SUBGRAPH EXTRACTION AND MEMETIC ALGORITHM FOR THE MAXIMUM CLIQUE PROBLEM

Duc-Cuong Dang and Aziz Moukrim
Université de Technologie de Compiègne
Laboratoire Heudiasyc, UMR UTC-CNRS 6599, BP 20529, 60205 Compiègne, France

Keywords: Maximum clique problem, Memetic algorithm, Triangulated graph, Circular-arc graph.

Abstract: The maximum clique problem involves finding the largest set of pairwise adjacent vertices in a graph. The problem is classic but still attracts much attention because of its hardness and its prominent applications. Our work is based on the existence of an order on all the vertices whereby those belonging to a maximum clique stay close enough to each other. Such an order can be identified via the extraction of a particular subgraph from the original graph. The problem can consequently be seen as a permutation problem that can be addressed efficiently with an evolutionary-like algorithm, here we use a memetic algorithm (MA). Computational experiments conducted on DIMACS benchmark instances clearly show our MA which not only outperforms existing genetic approaches, but also compares very well to state-of-the-art algorithms.

1 INTRODUCTION

A clique in a graph is a set of pairwise adjacent vertices, that is to say an induced subgraph which is itself a complete graph. The Maximum Clique Problem (MCP) involves finding a clique with the greatest cardinality. The MCP is one of the important problems of graph theory, and is computationally equivalent to both the Maximum Independent Set Problem (MISP) and the Minimum Vertex Cover Problem (MVCP). The problem has been proved to be NP-Hard (Garey and Johnson, 1979) for an arbitrary graph. For some special graphs such as triangulated graphs, interval graphs and circular-arc graphs, the MCP can be solved in linear time (Golumbic, 2004). The MCP has numerous applications in a variety of domains including information retrieval, signal transmission and bioinformatics (Balas and Yu, 1986; Ji et al., 2004). Therefore, it is important to design efficient heuristics to solve different instances in a reasonable computation time.

Many heuristic approaches have been developed for the MCP, most of which are modified version of different local search techniques, while others are metaheuristics (for an overview of these methods, see (Bomze et al., 1999; Pardalos and Xue, 1994)). Recent published methods have been tested using benchmark instances from the Second DIMACS Challenge (Johnson and Trick, 1996), but comparing experimental results remains difficult, from the point of view of both the quality of the solutions and their computation time, owing to differences in experimental testing protocols and the large variety of machines used. However, if we consider only the quality of the solutions reported by authors of these methods, two stand out as being state-of-the-art for the MCP. First, *Reactive Local Search* (RLS) (Battiti and Protasi, 2001) is a tabu search with a feedback scheme to adapt the tenure parameter and to restart the process when required. A recent evolutionary approach extending the reactive scheme has been reviewed by the authors[1]. Secondly, *Variable Neighborhood Search* (VNS) (Hansen et al., 2004) is, as its name suggests, a variable neighborhood search using the simplicial vertex test during its descent steps. Certain other methods, comparable to the state-of-the-art, are worthy of note. *K-opt Local Search* (KLS) (Katayama et al., 2005) is a quick variable-depth search tested in a basic multi-start scheme. *Ant Colony Optimization* (ACO) (Solnon and Fenet, 2006) is a recent ant colony optimization approach for the MCP. *Quick Almost Exact Maximum Weight Clique/Independent Set Solver* (QUALEX-MS) (Busygin, 2006) is a deterministic greedy construction algorithm based on a quadratic formulation of the MCP. *Dynamic Local Search* (DLS) (Pullan and Hoos, 2006) is a stochastic

[1] http://rtm.science.unitn.it/~battiti/

Dang D. and Moukrim A..
SUBGRAPH EXTRACTION AND MEMETIC ALGORITHM FOR THE MAXIMUM CLIQUE PROBLEM.
DOI: 10.5220/0003081000770084
In *Proceedings of the International Conference on Evolutionary Computation* (ICEC-2010), pages 77-84
ISBN: 978-989-8425-31-7

local search with dynamic penalty adjustment for vertex selection. The DLS focuses on finding a clique of given size. It was tested on different global schemes (Pullan and Hoos, 2006; Pullan, 2006; Grosso et al., 2008).

Genetic Algorithms (GA) have been shown to be efficient for many hard combinatorial optimization problems, but pure genetic schemes applied for the MCP have poor performances when compared to local search techniques (Park and Carter, 1995). Consequently, most efficient GA designs for the MCP are hybrids between a genetic scheme and a local search (Bui and Eppley, 1995; Marchiori, 1998; Singh and Gupta, 2006). One common feature of these GA is that they use a binary representation of the chromosome.

In this paper we focus on GA designs in which local search techniques replace the mutation procedures of classic GA, and we refer to Memetic Algorithms (Moscato, 1999). Moreover, we propose using a permutation of the set of vertices as a representation, meaning that an evaluation process is then required to identify the associated clique. This identification is performed using a subgraph extraction approach which targets interval graphs, circular-arc graphs and triangulated graphs. Three algorithms, that we called VMTG (vertex-maximal extraction using triangulated graphs), EMTG (edge-maximal extraction using triangulated graphs) and CAG (extraction using circular-arc graphs) are developed for this purpose.

The paper is organized as follows. We first present, in Section 2, a formal description of the MCP with a short description of our approach to solve the problem. Different methods to evaluate a permutation of vertices using subgraph extraction are also described in the section. The memetic algorithm is presented and discussed in Section 3. Numerical results are provided in Section 4, and finally some conclusions are drawn.

2 SUBGRAPH EXTRACTION APPROACH

Let $G = (V, E)$ be an arbitrary undirected graph where V is the set of vertices and $E \subseteq V \times V$ the set of edges. The number of vertices in V will be denoted n, and the number of edges in E will be denoted m. Given a subset S of V, we refer to $G[S] = (S, E \cap (S \times S))$ as the *subgraph induced by S*. A graph $G = (V, E)$ is *complete* if all its vertices are pairwise adjacent, meaning that $\forall x, y \in V, [x, y] \in E$. A *clique* K is a subset of V such that $G[K]$ is complete. The MCP ask to find a clique K with the greatest cardinality.

Our approach to solving the MCP involves, first, considering a permutation of all the vertices in V, i.e. a sequence which we shall denote π, and then using an extraction procedure to identify associated solutions, i.e. the most profitable subsequences. Both the sequence and the identified solution can be improved using appropriate optimization techniques.

The most intuitively straightforward extraction method for a given sequence π is to find the subsequence $(\pi(i), ..., \pi(i + l_i))$ that can form a clique and that has the greatest length l_i. Every permutation containing the vertices of a maximum clique K_{max} as a subsequence yields K_{max} in return. A naive algorithm with complexity $O(n^3)$ can be used to extract such a subsequence. More efficient extraction methods involve first extracting a subgraph from the original one, then identifying the maximum clique from the subgraph. The subgraph types are chosen such that the MCP can be solved in polynomial time, or linear time in the ideal case. For this reason, in this section we investigate different extraction methods using interval graphs, circular-arc graphs and triangulated graphs.

2.1 Triangulated Graph Extraction

A graph is triangulated or chordal if every one of its cycles of four or more nodes has a chord, that is to say an edge joining two nodes that are not consecutive in the cycle. We are interested in triangulated graph extraction, mainly because this method has already been proposed by (Balas and Yu, 1986) (vertex-maximal version) and by (Xue, 1994) (edge-maximal version). Neither of these methods starts out with a predefined permutation of vertices, but both attempt to build one dynamically while maximizing the edge or vertex criteria. We have adapted their methods for our purposes by including a predefined permutation.

For a given graph $G = (V, E)$, a permutation of its vertices set is defined as bijection $\pi : [1..n] \rightarrow V$. We set $succ_{G,\pi}(x) = \{y | \pi^{-1}(y) > \pi^{-1}(x) \text{ and } [x, y] \in E\}$ to denote the set of *successors* of x in π and $t_{G,\pi}(x)$ to denote the first successor (having the smallest $\pi^{-1}(y)$ value). A vertex x is said to be *quasi-simplicial* in π if $t_{G,\pi}(x)$ is connected to every other element of $succ_{G,\pi}(x)$. A graph is triangulated if and only if there exists a permutation of vertices π, such that the induced subgraph $G[\{x\} \cup succ_{G,\pi}(x)]$ is complete for every $x \in V$. Such a permutation is called a *perfect elimination scheme* (PES). A permutation π of V is a PES of G if and only if for all $x \in V$, x is quasi-simplicial in π (Balas and Yu, 1986).

For a given permutation π and a subset S of V, we refer to $\pi_{|S}$: $[1..|S|] \rightarrow S$ as the *permutation restricted*

to S, such that $\pi^{-1}_{|S}(x) < \pi^{-1}_{|S}(y)$ with $(x,y) \in S \times S$ and $x \neq y$ if and only if $\pi^{-1}(x) < \pi^{-1}(y)$. A subgraph $G' = (V', E')$ of G is said to be π-triangulated if and only if $\pi_{|V'}$ is a PES of G'. In particular, when G' is an edge-induced subgraph, that is to say $V' = V$, it is said to be edge-maximal if and only if $\forall e \in E \setminus E'$, the subgraph $G'' = (V, E' \cup \{e\})$ is not π-triangulated. In the same way, when G' is a vertex-induced subgraph, usually noted $G[S]$, it is said to be vertex-maximal if and only if $\forall x \in V \setminus S$, $G[S \cup \{x\}]$ is not π-triangulated.

Our first triangulated subgraph extraction for a given permutation π, noted VMTG, is as follows. A subset S of vertices is initialized to $\pi(n)$, then we browse vertices in π from right to left. If the current vertex x is quasi-simplicial in $\pi_{|S\cup\{x\}}$, it will be added to S. An example of this extraction is given in Figure 1.b. In this example, according to permutation $\pi_1 = (3,2,6,7,1,5,4)$, S is initialized to vertex 4 from the original graph (Figure 1.a), then vertices 5, 1, 7, 6 are progressively added to S because their first successors are connected to their all other successors in $\pi_{1|S}$ during the process. Then vertex 2 is not added to S because its first successor in the current $\pi_{1|S}$ is 7 and it is not connected to vertex 1, the other successor of 2. Finally, vertex 3 is added to S and we obtain the π_1-triangulated vertex-induced subgraph. The largest clique is memorized for vertex 6, because its set of successors in $\pi_{1|S}$ has the greatest cardinality. Hence, for an arbitrary graph $G = (V,E)$, the following proposition holds.

Proposition 1. *Algorithm VMTG finds a vertex-maximal π-triangulated subgraph of G in O(n+m).*

The second extraction, noted EMTG, is based on an almost identical principle, but instead of removing the vertex we try to maximally integrate relative edges into the extracted graph while keeping the quasi-simplicial property. An example is given in Figure 1.c. Here vertices are always added to G' at the same time as necessary edges. With the same permutation π_1 from previous example, the algorithm initializes G' with vertex 4 and empty edge set, then progressively adds vertex 5 and edge 5-4, vertex 1 and edges 1-5, 1-4, vertex 7 and edges 7-5, 7-4, vertex 6 and edges 6-7, 6-5, 6-4, vertex 2 and edge 2-7 (but not edge 2-1), vertex 3 and edge 3-2 (but not edge 3-1) to G'. The largest clique is also memorized at adding step of vertex 6. Similarly, for an arbitrary graph $G = (V,E)$, the following proposition holds.

Proposition 2. *Algorithm EMTG finds an edge-maximal π-triangulated subgraph of G in O(n+m).*

2.2 Interval Graph and Circular-arc Graph Extraction

An *interval graph* is an *intersection graph* of multiple segments of a line. Its generalization on a circle (*circular-arc graph*) is an intersection graph of multiple arcs of a circle. Intersection containing the largest number of segments or circular-arcs is a maximum clique of the graph, so finding the maximum clique in these graphs can be done is linear time when all segments or circular-arcs are well defined (Golumbic, 2004). In this section, we propose extraction methods for an arbitrary graph and a permutation of vertices using interval graph and circular-arc graph.

Let $G = (V,E)$ be an arbitrary graph and π be a permutation of vertices of V. For any i in $[1..n]$, we denote $I_i = [i, i+l_i]$ where l_i is such that $0 \leq l_i \leq n-i$ and $\forall l \in [1..l_i], [\pi(i), \pi(i+l)] \in E$ and $X = \{I_i | 1 \leq i \leq n\}$. The interval set X can be seen as the representation of an interval graph $G_\pi = (V, E_\pi)$. For any couple of vertices $x, y \in V$, $[x,y] \in E_\pi$ if and only if $I_i \cap I_j \neq \emptyset$ with $i = \pi^{-1}(x)$ and $j = \pi^{-1}(y)$. Also for any i in $[1..n]$, we denote as J_i an extension of I_i in which $J_i = \emptyset$ if $l_i < n-i$ or $[\pi(i), \pi(1)] \notin E$. Otherwise $J_i = [1, p_i]$ where p_i is such that $1 \leq p_i < i$ and $\forall p \in [1..p_i], [\pi(i), \pi(p)] \in E$. Therefore $Y = \{I_i \cup J_i | 1 \leq i \leq n\}$ is the representation of a circular-arc graph that we denote $\hat{G}_\pi = (V, \hat{E}_\pi)$. For any couple of vertices $x, y \in V$, $[x,y] \in \hat{E}_\pi$ if and only if $(I_i \cup J_i) \cap (I_j \cup J_j) \neq \emptyset$ with $i = \pi^{-1}(x)$ and $j = \pi^{-1}(y)$.

Proposition 3. *For an arbitrary graph $G = (V,E)$ and a permutation π of V, we have $E_\pi \subset \hat{E}_\pi \subset E$.*

A direct corollary of Proposition 3 is that any clique in G_π is a clique in $\hat{G}_\pi$ and so on any clique in $\hat{G}_\pi$ is also a clique in G. From now we are interested in circular-arc graph extraction instead of interval graph one. The clique extraction using circular-arc graphs, noted CAG, can be described as follows. For each position i in the permutation, we maintain a list L_i of the vertices $\pi(j)$ in which the circular-arcs $(I_j \cup J_j)$ pass through position i. Formally, $L_i = \{\pi_j | j = [1..n] \text{ and } i \in (I_j \cup J_j)\}$. We browse circular-arcs all $\{I_j \cup J_j\}$ with $j \in [1..n]$ and progressively update L_i. At the end of this process the longest list will correspond to the maximum clique of $\hat{G}_\pi$. Since the circular-arc length starting from vertex $\pi(j)$ does not exceed the degree of vertex $\pi(j)$ in G, the final complexity of this process is obviously $O(n+m)$. An example of this subgraph extraction is shown in Figure 1.d. We consider permutation $\pi_2 = (4,5,1,7,6,2,3)$, the algorithm can start from any vertex in the order and for simple, following this order until returning to the same vertex. For example we begin with vertex 5, its first unlinked vertex in the order is vertex 2, so 5

is added to L_5, L_1, L_7 and L_6. In the same way 1 is added to L_1, 7 is added to L_7, L_6 and L_2, 6 is added to L_6, 2 is added to L_2 and L_3, 3 is added to L_3 and finally 4 is added to L_4, L_5, L_1, L_7 and L_6. The longest list is now L_6 contains the maximum clique (4, 5, 6, 7) of the extracted graph.

All algorithms VMTG, EMTG and CAG have the following property.

Property 1. *For an arbitrary graph $G = (V,E)$, there exists a permutation of V as input of the algorithm that gives in output a maximum clique of G.*

For VMTG and EMTG algorithms, it is enough for vertices of a maximum clique to stick together in the end of the permutation in order to find the clique. For CAG algorithm, it is enough for those vertices to stick together at any position of the permutation. So the CAG algorithm dominates the earlier mentioned straight forward method. These permutations are sufficient to show the correctness of Property 1, even though other kinds of permutation giving a maximum clique in output exist as seen in the example of Figure 1. In the next section, we describe how to find such a permutation using evolutionary approaches.

3 MEMETIC ALGORITHM

One of the metaheuristics that have proved to be effective in addressing permutation problem is the genetic algorithm (GA). When local search techniques (LS) are used instead of mutation processes in classical GA, the method is termed a memetic algorithm (MA) (Moscato, 1999). This section describes the detail of our MA design.

3.1 Population Management

In GA design, the population is a set of individuals or solutions. In our MA each individual is represented by a permutation of vertices or by a sequence. In order to transform a sequence into a solution or a clique, an evaluation process is required. We make this evaluation process correspond to calling an *eval*() function, using one of the three methods described in Section 2. For convenience we consider all permutations in circular order. VMTG and EMTG algorithms in particular require a linear order to operate, so the circular order can be transformed into linear one starting from a specific *open point*. The position of this open point is initialized to random for new created permutations, i.e at initialization or through the crossover operator, but for improved permutations from local search process the value is especially given (see Section 3.2). This value has no signification for CAG algorithm. We now discuss how to evolve these individuals toward better ones.

The population is sorted hierarchically, first by identified clique size and then by generation number. A new individual created via crossover and mutation operations is a candidate for insertion into the population. This insertion requires the new individual to have a performance at least equal to that of the worst individual, and sufficiently different from the two closest individuals in the population hierarchy, for example with at least 5 distinct vertices. To distinguish two individuals, we use the Hamming distance on extracted cliques. A successful insertion is followed by the ejection of the last individual in the hierarchy, so as to maintain the existing population size.

3.2 Crossover Operator and Local Search as Mutation

At each iteration of the algorithm two parents are selected via a binary tournament for crossover operation. Our crossover operator, called MOX, works with a *heritage mask*, a binary string of size n. The child sequence will inherit from one parent all the vertices in the mask having the value 1, and from the other parent the remaining vertices, in the same order. There are different ways to generate an efficient heritage mask, the most simple is to put randomly values 0 and 1 with equal probability. Then, from a random position in the mask, we create a subsequence of $\frac{|P_{max}|}{2}$ size with values 1 ($|P_{max}|$ is the best known clique size of two parents). The idea is to break and merge randomly different cliques from the permutations.

After a child sequence is created using MOX and evaluated, it will have *pm* probability of being mutated through local search (LS). During this process, three neighborhoods are applied in stochastic order until no further improvement can be found. Here we denote as K the current clique, as $N_{|K|}$ the set of vertices that connect to all the vertices in K, and as $N_{|K|-1}(x)$ (with $x \in K$) the set of vertices that connect to all vertices in K except x. The number in parentheses is the stochastic priority *ps* of the neighborhood, that means the neighborhood has probability $\frac{ps}{\sum ps}$ to be selected.

- *k-opt inner loop* (5), the inner loop of the heuristic k-opt (Katayama et al., 2005), for each iteration, remove one vertex in K having greatest value of $|N_{|K|-1}(x)|$ then try to complete the clique with vertices having the greatest degree in the induced subgraph $G[N_{|K|}]$. Removed and added vertices

Table 1: Overall performance comparison on DIMACS benchmark.

Method	RLS	HSSGA	GA	VNS	KLS	PLS	MA			**Best**
							VMTG	EMTG	CAG	
$\sum\|K\|$	2849	2832	2819	2851	2845	2856	2854	2859	2855	2862
$\sum CPU$	6715.08	4774.65	926.8	6453.48	148.24	1017.5	4332.88	11264.46	4062.25	

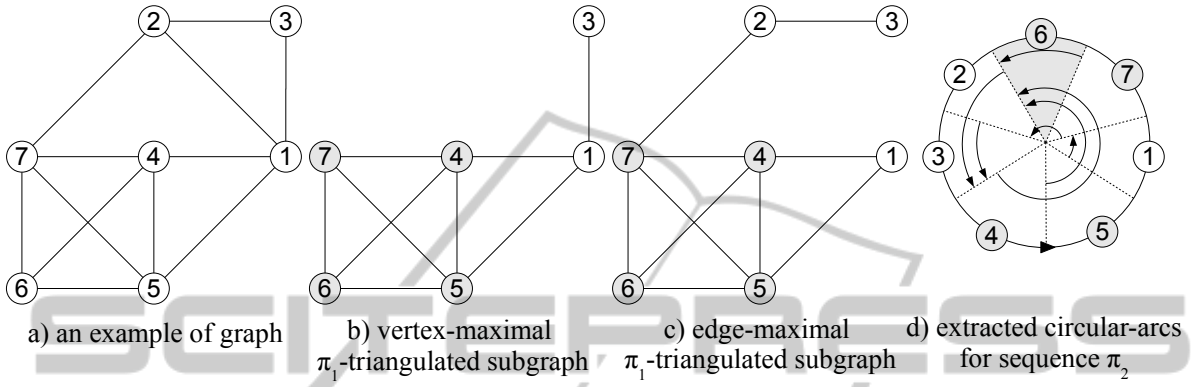

Figure 1: Examples of subgraph extraction with $\pi_1 = (3,2,6,7,1,5,4)$ and $\pi_2 = (4,5,1,7,6,2,3)$. In this particular example, π_1 is reversed order of π_2 and algorithm CAG (d) gives the same graph as EMTG (c) in return.

are marked to avoid using in next iterations. The loop is stopped when the current clique does not contain any vertex from the original one. The highest score is maintained during the process. At the end of the loop, it is compared with the original clique in order to detect an improvement.

- *heuristic drop/add* (3), for each iteration, remove d vertices with priority to vertices having the greatest value of $|N_{|K|-1}(x)|$ from the original clique, then complete it with vertices having the greatest degree in the induced subgraph $G[N_{|K|}]$. The value of d is initialized to 1 and duplicated after each iteration. The loop is stopped when an improvement is found or when d is gone over $|K|-1$.
- *random drop/add* (2), similar to heuristic drop/add except vertices are randomly selected during dropping and adding step.

After the local search, to convert the improved clique back into a sequence, vertices from the clique are first placed in an empty sequence at a random starting position. The sequence is then randomly completed with the other vertices in the remaining position. The open point for transforming a circular order to linear order is set to the ending position of placed clique.

4 NUMERICAL RESULTS

The DIMACS instance set for the MCP and Graph Coloring Problem (GCP) was released in 1993 for the DIMACS series of challenges in combinatorial optimization. Since that time it has been used as a benchmark to evaluate the efficiency of heuristics developed for the MCP. We therefore tested our algorithm on this set in order to compare its performance with other methods in the literature. The set consists of 80 instances, generation methods being described in (Johnson and Trick, 1996). Most recent published heuristics focus on 37 instances only, rather than on the full set. These focused instances range in size from 125 vertices and 6000 edges to 4000 vertices and 5000000 edges.

Following a large number of experiments on DIMACS instances, we decided to set the parameters of our algorithm as follows: the population size is fixed at 40 individuals, and it is unnecessary to include some good individuals when initializing the population, i.e. by using a heuristic rather than a random generation. We denote as $iter_{ineffective}$ the current number of consecutive iterations without improvement. The algorithm is stopped after $iter_{ineffective}$ reaches a value $iter_{max}$. For small instances (less than 1000 vertices) we set $iter_{max}$ to $20.n$, and for larger instances we set $iter_{max}$ to n. Every individual created though crossover has a probability $pm = 1 - \frac{iter_{ineffective}}{iter_{max}}$ of being mutated. All results for each instance are reported

after 100 executions. Our experiments were run on an Intel Core 2 Duo E6750 - 2.67 GHz personal computer with 2 GB RAM, using a single-threaded program. Computer performance, reported by Sandra SiSoft[2], in single core mode is 9280 MFLOPS and 10770 MIPS. The DIMACS Machine Benchmark[3] required respectively 0.02, 0.23, 1.40 and 5.36 CPU seconds for solving r200.5, r300.5, r400.5 and r500.5 sample instances.

Table 2 reports our detailed results for DIMACS benchmarks. The column $|K|$ indicates clique size found in solution over executions, here $|K|_{worst}$, $|K|_{best}$ and the standard deviation are reported. The column *CPU* indicates average CPU time per execution with standard deviation. Finally the column N reports the number of executions where $|K|_{best}$ is reached. Over these results, it should be observed EMTG extraction gives the best performance for MA but it requires extended execution time. This is likely due to additional memory usages for saving edge structures of the extracted subgraph during evaluation process. VMTG and CAG extractions give quite similar performance but CAG runs slightly faster than VMTG as it has very simple implementation.

Our overall results on 37 DIMACS instances, using different extractions VMTG, EMTG and CAG, are compared to those obtained by genetic algorithms reported by (Marchiori, 1998) (GA) and by (Singh and Gupta, 2006) (HSSGA), and to the state-of-the-art reported by (Battiti and Protasi, 2001) (RLS) and by (Hansen et al., 2004) (VNS). We also compare our results to those reported by (Katayama et al., 2005) (KLS) and by (Pullan, 2006) (PLS). Table 1 reports for each method the overall performance with respect to the total number of vertices in identified cliques, denoted $\sum|K|$ and the average CPU time for executing full DIMACS set, denoted $\sum CPU$. For our MA, CPU time for each execution is full running time of the algorithm. For other methods, the authors reported this value only as time from starting of the algorithm until the discovery of the best solution. Results from other excellent methods, sometimes outperforming state-of-the-art methods, are not shown in Table 1, either because certain results were lacking or because different testing protocols were used. Regarding these methods, we remarked the following: ACO by (Solnon and Fenet, 2006) has an overall score of 1756 without reporting the result for the MAN_a81 instance; DLS by (Pullan and Hoos, 2006) has excellent overall score of 2856 with parameters customized for each instance (as early development of PLS); ILS by (Grosso et al., 2008) did not report some results, but the algorithm obtained a record clique size for the C2000.9 instance (80 vertices instead of the 78 in the literature).

These results clearly demonstrate that our Memetic Algorithm compares very well with the other genetic algorithms. MA outperforms the GA proposed by (Marchiori, 1998) and the HSSGA proposed by (Singh and Gupta, 2006) in terms of efficiency, and it is also competitive with state-of-the-art algorithms. Other details of our results including experiments on BHOSLIB (Benchmarks with Hidden Optimum Solutions for Graph Problems) instances are available at http://www.hds.utc.fr/∼dangducc/mclique/.

5 CONCLUSIONS AND FUTURE WORK

We have proposed a memetic algorithm for the MCP which not only outperforms other genetic approaches, but is competitive with state-of-the-art approaches. For the first time the order of the vertices has been used to represent a chromosome in the genetic approach for solving the MCP. Finally, it is sure that the MA can be more engineered to improve stability, efficiency and execution time but the current results have been shown the subgraph extraction approach and memetic algorithms are promising research directions for solving the MCP and related combinatorial problems.

Given that a large number of fast and effective local searches have been developed for the MCP, as future work we plan to investigate the performances of some of these inside the memetic scheme. Moreover, since the subgraph extraction approach is not restricted to using genetic algorithms as a global search method, we also intend to test other schemes.

REFERENCES

Balas, E. and Yu, C. (1986). Finding a maximum clique in an arbitary graph. *SIAM Journal of Computing*, 15:1054–1068.

Battiti, R. and Protasi, M. (2001). Reactive local search for the maximum clique problem. *Algorithmica*, 29:610–637.

Bomze, I., Budinich, M., Pardalos, P., and Pelillo, M. (1999). *Handbook of Combinatorial Optimization*, chapter The maximum clique problem. Kluwer Academic Publishers.

Bui, T. N. and Eppley, P. H. (1995). A hybrid genetic algorithm for the maximum clique problem. In *Proceed-*

[2]http://www.sisoftware.net/

[3]ftp://dimacs.rutgers.edu/pub/dsj/clique/

ings of the 6th International Conference on Genetic Algorithms, pages 478–484.

Busygin, S. (2006). A new trust region technique for the maximum weight clique problem. *Discrete Applied Mathematics*, 154:2080–2096.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, USA.

Golumbic, M. C. (2004). *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol 57)*. North-Holland Publishing Co.

Grosso, A., Locatelli, M., and Pullan, W. (2008). Simple ingredients leading to very efficient heuristics for the maximum clique problem. *Journal of Heuristics*, 14(6):587–612.

Hansen, P., Mladenovic, N., and Urosevic, D. (2004). Variable neighborhood search for the maximum clique. *Discrete Applied Mathematics*, 145:117–125.

Ji, Y., Xu, X., and Stormo, G. D. (2004). A graph theorical approach for predicting common RNA secondary structure motifs including psudoknots in unaligned sequences. *Bioinformatics*, 20:1591–1602.

Johnson, D. and Trick, M. (1996). *Cliques, Cloroing And Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series*. American Mathematical Society.

Katayama, K., Hamamot, A., and Narihisa, H. (2005). Solving the maximum clique problem by k-opt local search. *Information Processing Letters*, 95:503–511.

Marchiori, E. (1998). A simple heuristic based genetic algorithm for the maximum clique problem. In *ACM Symposium on Applied Computing*, pages 366–373.

Moscato, P. (1999). *New Ideas in Optimization*, chapter Memetic Algorithms: a short introduction, pages 219–234. McGraw-Hill Ltd., UK.

Pardalos, P. and Xue, J. (1994). The maximum clique problem. *Journal of Global Optimization*, 4:301–328.

Park, K. and Carter, B. (1995). On the effectiveness of genetic search in combinatorial optimization. In *Proceedings of the 10th ACM Symposium on Applied Computing*, pages 329–336.

Pullan, W. (2006). Phased local search for the maximum clique problem. *Journal of Combinatorial Optimization*, 12:303–323.

Pullan, W. and Hoos, H. H. (2006). Dynamic local search for the maximum clique problem. *Journal of Artificial Intelligence Research*, 25:159–185.

Singh, A. and Gupta, A. K. (2006). A hybrid heuristic for the maximum clique problem. *Journal of Heuristics*, 12:5–22.

Solnon, C. and Fenet, S. (2006). A study of ACO capabilities for solving the maximum clique problem. *Journal of Heuristics*, 12.

Xue, J. (1994). Edge-maximal triangulated subgraphs and heuristics for the maximum clique problem. *Network*, 24:109–120.

No.	Instance	Best	VMTG			EMTG			CAG		
			$\|K\|$	CPU	N	$\|K\|$	CPU	N	$\|K\|$	CPU	N
1	C125.9	34*	34	1.82(0.03)	100	34	2.09(0.03)	100	34	1.86(0.03)	100
2	C250.9	44*	44	8.97(0.12)	100	44	11.62(0.12)	100	44	8.18(0.11)	100
3	C500.9	57	57	53.77(2.47)	100	57	88.9(6.4)	100	57	43.59(3.66)	100
4	C1000.9	68	65-68(0.74)	24.16(6.69)	1	65-68(0.59)	57.71(11.96)	10	66-68(0.54)	22.27(5.49)	12
5	C2000.9	80	74-77(0.78)	183.76(50.46)	2	75-78(0.83)	560.02(119.21)	3	75-78(0.7)	160.58(35.38)	1
6	DSJC500.5	14*	13	33.43(0.33)	100	13	61.61(0.46)	100	13	30.13(0.35)	100
7	DSJC1000.5	15*	14-15(0.5)	16.06(3.9)	56	14-15(0.45)	32.84(6.08)	28	14-15(0.5)	13.16(2.9)	45
8	C2000.5	16	15-16(0.46)	87.58(18.69)	69	15-16(0.47)	292.76(56.52)	66	15-16(0.45)	78.19(17.65)	71
9	C4000.5	18	16-18(0.22)	595.99(101.41)	3	16-18(0.22)	2328.3(373.51)	2	16-18(0.26)	506.17(83.25)	5
10	MANN_a27	126*	126	26.4(0.24)	100	126	50.84(0.35)	100	126	38.26(0.32)	100
11	MANN_a45	345*	344-345(0.2)	33.26(4.97)	4	344-345(0.17)	77.11(10.48)	3	344-345(0.14)	44.35(5.2)	2
12	MANN_a81	1100	1098-1099(0.36)	973.2(166.16)	15	1098-1100(0.35)	2877.36(333.02)	1	1098-1099(0.31)	1282.74(240.67)	11
13	brock200_2	12*	11-12(0.2)	3.6(0.63)	96	11-12(0.31)	5.66(1.11)	89	11-12(0.37)	3.74(0.76)	84
14	brock200_4	17*	16-17(0.41)	3.94(0.84)	79	16-17(0.5)	5.71(1.36)	48	16-17(0.49)	3.43(0.72)	43
15	brock400_2	29*	25-29(1.6)	27.58(5.45)	20	25-29(0.78)	41.51(5.17)	4	25-29(1.2)	21.68(2.51)	10
16	brock400_4	33*	25-33(3.01)	31.5(6.73)	83	25-33(3.82)	44.87(8.73)	35	25-33(3.84)	24.42(6.29)	36
17	brock800_2	24*	21	177.49(7.03)	100	21-24(0.3)	354.63(29.76)	1	21	149.49(11.25)	100
18	brock800_4	26*	21-26(0.5)	189.52(22.01)	1	21-26(0.5)	386.58(55)	1	21-26(0.7)	162.54(21.58)	2
19	gen200_p0.9_44	44*	44	5.31(0.07)	100	44	6.4(0.09)	100	44	5.01(0.08)	100
20	gen200_p0.9_55	55*	55	5.33(0.07)	100	55	6.59(0.08)	100	55	5.2(0.08)	100
21	gen400_p0.9_55	55*	55	36(5.95)	100	53-55(0.84)	55.24(11.89)	77	53-55(0.94)	30.3(6.89)	67
22	gen400_p0.9_65	65*	65	29.87(0.23)	100	65	48.42(0.4)	100	65	28.03(0.32)	100
23	gen400_p0.9_75	75*	75	31(0.31)	100	75	47.97(0.8)	100	75	27.66(0.92)	100
24	hamming8-4	16*	16	5.09(0.05)	100	16	8.32(0.08)	100	16	5.15(0.05)	100
25	hamming10-4	40*	40	19.94(0.54)	100	40	45.81(0.82)	100	40	16.59(0.44)	100
26	keller4	11*	11	2(0.03)	100	11	2.84(0.03)	100	11	2.11(0.03)	100
27	keller5	27*	27	166.38(1.27)	100	27	323.29(1.39)	100	27	131.24(0.75)	100
28	keller6	59	56-59(1.06)	892.84(244.5)	28	56-59(0.58)	2348.99(441.94)	92	57-59(0.78)	658.66(144.47)	81
29	p_hat300-1	8*	8	6.13(0.1)	100	8	8.59(0.08)	100	8	5.91(0.05)	100
30	p_hat300-2	25*	25	15.04(0.18)	100	25	16.67(0.14)	100	25	12.36(0.1)	100
31	p_hat300-3	36*	36	16.48(0.16)	100	36	19.73(0.17)	100	36	13.73(0.14)	100
32	p_hat700-1	11*	11	70.56(0.83)	100	11	133.16(1.7)	100	11	64.73(0.67)	100
33	p_hat700-2	44*	44	215.34(1.62)	100	44	273.2(1.66)	100	44	174.63(1.3)	100
34	p_hat700-3	62	62	152.39(0.98)	100	62	255.74(1.18)	100	62	123.5(0.82)	100
35	p_hat1500-1	12*	11-12(0.41)	34.66(8.14)	22	11-12(0.24)	74.9(7.21)	6	11-12(0.27)	26.83(4.25)	8
36	p_hat1500-2	65	65	82.21(1.5)	100	65	151.2(2.2)	100	65	76.85(1.65)	100
37	p_hat1500-3	94	94	74.29(1.73)	100	94	157.28(2.67)	100	94	58.97(1.29)	100
	Total	2862	2854	4332.88		2859	11264.46		2855	4062.25	

Table 2: Detailed results on DIMACS benchmark (* if optimality is proved).