European Journal of Operational Research 242 (2015) 693–709

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Invited Review

A review on algorithms for maximum clique problems

Qinghua Wu [a], Jin-Kao Hao [b,*]

[a] School of Management, Huazhong University of Science and Technology, No. 1037, Luoyu Road, Wuhan, China
[b] LERIA, Université d'Angers, 2 bd Lavoisier, 49045 Angers, Cedex 01, France

ARTICLE INFO

Article history:
Received 23 November 2013
Accepted 28 September 2014
Available online 17 October 2014

Keywords:
Maximum clique problems
Exact algorithms
Heuristics
Applications

ABSTRACT

The maximum clique problem (MCP) is to determine in a graph a clique (i.e., a complete subgraph) of maximum cardinality. The MCP is notable for its capability of modeling other combinatorial problems and real-world applications. As one of the most studied NP-hard problems, many algorithms are available in the literature and new methods are continually being proposed. Given that the two existing surveys on the MCP date back to 1994 and 1999 respectively, one primary goal of this paper is to provide an updated and comprehensive review on both exact and heuristic MCP algorithms, with a special focus on recent developments. To be informative, we identify the general framework followed by these algorithms and pinpoint the key ingredients that make them successful. By classifying the main search strategies and putting forward the critical elements of the most relevant clique methods, this review intends to encourage future development of more powerful methods and motivate new applications of the clique approaches.

1. Introduction

The maximum clique problem (MCP) is to find a complete subgraph of maximum cardinality in a general graph. Its decision version is among the first 21 NP-complete problems presented in Karp's seminal paper on computational complexity (Karp, 1972). The MCP is among the most studied combinatorial problems.

The MCP has a wide range of practical applications in numerous fields. Early applications can be found for instance in Ballard and Brown (1982); Barahona, Weintraub, and Epstein (1992) and Christofides (1975) and are surveyed in Bomze, Budinich, Pardalos, and Pelillo (1999) and Pardalos and Xue (1994). Nowadays, more and more practical applications of clique problems arise in a number of domains including bioinformatics and chemoinformatics (Malod-Dognin, Andonov, & Yanev, 2010; Ravetti & Moscato, 2008), coding theory (Etzion & Östergård, 1998), economics (Boginski, Butenko, & Pardalos, 2006), examination planning (Carter, Laporte, & Lee, 1996; Carter & Johnson, 2001), financial networks (Boginski et al., 2006), location (Brotcorne, Laporte, & Semet, 2002), scheduling (Dorndorf, Jaehn, & Pesch, 2008; Weide, Ryan, & Ehrgott, 2010), signal transmission analysis (Chen, Zhai, & Fang, 2010), social network analysis (Balasundaram, Butenko, & Hicks, 2011; Pattillo, Youssef, & Butenko, 2012), wireless networks and telecommunications (Balasundaram & Butenko, 2006; Jain, Padhye, Padmanabhan, & Qiu, 2005). In addition to these applications, the MCP is tightly related to some important combinatorial optimization problems such as clique partitioning (Wang, Alidaee, Glover, & Kochenberger, 2006), graph clustering (Schaeffer, 2007), graph vertex coloring (Chams, Hertz, & Werra, 1987; Wu & Hao, 2012a), max-min diversity (Croce, Grosso, & Locatelli, 2009), optimal winner determination (Shoham, Cramton, & Steinberg, 2006; Wu & Hao, 2015), set packing (Wu, Hao, & Glover, 2012) and sum coloring (Wu & Hao, 2012b). These problems can either be directly formulated as a maximum clique problem or have a sub-problem which requires to find a maximum clique.

Given its theoretical importance and practical relevance, considerable effort has been devoted to the development of various solution methods for the MCP. On the one hand, effective exact methods have been designed mainly based on the general branch-and-bound (B&B) framework. These methods have the theoretical advantage of guaranteeing the optimality of the solution found. However, due to the inherent computational complexity of the MCP, exact methods can require a prohibitive computing time in the general case and are often applicable only to problems of limited sizes. On the other hand, to handle problems whose optimal solutions cannot be reached within a reasonable time, various heuristic and metaheuristic algorithms have been devised with the purpose of providing sub-optimal solutions as good as possible to large problems within an acceptable time. It is clear that exact and heuristic methods constitute two complementary solution approaches which can be applied to face different situations and fulfill different objectives. These two approaches can even be combined to create more powerful search methods.

* Corresponding author. +33 2 41 73 50 76.
E-mail addresses: qinghuawu1005@gmail.com (Q. Wu), hao@info.univ-angers.fr, jin-kao.hao@univ-angers.fr (J. K. Hao).

http://dx.doi.org/10.1016/j.ejor.2014.09.064
0377-2217/

Since the Second DIMACS Implementation Challenge dedicated to Maximum Clique, Graph Coloring, and Satisfiability organized during 1992–1993 (Johnson & Trick, 1996), studies on these NP-hard problems are becoming more and more intense. In particular, significant progresses have been achieved regarding the MCP, its important generalizations (e.g., maximum vertex weight clique and maximum edge weight clique) and relaxations (e.g., quasi-clique and densest k-subgraph). Advances on new algorithms have helped to find improved results to benchmark problems and deliver effective solutions to new applications (social network analysis, protein structure alignment, wireless network etc.).

At the same time, we observe that the two most influential surveys on the MCP date back to 1994 and 1999 respectively (Bomze et al., 1999; Pardalos & Xue, 1994). To the best of our knowledge, there is no updated review to report the newest advances achieved during the past 15 years. This paper thus aims to fill this gap by providing a detailed review of different solution approaches proposed in the recent literature for maximum clique problems. We will not only make a general and large survey of the most representative exact and heuristic algorithms, but also carry out an in-depth analysis of the studied methods to identify their most relevant ingredients that make these methods successful.

2. Definitions, problem formulations and computational complexity

Let $G = (V, E)$ be an undirected graph with vertex set $V = \{1, \ldots, n\}$ and edge set $E \subseteq V \times V$. A *clique* C of G is a subset of V such that every two vertices in C are adjacent, i.e., $\forall u, v \in C, \{u, v\} \in E$. A clique is *maximal* if it is not contained in any other clique, a clique is *maximum* if its cardinality is the largest among all the cliques of the graph. The *maximum clique problem* (MCP) is to find a maximum clique of a given graph in the general case. The clique number $\omega(G)$ of G is the number of vertices in a maximum clique in G.

The maximum clique problem is strictly equivalent to two other well-known combinatorial optimization problems: the *maximum independent set problem* (MIS) and the *minimum vertex cover problem* (MVC). Given $G = (V, E)$, an independent set (also called a stable set) I of G is a subset of V such that every two vertices in I are not connected by an edge, i.e., $\forall u, v \in I, \{u, v\} \notin E$. The MIS is to determine an independent set of maximum cardinality. A vertex cover V' of G is a subset of V, such that every edge $\{i, j\} \in E$ has at least one endpoint in V'. The MVC is to find a vertex cover of minimum cardinality.

Let $\overline{G} = (V, \overline{E})$ be the complementary graph of G such that $\{i, j\} \in \overline{E}$ if $\{i, j\} \notin E$. One observes that C is a maximum clique of G if and only if C is a maximum independent set of $\overline{G}$, and if and only if $V \backslash C$ is a minimum vertex cover of $\overline{G}$. An illustration of the relation between maximum clique, maximum independent set and minimum vertex cover is given in Fig. 1. Due to the close connection between the MCP and MIS, we will operate with both problems while describing the properties and algorithms for the MCP. Clearly a result which holds for the MCP in G will also be true for the MIS in $\overline{G}$. This paper focuses thus on the MCP (and the MIS) while putting aside the MVC which itself has a large body of studies in the literature.

On the other hand, one notices that some generalizations and relaxations of the MCP are attracting increasing attention recently due to their wide applications in some emerging areas like bioinformatics and social network analysis. We will discuss these cases in Section 5. The rest of this section is dedicated to the issue of problem formulations and complexity of the MCP.

There are numerous studies on the formulation of the MCP. These studies are of great interest since they can lead to deep understandings of the problem and the discovery of new results of theoretical and practical nature. For a comprehensive review of the existing formulations of the MCP, the reader is refereed to Bomze et al. (1999); Butenko (2003) and Pardalos and Xue (1994). Below, we review some typical and recently developed formulations.

The simplest formulation is given by the following binary program:

$$\text{maximize} \quad \sum_{i=1}^{n} x_i \tag{1}$$

$$\text{subject to} \quad x_i + x_j \leq 1, \quad \forall \{i, j\} \in \overline{E} \tag{2}$$

$$x_i \in \{0, 1\}, \quad i = 1, \ldots, n. \tag{3}$$

In this edge formulation, any feasible solution defines a clique C in G as follows: vertex i is in the clique if $x_i = 1$ and otherwise $x_i = 0$. The linear relaxation of this formulation is significant as well. If a variable $x_i = 1$ holds for an optimal solution to the linear relaxation of the above formulation, then $x_i = 1$ holds for at least one optimal solution to the integer formulation (Nemhauser & Trotter, 1975). Clearly this result can be used by an algorithm to reduce the explored search space when seeking an optimal clique.

Let S denote the set of all maximal independent sets in G, an alternative formulation based on independent sets imposes that any clique of G can contain no more than a single vertex from any maximal independent set of G:

$$\text{maximize} \quad \sum_{i=1}^{n} x_i \tag{4}$$

$$\text{subject to} \quad \sum_{i \in s} x_i \leq 1, \quad \forall s \in S \tag{5}$$

$$x_i \in \{0, 1\}, \quad i = 1, \ldots, n. \tag{6}$$

This formulation has the advantage that the expected gap between the optimal solution and its linear relaxation is small. However, it is difficult to enumerate all independent sets in an arbitrary graph. Furthermore, as the number of independent sets in the graph grows exponentially with the graph size, it is not trivial to solve the relaxation of the independent set formulation.

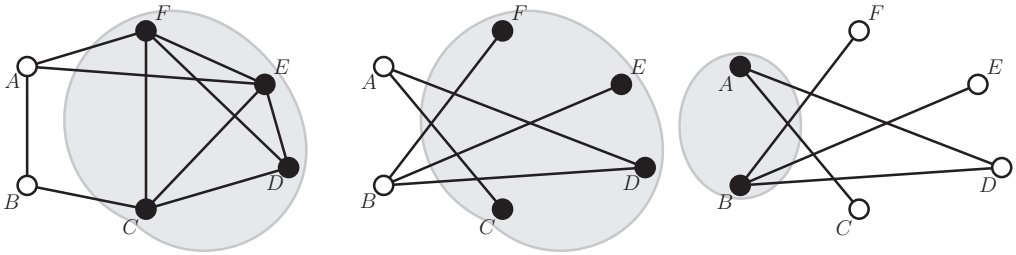

Fig. 1. An illustration of the relation between maximum clique, maximum independent set and minimum vertex cover. Given the initial graph G with $V = \{A, B, C, D, E, F\}$ (left) and its complementary graph $\overline{G}$ (middle/right), the set of vertices $\{C, D, E, F\}$ is a maximum clique of G and an maximum independent set of $\overline{G}$ while $\{A, B\} = V \setminus \{C, D, E, F\}$ is a minimum vertex cover of $\overline{G}$.

The above edge formulation can be modified to model the maximum weight independent set problem (Shor, 1990), leading to a quadratically constrained problem:

$$\text{maximize} \quad \sum_{i=1}^{n} w_i x_i \tag{7}$$

$$\text{subject to} \quad x_i x_j = 0, \quad \forall \{i,j\} \in E \tag{8}$$

$$x_i^2 - x_i = 0, \quad i = 1, \ldots, n. \tag{9}$$

This formulation combined with dual quadratic estimates generates very good computational results (Shor, 1990).

A remarkable connection exists between the MCP and a certain standard quadratic programming problem (Motzkin & Straus, 1965). For a given set of vertices $S \subseteq V$, let x^S be the characteristic vector of S, that is a vector that satisfies $x_i^S = \frac{1}{|S|}$ if vertex $i \in S$ and $x_i^S = 0$ otherwise, for $i \in V$. Let Δ be the standard simplex in the n-dimensional Euclidean space $\Re^n$: $\Delta = \{x \in \Re^n : \sum_{i=1}^n x_i = 1, x_i \geq 0, i = 1, \ldots, n\}$.

Let $A_G = (a_{ij})_{i,j \in V}$ be the adjacency matrix of the graph G, and for $x \in \Re^n$, let $g(x) = x^T A_G x$. Then the global optimal solution x^* to $\max_{x \in \Delta} g(x)$ is related to the clique number $\omega(G)$ by the following formula:

$$\omega(G) = \frac{1}{1 - g(x^*)} \geq \frac{1}{1 - g(x)}, \quad \forall x \in \Delta \tag{10}$$

Additionally, S is a maximum clique of G if and only if its characteristic vector x^S is a global maximizer of g on Δ.

More recently, discretized formulations for the MCP are proposed in Martins (2011), leading to tight upper bounds on many benchmark graphs based on linear relaxations. These formulations use an additional set of auxiliary variables:

$$w^q = \begin{cases} 1 & \text{if the clique size is equal to } q, \\ 0 & \text{otherwise,} \end{cases} \quad \forall q \in Q \tag{11}$$

with $Q = \{q_{\min}, \ldots, q_{\max}\}$ $(1 \leq q_{\min}, \omega(G) \leq q_{\max})$ being an interval containing all cliques' sizes. Then the MCP is formulated as follows:

$$\text{maximize} \quad \sum_{i=1}^{n} x_i \tag{12}$$

$$\text{subject to} \quad \sum_{j \in N(i)} x_j \geq (q-1)x_i - (q-2)(1 - w^q), \quad \forall i \in V, \forall q \in Q \tag{13}$$

$$\sum_{i \in V} x_i = \sum_{q \in Q} q w^q \tag{14}$$

$$\sum_{q \in Q} w^q = 1 \tag{15}$$

$$x_i \in \{0,1\}, \forall i \in V \tag{16}$$

$$w^q \in \{0,1\}, \forall q \in Q \tag{17}$$

$N(i)$ in constraint (13) denotes the set of vertices adjacent to i in the graph. Constraints (15) and (17) impose that only a single clique size is determined. Constrains (13), (14) and (16) force that each of the q vertices belonging to the clique must have at least $q-1$ neighbors in the clique. Constraint (13) can be further strengthened in an extended and discretized variable space, leading to an extended and discretized formulation to the MCP. This formulation requires the auxiliary variables $\{w^q\}$ and a new set of node variables that include an extra index with information on clique size:

$$x_i^q = \begin{cases} 1 & \text{if vertex } i \text{ is in a clique of size } q, \\ 0 & \text{otherwise,} \end{cases} \quad \forall i \in V, \forall q \in Q \tag{18}$$

Using these variables, the following extended and discretized formulation for the MCP can be given:

$$\text{maximize} \quad \sum_{i \in V} \sum_{q \in Q} x_i^q \tag{19}$$

$$\text{subject to} \quad \sum_{q \in Q} w^q = 1 \tag{20}$$

$$\sum_{j \in N(i)} x_j^q \geq (q-1)x_i^q, \quad \forall i \in V, \ q \in Q \tag{21}$$

$$\sum_{i \in V} x_i^q = q w^q, \quad \forall q \in Q \tag{22}$$

$$w^q \in \{0,1\}, \quad \forall q \in Q \tag{23}$$

$$x_i^q \in \{0,1\}, \quad \forall i \in V, \forall q \in Q \tag{24}$$

Obviously, the number of variables and constraints of these formulations depends on the range of variation of an interval containing the clique number (ω) of the graph. When a short interval containing $\omega(G)$ is known and for sparse graphs, the LP relaxation of these discretized formulations are able to produce much stronger upper bounds for $\omega(G)$ than other formulations.

The MCP is well studied from the complexity viewpoint, early complexity results on the problem being reviewed in Bomze et al. (1999) and Pardalos and Xue (1994). During the last 15 years, a number of advances on the research of the computational complexity of the MCP appeared, including inapproximability results, parametric complexity completeness, refined complexity analysis of exponential algorithms. Although a full review of these theoretical results is clearly beyond the scope of the paper, we provide a brief summary of some main results.

The current best-known polynomial-time approximation algorithm achieves only an approximation guarantee of $O(n(\log\log n)^2/(\log n)^3)$ (Feige, 2004). On the other hand, the study of Engebretsen and Holmerin (2003) shows that the MCP is not approximable within a factor of $n/2^{O(\log n/\sqrt{\log\log n})}$ under the assumption that NP $\not\subseteq$ ZPTIME $(2^{O(\log n(\log\log n)^{3/2})})$. The work of Khot (2001) further demonstrates that the MCP cannot be approximated within a factor of $n/2^{(\log\ n)^{1-\gamma'}}$ for some small constant $\gamma' > 0$, assuming NP $\not\subseteq$ ZPTIME$(2^{(\log\ n)^{O(1)}})$. In Håstad (1999), it is shown that the MCP is not approximable within $n^{1-\epsilon}$ for any $\epsilon > 0$, unless NP = ZPP. An improved result shows that the MCP is not approximable within $n^{1-\epsilon}$ for any $\epsilon > 0$ unless NP = P (Zuckerman, 2006). Another study (Bourgeoisa, Escoffiera, & Paschosa, 2011) confirms that if there exists an exact exponential time algorithm for the related MIS with a worst-case complexity $O^*(\gamma^n)$ $(\gamma < 2)$ where $O^*(\cdot)$ is $O(\cdot)$ ignoring polynomial factors and n is the order of the graph, then for any $\rho \in (0,1]$, there exists a ρ-approximation algorithm for the MIS that runs in time $O^*(\gamma^{\rho n})$. Recently, it is established that unless NP $\subseteq$ SUBEXP, for every $0 < \delta < 1$, there exists a constant $F(\delta) > 0$ such that the MCP has no Fixed-Parameter Tractable (FPT) optimum approximation with ratio $\rho(\text{OPT}) = \text{OPT}^{1-\delta}$ in $2^{\text{OPT}^F} \cdot \text{poly}(|V|)$ time (Chitnis, Hajiaghayi, & Kortsarz, 2013). This hardness result in the domain of parameterized complexity is further enhanced very recently in Chalermsook, Laekhanukit, and Nanongkai (2013), confirming that, for any r larger than some constant $\epsilon > 0$, any r-approximation algorithm for the MIS must run in at least $2^{n^{1-\epsilon/r^{1+\epsilon}}}$ time under the exponential time hypothesis.

3. Exact approaches to the MCP

This section is dedicated to an in-depth review of the most recent and influential exact methods for the MCP. Unsurprisingly, most

of them are based on the general B&B framework. They differ from each other mainly by (1) their specific techniques to determine the lower and upper bounds and (2) their branching strategies. Below, we begin with the general B&B scheme which is common to many studied methods and then review the improvements introduced by the most representative methods. Comprehensive overviews covering early exact methods prior to 1999 can be found in Bomze et al. (1999) and Pardalos and Xue (1994).

Algorithm 1 A simple algorithm to find the maximum clique C^*

```
1: Function Main
2:    C* ← ∅ {the maximum clique}
3 :   Clique(∅, V)
4:    return C*
5: End function
6: Function Clique(set C, set P)
7:    if (|C| > |C*|) then
8:       C* ← C
9:    End if
10:   if (|C| + |P| > |C*|) then
11:      for all p ∈ P in predetermined order, do
12:         P ← P \ {p}
13:         C' ← C ⋃ {p}
14:         P' ← P ⋂ N(p)
15:         Clique(C', P')
16:      End for
17:   End if
18: End function
```

3.1. General framework

An early and well-known exact algorithm (denoted by CP) is developed by Carraghan and Pardalos (1990) which is shown in Algorithm 1. Despite its simplicity, this algorithm constitutes an important step for exact solving of the MCP since it provides the basis for many later improved exact clique algorithms. For this reason, we discuss in detail the functioning of this algorithm and identify the key elements which impact critically its performance.

The CP algorithm uses two key vertex sets: C (called current solution or clique) and P (called candidate vertex set or candidate set). C designates the clique under construction while P is a subset of $V \backslash C$ such that $v \in P$ if and only if $\forall u \in C, \{u, v\} \in E$. In other words, each vertex of P must be connected to *all* the vertices of C (the reverse does not necessarily hold). Let $N(v)$ be the set of the vertices adjacent to vertex v, then P can equivalently be defined by $P = \cap_{v \in C} N(v)$. Given the property of P, it is clear that any vertex v of P can be added to C to obtain a larger clique $C' = C \cup \{v\}$. This property constitutes one of the key foundations of Algorithm 1.

The algorithm operates by calling recursively the function $Clique(C, P)$ (starting with an empty clique $C = \emptyset$ and $P = V$, see lines 2 and 3, Algorithm 1) and uses a global variable C^* to maintain the largest clique discovered so far ($|C^*|$ is thus the current best *lower bound* of the maximum clique). The function $Clique(C, P)$ is mainly defined by two key components: the bounding (line 10) and branching (line 11) procedures. Precisely, given the current clique C and its corresponding P set, one observes that $|C| + |P|$ naturally defines an *upper bound* for the maximum clique. As a consequence, if $|C| + |P| \leq |C^*|$, C cannot lead to a clique larger than C^* and thus can be safely closed. Otherwise, the subtree rooted at the clique C needs to be further explored. In this case, a branching strategy is employed to determine the next vertex $v \in P$ to be selected to expand the current clique C (line 11). In Carraghan and Pardalos (1990), this is achieved by selecting the first vertex $v \in P$ to insert into the current clique. Initially, the vertices $v_1, v_2, ..., v_n$ of the original graph G are ordered in such a way that v_1 has the smallest degree in V, v_2 has the smallest degree in $V \backslash \{v_1\}$ and so on. After each branching step, P is updated by the union $P = P \cap N(v)$ (line 14, Algorithm 1), making the required property of the set P always verified.

Since the introduction of the CP algorithm, many refinements have been devised to improve its performance with a focus on two key issues. The first one is to tighten the upper bound of the maximum clique (other than to use the size of P) during the search for the purpose of more efficient subtree pruning. The second one is to improve the branching rule in order to choose the most promising vertex of P to expand the clique.

Around these two issues, four typical methods can be found in the literature.

(1) The first one uses an iterative deepening strategy, and tries to obtain improved upper bounds on the size of the maximum clique of P with the aid of information obtained in some previously computed smaller subgraphs (Section 3.2).
(2) The second one is based on vertex coloring, which serves both as a bounding strategy to approximate the upper bound, as well as a branching strategy to guide the choice of the vertex from P (Section 3.3).
(3) The third one is to directly reduce the set P by removing certain vertices which can no way extend the current clique to a maximum clique or by using special rules to identify vertices in P which are part of a maximum clique that contains the current clique C (Section 3.4).
(4) The fourth one combines vertex coloring and the most recent MaxSAT technology to obtain tighter upper bounds (Section 3.5).

In what follows, we review representative works dealing with these issues.

3.2. Exact algorithms based on solving sub-clique problems

Östergård (2002) proposes an iterative deepening strategy which uses a basic idea similar to dynamic programming to improve the upper bound of the above CP algorithm. Let $S_i = \{v_i, v_{i+1}, \ldots, v_n\}$ be a subgraph of the initial graph G including the vertices from v_i through v_n. The CP algorithm seeks a maximum clique by first considering cliques in S_1 that contains $\{v_1, v_2, \ldots, v_n\}$ then cliques in S_2 that contains $\{v_2, \ldots, v_n\}$ and so on. In Östergård's algorithm (called Cliquer), this ordering is reversed: Cliquer starts on the smallest subgraph S_n containing only vertex v_n and determines a maximum clique of this graph (which is trivial). Then it considers S_{n-1} composed of two vertices $\{v_{n-1}, v_n\}$ and determines a maximum clique. As such, Cliquer iteratively finds a maximum clique for subgraphs $S_{n-2}, \ldots, S_1$ and ends up with a maximum clique in the last subgraph S_1 which is the original graph to be solved. Instead of using the size of P to bound the maximum clique, a new strategy is introduced in Cliquer using information from the previously computed maximum cliques of smaller graphs.

Let $c(i)$ $(i = n, n-1, \ldots, k)$ be the size of the maximum clique of the subgraph S_i, which is previously computed for each subgraph $S_n, S_{n-1}, \ldots, S_k$. Then we can use these $c(i)$ values to define a new pruning strategy when searching on the subgraph S_{k-1}. In fact, to obtain a clique greater than $|C^*|$ from the current clique C (and its associate P), one can safely prune the search if $|C| + c(i) \leq |C^*|$ where $i = \min\{j : v_j \in P\}$, since P is a subset of S_i and $c(i)$ is an upper bound to the maximum clique of the subgraph induced by P. With this technique, Cliquer is able to boost its performance significantly. Indeed, comparisons with several other approaches before Cliquer on random and DIMACS instances show that Cliquer is superior in many cases to the reference approaches. Cliquer, however, has not proved for the moment as successful as some sophisticated vertex coloring based exact algorithms like MCQ (Tomita & Seki, 2003) and BB-MaxClique (Segundo, Rodríguez-Losada, & Jiménez, 2011).

3.3. Exact methods based on vertex coloring

To estimate the upper bound of the maximum clique, graph coloring techniques are frequently applied to the subgraph induced by the candidate set P. This is based on the general fact that if a graph can be colored with k colors, then the maximum clique in this graph is smaller or equal to k. As such, a smaller k for the subgraph induced by P corresponds to a better upper bound of the maximum clique (see Section 3.1). Since the number of color classes is a much more precise estimation than the number of vertices in P, using a coloring instead of $|P|$ improves generally the upper bound and consequently reduces the size of the search tree. However, since coloring itself is NP-hard, finding k-coloring with k close to the chromatic number may be extremely difficult and time consuming. It is therefore important to find a proper trade-off between computing time and bound quality. In addition, vertex coloring can be also served in defining branching rules to guide the choice of the vertices from the candidate vertex set P during the search process.

Basically, two different ways of using vertex coloring for branching and bounding are explored in the literature. The first one colors the initial graph only once before the B&B routine starts and uses this coloring throughout the search. This strategy has the main advantage of running the coloring algorithm only once. However, since the clique algorithm manipulates many and *different* subgraphs of the initial graph G, the coloring for G is not necessarily appropriate for bound estimation of these reduced subgraphs. The second strategy applies repeatedly a coloring algorithm to different subgraphs at different nodes of the search tree. This strategy makes it possible to obtain tighter bounds of the maximum clique of the subgraphs. However, coloring multiple graphs may be also time consuming. For the purpose of saving computing time, fast greedy coloring heuristics are usually employed.

3.3.1. Branch-and-bound strategies based on subgraphs coloring

An early classic B&B algorithm (denoted by BT) which employs a heuristic coloring for finding maximum cliques can be find in Babel and Tinhofer (1990). This algorithm employs two strategies for bounding and pruning. The first strategy, which uses the number of vertices in P, is done exactly in the same way as the algorithm of Carraghan and Pardalos. In the second pruning strategy, the algorithm employs a coloring algorithm based on the greedy DSATUR procedure (Brélaz, 1979) to find a better upper bound. In addition, DSATUR also provides a heuristic maximum clique of P which is then used to update the lower bound of the original graph. At each node of the search tree, the algorithm first computes a coloring of the candidate vertex set P, then it chooses a vertex according to the lexicographical order and adds it into the current clique C.

The MCQ algorithm (Tomita & Seki, 2003) goes one step further, based on the idea that the coloring of the subgraph induced by the candidate set P not only provides a bound on the size of the maximum clique, but also serves as a branching strategy. The main ingredient of MCQ is to use a numbering and sorting procedure to sort the vertices of P in an ascending order with respect to the color numbers at each branching step. The numbering and sorting procedure first employs a greedy coloring algorithm to color the vertices of P in a predetermined order as follows. For each coloring step, a vertex $v \in P$ is inserted into the first possible color class, so that v is non-adjacent to any vertex already in this color class. If such a color class does not exist, a new color class is opened to insert v. After all vertices in P are assigned to their respective color classes, these vertices are sorted in an ascending order with respect to their color numbers, and then copied back to P according to this sorting order. Then, at each search step of the algorithm, MCQ selects a vertex $v \in P$ in reverse order (the last vertex in the reordered set P belongs to the highest color class) and the color number associated with each vertex becomes an upper bound for the maximum clique in the remaining subgraph to be searched.

There are two noteworthy differences in the use of coloring in the MCQ algorithm with respect to an early algorithm also using coloring (Babel & Tinhofer, 1990). First, instead of computing a coloring at *each node* of the search tree, the MCQ algorithm looks for a heuristic coloring only for a *new branch*. Second, at each step, a vertex in the candidate set is selected from the final color class of the coloring of the candidate set. With this strategy, the number of colors of the coloring for the candidate set tends to be reduced more quickly than if a vertex is chosen according to the lexicographical order.

MCR (Tomita & Kameda, 2007) improves MCQ with a better initial sorting of vertices of the input graph G which is similar to that of the CP algorithm, but uses the same coloring process to compute the upper bound as in MCQ. In MCR, the vertices of G is ordered into a list $L = (v_1, \ldots, v_n)$ where v_n is the vertex of minimum degree in G, v_{n-1} is the vertex of minimum degree in $G\backslash\{v_n\}$, v_{n-2} is the vertex of minimum degree in $G\backslash\{v_{n-1}, v_n\}$, and so on. Furthermore, the authors of MCR carried out additional computational experiments to confirm the usefulness of the initial sorting of vertices by comparing the performance of MCR with MCQ.

The numbering and sorting procedure of MCQ is further improved in the MaxCliqueDyn algorithm (Konc & Janežič, 2007). This is based on the observation that a better upper bound is achieved, when the vertices in P are presented to the greedy coloring procedure in a non-increasing order of their degrees (Carraghan & Pardalos, 1990). The authors of MaxCliqueDyn realize that it is necessary to reorder only those vertices in the candidate set P with color numbers large enough to be added to the current clique C in a direct descendant order with respect to the color numbers. Any vertex $v \in P$ with a color number below a threshold $K_{\min} < |C^*| - |C| + 1$ will never be added to the current clique and is kept in the same order as it was presented to the coloring algorithm. In addition, MaxCliqueDyn dynamically recomputes the degree of vertices at some nodes near the root of the search tree, and re-orders the vertices in the non-increasing order with respect to their degrees before coloring these vertices. This coloring strategy makes MaxCliqueDyn faster than MCQ, especially for dense graphs.

In Segundo et al. (2011), an exact bit-parallel algorithm (BB-MaxClique) for the MCP is proposed. BB-MaxClique uses an improved approximate coloring procedure which relies on a new implicit global degree branching rule to obtain tighter upper bounds to the candidate set P. The basic idea of this branching rule is to keep the vertices in the candidate set P in the same order as they are presented initially to the BB-MaxClique procedure (conveniently sorted by non-increasing degree) at each level of recursion so that the vertices in the candidate set P can be presented to the coloring procedure in a roughly non-increasing order with respect to their degrees. Moreover, BB-MaxClique makes full use of bit strings to efficiently compute basic operations during the search. Their experiments confirm that the implicit global degree branching rule prunes the search tree better than the two reference algorithms MCQ and MaxCliqueDyn for a wide range of DIMACS graph, notably for dense graphs.

The greedy coloring procedure in the MCQ algorithm (Tomita & Seki, 2003) is further improved by MCS (Tomita, Sutani, Higashi, Takahashi, & Wakatsuki, 2010), which uses a recoloring strategy to improve the coloring obtained by the greedy coloring procedure. The basic idea of MCS is that if a vertex $v \in P$ with color number $k_v > k_{\min}$ ($k_{\min} = |C^*| - |C|$) is moved to a color class with color index $l \leq k_{\min}$, then the number of vertices to be searched in the candidate set P can be reduced since all vertices with a color number below $k_{\min}$ will be pruned in the derived child subproblem. It is possible to reassign a vertex v belonging to a color class C_k ($k > k_{\min}$) to a different color class C_j ($j \leq k_{\min}$) if the following two conditions hold:

(1) There exists a vertex $w \in C_j$ such that $N(v) \bigcap C_j = \{w\}$, i.e., w is the only member of the neighbor set of v in C_j.
(2) There exists a color class C_l such that $N(w) \bigcap C_l = \varnothing$ and $l \leq k_{\min}$, i.e., C_l does not contain any neighbor of w.

In this case, v is moved from its current color class to another color class C_j with $j \leq k_{\min}$, the subproblem hanging from v will be pruned. Very recently, this recoloring strategy is integrated in a bit string framework, leading to an improved bit parallel exact algorithm (Segundo, Matia, Rodriguez-Losada, & Hernando, 2013).

3.3.2. Branch-and-bound strategies based on a unique coloring

B&B algorithms like MCQ, MCR, MaxCliqueDyn and BB-MaxClique need to compute a coloring at each branching step. Since finding a coloring is time consuming, frequently calculating colorings could affect the performance of these algorithms. Another class of B&B algorithms which use coloring as an aid for bounding and branching apply only once a coloring algorithm to the initial graph before the B&B routine starts and then use the obtained coloring on the permanent base during the search. A typical algorithm of this kind (denoted by DK) is shown in Kumlander (2005).

The DK algorithm integrates a pruning rule similar to MCQ and a backtracking search technique which examines the graph in the opposite order of a standard B&B algorithm. The first step of the algorithm is to obtain a coloring $c = \{C_1, C_2, \ldots, C_k\}$ of the vertices of G and re-order the vertices by color classes, so that color classes will appear in the decreasing order with respect to the color index, i.e., $V = \{C_k, C_{k-1}, \ldots, C_1\}$. One first considers all cliques that could be built using only the vertices of the first color class C_1. Then one considers all cliques that could be built using vertices of C_1 and C_2, i.e., of the first and second color classes, and so forth. The ith step considers all cliques that contain vertices of $\{C_i, C_{i-1}, \ldots, C_1\}$, the last considered subgraph being the original graph. The algorithm also uses a special array b to remember the maximum clique found for each level of the subgraphs during the backtracking search. So $b[i]$ is the maximum clique for a subgraph formed by the vertices of $\{C_i, C_{i-1}, \ldots, C_1\}$. The branching rule used by this algorithm is similar to MCQ, it always selects a vertex $v \in P$ with the largest color number to insert into the current clique. So when one considers v to become the $(j+1)$th vertex in the current clique and v belongs to the kth color class, the algorithm can prune the search if $j + b[k] \leq |C^*|$ (C^* is the largest clique found so far), since P is a subset of $C_k \cup C_{k-1} \cup \cdots \cup C_1$ and $b[k]$ is an upper bound to the maximum clique of the subgraph induced by $C_k \cup C_{k-1} \cup \cdots \cup C_1$. Computational experiments on the DIMACS benchmarks show that this algorithm outperforms the CP and Cliquer algorithms.

3.4. Tightening the candidate set P by filtering algorithms

Another method to improve the basic CP algorithm is to directly tighten the candidate set P by removing certain vertices which cannot be used to extend the current clique C to a maximum clique or by fixing vertices in P which are part of a maximum clique of G that contains the current clique C. Following this line, three B&B algorithms, denoted by *DF*, χ and $\chi + \text{DF}$, are studied in Fahle (2002). The basic idea of *DF* is to use two cost based domain filtering algorithms to "clean-up the candidate set" before each branching step. Given the current clique C, its corresponding candidate set P and the largest clique discovered so far C^*, let $N_P(v) = \{u \in P : \{u, v\} \in E\}$, the *DF* algorithm is based on the following filtering properties:

- *Lemma 1.* For each $v \in P$ such that $|C| + |N_P(v)| < |C^*|$, v cannot extend C to a maximum clique of G.
- *Lemma 2.* For each $v \in P$ such that $|N_P(v)| = |P| - 1$ is contained in any maximum clique of G that also contains C.

Lemma 1 allows the algorithm to safely remove from P every vertex of degree less than $|C^*| - |C|$, since such a vertex cannot be part of a clique larger than $|C^*|$ in G. With *Lemma* 2, one can displace every vertex of degree $|P| - 1$ from the candidate set P to the current clique C to increase the clique size.

In the χ algorithm, to approximate the chromatic number of the candidate set $P = \cap_{v \in C} N(v)$, one computes four heuristic colorings of P and retains the coloring with the smallest number of colors for bounding. These four colorings are obtained using two different greedy coloring algorithms. Algorithm $\chi + \text{DF}$ combines the domain filtering strategy of *DF* and the coloring-based bounding strategy: for each vertex $v \in P$, compute with a coloring algorithm an upper bound of the chromatic number for the subgraph induced by all the neighbors of v in P. If this upper bound plus the size of the current clique is smaller than the lower bound of the graph, then v can be safely removed from P. The computational experiments reported in the paper show that the combined use of these two strategies outperforms the CP algorithm as well as approaches that only apply either of these techniques. Furthermore, $\chi + \text{DF}$ is shown to be able to solve to optimality 45 out of the 66 tested DIMACS instances.

One obvious drawback of the filtering algorithm in $\chi + \text{DF}$ is the time required to compute the upper bound of the chromatic number of the subgraph induced by a vertex and its neighbors in the candidate set P. In Régin (2003), a constraint programming approach (CPR) is used to determine an upper bound based on a fast matching algorithm rather than a coloring algorithm. This new upper bound roughly corresponds to the number of independent sets of cardinality 2 in the subgraph induced by P. This bound is then used to remove some vertices from P after each branching step in the following way: select a vertex v and compute the upper bound using the matching algorithm for the subgraph induced by all the neighbors of v; if this upper bound plus the size of the current clique is smaller than the lower bound of the graph, then v can be safely removed from P. In addition to the filtering algorithm, CPR also uses the diving technique in conjunction with a MIP approach to solve subproblems and a specific strategy to select the vertex to expand the current clique. CPR is able to solve seven DIMACS instances to optimality for the first time.

3.5. Exact methods based on MaxSAT

Exact methods based on vertex coloring use the number of colors required by the subgraph to approximate the upper bound of the maximum clique on the subgraph induced by the candidate set P. This approach is further improved by using methods developed for MaxSAT (Li & Quan, 2010). The resulting method (denoted by MaxCLQ) is based on the following proposition: If G can be partitioned into k independent sets (thus can be colored with k colors), and there are s disjoint inconsistent subsets of soft clauses in the independent set using MaxSAT encoding, then the maximum clique in the graph is smaller or equal to $k - s$. To obtain a tighter bound of the maximum clique on the subgraph G' induced by P, MaxCLQ first partitions G' into k independent sets in a similar way to MCQ (Tomita & Seki, 2003). Based on this partition, one first encodes the graph into a MaxSAT instance where each independent set is encoded into a soft clause, and then uses an dedicated approach to detect the number (denoted by s) of disjoint inconsistent soft clauses in the MaxSAT instance. Finally $k - s$ is used as an upper bound to the maximum clique of the subgraph G' induced by P. This approach leads generally to tighter bounds than the pure coloring based approaches. Experimental evaluations on the DIMACS benchmarks show the B&B algorithm integrating this MaxSAT based bounding technique is very effective and can even close one open problem (p_hat1000-3).

Later, a complex approach combining MaxCLQ (Li & Quan, 2010) and MCS (Tomita et al., 2010) with the ILS algorithm (Andrade, Resende, & Werneck, 2012) is studied in Maslov, Batsyn, and Pardalos (2014). The ILS algorithm is used to obtain an initial solution to the maximum clique problem, which is then served as a lower bound to prune branches in the main B&B algorithm. The study shows that running ILS before MaxCLQ and MCS algorithms considerably reduces the total computing time for some hard DIMACS instances.

Table 1
Main exact algorithms for the maximum clique problem.

Algorithm name	Reference	Type of approach	Comments on performance
CP	Carraghan and Pardalos (1990)	The basic B&B algorithm	A landmark B&B algorithm providing the basis for many later B&B algorithms
BT	Babel and Tinhofer (1990)	B&B based on subgraphs coloring	Another classic B&B algorithm which computes a coloring for bounding at each search step, slower than modern B&B algorithms based on graph coloring
Cliquer	Östergård (2002)	B&B based on solving sub-clique problems	Reports better results than its predecessors, but performs less well than sophisticated vertex coloring based algorithms like MCQ, MCS and BB-MaxClique
$\chi + DF$	Fahle (2002)	B&B using filtering algorithms to tighten the candidate set P	Slower than Cliquer on an amount of DIMACS instances
MCQ	Tomita and Seki (2003)	B&B based on subgraphs coloring	Faster than Cliquer and a number of algorithms before MCQ on many DIMACS instances
CPR	Régin (2003)	B&B using filtering algorithms to tighten the candidate set P	Closes seven DIMACS instances for the first time, faster than another B&B algorithm using filtering algorithms $\chi + DF$ on most DIMACS instances
DK	Kumlander (2005)	B&B based on a unique coloring	Performs better than CP, but seems less competitive as some sophisticated vertex coloring based algorithms like MCQ, MCS and BB-MaxClique
MCR	Tomita and Kameda (2007)	B&B based on subgraphs coloring	An improved version of MCQ with a better initial sorting of vertices of the initial graph
MaxCliqueDyn	Konc and Janežič (2007)	B&B based on subgraphs coloring	Faster than MCQ on most DIMACS instances, especially for dense graphs
MaxCLQ	Li and Quan (2010)	B&B based on MaxSAT	A highly effective exact algorithm, closes a hard instance p_hat1000-3 for the first time
MCS	Tomita et al. (2010)	B&B based on subgraphs coloring	An improved version of MCQ, performs better than its two predecessors MCQ and MCR on many DIMACS instances
BB-MaxClique	Segundo et al. (2011)	B&B based on subgraphs coloring	Faster than MCQ and MaxCliqueDyn on a number of DIMACS instances, especially effective for dense graphs
ILS&MaxCLQ	Maslov et al. (2014)	B&B based on MaxSAT using local search for initial bounds	Reaches some speedup for some hard DIMACS instances compared to MaxCLQ, but needs more computing time for some easy instances

3.6. Other exact methods

In addition to the above algorithms, there are some other related studies. A branch and cut (B&C) algorithm for the equivalent MIS can be found in Rebennack, Oswald, Theis, Seitz, Reinelt, and Pardalos (2011). This algorithm is based on the edge-projection theory first introduced in Mannino and Sassano (1996) and further explored in Rossi and Smriglio (2001). An overview of various characteristics of effective B&C algorithms for the MCP and MIS is given in Rebennack, Reineltb, and Pardalos (2012). Some parallel and multi-threading exact algorithms appear very recently in the literature. For instance, a scalable and fault-tolerant solution for the MCP based on the MapReduce framework is presented in Xiang, Guo, and Aboulnaga (2013). The main idea of this approach is to first partition the graph into smaller subgraphs, which are then independently solved to optimality with a B&B method on different nodes of a computing cluster. Another recent multi-threading parallel algorithm adapts a B&B algorithm similar to MCQ and explores the multi-core parallelism with a bit-set encoding mechanism (McCreesh & Prosser, 2013). This algorithm achieves excellent performances on many DIMACS instances and some BHOSLIB instances.

To conclude this section on exact algorithms, Table 1 summarizes in chronological order the reviewed algorithms. One observes that very few exact algorithms report their results on the set of 40 BHOSLIB instances with the exception of the study reported in McCreesh and Prosser (2013). In Section 6, we provide more information about the relative performance of these exact algorithms.

4. Heuristic approaches to the MCP

In addition to exact algorithms, notable progresses on heuristic algorithms for the MCP have been made in the recent years. In this section we review the most representative and effective MCP heuristics.

4.1. Greedy algorithms

The majority of greedy heuristics for the MCP are called "sequential greedy heuristics". They generate a maximal clique by starting with an empty clique and then iteratively adding vertices using greedy rules until no further additions are possible, or by repeatedly removing vertices from a set of vertices which do not form a clique. A comprehensive survey of early greedy heuristics are provided in Bomze et al. (1999); Butenko (2003) and Pardalos and Xue (1994).

For these early greedy heuristics, decisions on which vertex to be added in or moved out are usually based on some static information associated with the vertices in the candidate set like their degrees. However, static greedy algorithms can easily fall into the usual greedy traps due to their short-sighted nature. Several improvements to the static greedy heuristics have been proposed in the literature. For instance, three adaptive restart randomized greedy heuristics (denoted by AI, AW and NA) are studied in Jagota and Sanchis (2001). The main ingredient of these heuristics is a probabilistic greedy vertex selection rule combined with a multi-start strategy. During each run of the greedy algorithm, the probability distributions used by the selection rule are updated through a learning-like mechanism. QUALEX-MS (Busygin, 2006) is another deterministic iterated greedy algorithm. In QUALEX-MS, each vertex is assigned a weight which represents its importance towards inclusion in the evolving clique. Vertex weights are calculated on the basis of the coordinates of stationary points of nonlinear programs derived from the Motzkin–Straus nonlinear formulation of the MCP. The deep adaptive greedy search (DAGS) by Grosso, Locatelli, and Croce (2004) integrates swap moves and vertex weights into a two-phase greedy construction procedure. Two important vertex selection strategies are introduced in DAGS to avoid usual greedy traps. First, DAGS uses swap moves to transform the current partial clique into an equal-sized but more promising clique instead of using only the short-sighted add move operator. Second, DAGS adaptively adjusts the vertex weights used for vertex selection

by a restart mechanism. This dynamic weighting technique guides the search towards less explored areas. Computational results show that DAGS is superior to QUALEX-MS for most of the tested DIMACS instances.

4.2. Local search heuristics

Local search is the most successful framework for designing effective MCP heuristics. We provide here an in-depth analysis of the most influential heuristic algorithms mainly developed during the two past decades.

4.2.1. Local search strategies for the MCP

When designing a local search algorithm for solving a particular problem, one has to define the search space to be explored, the evaluation function and the neighborhood function. This triplet forms a search strategy. We propose to classify the search strategies for the MCP into two categories.

- The k-fixed penalty strategy: A target clique size k is provided and the goal is to find a feasible clique of size k (a k-clique). The search space Ω is usually composed of all vertex subsets of size k (called k-subsets) including both feasible and infeasible cliques. To approximate the maximum clique, one searches for legal k-cliques with increasing k values.
- The legal strategy: The search space Ω contains all legal cliques and the goal is to find a clique $c \in \Omega$ whose size is as large as possible.

These two general strategies can be further classified according to the move operators used to explore the search space. For the k-fixed penalty strategy, existing local search algorithms mainly rely on either a constrained swap operator or a unconstrained swap operator which basically exchanges one (or more) vertex of the current clique with another vertex (or more vertices) out of the clique. As to the legal strategy, the algorithms typically employ three different move operators (add, swap and drop). Fig. 2 summarizes this classification which is also used in this review to guide our presentation of local search algorithms.

4.2.2. Local search algorithms based on the k-fixed penalty strategy

For the k-fixed penalty strategy, as noted in Friden et al. (1989), the maximum clique problem can be approximated by finding a series of k-cliques for increasing values of k (a k-clique is a clique of size k). Each time a k-clique is found, k is incremented by one and a new (larger) k-clique is sought. This process is repeated until no k-clique can be found. The last k-clique constitutes an approximation of the maximum clique of the graph. Consequently, the maximum clique problem comes down to the problem of finding k-cliques.

In the k-fixed penalty strategy, a solution is represented as a subset C of k vertices. The evaluation function $f(C)$ counts the number of edges induced by C, and the goal of the search algorithm is to maximize function $f(C)$ such that $f(C)$ reaches its largest value $f(C) = k \times (k-1)/2$, which means any two vertices of C are connected by an edge and C is a legal k-clique.

To obtain a neighboring solution from the current solution C, one typically swaps a vertex in C with another vertex in $V\backslash C$. Basically, there are two different techniques to apply the swap operation in the literature. The first technique considers all the possible moves induced by C and $V\backslash C$; any vertex from C and any vertex from $V\backslash C$ can take part in a swap operation. The second technique constrains the consideration of the two exchanged vertices to some specific subsets of C and $V\backslash C$. This technique is useful to reduce the computing time needed to explore the neighborhood.

As early as 1989, the first tabu search application to the equivalent MIS (called STABULUS) is presented in Friden et al. (1989) using the k-fixed penalty strategy. STABULUS tries to minimize the number of edges contained in the current subset of k vertices to zero (notice that here we are solving the MIS in the complementary graph, see Section 2). The neighborhood employed by STABULUS is defined by the unconstrained swap move operator, which swaps a vertex $u \in C$ with another vertex $v \in V\backslash C$. At each iteration, STABULUS selects the best swap move which decreases the most the evaluation function to generate the next neighboring solution. To determine whether a move is eligible, STABULUS first uses one tabu list T_1 containing the $|T_1|$ last solutions. In addition, STABULUS uses two other lists T_2 and T_3 containing the last vertices which were removed from (or introduced into) the independent set for the purpose of preselection. This algorithm reported remarkable results at the time it was published. STABULUS was later improved in Fleurent and Ferland (1996b) by introducing a different tabu list and tabu tenure management mechanism. Instead of putting the visited solutions in the tabu list, each time a vertex $u \in C$ is swapped with another vertex $v \in V\backslash C$, these two exchanged vertices are added into the tabu list and are forbidden to move for the next T_u and T_v iterations, where $T_u = 0.3 \times |V\backslash C|$ and $T_v = 0.3 \times |C|$.

Recently, an adaptive multi-start tabu search algorithm (AMTS) is introduced in Wu and Hao (2011) which is based on the k-fixed penalty strategy and the constrained swap operator. The constrained neighborhood limits the swap move to two critical subsets $X \subseteq C$ and $Y \subseteq V\backslash C$ such that $|X|$ and $|Y|$ are as small as possible, and the resulting neighborhood contains always the best solutions of the unconstrained neighborhood induced by C and $V\backslash C$. The critical subset $X \subseteq C$ identifies the vertices of the current solution C that are not in the tabu list and have the highest number of adjacent vertices in C, and set Y identifies the vertices of $V\backslash C$ that are not in the tabu list and have the smallest number of adjacent vertices in C. A neighboring

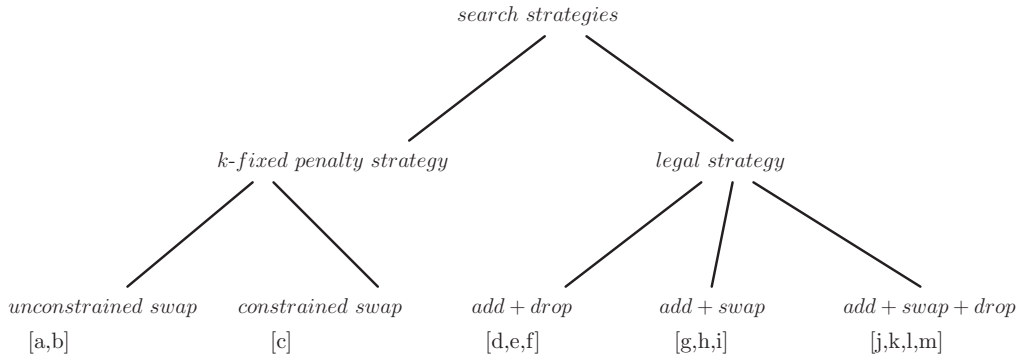

Fig. 2. Local search strategies for the MCP. a(Fleurent and Ferland (1996b)), b(Friden, Hertz, and Werra (1989)), c(Wu and Hao (2011)), d(Battiti and Protasi (2001)), e(Gendreau, Soriano, and Salvail (1993)), f(Katayama et al. (2005)), g(Pullan (2006)), h(Pullan and Hoos (2006)), i(Pullan, Mascia, and Brunato (2011)), j(Benlic and Hao (2013)), k(Jin and Hao (2015)), l(Hansen, Mladenović, and Urošević (2004)), m(Wu, Hao, and Glover (2012)).

solution C' is then obtained by swapping one vertex $u \in X$ and one vertex $v \in Y$. The constrained neighborhood is both more focused and smaller-sized compared to the unconstrained swap neighborhood and largely improves the computational efficiency of the search procedure. This algorithm competes well with reference heuristics like RLS (Battiti & Protasi, 2001) and DLS (Pullan & Hoos, 2006).

4.2.3. Local search algorithms based on the legal strategy

The algorithms presented in Andrade et al. (2012); Battiti and Protasi (2001); Jin and Hao (2015); Katayama et al. (2005); Pullan (2006); Pullan and Hoos (2006); Pullan et al. (2011) and Wu et al. (2012) follow the legal strategy. Here the search space Ω is defined by the set of all legal cliques. The function to be maximized is the clique size $f(C) = |C|$, and the neighborhood $N(C)$ is given by subsets of legal cliques which can be reached by applying some eligible move operators. From the literature, we can identify three such operators: add, drop and swap. The add operator displaces to C a vertex in $V\backslash C$ that is adjacent to all vertices in C. The drop operator deletes a vertex from C. The standard swap operator exchanges a vertex v in C with a vertex u in $V\backslash C$ which must be connected to all vertices in C except v. Note that a swap move can be trivially decomposed into two separate moves, i.e., a drop move followed by an add move (Battiti & Mascia, 2010). Most local search methods using the legal strategy jointly rely on two or three of these move operators. For instance, methods that employ the add and drop moves include the tabu search algorithms (Gendreau et al., 1993), the RLS algorithm (Battiti & Protasi, 2001) and the KLS algorithm (Katayama et al., 2005) while examples of using all three moves (add, drop and swap) are the VNS algorithm (Hansen et al., 2004), the family of PLS (Pullan, 2006), DLS (Pullan & Hoos, 2006), CLS (Pullan et al., 2011) algorithms, the MN/TS algorithm (Wu et al., 2012), the BLS algorithm (Benlic & Hao, 2013) and the SBTS algorithm (Jin & Hao, 2015). Notice that in some rare cases, more general swap(a, b) could be also useful allowing to exchange a and b vertices between C and $V\backslash C$.

For the algorithms using only add and drop moves, add moves are always preferred since they increase the clique size, drop moves are applied only when no add move exists. Associated with the add moves is the vertex set PA (which corresponds to the candidate set P of Section 3.1), which is composed of the vertices that are excluded from the clique C and connected to all the vertices of C: $PA = \{v : v \in V\backslash C, \{v, u\} \in E, \forall u \in C\}$. Algorithms which rely on the add and drop moves differ from each other chiefly in the scheme of vertex selection and the prohibition mechanism applied to the performed moves.

For instance, two early tabu search variants (denoted by DT and PT) using add and move are studied in Gendreau et al. (1993). The first variant is deterministic: when add moves are possible, one vertex $v \in PA$ with the maximum adjacent vertices in PA is selected to join the current clique C. When no add move exists, one vertex $v \in C$ that results in the largest PA is removed from C. The general prohibition rule adopted by the deterministic variant is as follows: Only vertices that leave the clique C are forbidden to move back to C during the prohibition period, vertices that join the clique C can leave C without restriction. The second variant (PT) is probabilistic: if add moves are possible, a random vertex in PA is selected for a possible addition, otherwise, the current solution C is a local optimum and additions are impossible, a number of randomly selected vertices in C are removed from it.

Reactive Local Search (RLS) by Battiti and Protasi (2001) is a landmark algorithm to the MCP which applies the add and drop operations. Starting from an empty clique, RLS explores the search space of cliques by adding/removing one vertex to/from the current clique. RLS applies add moves whenever this is possible and selects a vertex with the highest number of adjacent vertices in PA to add to the clique C. If no allowed addition exists, RLS searches for an allowed vertex to drop from C such that its removal leads to the largest set of vertex additions to C. If no allowed moves are available, a random vertex is picked from C and is dropped. As soon as a vertex is moved, it is put into the tabu list and remains prohibited for the next T iterations. The prohibition period T is related to the amount of desired diversification, and is determined by feedback information from the search history. RLS also uses a dynamic restart strategy to provide additional long-term diversification, assuring that each vertex is eventually tried as part of the current clique. Computational results on DIMACS instances show that RLS performs better than its predecessors, both in terms of efficiency and quality.

The k-opt local search (KLS) of Katayama et al. (2005) also relies on the add and drop operations. KLS is generalized from the basic 1-opt local search by moving multiple vertices at each iteration instead of a single vertex for the purpose of obtaining a larger clique. At each iteration of KLS, a variable number of k vertices are added to or removed from the current clique simultaneously by applying a sequence of add and drop move operations. The vertex selection rule of KLS is similar to that of the deterministic variant of the tabu search algorithm of Gendreau et al. (1993). Each time a vertex is added to (or removed from) C, it is forbidden to remove from (or add back to) C during this round of k-opt neighborhood search. A restart strategy is also employed by the KLS algorithm: The largest clique obtained in the previous execution becomes a new initial solution for the next round of k-opt neighborhood search. The reported results on 37 DIMACS benchmarks show that KLS remains competitive compared to RLS (Battiti & Protasi, 2001).

For the algorithms relying on the add, swap and drop moves, add moves are always applied whenever this is possible as they are the only moves that augment the current clique. Drop moves are considered only when no add or swap move exists. Associated with swap moves is the vertex set OM, which is composed of the vertices that are excluded from the clique C and connected to *all but one* vertex of C: $OM = \{v : v \in V\backslash C, |N(v) \bigcap C| = |C| - 1\}$ where $N(v) = \{j : j \in V, \{j, v\} \in E\}$ is the set of vertices adjacent to v.

The variable neighborhood search algorithm (VNS) of Hansen et al. (2004) uses a distance metric between two solutions T and T' which is the difference in cardinality of T and T'. Then the neighborhood $N_k(T)$ consists of all solutions at distance k from T. Three types of moves (add, swap and drop) are considered for the selected neighborhoods $N_k, k = 1, \ldots, k_{\max}$. An initial solution is obtained by a variable neighborhood descent (VND) heuristic, which is also used as the local search procedure of the VNS approach. VNS uses add moves in its descent phase and swap moves in a plateau phase. Upon reaching a local optimum with respect to both the add and swap neighborhoods, a new solution is randomly generated from the kth neighborhood by dropping k vertices from the current clique. Different selection rules are used by VND to guide the choice of the vertex to be added to the clique C. The results on a subset of the DIMACS instances show a globally good performance.

The family of stochastic local search algorithms (dynamic local search (DLS) (Pullan & Hoos, 2006), phased local search (PLS) (Pullan, 2006) and cooperating local search (CLS) (Pullan et al., 2011)) employ the add and swap moves in their main search procedure and the drop moves for overcoming search stagnation. These algorithms alternate between a clique expansion phase and a plateau search phase. The expansion phase seeks to expand the clique by adding a vertex from PA. When the current clique cannot be extended, a plateau search phase is triggered during which vertices of the current clique are swapped with vertices in OM. If the current clique becomes expandable, the search switches back to the expansion phase. When no add and swap moves are possible, some perturbation strategies are applied to the clique C. DLS (Pullan & Hoos, 2006) follows this scheme and uses dynamic vertex penalties to guide vertex selection during the clique expansion and plateau search phases. The perturbation mechanism simply reduces the clique C to the last added vertex v, or adds a vertex v that is chosen at random and then removes all vertices from C that are not connected to v. To cope with graphs of different

structures, PLS (Pullan, 2006) combines three sub-algorithms which use different vertex selection rules: random selection, random selection among those with the highest vertex degree or random selection within those with the lowest vertex penalty. In addition, the perturbation mechanism differs between the sub-algorithms. CLS (Pullan et al., 2011) further improves over PLS by following the idea of the hyper-heuristic scheme (Burke, Hart, Kendall, Newall, Ross, & Schulenburg, 2003). CLS incorporates four low level heuristics which are effective for different instance types. The difference between these low level heuristics is their vertex selection techniques and the way they deal with plateaus. To further enhance the performance of CLS, relevant information is passed between low level heuristics in order to guide the search to particular areas of the search domain. DLS, PLS and in particular CLS show excellent performances on both DIMACS and BOSHLIB benchmarks.

The study of Andrade et al. (2012) explores general swap operator in the context of the legal strategy: The $(1, 2)$-swap move, which exchanges a single vertex from C against two other vertices, and the $(2, 3)$-swap move, which removes two vertices from C and adds three new vertices. This study also shows that, given any maximal clique, a $(1, 2)$-swap move can be implemented in a linear time, while a $(2, 3)$-swap move can be implemented in $O(m\Delta)$, where m is the number of edges and Δ is the highest degree in the graph. These swap operators are integrated in an iterated local search (ILS) method. The results on the DIMACS and BHOSLIB benchmarks show that ILS performs very well on two large and difficult MANN instances. However, it is dominated by some best MCP algorithms on some large brock family instances from DIMACS as well as some BHOSLIB instances.

The multi-neighborhood tabu search (MN/TS) (Wu et al., 2012) for the weighted and unweighted MCP is based on a combined neighborhood induced by the add, swap and drop moves. For the unweighted case, the vertex selection rule is based on a random scheme: For the add move, a vertex $v \in$ PA is randomly picked to expand C, and for the swap move, a vertex $v \in$ OM is randomly selected to exchange with the only vertex $u \in C$ which is not connected to v, while for the drop move, a vertex is randomly removed from C. The general prohibition rule in MN/TS is similar to that of the deterministic tabu search procedure of Gendreau et al. (1993): Only vertices that leave the clique C are forbidden to move back to C during the prohibition period, vertices that join the clique can be removed without restriction. MN/TS obtains the best-known solutions for all the BHOSLIB instances and 78 out of the 80 DIMACS instances.

The breakout local search (BLS) (Benlic & Hao, 2013) for the MCP and its vertex weight version uses the union of the add and swap moves in its search procedure and the drop and add-repair moves during its perturbation procedure. BLS explores first search areas around the current local optimum and displaces the search into a new distant area only if the search seems to be stagnating. This is achieved by its specific perturbation strategy which determines dynamically and adaptively the type and strength of perturbations. BLS reaches competitive results on both DIMACS and BHOSLIB instances.

Very recently, a general swap-based tabu search (SBTS) algorithm (Jin & Hao, 2015) is introduced for the equivalent MIS. Each iteration of SBTS corresponds to either an intensification step or a diversification step by applying a general $(k, 1)$-swap ($k \geq 0$) operator. Given an independent set S, $(k, 1)$-swap exchanges one vertex (which is strategically selected) in $V \backslash S$ against its k adjacent vertices in S. SBTS alternates dynamically between its $(k, 1)$-swap based neighborhoods to find improved solutions or escape from local optima. SBTS is currently the single heuristic able to attain the best-known result for all DIMACS and BHOSLIB instances.

4.3. Evolutionary algorithms and other popular heuristics for the MCP

Evolutionary algorithms like genetic algorithms have long been a popular approach for the MCP. As opposed to local search, this approach uses a population of solutions that are improved via an "evolutionary" process. Early attempts are reported in 1990s. However, pure genetic algorithms are not effective for the MCP (Carter & Park, 1993). This approach is often enhanced by incorporating other techniques like local optimization. A first example is the genetic tabu search algorithm of Fleurent and Ferland (1996a) which combines a standard uniform crossover and a tabu search procedure. Another hybrid genetic algorithm (GENE) mixes a uniform crossover, a swap mutation and a descent method (Marchiori, 2002). A hybrid evolutionary algorithm with a guided mutation (EA/G) for the problem is introduced in Zhang, Sun, and Tsang (2005). The HSSGA algorithm by Singh and Gupta (2006a) is a combination of a steady-state genetic algorithm, a randomized sequential greedy approach and the exact algorithm of Carraghan and Pardalos (1990) while IEA-PTS blends an impatient evolutionary algorithm and a probabilistic tabu search (Guturu & Dantu, 2008).

Despite numerous attempts, the reported evolutionary algorithms for the MCP are not competitive compared to simpler local search algorithms. This may be partially explained by the fact that for the MCP, no meaningful recombination (or crossover) operator is known yet. Indeed it is not clear how random crossover operators can help the search to discover improved cliques. Even the most effective hybrid evolutionary algorithms for the MCP are basically driven by some local search procedures. However, better performances could be achieved if a semantically meaningful crossover for the MCP can be devised.

Other popular heuristics for the MCP include ant colony optimization (Solnon & Fenet, 2006), artificial neural networks (Yang, Yi, Zhang, & Tang, 2009) and methods based on continuous optimization (Burer, Monteiro, & Zhang, 2002; Busygin, 2006). According to the reported results, the overall performance of these heuristics are not competitive compared with recent local search methods. Finally, it is worthwhile to mention that some recent edge weighting local search approaches for the equivalent minimum vertex cover problem, such as COVER (Richter, Helmert, & Gretton, 2007) and EWCC (Cai, Su, & Sattar, 2011), report very good results on the DIMACS and BHOSLIB benchmarks.

To conclude this section, we summarize in Table 2 the main reviewed heuristic algorithms (listed in chronological order). In Section 6, we provide more information about the relative performance of these heuristic algorithms.

5. Generalizations and relaxations of the MCP

The MCP has some relevant generalizations and relaxations which are useful to formulate a number of practical applications where the standard MCP is not appropriate. These problems are not only gaining increasing popularity in a number of practical applications in particular in graph data analysis and mining, but also keeps garnering attention as a promising avenue for theoretical investigations.

5.1. Maximum vertex weight clique problem (MVWCP)

Given $G = (V, E)$, let $w : V \rightarrow Z^+$ be a weighting function that assigns to each vertex $i \in V$ a positive value. For a clique C of G, define its weight as $W(C) = \sum_{i \in C} w_i$. The MVWCP is to determine a clique C^* of maximum weight, i.e., $\forall C \in \Omega$, $W(C^*) \geq W(C)$ where Ω is the set of all possible cliques of the graph. The MCP can be considered as a special case of the MVWCP where the weight of each vertex is set equal to 1. It is clear that a maximum clique of the MCP does not necessarily lead to a maximum vertex weight clique of the MVWCP, and the MVWCP has at least the same computational complexity as the MCP.

Some exact algorithms for the MVWCP come from and generalize previous methods designed for the unweighted MCP (Kumlander, 2004; Östergård, 1999). Three other B&B algorithms are introduced in Warren and Hicks (2006), which use weighted clique covers to

Table 2
Main heuristic algorithms for the maximum clique problem.

Algorithm name	Reference	Type of approach	Comments on performance
STABULUS	Friden et al. (1989)	Tabu search based on the k-fixed penalty strategy	The first MIS algorithm using tabu search
DT, PT	Gendreau et al. (1993)	Tabu search based on the legal strategy	Two early tabu search algorithms without swap moves
AI, AW, NA	Jagota and Sanchis (2001)	Greedy algorithm	Three adaptive restart randomized greedy heuristics, the results of these three heuristics are not available on the DIMACS and BHOSLIB benchmarks
RLS	Battiti and Protasi (2001)	Reactive tabu search based on the legal strategy	A landmark MCP algorithm, reports better results than its predecessors
GENE	Marchiori (2002)	Evolutionary algorithm	An early hybrid genetic algorithm, outperformed by many local search heuristics
DAGS	Grosso et al. (2004)	Greedy algorithm	A sophisticated greedy algorithm achieving better performances than other greedy algorithms like QUALEX-MS on most DIMACS instances
VNS	Hansen et al. (2004)	Variable neighborhood search based on the legal strategy	Shows a good performance on the MANN instances from DIMACS, its overall performance is comparable in terms of quality with RLS
KLS	Katayama et al. (2005)	Local search based on the legal strategy	Achieves a good performance on the MANN instances from DIMACS but a bad performance on the keller and brock instances
EA/G	Zhang et al. (2005)	Evolutionary algorithm	Achieves better performances than an early evolutionary algorithm GENE, but performs slightly less well than RLS
QUALEX-MS	Busygin (2006)	Greedy algorithm	Shows good performances on the brock instances from DIMACS
DLS	Pullan and Hoos (2006)	Dynamic local search based on the legal strategy	One of the best performing algorithms, shows highly competitive results compared with a number of state-of-the-art algorithms before DLS
PLS	Pullan (2006)	Phased local search based on the legal strategy	Achieves similar performances compared to DLS on the DIMACS benchmarks
HSSGA	Singh and Gupta (2006a)	Evolutionary algorithm	Performs worse than many modern local search algorithms like RLS, DLS, RLS, PLS and AMTS
COVER	Richter et al. (2007)	Edge weighting local search approach	Shows a good performance for the two large MANN instances, but fails to reach the best-known results for some brock family instances from DIMACS
IEA-PTS	Guturu and Dantu (2008)	Evolutionary algorithm	Outperforms two other evolutionary algorithms EA/G and HSSGA, reports competitive results on the DIMACS benchmarks, but fails to reach the best-known results for some BHOSLIB benchmarks
CLS	Pullan et al. (2011)	Cooperating local search based on the legal strategy	A fast algorithm using multiple strategies, one of the current best performing MCP algorithms
EWCC	Cai et al. (2011)	Edge weighting local search approach	A recent edge weighting local search approach for the equivalent minimum vertex cover problem, reports competitive results on the DIMACS and BHOSLIB benchmarks.
MN/TS	Wu et al. (2012)	Multi-neighborhood tabu search based on the legal strategy	Fails to reach the best-known results for two large MANN instances
ILS	Andrade et al. (2012)	Local search based on the legal strategy	Achieves an excellent performance on two large MANN instances, but fails to reach the best-known results for some brock instances from DIMACS and some BHOSLIB instances
AMTS	Wu and Hao (2011)	Tabu search based on the k-fixed penalty strategy	Shows an overall good performance on the DIMACS benchmarks, but slower than DLS
BLS	Benlic and Hao (2013)	Breakout local search based on the legal strategy and adaptive perturbation	Attains, except two large MANN instances, the best-known results for the DIMACS and BHOSLIB benchmarks, but slower than DLS
SBTS	Jin and Hao (2015)	General $(k, 1)$-swap tabu search based on the legal strategy and	Shows an overall good performance on the DIMACS and BHOSLIB benchmarks

generate upper bounds and branching rules. Several heuristics are also available to find sub-optimal solutions for the MVWCP: Augmentation algorithm (Mannino & Stefanutti, 1999), distributed computational network algorithm (Bomze, Pelillo, & Stix, 2000), complementary pivoting algorithm (Massaro, Pelillo, & Bomze, 2001), hybrid evolutionary approach (Singh & Gupta, 2006b), adaptation to the weight version of the phased local search (Pullan, 2008) for the unweighted case, multi-neighborhood tabu search algorithm (Wu et al., 2012) and breakout local search (Benlic & Hao, 2013).

5.2. Maximum edge weight clique problem (MEWCP)

Given an integer m and a complete graph $G = (V, E)$, each edge $\{i, j\} \in E$ being associated with a weight d_{ij}. The MEWCP is to find a clique C in G such that the sum of the weights of the edges in C is maximized and the number of vertices in C is no more than m. Like the MVWCP, the decision version of the MEWCP is NP-complete (Ausiello, Crescenzi, Gambosi, Kann, Marchetti-Spaccamela, & Protasi, 1999), as far as it reduces to the maximum clique problem. The MEWCP or its variants are also studied under other names: Maximum diversity (Martí, Gallego, Duarte, & Pardo, 2013; Palubeckis, 2007), maxisum dispersion (Kuby, 1987), MAX-AVG dispersion (Ravi, Rosenkrantz, & Tayi, 1994), remote-clique (Chandra & Halldórsson, 2001), and maximum edge-weighted subgraph (Macambira, 2003).

Early exact methods for the MEWCP can be found in Dijkhuizen and Faigle (1993) and Park, Lee, and Park (1996). Three more recent examples of B&B algorithms are provided in Aringhieri, Bruglieri, and Cordone (2009); Martí, Gallego, and Duarte (2010) and Sørensen (2004) which can solve problems with up to 150 vertices. On the other hand, there are many heuristics devoted to the MEWCP: Tabu search (Aringhieri & Cordone, 2011; Macambira, 2003), iterated tabu search (Palubeckis, 2007), iterated greedy algorithm (Lozano, Molina, & García-Martínez, 2011), greedy randomized adaptive search procedure (Andrade, de Andrade, Martins, & Plastino, 2005; Silva, De Andrade, Ochi, Martins, & Plastino, 2007), genetic algorithm (Feng, Jiang, Fan, & Fu, 2010), variable neighborhood search (Brimberg, Mladenović, Urošević, & Ngai, 2009), scatter search (Gallego, Duarte, Laguna, & Martí, 2009; Gortázar, Duarte, Laguna, & Martí, 2010), path-relinking method (Andrade et al., 2005) and memetic search (Wang, Hao, Glover, & Lü, 2014; Wu & Hao, 2013).

Comprehensive surveys and comparisons of the most significant heuristic and metaheuristic methods for the MEWCP before 2011 can be found in Aringhieri and Cordone (2011) and Martí et al. (2013).

5.3. Maximum quasi-clique problem

Given a graph $G = (V, E)$ and a real number $\gamma \in [0, 1]$, a γ-quasi-clique (γ-clique for short) is defined as a subset of vertices $S \subseteq V$ such that the number of edges in the subgraph induced by S is at least $\gamma\binom{|S|}{2}$. Then the maximum γ-clique problem is to find a γ-clique of maximum cardinality in the given graph (Definition 1, see references like Bourgeois, Giannakos, Lucarelli, Milis, Paschos, and Pottié 2012). An alternate definition of a γ-quasi-clique can also be found in the literature like Pei, Jiang, and Zhang (2005) and Zeng, Wang, Zhou, and Karypis (2007) where $S \subseteq V$ is called a γ-quasi-clique if every vertex in S has at least $\gamma(|S| - 1)$ adjacent vertices in S. Obviously, this definition is more restrictive than Definition 1 since $S \subseteq V$ satisfying the minimum degree requirement is also a γ-quasi-clique according to Definition 1. However, the converse is not necessarily true as a γ-quasi-clique by Definition 1 could include vertices which have less than $\gamma(|S| - 1)$ adjacent vertices in S. A more general definition is provided in Brunato, Hoos, and Battiti (2008) where $S \subseteq V$ is called a (λ, γ)-quasi-clique if the number of edges in the subgraph induced by S is at least $\gamma\binom{|S|}{2}$ and every vertex in S has at least $\lambda(|S| - 1)$ adjacent vertices in S. The maximum γ-clique problem is shown to be NP-complete in Patillo, Veremyev, Butenko, and Boginski (2013) for any fixed γ satisfying $0 < \gamma < 1$.

Mathematical programming formulations, effective pruning techniques and exact methods are also proposed in the literature to solve the problem exactly. Mixed-integer programming formulations for the maximum γ-clique problem are described in Patillo et al. (2013) and report preliminary results obtained with a modern commercial MIP solver. A B&B algorithm for the problem is recently developed in Pajouh, Miao, and Balasundaram (2014). There are several heuristics available to find sub-optimal solutions in various application scenarios. A heuristic algorithm (Matsuda, Ishihara, & Hashimoto, 1999) is developed to find large γ-cliques for classifying a large and mixed set of uncharacterized biological sequences. GRASP procedures are studied in Abello, Resende, and Sudarsky (2002) for detecting large quasi-cliques in graphs representing telecommunications data. Two existing local search algorithms for the MCP are extended to the quasi-clique problem (Brunato et al., 2008). A greedy algorithm for a quasi-clique finding problem is proposed within the context of studying the human protein–protein interaction networks (Bhattacharyya & Bandyopadhyay, 2009).

5.4. Densest k-subgraph problem

Given a graph $G = (V, E)$ and an integer $k \leq |V|$, the densest k-subgraph problem is to find a subgraph of k vertices with the largest number of edges in the subgraph. Its edge weight version (called the heaviest k-subgraph problem) is to find a k-vertices induced subgraph with the maximum total edge weight. The densest k-subgraph problem is NP-hard in the general case (Feige, Kortsarz, & Peleg, 2001).

There are some solution approaches for the problem. For instance, several integer programming formulations are studied in Billionnet (2005). A B&C algorithm to solve the problem exactly is introduced in Bourgeois, Giannakos, Lucarelli, Milis, and Paschos(2013) (2013). A variable neighborhood search algorithm is described in Brimberg et al. (2009) for an edge weight version of the problem.

Finally, there are some other interesting models related to the MCP, such as the maximum k-plex problem (Balasundaram et al., 2011), the maximum k-club problem (Bourjolly, Laporte, & Pesant, 2002) and the maximum/minimum edge neighborhood density clique problem (Martins, 2012).

6. Benchmarks and performance evaluation

This section is devoted to performance evaluation of MCP algorithms. First of all, one must keep in mind that performance evaluation is a delicate task since many factors can affect such an assessment like the programming language and the data structures used to code the algorithm, the computing plate-form, the experimental protocol (stop condition, parameter tuning...). Moreover, evaluation criteria may also become a delicate issue since a criterion suitable for an exact algorithm may be unsuitable for a heuristic, and vice versa. Even for a given type of algorithms, several criteria would be possible. To simplify the presentation and highlight some main trends, we focus on two most popular indicators: the quality of solutions (main indicator) and the computing time (secondary indicator). In any case, the comparisons presented in this section are only for indicative purposes and should be interpreted with caution.

6.1. Benchmarks

For the purpose of computational assessment of MCP algorithms, there are two main benchmarks available in the literature. The DIMACS benchmark is historically the most popular while the BHOSLIB benchmark is more recent.

6.1.1. DIMACS benchmark

The DIMACS benchmark set is created in the 1990's for the second DIMACS challenge on Clique, Satisfiability and Graph Coloring (Johnson & Trick, 1996).[1] It is composed of 80 graphs and represents the standard set for evaluating MCP algorithms. The benchmark covers a number of real-world problems from e.g., coding theory, fault diagnosis problems and Keller's conjecture, etc., in addition to randomly generated graphs and graphs where the maximum clique is hidden by incorporating low-degree vertices. The size of the DIMACS instances ranges from less than 50 vertices and 1000 edges up to more than 3300 vertices and 5,000,000 edges. According to the results reported in the literature, among these 80 DIMACS instances, the maximum clique is now known for most of them except for four graphs: three (large and dense) random graphs with at least 500 vertices (C500.9, C1000.9, C2000.9) and 1 structured graph (johnson32_2_4).

6.1.2. BHOSLIB benchmark

The BHOSLIB benchmark set is composed of 40 graphs with hidden optimal solutions (Xu, Boussemart, Hemery, & Lecoutre, 2007),[2] which are translated from hard random SAT instances generated at the exact phase transition of the RB model (Xu, Boussemart, Hemery, & Lecoutre, 2005). These instances have sizes ranging from 450 vertices and 17,794 edges up to 1534 vertices and 12,7011 edges. These instances are supposed to be hard theoretically for exact algorithms and only few exact algorithms (see for instance McCreesh and Prosser 2013) report results on these instances. On the other hand, several recent heuristic algorithms can attain the known optimal solutions for these instances with no particular difficulty (Benlic & Hao, 2013; Cai, Su, & Chen, 2010; Cai et al., 2011; Grosso, Locatelli & Pullan, 2008; Jin & Hao, 2015; Pullan et al., 2011; Richter et al., 2007; Wu et al., 2012).

In addition to these benchmarks, a set of instances from code theory (less used in the literature) is available at http://neilsloane.com/doc/graphs.html.

6.2. Indicative performance evaluation

To compare different exact algorithms for the MCP, we summarize in Table 3 the computational results of 10 recent exact algorithms on

[1] ftp://dimacs.rutgers.edu/pub/challenge/graph/.
[2] http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm.

Table 3
Performance comparison of 10 stat-of-art exact MCP algorithms on 35 popular DIMACS instances. Entries "–" signify that the data is unavailable. The average running time of each algorithm is adjusted and expressed in seconds of the computer used in Li and Quan (2010). The compared algorithms are: $\chi + DF$ (Fahle, 2002), Cliquer (Östergård, 2002), CPR (Régin, 2003), MaxCliqueDyn (Konc & Janežič, 2007), MCQ (Tomita & Seki, 2003), MCR (Tomita & Kameda, 2007), MCS (Tomita et al., 2010), MaxCLQ (Li & Quan, 2010), BBclique (Segundo et al., 2011) and ILS&MaxCLQ (Maslov et al., 2014).

Instance	$\chi + DF$	Cliquer	CPR	MaxCliqueDyn	MCQ	MCR	MCS	MaxCLQ	BBclique	ILS&MaxCLQ
brock200_1	52.83	6.37	4.60	0.96	1.79	1.13	–	0.67	0.43	11.05
brock400_1	–	22182.00	4867.00	703.50	1905.22	1137.00	521.05	370.84	433.44	428.70
brock400_2	–	5617.00	3395.00	309.00	787.49	465.10	223.30	178.70	182.08	214.83
brock400_3	–	1667.00	1922.00	565.00	1501.30	766.51	351.87	290.06	289.81	341.69
brock400_4	–	247.70	2597.00	320.40	772.52	409.43	186.466	167.30	172.57	160.34
brock800_1	–	–	–	8821.00	–	10712.00	7027.81	8815.00	7413.90	8649.49
brock800_2	–	–	–	8125.00	–	9679.00	6305.26	7690.00	6768.85	7575.36
brock800_3	–	26014.00	–	5565.00	11143.79	6546.00	4327.06	5285.00	4364.83	5008.19
brock800_4	–	6108.00	–	4240.00	8589.54	4561.00	3005.26	3880.00	3080.45	3800.02
MANN_a27	5891.03	–	7.93	3.10	4.52	1.98	0.60	0.66	0.47	3.07
MANN_a45	–	–	–	2006.00	3581.69	2931.00	211.27	255.67	209.13	58.28
hamming10-2	2.92	0.19	0.45	2.26	0.84	0.16	–	7.92	0.16	10.07
keller5	–	–	–	31038.00	–	–	–	9687.00	–	9227.09
p_hat300-3	487.73	496.60	17.47	4.91	20.01	7.45	1.87	2.07	1.65	18.38
p_hat500-2	116.03	134.60	14.03	1.53	4.29	2.12	0.52	0.90	0.52	70.48
p_hat500-3	–	–	5470.00	349.40	3714.37	1256.00	112.78	55.95	109.88	115.49
p_hat700-1	1.50	0.09	2.58	0.14	0.11	0.07	–	0.80	0.06	251.86
p_hat700-2	1187.73	15417.00	109.80	12.60	57.18	30.19	4.21	4.87	4.71	135.83
p_hat700-3	–	–	–	6187.00	–	–	1798.49	1033.00	2344.24	1402.21
p_hat1000-1	9.43	1.11	11.93	0.58	0.67	0.35	–	2.53	0.39	519.46
p_hat1000-2	–	–	7230.00	412.90	3391.50	1656.00	166.16	146.54	234.98	474.47
p_hat1000-3	–	–	–	–	–	–	–	200760.00	–	228357.60
p_hat1500-1	68.39	8.01	206.40	4.31	5.48	2.97	–	15.85	3.81	1202.00
p_hat1500-2	–	–	–	61461.00	–	–	12415.03	8848.00	–	14136.59
san1000	1732.54	0.08	44.12	0.74	7.43	3.35	1.57	1.46	0.75	794.33
san200_0.9_2	1099.05	13.36	1.12	0.79	3.60	2.92	0.30	0.10	0.16	3.44
san200_0.9_3	110.84	503.40	78.41	3.43	11.69	0.11	–	0.22	0.04	4.04
san400_0.7_1	242.45	–	9.99	0.52	1.85	1.09	0.40	0.21	0.30	44.10
san400_0.7_2	91.03	3081.00	28.98	0.20	1.17	0.21	0.09	0.09	0.12	56.21
san400_0.7_3	351.41	4.47	117.30	2.04	3.79	2.12	1.05	0.75	0.66	59.26
san400_0.9_1	4109.43	–	729.60	30.95	1377.77	2.50	0.07	1.97	0.20	10.90
sanr200_0.7	14.15	1.88	1.85	0.39	0.60	0.37	0.25	0.38	0.16	12.75
sanr200_0.9	–	46593.00	64.41	51.57	381.21	204.72	30.82	5.72	23.96	8.37
sanr400_0.5	13.20	0.97	7.35	0.74	0.87	0.52	–	1.73	0.37	85.49
sanr400_0.7	9065.09	2852.00	1347.00	177.80	394.05	237.26	136.09	141.48	115.15	198.61

the DIMACS benchmark. To be concise, we exclude the very easy instances that are solved by most of the reviewed algorithms in less than 2 seconds and the 10 hard instances on which few exact algorithms reports their results (C500.9, C1000.9, C2000.5, C2000.9, C4000.5, MANN_a81, hamming10-4, johnson32-2-4, keller6, and p_hat1500-3). The results of Cliquer, CPR, MaxCliqueDyn, MCR and MaxCLQ are directly extracted from Li and Quan (2010), while the results of $\chi + DF$, MCS, BBclique and ILS&MaxCLQ are taken from the papers describing these algorithms and the results of MCQ are obtained from Konc and Janežič (2007). The running times shown in Table 3 are all adjusted and expressed in seconds of the computer used in Li and Quan (2010) to run MaxCLQ. The method for adjusting the running times is as follows. If the original paper provides machine benchmarking information on running the DIMACS MCP Machine Benchmark program (ftp://dimacs.rutgers.edu in directory /pub/dsj/clique) on the three benchmark graphs r300.5, r400.5 and r500.5, then the CPU time is transformed using the benchmark information. Otherwise, the CPU time is adjusted using the SPEC - Standard Performance Evaluation Corporation (www.spec.org). To interpret the data in Table 3, we should keep in mind that the results are often obtained under different time limits and the SPEC time conversion is not as precise as running the DIMACS Machine Benchmark program.

Comparative studies of exact algorithms are available in a number of studies like Carmo and Züge (2012); Konc and Janežič (2007); Li and Quan (2010); Östergård (2002); Prosser (2012); Régin (2003); Segundo et al. (2013, 2011); Tomita and Kameda (2007); Tomita and Seki (2003) and Tomita et al. (2010). A summary of 10 most effective exact algorithms is provided in Table 3. From these results, one finds that the most effective algorithms are those that use vertex coloring for both bounding and branching like MCQ (Tomita & Seki, 2003), MCR (Tomita & Kameda, 2007), MCS (Tomita et al., 2010), MaxCliqueDyn (Konc & Janežič, 2007) and BB-MaxClique (Segundo et al., 2011), and the MaxSAT based algorithms like MaxCLQ (Li & Quan, 2010). In Carmo and Züge (2012); Li and Quan (2010) and Tomita and Seki (2003), extensive comparisons are presented between the coloring based algorithm MCQ and some other algorithms before MCQ, like Cliquer (Östergård, 2002) and $\chi + DF$ (Fahle, 2002), disclosing that MCQ is superior on a wide range of DIMACS instances. Given that $\chi + DF$ uses coloring only for bounding, the comparative results between MCQ and $\chi + DF$ also indicate that it is better to use coloring as an aid for both branching and bounding than for bounding only. MCR, MCS, MaxCliqueDyn and BB-MaxClique can be viewed as improved versions of MCQ by embedding more effective branching rules. When comparing the MaxSAT based algorithms like MaxCLQ and the coloring based algorithms like MCR, MCS and MaxCliqueDyn, one observes that MaxCLQ solves most of the large and dense DIMACS instances faster than other coloring based algorithms while the reverse is true for some small instances (Li & Quan, 2010; Maslov et al., 2014).

Similarly, Table 4 summarizes the results of 21 heuristics on 9 representative DIMACS instances. These instances are selected since they are known to be hard for most existing MCP heuristic algorithms and thus can be used to differentiate the compared heuristics. The results in terms of solution quality (i.e., best and average clique size) are directly extracted from the original papers while the average CPU running times are adjusted and expressed in seconds of the computer used by EWCC (Cai et al., 2011) using the same method as for exact algorithms of Table 3. The solution quality (i.e., clique size)

Table 4
Performance of 21 effective heuristics on the 9 hard DIMACS instances, entries of "–" signify that the data is not available. The average clique size is indicated in the brackets when a 100 percent success rate is not reached. The average running time required by each algorithm is adjusted and expressed in seconds of the computer used in Cai et al. (2011). The compared algorithms are: RLS (Battiti & Protasi, 2001), GENE (Marchiori, 2002), DAGS (Grosso et al., 2004), VNS (Hansen et al., 2004), KLS (Katayama et al., 2005), EA/G (Zhang et al., 2005), QUALEX-MS (Busygin, 2006), DLS (Pullan & Hoos, 2006), PLS (Pullan, 2006), HSSGA (Singh & Gupta, 2006a), COVER (Richter et al., 2007), IEA-PTS (Guturu & Dantu, 2008), CLS (Pullan et al., 2011), EWCC (Cai et al., 2011), MN/TS (Wu et al., 2012), ILS (Andrade et al., 2012), AMTS (Wu & Hao, 2011), BLS (Benlic & Hao, 2013) and SBTS (Jin & Hao, 2015).

Instance	brock400_2		brock400_4		brock800_2		brock800_4		C2000.9		C4000.5		MANN_a45		MANN_a81		keller6	
	Best(*Avg*)	*Time*	*Best*(*Avg*)	*Time*	*Best*(*Avg*)	*Time*	*Best*(*Avg*)	*Time*	*Best*(*Avg*)	*Time*	*Best*(*Avg*)	*Time*	*Best*(*Avg*)	*Time*	*Best*(*Avg*)	*Time*	*Best*(*Avg*)	*Time*
RLS	29(26.06)	3.06	33(32.42)	7.89	21	0.34	21	0.48	78(77.58)	59.83	18	158.65	345(343.60)	28.98	1098	205.72	59	13.79
GENE	24(22.5)	0.27	25(23.6)	0.19	20(19.3)	0.75	20(18.9)	1.12	72(68.2)	4.89	16(15.4)	1.95	343(342.4)	19.56	1097(1096.3)	401.41	55(51.8)	8.71
DAGS	29(28.10)	0.62	33	0.62	24(20.82)	3.73	26(22.60)	3.75	76(75.40)	405.33	18(17.50)	717.55	344(343.95)	426.95	–	–	57(56.40)	2739.11
VNS	29(27.4)	4.17	33	2.69	21	0.85	21	3.16	78(77.2)	22.74	18	310.71	345(344.5)	1.51	1100(1099.3)	65.47	59(58.2)	17.90
KLS	25(24.84)	0.04	25	0.01	21(20.86)	0.16	21(20.67)	0.39	77(74.90)	4.80	18(17.02)	7.76	345(343.88)	2.02	1100(1098.07)	12.88	57(55.59)	17.08
EA/G	25(24.7)	1.42	33(25.1)	1.42	21(20.1)	3.42	21(19.9)	3.42	72(70.9)	17.38	17(16.1)	23.46	345(343.7)	30.84	1098(1097.2)	319.04	56(53.4)	24.26
QUALEX-MS	29	0.67	33	0.44	24	4.00	26	4.00	72	47.78	17	521.11	342	3.78	1096	106.00	53	286.89
DLS	29	0.12	33	0.02	24	3.97	26	2.24	78(77.93)	48.79	18	45.76	344	13.12	1098(1097.96)	66.66	59	43.05
PLS	29	0.12	33	0.02	24	4.08	26	2.30	78	50.11	18	47.01	344	84.27	1098	172.17	59(57.75)	172.17
HSSGA	29(25.1)	0.14	33(27.0)	0.29	21(20.7)	1.35	21(20.1)	0.38	74(71.0)	14.83	17(16.8)	19.97	343(342.6)	8.22	1095(1094.2)	503.99	57(54.2)	39.67
COVER	28(–)	–	33(–)	42.63	–	–	–	–	78(77.84)	139.36	18	260.27	345(344.41)	–	1100(–)	–	59	5.89
IEA-PTS	29(27.52)	1.08	33	0.84	24(21.06)	1.03	26(21.4)	2.07	79(76.4)	19.71	18(17.66)	104.21	345(343.97)	9.43	1099(1097.01)	237.55	59(57.06)	45.40
CLS	29	0.08	33	0.02	24	1.73	26	0.58	78	7.28	18	13.63	344	15.40	1098	20.80	59	0.91
EWCC	29(25.48)	374.52	33	25.37	21	0.49	21	0.62	79(78.56)	858.73	18	738.91	345(344.94)	698.50	1100(1098.11)	634.46	59	3.76
MN/TS	29	0.81	33	0.17	24(23.88)	94.25	26	37.57	80(78.37)	339.57	18	86.96	340	54.56	1090	380.87	59	58.96
ILS	25	11.57	33(30.3)	11.57	21	63.15	26(21.3)	63.15	77(76.9)	108.42	18(17.1)	1996.84	345(344.5)	3.15	1100	10.52	59	546.31
AMTS	29	0.69	33	0.35	24	19.61	26	9.01	80(78.95)	266.33	18	74.93	345(344.04)	66.77	1098	16.30	59	6.39
BLS	29	10.29	33	1.87	24(23.04)	637.94	26	356.05	80(78.6)	2846.84	18	387.33	342(340.82)	–	1094(1092.17)	–	59	14.67
SBTS	29	11.97	33	0.49	24(22.29)	464.12	26(25.90)	249.47	80(77.29)	896.78	18	919.07	345	16.30	1100	13.43	59	446.67

is the primary criterion while the computing times are provided for indicative purposes.

From Table 4 as well as the results reported by the original papers, one first observes that the newest heuristic SBTS (Jin & Hao, 2015) is the only method able to attain the best-known clique size for these 9 graphs (in fact this remains true for all DIMACS and BHOSLIB instances) while the other heuristics miss at least one best-known result. Secondly, algorithms based on the vertex penalty mechanism such as DLS (Pullan & Hoos, 2006), PLS (Pullan, 2006), CLS (Pullan et al., 2011) perform very well on the brock instances, but have troubles to find an optimal (or best known) solution for the large MANN instances. On the other hand, algorithms designed for the vertex cover problem (EWLS (Cai et al., 2010), EWCC (Cai et al., 2011), COVER (Richter et al., 2007)) and two other algorithms (VNS (Hansen et al., 2004) and KLS (Katayama et al., 2005)) are able to reach the optimal (or best known) solution for the two large MANN instances, but perform less well on some large brock graphs.

In light of the computational results of Table 4 as well as the analysis in Cai et al. (2011); Jin and Hao (2015); Pullan and Hoos (2006); Pullan (2006); Pullan et al. (2011); Richter et al. (2007); Wu and Hao (2011), the dominant heuristics are DLS (Pullan & Hoos, 2006), PLS (Pullan, 2006), COVER (Richter et al., 2007), CLS (Pullan et al., 2011), EWCC (Cai et al., 2011), AMTS (Wu & Hao, 2011), SBTS (Jin & Hao, 2015). As shown in Pullan and Hoos (2006), DLS dominates previous state-of-the-art heuristics KLS (Katayama et al., 2005), RLS (Battiti & Protasi, 2001) and DAGS (Grosso et al., 2004) on several DIMACS instances. In Pullan (2006), PLS is compared directly with DLS using the assessment criteria listed in Pullan and Hoos (2006), showing a comparable or better performance than DLS for almost all DIMACS instances. In Richter et al. (2007); Cai et al. (2011); Pullan et al. (2011) and Wu and Hao (2011), EWCC, CLS, COVER and AMTS are compared with DLS or PLS, showing that they are as competitive as or even more efficient than DLS or PLS. Therefore, we can roughly conclude that EWCC, SBTS, DLS, PLS, CLS, COVER and AMTS are among the most effective heuristic algorithms currently available for the MCP.

7. Perspectives and conclusion

Based on this review, we now take on the challenge of discussing some perspective research directions.

7.1. Improving exact methods

From the perspective of exact solution of the MCP, we consider two directions for further improvement. First, as shown in Section 3, the most popular exact algorithms use a (very simple) graph coloring procedure to estimate their upper bounds. Since the quality of the coloring impacts directly the quality of the bounds, it would be interesting to investigate recent and more powerful coloring algorithms. At the same time, since the coloring procedure is repeatedly called as a subroutine in the B&B process, a compromise between the coloring quality and the computing time needs to be reached.

Second, it is possible to take advantages of both exact and heuristic methods and combine these two approaches. For instance, before the branch-and-bound routine starts, we could use a highly effective local search procedure to obtain some high quality initial solutions, which could be served then as lower bounds of the main B&B algorithm. Such a combination was very recently investigated in Maslov et al. (2014) and proves to be effective. In addition to this loose combination, we can consider more subtle cooperation between these two search methods. For instance, during the B&B process, for each active node of the search tree, one can call a (fast) local search procedure which determines a best possible path from the current node to the leaf of the search tree, leading thus to a hopefully good lower bound of the maximum clique.

Other ways of integrating heuristics with exact methods can also be considered. For instance, information from high quality solutions found in several runs of a heuristic can be used to define smaller problems solvable by an exact algorithm. This is motivated by the observation that many optimization problems show some type of "backbone" structure in the sense that high quality solutions share a number of solution components with optimal solutions. We could use an effective local search method to collect some high quality solutions, then fix the vertices which are shared by all these high quality solutions. By excluding the vertices which are adjacent to all these preserved vertices shared by the high quality solutions, we obtain a subproblem which may be small enough to be solved by exact algorithms. Notice that such an approach does not always guarantee the completeness of the search.

7.2. Improving heuristic methods

As shown in Section 4.2, local search is the most popular and globally the most effective heuristic approach for the clique problems. However, if we examine the existing local search algorithms on an individual base, we observe that very few single algorithm dominates all the other algorithms. This happens because when the search operators used in an algorithm are suitable to explore the structures of some graphs, these same operators may fail to handle other graphs with quite different structures. One possible way to mitigate this deficiency would be to incorporate multiple search operators or strategies within a single algorithm and equip the algorithm with a capacity to dynamically decide the most suitable operators to trigger during the search process. Moreover, it would also be interesting to explore a portfolio approach using several algorithms.

From the population-based search perspective, as discussed in Section 4.3, the most effective algorithms of this family for the MCP are memetic methods which combine the genetic search framework and local search. Unfortunately, these complex hybrid methods are often surpassed by simpler local search heuristics for the MCP. One reason is that the recombination (crossover) operators used in these algorithms are not really meaningful with respect to the clique problem. On the other hand, it is now well acknowledged that with the memetic framework, a carefully designed recombination operator, i.e., able to recombine the "building blocks" of the given problem, constitutes a key factor for the effectiveness of the algorithm (Hao, 2012). Consequently, an interesting perspective to boost the performance of population-based search is to devise dedicated recombination operators. For this purpose, it is necessary to obtain a deep understanding of the properties and structures of the solutions in order to identify meaningful building blocks. Dedicated recombination operators can then be designed which can generate new promising solutions by blending properly existing solutions. Such an recombination operator will allow the memetic algorithm to make a difference when it is combined with a powerful local search procedure.

Finally, exact methods could be used to help the heuristics to visit promising regions of the search space. For instance, as the maximum clique problem can be formulated as an integer programming problem, its LP relaxation can be solved easily by any LP solver. The resulting LP optimal solution could be a sources of useful information. One possibility would be to fix some variables according to the LP optimum and exclude them from the search space examined by the heuristic method. Such a strategy helps the heuristic method to intensify its search within more focuses areas.

To conclude, the maximum clique problem and its generalized and relaxed variants are generic models which find more and more applications in numerous domains. Advances in exact and heuristic methods for these clique problems will help to find satisfying solutions to many practical problems. On the other hand, studies of challenging real-world problems will encourage the development of more effective solution methods. Given the increasing interest in clique problems and their applications, it is reasonable to believe that research in these domains will become even more intense and fruitful in the forthcoming years.

Acknowledgments

We are grateful to the anonymous referees for valuable suggestions and comments which helped us to improve the paper. This work is partially supported by the RaDaPop (2009–2013) and LigeRO projects (2009–2013) from the Region of Pays de la Loire, France, and the National Natural Science Foundation of China [grant nos. 71332001, 71401059].

References

Abello, J., Resende, M. G. C., & Sudarsky, S. (2002). Massive quasi-clique detection. *Lecture Notes in Computer Science*, *2286*, 598–612.
Andrade, D. V., Resende, M. G. C., & Werneck, R. F. (2012). Fast local search for the maximum independent set problem. *Journal of Heuristics*, *18*(4), 525–547.
Aringhieri, R., Bruglieri, M., & Cordone, R. (2009). Optimal results and tight bounds for the maximum diversity problem. *Foundations of Computing and Decision Sciences*, *34*(2), 73–86.
Aringhieri, R., & Cordone, R. (2011). Comparing local search metaheuristics for the maximum diversity problem. *Journal of the Operational Research Society*, *62*(2), 266–280.
Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., & Protasi, M. (1999). *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Berlin: Springer.
Babel, L., & Tinhofer, G. (1990). A branch and bound algorithm for the maximum clique problem. *Methods and Models of Operations Research*, *34*(3), 207–217.
Balasundaram, B., & Butenko, S. (2006). Graph domination, coloring and cliques in telecommunications. In *Handbook of optimization in telecommunications* (pp. 865–890). USA: Springer.
Balasundaram, B., Butenko, S., & Hicks, I. V. (2011). Clique relaxations in social network analysis: The maximum k-plex problem. *Operations Research*, *59*(1): 133–142.
Ballard, D., & Brown, C. (1982). *Computer vision*. Englewood Cliffs: Prentice-Hall.
Barahona, F., Weintraub, A., & Epstein, R. (1992). Habitat dispersion in forest planning and the stable set problem. *Operations Research*, *40*(1), 14–21.
Battiti, R., & Protasi, M. (2001). Reactive local search for the maximum clique problem. *Algorithmica 29*(4), 610–637.
Battiti, R., & Mascia, F. (2010). Reactive and dynamic local search for the Max-Clique problem: Engineering effective building blocks. *Computers & Operations Research*, *37*(3), 534–542.
Benlic, U., & Hao, J. K. (2013). Breakout local search for maximum clique problems. *Computers & Operations Research*, *40*(1), 192–206.
Bhattacharyya, M., & Bandyopadhyay, S. (2009) Mining the largest quasi-clique in human protein interactome. In *Proceedings of IEEE international conference on artificial intelligence systems* (pp. 194–199). Los Alamitos, CA, USA: IEEE Computer Society.
Billionnet, A. (2005). Different formulations for solving the heaviest *k*-subgraph problem. *Information Systems and Operational Research*, *43*(3), 171–186.
Boginski, V., Butenko, S., & Pardalos, P. M. (2006). Mining market data: A network approach. *Computers & Operations Research*, *33*(11), 3171–3184.
Bomze, I. M., Budinich, M., Pardalos, P. M., & Pelillo, M. (1999). The maximum clique problem. In *Handbook of combinatorial optimization* (pp. 1–74). USA: Springer.
Bomze, I. M., Pelillo, M., & Stix, V. (2000). Approximating the maximum weight clique using replicator dynamics. *IEEE Transactions on Neural Networks*, *11*(6), 1228–1241.
Bourgeoisa, N., Escoffiera, B., & Paschosa, V. T. (2011). Approximation of max independent set, min vertex cover and related problems by moderately exponential algorithms. *Discrete Applied Mathematics*, *159*(17), 1954–1970.
Bourgeois, N., Giannakos, A., Lucarelli, G., Milis, I., Paschos, V.Th., & Pottié, O. (2012). The max quasi-independent set problem. *Journal of Combinatorial Optimization*, *23*(1), 94–117.
Bourgeois, N., Giannakos, A., Lucarelli, G., Milis, I., & Paschos, V.Th. (2013). Exact and approximation algorithms for densest *k*-subgraph. *Lecture Notes in Computer Science*, *7748*, 114–125.
Bourjolly, J. M., Laporte, G., & Pesant, G. (2002). An exact algorithm for the maximum *k*-club problem in an undirected graph. *European Journal of Operational Research*, *138*, 21–28.
Brélaz, D. (1979). New methods to color the vertices of a graph. *Communications of the ACM*, *22*(4), 251–256.
Brimberg, J., Mladenović, N., Urošević, D., & Ngai, E. (2009). Variable neighborhood search for the heaviest *k*-subgraph. *Computers & Operations Research*, *36*(11), 2885–2891.
Brotcorne, L., Laporte, G., & Semet, F. (2002). Fast heuristic for large scale covering location problems. *Computers & Operations Research*, *29*(6), 651–665.
Brunato, M., Hoos, H., & Battiti, R. (2008). On effectively finding maximal quasi-cliques in graphs. *Lecture Notes in Computer Science*, *5313*, 41–55.
Burke, E., Hart, E., Kendall, G., Newall, J., Ross, P., & Schulenburg, S. (2003). Hyper-heuristics: An emerging direction in modern search technology. In F. Glover (Ed.), *Handbook of meta-heuristics* (pp. 457–474). Kluwer Academic.

Jin, Y., & Hao, J. K. (2015). General swap-based multiple neighborhood tabu search for finding maximum independent set. *Engineering Application of Artificial Intelligence*, *37*, 20–33.

Östergård, P. R. J. (2002). A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, *120*(1), 197–207.
Pajouh, F. M., Miao, Z., & Balasundaram, B. (2014). A branch-and-bound approach for maximum quasi-cliques. *Annals of Operations Research*, *216*(1), 145–161.
Palubeckis, G. (2007). Iterated tabu search for the maximum diversity problem. *Applied Mathematics and Computation*, *189*(1), 371–383.
Pardalos, P. M., & Xue, J. (1994). The maximum clique problem. *Journal of Global Optimization*, *4*(3), 301–328.
Park, K., Lee, K., & Park, S. (1996). An extended formulation approach to the edge-weighted maximal clique problem. *European Journal of Operational Research*, *95*(3), 671–682.
Patillo, J., Veremyev, A., Butenko, S., & Boginski, V. (2013). On the maximum quasi-clique problem. *Discrete Applied Mathematics*, *161*(1–2), 244–257.
Pattillo, J., Youssef, N., & Butenko, S. (2012). Clique relaxation models in social network analysis. In M. T. Thai, & P. M. Pardalos (Eds.), *Handbook of optimization in complex networks: Theory and applications*. Springer optimization and its applications (Vol. 58, pp. 143–162).
Pei, J., Jiang, D., & Zhang, A. (2005). On mining cross-graph quasi-cliques. In *Proceedings of the 11th ACM SIGKDD international conference on knowledge discovery in data mining* (pp. 228–238).
Prosser, P. (2012). Exact algorithms for maximum clique: A computational study. *Algorithms*, *5*(4), 545–587.
Pullan, W. (2006). Phased local search for the maximum clique problem. *Journal of Combinatorial Optimization*, *12*(3), 303–323.
Pullan, W., & Hoos, H. H. (2006). Dynamic local search for the maximum clique problem. *Journal of Artificial Intelligence Research*, *25*, 159–185.
Pullan, W. (2008). Approximating the maximum vertex/edge weighted clique using local search. *Journal of Heuristics*, *14*(2), 117–134.
Pullan, W., Mascia, F., & Brunato, M. (2011). Cooperating local search for the maximum clique problem. *Journal of Heuristics*, *17*(2), 181–199.
Ravetti, M. G., & Moscato, P. (2008). Identification of a 5-protein biomarker molecular signature for predicting Alzheimer's disease. *PLoS One*, *3*(9), e3111.
Ravi, S. S., Rosenkrantz, D. J., & Tayi, G. K. (1994). Heuristic and special case algorithms for dispersion problems. *Operations Research*, *42*(2), 299–310.
Rebennack, S., Oswald, M., Theis, D. O., Seitz, H., Reinelt, G., & Pardalos, P. M. (2011). A branch and cut solver for the maximum stable set problem. *Journal of Combinatorial Optimization*, *21*(4), 434–457.
Rebennack, S., Reineltb, G., & Pardalos, P. M. (2012). A tutorial on branch and cut algorithms for the maximum stable set problem. *International Transactions in Operational Research*, *19*(1–2), 161–199.
Régin, J. C. (2003). Solving the maximum clique problem with constraint programming. *Lecture Notes in Computer Science*, *2833*, 634–648.
Richter, S., Helmert, M., & Gretton, C. (2007). A stochastic local search approach to vertex cover. In *Proceedings of the 30th German conference on artificial intelligence* (pp. 412–426).
Rossi, F., & Smriglio, S. (2001). A branch-and-cut algorithm for the maximum cardinality stable set problem. *Operations Research Letters*, *28*(2), 63–74.
Schaeffer, S. E. (2007). Graph clustering. *Computer Science Review*, *1*(1), 27–64.
Segundo, P. S., Matia, F., Rodriguez-Losada, D., & Hernando, M. (2013). An improved bit parallel exact maximum clique algorithm. *Optimization Letters*, *7*(3), 467–479.
Segundo, P. S., Rodríguez-Losada, D., & Jiménez, A. (2011). An exact bit-parallel algorithm for the maximum clique problem. *Computers & Operations Research*, *38*(2), 571–581.
Shoham, Y., Cramton, P., & Steinberg, R. (2006). *Combinatorial auctions*. MIT Press.
Shor, N. Z. (1990). Dual quadratic estimates in polynomial and Boolean programming. *Annals of Operations Research*, *25*(1), 163–168.
Silva, G. C., De Andrade, M. R. Q., Ochi, L. S., Martins, S. L., & Plastino, A. (2007). New heuristics for the maximum diversity problem. *Journal of Heuristics*, *13*(4), 315–336.
Singh, A., & Gupta, A. K. (2006a). A hybrid heuristic for the maximum clique problem. *Journal of Heuristics*, *12*(1–2), 5–22.
Singh, A., & Gupta, A. K. (2006b). A hybrid evolutionary approach to maximum weight clique problem. *International Journal of Computational Intelligence Research*, *2*(4), 349–355.
Solnon, C., & Fenet, S. (2006). A study of aco capabilities for solving the maximum clique problem. *Journal of Heuristics*, *12*(3), 158–180.
Sørensen, M. M. (2004). New facets and a branch-and-cut algorithm for the weighted clique problem. *European Journal of Operational Research*, *154*(1), 57–70.
Tomita, E., & Seki, T. (2003). An efficient branch-and-bound algorithm for finding a maximum clique. *Lecture Notes in Computer Science*, *2731*, 278–289.
Tomita, E., & Kameda, T. (2007). An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *Journal of Global Optimization*, *37*(1), 95–111.
Tomita, E., Sutani, Y., Higashi, T., Takahashi, S., & Wakatsuki, M. (2010). A simple and faster branch-and-bound algorithm for finding a maximum clique. *Lecture Notes in Computer Science*, *5942*, 191–203.
Wang, H., Alidaee, B., Glover, F. & Kochenberger, G. (2006). Solving group technology problem via clique partitioning. *International Journal of Flexible Manufacturing Systems*, *18*(2), 77–97.
Wang, Y., Hao, J. K., Glover, F., & Lü, Z. P. (2014). A tabu search based memetic algorithm for the maximum diversity problem. *Engineering Applications of Artificial Intelligence*, *27*, 103–114.
Warren, J. S., & Hicks, I. V. (2006). *Combinatorial branch-and-bound for the maximum weight independent set problem* (Technical report). Texas A&M University.
Weide, O., Ryan, D., & Ehrgott, M. (2010). An iterative approach to robust and integrated aircraft routing and crew scheduling. *Computers & Operations Research*, *37*(5), 833–844.
Wu, Q., & Hao, J. K. (2011). An adaptive multistart tabu search approach to solve the maximum clique problem. *Journal of Combinatorial Optimization*, *26*(1), 86–108.
Wu, Q., & Hao, J. K. (2012a). Coloring large graphs based on independent set extraction. *Computers & Operations Research*, *39*(2), 283–290.
Wu, Q., & Hao, J. K. (2012b). An effective heuristic algorithm for sum coloring of graphs. *Computers & Operations Research*, *39*(7), 1593–1600.
Wu, Q., Hao, J. K. & Glover, F. (2012). Multi-neighborhood tabu search for the maximum weight clique problem. *Annals of Operations Research*, *196*(1), 611–634.
Wu, Q., & Hao, J. K. (2013). A hybrid metaheuristic method for the maximum diversity problem. *European Journal of Operational Research*, *231*(2), 452–464.
Wu, Q., & Hao, J. K. (2015). Solving the winner determination problem via a weighted maximum clique heuristic. *Expert Systems with Applications*, *42*(1), 355–365.
Xiang, J., Guo, C., & Aboulnaga, A. (2013). Scalable maximum clique computation using MapReduce. In *IEEE 29th international conference on data engineering* (pp. 74–85).
Xu, K., Boussemart, F., Hemery, F., & Lecoutre, C. (2005). A simple model to generate hard satisfiable instances. In *Proceedings of the nineteenth international joint conference on artificial intelligence (IJCAI-05)* (pp. 337–342).
Xu, K., Boussemart, F., Hemery, F., & Lecoutre, C. (2007). Random constraint satisfaction: Easy generation of hard (satisfiable) instances. *Artificial intelligence*, *171*(8–9), 514–534.
Yang, G., Yi, J., Zhang, Z., & Tang, Z. (2009). A TCNN filter algorithm to maximum clique problem. *Neurocomputing*, *72*(4–6), 1312–1318.
Zeng, Z., Wang, J., Zhou, L., & Karypis, G. (2007). Out-of-core coherent closed quasi-clique mining from large dense graph databases. *ACM Transactions on Database System*, *32*(2): Article 13.
Zhang, Q., Sun, J., & Tsang, E. (2005). An evolutionary algorithm with guided mutation for the maximum clique problem. *IEEE Transactions on Evolutionary Computation*, *9*(2), 192–200.
Zuckerman, D. (2006). Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the 38th ACM symposium on theory of computing* (pp. 681–690).