, and the line separating both quartiles is the median of the reported CPU time distribution; the lines extending vertically from the boxes (known as *whiskers*) indicate the variability outside the upper and lower quartiles. Outside the whiskers, the outliers (heterogeneous results) are plotted as individual points. The numbers in the top row (header #OPT) show the number of instances solved to proven optimality within the time limit of 1800 seconds by each of the algorithms. Fig. 9 evidences the superior computing times of CliSAT, and is consistent with the results reported in the performance profile. It solves the largest amount of instances within the time limit (453), and presents the best median, which is slightly below 1 second.

4. Conclusions and future work

In this paper we present a very efficient combinatorial branch-and-bound exact algorithm CliSAT for the maximum clique problem. CliSAT combines all the recent state-of-the-art techniques with two new bounding procedures: (i) a filtering phase which exploits the notion of $(\kappa+1)$-partite branching nodes, i.e., nodes that are associated to a $(\kappa+1)$-partite graph and which require precisely a $(\kappa+1)$-clique to improve the incumbent solution; (ii) a partial maximum satisfiability-based procedure that prunes branching candidate vertices grouped according to independent sets, instead of individually. Our implementation has been extensively tested over a dataset of more than 700 instances from the literature, where it outperforms the state-of-the-art algorithms sometimes by several orders of magnitude.

A number of conclusions may be drawn from the tests. To begin with, empirical evidence suggests that the two new bounding techniques presented do not dominate each other and that the filtering phase of CliSAT is more effective in those instances where the gap between the chromatic number and the clique number is "small". Another conclusion is that, contrary to what is suggested in the recent paper entitled *Why is Maximum Clique Often Easy in Practice?* (Walteros & Buchanan, 2020), the problem remains very hard in practice, as witnessed by the instances that could not be solved to proven optimality by any of the algorithms tested. Clearly, further breakthroughs will be required to solve these very hard instances. An open question is whether these breakthroughs will come in the form of new heuristics for partial maximum satisfiability or in some other form. Another open question is the impact that the incremental branching scheme of CliSAT has on the effectiveness of its filtering phase. Intuitively, it would seem that incremental branching favours the appearance of $(\kappa+1)$-partite branching nodes in the shallow levels of the branch-and-bound tree, which, in turn, might be pruned with higher probability by the filtering phase of CliSAT.

Acknowledgements

The authors would like to thank three anonymous referees whose comments helped to improve the quality of the paper. This publication is part of the R&D project "Cognitive Personal Assistance for Social Environments (ACOGES)", reference PID2020-113096RB-I00, funded by MCIN/AEI/10.13039/501100011033.

References

Balas, E., & Yu, C. S. (1986). Finding a maximum clique in an arbitrary graph. *SIAM Journal on Computing, 15*(4), 1054–1068.
Bettinelli, A., Cacchiani, V., & Malaguti, E. (2017). A branch-and-bound algorithm for the knapsack problem with conflict graph. *INFORMS Journal on Computing, 29*(3), 457–473.
Carraghan, R., & Pardalos, P. M. (1990). An exact algorithm for the maximum clique problem. *Operations Research Letters, 9*(6), 375–382.
Coniglio, S., Furini, F., & Segundo, P. S. (2021). A new combinatorial branch-and-bound algorithm for the knapsack problem with conflicts. *European Journal of Operational Research, 289*(2), 435–455.
Davis, M., & Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the ACM (JACM), 7*(3), 201–215.
Dolan, E., & Moré, J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming, 91*, 201–203.
Fahle, T. (2002). Simple and fast: Improving a branch-and-bound algorithm for maximum clique. In *European symposium on algorithms* (pp. 485–498).
Furini, F., Ljubic, I., Martin, S., & Segundo, P. S. (2019). The maximum clique interdiction problem. *European Journal of Operational Research, 277*(1), 112–127.
Furini, F., Ljubic, I., Segundo, P. S., & Zhao, Y. (2021). A branch-and-cut algorithm for the edge interdiction clique problem. *European Journal of Operational Research, 294*(1), 54–69.
Hespe, D., Lamm, S., Schulz, C., & Strash, D. (2020). Wegotyoucovered: The winning solver from the pace 2019 challenge, vertex cover track. In *2020 proceedings of the SIAM workshop on combinatorial scientific computing* (pp. 1–11).
Håstad, J. (1999). Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica, 182*, 105–142.
Jiang, H., Li, C.-M., Liu, Y., & Manya, F. (2018). A two-stage maxsat reasoning approach for the maximum weight clique problem. In *Thirty-second AAAI conference on artificial intelligence*.
Jiang, H., Li, C.-M., & Manya, F. (2016). Combining efficient preprocessing and incremental maxsat reasoning for maxclique in large graphs. In *Proceedings of the twenty-second European conference on artificial intelligence* (pp. 939–947).
Li, C., Fang, Z., Jiang, H., & Xu, K. (2018a). Incremental upper bound for the maximum clique problem. *INFORMS Journal on Computing, 30*(1), 137–153.
Li, C., Jiang, H., & Manyà, F. (2017). On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem. *Computers and Operations Research, 84*, 1–15.
Li, C., Liu, Y., Jiang, H., Manya, F., & Li, Y. (2018b). A new upper bound for the maximum weight clique problem. *European Journal of Operational Research, 270*(1), 66–77.
Li, C.-M., Fang, Z., & Xu, K. (2013). Combining maxsat reasoning and incremental upper bound for the maximum clique problem. In *2013 IEEE 25th international conference on tools with artificial intelligence* (pp. 939–946).
Li, C.-M., & Quan, Z. (2010a). Combining graph structure exploitation and propositional reasoning for the maximum clique problem. In *2010 22nd IEEE international conference on tools with artificial intelligence: vol. 1* (pp. 344–351).
Li, C.-M., & Quan, Z. (2010b). An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In *Twenty-fourth AAAI conference on artificial intelligence* (pp. 128–133).
Malaguti, E., & Toth, P. (2010). A survey on vertex coloring problems. *International Transactions in Operational Research, 17*(1), 1–34.
Maslov, E., Batsyn, M., & Pardalos, P. (2014). Speeding up branch and bound algorithms for solving the maximum clique problem. *Journal of Global Optimization, 59*(1), 1–21.
Östergård, P. R. J (2002). A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics, 120*(1–3), 197–207.
Rossi, F., Van Beek, P., & Walsh, T. (2006). *Handbook of constraint programming*. Elsevier.
San Segundo, P., & Artieda, J. (2015). A novel clique formulation for the visual feature matching problem. *Applied Intelligence, 43*(2), 325–342.
San Segundo, P., Coniglio, S., Furini, F., & Ljubić, I. (2019a). A new branch-and-bound algorithm for the maximum edge-weighted clique problem. *European Journal of Operational Research, 278*(1), 76–90.
San Segundo, P., Furini, F., & Artieda, J. (2019b). A new branch-and-bound algorithm for the maximum weighted clique problem. *Computers and Operations Research, 110*, 18–33.
San Segundo, P., Furini, F., & León, R. (2022). A new branch-and-filter exact algorithm for binary constraint satisfaction problems. *European Journal of Operational Research, 299*(2), 448–467.
San Segundo, P., Lopez, A., Batsyn, M., Nikolaev, A., & Pardalos, P. M. (2016a). Improved initial vertex ordering for exact maximum clique search. *Applied Intelligence, 45*(3), 868–880.
San Segundo, P., Lopez, A., & Pardalos, P. M. (2016b). A new exact maximum clique algorithm for large and massive sparse graphs. *Computers and Operations Research, 66*, 81–94.
San Segundo, P., Matia, F., Rodriguez-Losada, D., & Hernando, M. (2013). An improved bit parallel exact maximum clique algorithm. *Optimization Letters, 7*(3), 467–479.
San Segundo, P., Nikolaev, A., & Batsyn, M. (2015). Infra-chromatic bound for exact maximum clique search. *Computers and Operations Research, 64*, 293–303.
San Segundo, P., Nikolaev, A., Batsyn, M., & Pardalos, P. M. (2016c). Improved infra-chromatic bound for exact maximum clique search. *Informatica, 27*(2), 463–487.
San Segundo, P., & Rodriguez-Losada, D. (2013). Robust global feature based data association with a sparse bit optimized maximum clique algorithm. *IEEE Transactions on Robotics, 29*(5), 1332–1339.
San Segundo, P., Rodríguez-Losada, D., & Jiménez, A. (2011). An exact bit-parallel algorithm for the maximum clique problem. *Computers and Operations Research, 38*(2), 571–581.
San Segundo, P., Rodriguez-Losada, D., Matia, F., & Galan, R. (2010). Fast exact feature based data correspondence search with an efficient bit-parallel MCP solver. *Applied Intelligence, 32*(3), 311–329.
San Segundo, P., & Tapia, C. (2014). Relaxed approximate coloring in exact maximum clique search. *Computers and Operations Research, 44*, 185–192.
Shimizu, S., Yamaguchi, K., & Masuda, S. (2018). A branch-and-bound based exact algorithm for the maximum edge-weight clique problem. In *International conference on computational science/intelligence & applied informatics* (pp. 27–47).

Stentiford, F. (2019). Face recognition by the construction of matching cliques of points. *Electronic Imaging, 2019*(8), 404-1.

Szabó, S. (2013). Monotonic matrices and clique search in graphs. *Annales Universitatis Scientiarum Budapestinensis, Sectio Computatorica, 41*, 307–322.

Szabó, S., & Zaválnij, B. (2019). Benchmark problems for exhaustive exact maximum clique search algorithms. *Informatica, 43*(2).

Tomita, E., Akutsu, T., & Matsunaga, T. (2011). Efficient algorithms for finding maximum and maximal cliques: Effective tools for bioinformatics. In *Intechopen*.

Tomita, E., Sutani, Y., Higashi, T., Takahashi, S., & Wakatsuki, M. (2010). A simple and faster branch-and-bound algorithm for finding a maximum clique. In *International workshop on algorithms and computation* (pp. 191–203).

Walteros, J. L., & Buchanan, A. (2020). Why is maximum clique often easy in practice? *Operations Research, 68*(6), 1866–1895.

Wu, Q., & Hao, J. (2013). An adaptive multistart tabu search approach to solve the maximum clique problem. *Journal of Combinatorial Optimization, 26*(1), 86–108.

Wu, Q., & Hao, J. (2015). A review on algorithms for maximum clique problems. *European Journal of Operational Research, 242*(3), 693–709.

Zhou, N.-F., Kjellerstrand, H., & Fruhman, J. (2015). *Constraint solving and planning with Picat*. Springer.