A Simple Heuristic Based Genetic Algorithm for the Maximum Clique Problem

Elena Marchiori*
CWI, Amsterdam, The Netherlands
and
Department of Computer Science, Leiden University, The Netherlands
`E-mail:elena@cs.leidenuniv.nl`

Keywords: maximum clique problem, heuristic algorithm, genetic algorithm, combinatorial optimization.

Abstract

This paper proposes a novel heuristic based genetic algorithm (HGA) for the maximum clique problem, which consists of the combination of a simple genetic algorithm and a naive heuristic algorithm. The heuristic based genetic algorithm is tested on the so-called DIMACS benchmark graphs, with up to 4000 nodes and up to 5506380 edges, consisting of randomly generated graphs with known maximum clique and of graphs derived from various practical applications. The performance of HGA on these graphs is very satisfactory both in terms of solution quality and running time. Despite its simplicity, HGA dramatically improves on all previous approaches based on genetic algorithms we are aware of, and yields results comparable to those of more involved heuristic algorithms based on local search. This provides empirical evidence of the effectiveness of heuristic based genetic algorithms as a search technique for solving the maximum clique problem, which is competitive with respect to other (variants of local) search techniques, like simulated annealing and tabu search.

1 Introduction

A clique of a (undirected) graph is a subgraph such that all pairs of distinct nodes are connected by an edge. A maximum clique of a graph is a clique of the graph having the largest number of nodes.

Computing the maximum clique of a graph (MC problem) is a paradigmatic combinatorial optimization problem which is encountered in many different real life applications, such as cluster analysis, information retrieval, mobile networks, computer vision (see, e.g., the survey paper [24]).

The MC problem is highly untractable. It is one of the first problems which has been proven to be NP-complete [17]. Moreover, even its approximations within a constant factor are NP-hard [10]. In particular, a recent result [14] states that if $\mathtt{NP} \neq \mathtt{P}$ then no polynomial time algorithm can approximate the maximum clique to within a factor of $N^{1/2-\epsilon}$ for any $\epsilon > 0$, where N is the number of nodes of the graph.

Due to these strong negative results on the computational complexity of the MC problem, many researchers have concentrated their effort to design efficient heuristics yielding satisfactory sub-optimal solutions for specific classes of graphs including random graphs with known maximum clique size and graphs obtained from various areas of applications (cf. [24, 15]).

In particular, a number of optimization and approximation algorithms for the MC problem have been presented at the second DIMACS International Implementation Challenge on NP-hard problems, organized in 1993 by the center for Discrete Mathematics and Theoretical Computer Science [15]. An interesting collection of test graphs for the MC problem has been made available for the comparative study of the performance of these algorithms, consisting of graphs derived from different problems such as fault diagnosis and coding theory (see the WWW site `http://dimacs.rutgers.edu/Challenges/index.html`).

Genetic algorithms (GA) have been successfully applied to various different hard combinatorial optimization problems (cf. [1]). However, the MC problem seems to be a particularly hard problem for genetic algorithms. The experimental results contained in previous works on this problem show that in general genetic algorithms have poor performance when compared with other local search techniques [7, 25], even when suitable representations and ad-hoc genetic operators are used, like in [12, 23], or when genetic operators are combined with other heuristic procedures and local optimization techniques [11, 6]. In particular, experiments on the DIMACS benchmark graphs indicate that genetic algorithms are rather inferior to other heuristic algorithms based on simulated annealing or tabu search, both with respect to the quality of solutions and the running time. It is not yet clear which graph properties render the MC problem a hard problem for genetic algorithms. In a recent work [31], various GA hardness measures like graph density, or Davidor's epistasis measure are applied to the MC problem: the results of the experiments do not allow to identify any GA hardness parameter for the MC problem.

It is well known that in genetic algorithms the fitness function plays a crucial role in directing the search: therefore, in order to enhance the GA performance on the MC

*On leave from University Cá Foscari of Venice, Italy. This research was supported by the *Stichting Informatica Onderzoek in Nederland* within the context of the project no. 612-32-001.

problem, previous works have considered definitions of the fitness function which combine information on various properties of the subgraph represented by a chromosome, like its density and size, possibly with the integration of some parameter in order to bias the search on better individuals as the execution of the GA proceeds. Moreover, the addition of various other features has been investigated, which vary from the use of sophisticated genetic operators to the incorporation of heuristics and of local optimization techniques.

In this paper we adopt an alternative approach which looks somehow naive, yet turns out to be rather effective. We consider a genetic algorithm with a rather simple fitness function, which describes just one property of the subgraph, and embed a simple heuristic as local optimizer for producing maximal cliques[1]. In this way the concerns of the exploration (determined by the fitness function) and exploitation (determined by the local optimizer) are neatly defined and separated, with the exploration looking for large subgraphs, and the exploitation trying to improve the quality of the individuals. This combination yields very satisfactory results on the DIMACS benchmark graphs both with respect to the quality of solutions (i.e., the size of the clique) and the running time. These results are comparable to the best results obtained by all the algorithms presented at the second DIMACS implementation challenge.

More precisely, the proposed heuristic algorithm (HA) consists of three steps, to be applied sequentially to a subgraph. First, few randomly chosen nodes are added to the subgraph; next, a clique is extracted from the subgraph using a naive randomized procedure; finally, this clique is extended by means of a sequential greedy heuristic. The genetic algorithm (GA) is also fairly simple: the population consists of strings of bits describing subgraphs of the given graph in the expected way, and the fitness function just counts the size of the subgraph represented by a chromosome, if this is clique, and is zero otherwise. The heuristic algorithm is combined with this genetic algorithm by applying HA to each member of the population at every iteration.

Despite its simplicity, the resulting heuristic based genetic algorithm (HGA) outperforms all the previous proposals based on genetic algorithms we are aware of, and provides experimental evidence that heuristic based genetic algorithms for the MC problem are competitive with more sophisticated heuristic algorithms based on local search like simulated annealing and tabu search (see, e.g., [15]).

The rest of the paper is organized as follows. The next section introduces a heuristic for the MC problem, and Section 3 describes its integration with a simple genetic algorithm. In Section 4 the resulting heuristic based GA is tested and its performance is compared with other heuristic algorithms. Section 5 discusses related work, and Section 6 contains some concluding observations.

2 A Novel Heuristic for the Maximum Clique Problem

In this section, we introduce a naive yet effective heuristic for the MC problem, which is used (in Section 3)to enhance the performance of a genetic algorithm. Here and in the sequel we assume that the nodes of the graph are arranged into a sequence $\langle n_1, \ldots, n_N \rangle$ for some N, in such a way that one can refer to the i-th node of the graph as the element of the sequence in the i-th position.

[1]A maximal clique is a clique that (viewed as set of nodes) is not as subset of another clique. Note that a maximum clique is maximal but not vice versa.

Search algorithms for finding sub-optimal solutions to combinatorial optimization problems are often based on heuristics (cf. [1]). One of the best known heuristics for the MC problem is the (sequential) greedy heuristic, which builds a maximal clique by the repeated addition of a node into a partial clique, starting from a (possibly empty) partial clique. The performance of an algorithm based on such heuristic depends on the starting point one chooses as well as on the way a node to be added to the partial clique is chosen at each iteration (see e.g. [24]).

We propose a heuristic algorithm which includes a naive greedy heuristic procedure. The idea is to build a maximal clique by starting with a random subgraph of the given graph: this graph is first enlarged by adding some nodes randomly selected, next it is reduced to a clique, and finally it is extended using a naive sequential greedy heuristic. More precisely, the algorithm consists of three steps to be applied sequentially to the input subgraph:

1. `Relax:` (Enlarge the subgraph)

   Add few new nodes randomly chosen from the graph.

2. `Repair:` (Extract a clique)

   Choose randomly a position idx with $1 \leq idx \leq N$:

   (a) for $i = idx$ to N: if n_i belongs to the subgraph then
   - either delete n_i, or
   - for $j = i + 1$ to N: delete n_j if it belongs to the subgraph and n_j is not connected with n_i;
     for $j = 1$ to $i - 1$: delete n_j if it belongs to the subgraph and n_j is not connected with n_i.

   (b) for $i = idx - 1$ downto 1: if n_i belongs to the subgraph then
   - either delete n_i, or
   - for $j = i - 1$ downto 1: delete n_j if it belongs to the subgraph and n_j is not connected with n_i.

3. `Extend:` (Enlarge the clique)

   Choose randomly a position idx with $1 \leq idx \leq N$;

   (a) for $j = idx$ to N: add n_j if it is connected with all the nodes of the subgraph (obtained so far).

   (b) for $j = 1$ to $idx - 1$: add n_j if it is connected with all the nodes of the subgraph (obtained so far).

It is easy to check that the outcome of HA applied to a subgraph is a *maximal* clique. Observe that HA is heavily based on random choices. In particular, in the repair step, the two substeps of each `for` statement are supposed to be selected with equal probability (1/2). Moreover, in steps 2 and 3 the random choice of the index idx determines different selection schedules of the graph nodes, which affect the construction of the maximal cliques.

This heuristic algorithm can be iteratively applied to a set of starting subgraphs randomly chosen, by keeping the best solution found as the final solution (multistart approach). We have tested the performance of HA on some DIMACS graphs. On a population of 50 elements, one iteration yields reasonably good results. For instance, HA performs very well on families of graphs such as the CFat, Johnson, and Hamming graphs, finding always the maximum clique. The performance of HA on more 'difficult' graph instances is positive: for instance, on the Keller6 graph, a popular test graph with 3361 nodes and 4619898 edges arising from code theory, which is consider a 'large' and 'difficult' instance for the MC problem having maximum clique of size ≥ 59, ten runs of HA generate as best solution a maximal clique of size 43 in 8.2 seconds, while on the average HA produces solutions of size 40.6 in 8.3 seconds.

Thus the performance of HA is reasonably satisfactory, and already competitive with other heuristic algorithms: for instance, on the DIMACS graphs the HA algorithm outperforms the evolutionary approach introduced in [4].

In general, the performance of a multistart version of HA iterated 100 times on the DIMACS graphs seems comparable to the Continuous Based Heuristic algorithm (CBH) introduced by Gibbons et al. [13].

We will illustrate in the next section how HA can be embedded into a genetic algorithm obtaining in such a way a more powerful algorithm for the MC problem.

```
begin
$t = 0$
initialize $P(t)$
apply HA to $P(t)$
evaluate $P(t)$
while (not termination-condition) do
begin
$t = t + 1$
select $P(t)$ from $P(t-1)$
apply diversification procedure to $P(t)$
alter $P(t)$ (via mutation and crossover)
apply HA to $P(t)$
evaluate $P(t)$
end
end
```

Figure 1: The heuristic based genetic algorithm HGA

3 A Heuristic Based Genetic Algorithm

We start by considering a simple genetic algorithm for the MC problem, based on a standard graph encoding technique and on a naive fitness function which has to be maximized, and then we refine such an algorithm by embedding the HA procedure introduced in the previous section.

The main features of a simple genetic algorithm (GA) for the MC problem can be summarized as follows:

Representation A chromosome consists of a string of bits, one bit for each node of the considered graph (thus of length equal to the number of nodes of the graph); it characterizes the subgraph consisting of those nodes whose relative entries in the string are equal to 1.

Fitness The fitness of a chromosome is equal to the size of the subgraph it represents (i.e., the number of 1's occurring in the string) if this is a clique, and it is equal to zero, otherwise.

GA type Generational genetic algorithm with elitist selection mechanism which copies the two best individuals of a population to the population of the next generation [16]. Moreover, the *keep-two-best* replacement mechanism of LibGa is used, which chooses the two best chromosomes among the set consisting of the two parents and their two offsprings.

Genetic operators Uniform crossover [32] and swap mutation.

We also employ a diversification procedure applied with very low probability (which depends on the total fitness of the current population), which replaces a chromosome selected for reproduction with a new one quasi-randomly generated.

As one would expect, such a genetic algorithm does not perform very well, getting easily stuck on local sub-optimal solutions of scarce quality. However, the combination of this GA with the HA procedure enhances its performance in a dramatic way.

The idea is to apply HA to the chromosomes of the population at each iteration of the GA before computing their fitness. In this way, the resulting population consists of chromosomes representing maximal cliques, thus the fitness of a chromosome is just the size of the clique it represents. Notice that a chromosome is not guaranteed to remain a clique after mutation and crossover are applied; however, the application of HA transforms the elements of the population into (maximal) cliques. Thus, HA plays the role of a repair algorithm for 'correcting' infeasible solutions generated by the application of the genetic operators (cf. [21]). However, HA does more than just repairing, since it acts also on chromosomes which are already cliques, and it can possibly transform a clique into a completely different one.

In this way one obtains a heuristic based genetic algorithm (HGA) which is summarized in Figure 1.

The above described simple genetic algorithm GA belongs to a class of genetic algorithms for which theoretical results on global convergence have been established (cf. [9, 27]). Notice that the integration of the HA procedure does not affect the global convergence properties of the GA.

Concerning the computational complexity of the algorithm, it can be easily shown that the worst case complexity per iteration is $O(N^2)$, where N is the number of nodes of the graph. In terms of memory usage the algorithm needs only to store the input graph and the GA population, which consists of fifty chromosomes of length N.

It is difficult to provide some theoretical study of the bounds on the performance of our algorithm, since this is in general a rather complex kind of analysis, for which few general results exist (cf. [5, 24, 26]). We can only state a rather weak qualitative theoretical result (cf. [20]) which follows from the fact that we use a greedy heuristic combined with a elitist selection mechanism, roughly stating that for a certain class of random graphs the algorithm will always return maximal cliques of size at least equal to about the half of the size of the maximum clique of that graph. This is not a very informative bound, as we will see in the next section that HGA performs very well on the DIMACS benchmark graphs, generating often maximum cliques.

4 Experimental Results

In this section, we evaluate experimentally the performance of our heuristic based genetic algorithm on the DIMACS benchmark graphs. These graphs provide a valuable source for testing the performance of algorithms for the MC problem, because they arise from various different areas of application. More specifically, eleven different families of graphs are considered, which depend on the method used to generate them: the CFat graphs arise from fault diagnosis problems

population size	mutation rate	crossover rate	number of iterations
50	0.1	0.8	100

Figure 2: GA parameters

[3], the Hamming and Johnson graphs are from coding theory problems [28]; the Keller graphs are based on Keller's conjecture on tilings using hypercubes [19]; the Gen graphs are artificially generated graphs with large known embedded clique, while the San and Sanr graphs are random graphs with known maximum clique (cf. [15]); the Brock graphs are generated in such a way that the expected maximum clique is much smaller than the real one (cf. [15]); the PHat graphs are random graphs with large variance in the node degree distribution and larger cliques than the usual random graphs [29]; the MANN graphs are generated for the set covering problem (cf. [15]); and the Cx.y and DISJx.y graphs are random graphs of size x and density[2] y.

The HGA algorithm has been implemented in C using the LibGa system [8], with slight code modifications in order to include the diversification procedure into the selection mechanism, and the HA procedure into the objective function. No attempt was made to optimize the code for this specific problem. The GA parameters are shown in Figure 2. The initial population has been generated randomly.

The algorithm was run on a multi-user Silicon Graphics IRIX Release 6.2 IP25 (194 MHZ clock, Main memory size: 512 Mbytes). The results of the experiments are based on 10 runs of HGA on each of the considered graphs, and are summarized in Tables 1 and 2. The first three columns indicate the graph name, size (i.e., the number of nodes) and density, respectively. The other columns indicate: the minimal and average user time (in seconds) of a run to reach the best solution; the minimal and average number of iterations of a run to reach the best solution; and the average and best size of the cliques generated by HGA. Finally, the last column contains the results obtained by other heuristic algorithms: the column of Table 1 labeled 'Best DIMACS' contains the best clique size found by all the fifteen heuristic algorithms presented at the second DIMACS Challenge [15]; while the column of Table 2 labeled 'SG Range' contains the range of the results found by Soriano and Gendrau [30] by using three different versions of tabu search (unfortunately, for these graphs we do not have the best results obtained by all the DIMACS heuristic algorithms). Moreover, values are labeled with a '*' if they have been proven (by means of exact algorithms) to be global optima.

The heuristic algorithms presented at the second DIMACS Challenge are based on various approaches, like tabu search, simulated annealing, and neural networks. The best four of these algorithms were able to reach the values of the DIMACS column on 23-27 among the 34 graph instances of Table 1; on the other hand, HGA could reach the best values on 25 graph instances. On the remaining 9 instances, the performance of HGA is rather satisfactory, yielding maximal cliques of size which is less than the corresponding value in the DIMACS column of 1 in four cases (DSJC500, MANN_a27, brock800_4 and p_hat1500-1), of 2 in three cases (C500.9, C4000.5, MANN_a81), of 3 in three cases (C1000.9, MANN_a45 and gen400_p0.9_55), and of 6 in the remaining two cases (C2000.9, keller6). Observe that these two latter graphs are considered rather difficult, and the results we obtained outperform all the previous approaches based on genetic algorithms we are aware of. Moreover, the amount of time required to find these sub-optimal solutions is very short, also due to the fact that the execution is stopped after 100 iterations. For instance, by setting the termination condition of the GA to 500 iterations, HGA could find a maximal clique of the keller6 graph having size 58 in about 1 hour and 10 minutes.

Concerning the graph instances of Table 2 the results show that the performance of HGA is comparable to that of the SG algorithms based on tabu search, thus showing the competitivity of heuristic genetic algorithms with respect to heuristic algorithms based on tabu search.

To give an idea of the performance of other genetic algorithms for the maximum clique problem, we have included in Table 3 the results of the hybrid genetic algorithm GMCA given in [6]. According to the authors, the performance of GMCA on the DIMACS graph instances seems to be comparable to the CBH algorithm by Gibbons et al. [13], which is outperformed by other heuristic based algorithms (cf. [15]) as well as by HGA, also with respect to the running time. We will compare GMCA and HGA in more detail in the next section.

In general, it seems that the MC problem is easy to solve on graph instances from the families Cfat, Johnson and Hamming for all kinds of heuristic based algorithms. On other graph families, the results seem to depend heavily on the type of heuristic used. For instance, even if both GMCA and HGA are based on genetic algorithms, the different heuristics heavily influence their relative performance: for instance, GMCA does not perform well on the San graphs, while HGA performs very well on these graph instances.

5 Related Work

The idea of incorporating heuristics into genetic algorithms for solving combinatorial optimization problems is not new [22, 18], and it has been successfully applied to various different combinatorial optimization problems. However, in the case of the MC problem, the various approaches based on genetic algorithms so far introduced did not seem to be competitive with respect to other local search techniques like simulated annealing and tabu search (cf., e.g., [15]). The performance of our heuristic based genetic algorithm on the DIMACS benchmark graphs shows that this is not the case, and provides empirical evidence of the effectiveness of heuristic based genetic algorithms for the approximate solution of the MC problem.

It is not easy to identify the reasons why rather sophisticated approaches based on genetic algorithms could not perform as well as other local search techniques for the MC problem. In this section, we describe the main features of these approaches and try to draw some conclusions by comparing them with our heuristic based genetic algorithm.

In [7] Carter and Park point out the scarce performance of a simple genetic algorithm for the MC problem; this GA is the one we have used in Section 3. They try to enhance the GA performance by using a more sophisticated fitness function, which incorporates a control variable that determines the degree to which subgraphs are penalized for not being cliques, and leads to an annealing schedule on that variable whereby the penalty is increased as time progresses. More-

[2]Recall that the density of a graph with N nodes is equal to the number of its edges divided by the maximum number of edges of a graph with N nodes, i.e. $N * (N-1)/2$

over, they add other features like specific crossover operators and greedy replacement. They perform various experiments and arrive to conclude that genetic algorithms have to be heavily customized in order to perform well. In particular, in [25], an experimental study is carried out whose results lead the authors to state that simulated annealing is better than genetic search, both with respect to quality of solutions and time complexity. For instance, on the `keller5` graph, which they describe as a 'difficult' instance for the MC problem, the various variants of genetic algorithms they apply can only find cliques of size at most 25, requiring a rather long running time (a parallel implementation on a set of six dedicated Sparc 1+ workstations required 21.2 hours of CPU time), while the simulated annealing algorithm could find the maximum clique in 1.6 hours. However, `HGA` could find the maximum clique of `keller5` in 10 seconds, thus outperforming the simulated annealing algorithm. This illustrates how difficult it is to compare empirically the performance of different local search techniques.

Also Foster and Soule [12] develop a genetic algorithm for the MC problem which involves a sophisticated fitness function. They consider an alternative encoding based on integer numbers instead of bits, in such a way that a chromosome represents a set of subgraphs instead of just one single subgraph as in the standard bit representation. As a consequence, they define the fitness function to be the maximum of the fitness of all the subgraphs described by a chromosome, where the fitness of a subgraph is defined by combining its density and size. Moreover, they consider a time weighting of the resulting fitness function which bias the selection of fitter elements in the population as the execution proceeds, in a way similar to the annealed schedule used by Carter and Park [7]. The results of the experiments on some `DIMACS` graphs, reported in [31] are still rather inferior to those obtained using other local search techniques, especially on the `San` graphs and on some instances of the `Brock` graphs (the authors do not report the running time). Notice that the performance of `HGA` on these graphs is very good, thus showing that the performance heavily depends on the specific GA used.

Other proposals are based on the direct incorporation of a heuristic into the GA. For instance, Murthy et al. in [23] introduce a GA for the MC problem using a fitness function which measures the density of the subgraph (which is 1 for cliques) plus a small term depending on the size of the subgraph which allows to discriminate between cliques of different sizes. Moreover, a simple heuristic is used consisting of two steps: some nodes are added to the elements of the population before the application of the fitness function, and some nodes are deleted from the offsprings. The results reported are based on experiments employing graphs of rather small size (up to 50 nodes) which renders difficult a comparison with other algorithms.

In [11] Fleurent and Ferlant investigate the combination of heuristics and genetic algorithms: they propose an object-oriented implementation of heuristic search methods for graph coloring, satisfiability and maximum clique. In particular, for the MC problem, experiments are conducted on hybrid genetic algorithms which incorporate tabu search or other local search techniques as alternative mutation operators. The performance of these algorithms seems to be rather satisfactory as regards the quality of solutions, but rather disappointing as regards the running time. For instance, on the `MANN_a45` graph, these hybrid genetic algorithms produce a clique of size 342 in more than 7 hours (average CPU time) on a SuperSPARC. While a clique of the same size could be generated by `HGA` in 2.4 minutes (average CPU time).

Finally, in [6] Bui and Eppley introduce a hybrid genetic algorithm for the MC problem. Again, the fitness function is the sum of the density and the size of the subgraph, with a weight attached to each addend in order to bias the search either on the size of the subgraph or on its quality as clique (i.e., how much the density of the subgraph approaches 1). The genetic algorithm is augmented with a pre-processing phase which orders the nodes of the graph in a suitable way by means of two kinds of procedures. Moreover, local optimization is applied to the offsprings: first, the nodes are reordered according to their degree, and then a *two-steps* greedy heuristic is applied which deletes some nodes and add some other nodes in order to improve the fitness of the chromosomes. The authors say that the performance of the resulting algorithm on the `DIMACS` graphs (see Table 3) is comparable to the CBH algorithm by Gibbons et al. [13]. However, the results are rather inferior to those obtained using other heuristic algorithms based on tabu search and simulated annealing. For instance, on the `keller5` graph, the hybrid genetic algorithm finds as best solution a clique of size 18 in more than 7 minutes (average CPU time) on a SUN SPARC LX, while `HGA` finds the maximum clique of size 27 in less than 20 seconds (average time).

A characteristic which seems shared by the above discussed approaches is the use of rather sophisticated fitness functions, which bias the search towards a combination of two different graph properties, namely the size and the density of the graph.

In contrast, the fitness function used in `HGA` describes just one property, that is the graph size. We can use such a simple fitness function because in `HGA` at each iteration the population consists of (maximal) cliques, thanks to the application of `HA` to the chromosomes. In this way, the search pressure determined by the fitness function is exclusively directed towards large graphs, while the application of the heuristic `HA` together with the genetic operators, are responsible for improving the quality of the graphs.

This could partly explain why our simple heuristic based genetic algorithm outperforms these previous GA approaches: it combines the search for a large graph and the search for a clique in a neat way, by incorporating the search for 'maximum' and 'clique' into the fitness function and the heuristic procedure, respectively.

6 Conclusion

This paper has tackled the problem of finding a maximum clique in an undirected graph by means of heuristic based genetic algorithms. We have proposed a combination of a simple genetic algorithm and a naive heuristic which turns out to give very satisfactory results, comparable to those obtained using more sophisticated heuristic algorithms based on simulating annealing and tabu search.

We intend to investigate whether the addition of more sophisticated diversification techniques, like for instance reactive local search (cf. [2]), can further improve the performance of `HGA`.

Acknowledgements

I would like to thank Guszti Eiben, Robert Giacomini, Giuliano Grossi, Marcello Pelillo, Jan Rutten and Adri Steenbeek for useful discussions on the subject of this paper.

References

[1] E. Aarts and J.K. Lenstra (Eds.). *Local Search in Combinatorial Optimization.* John Wiley and Sons, 1997.

[2] R. Battiti and M. Protasi. Reactive local search for the maximum clique problem. Technical report, ICSI, Berkeley, California, TR-95-052, September 1995.

[3] P. Berman and A. Pelc. Distributed fault diagnosis for multiporcessor systems. In *20th Annual Int. Symp. on Fault-Tolerant Computing*, pages 340–346, 1990.

[4] I. Bomze, M. Pelillo, and R. Giacomini. Evolutionary approach to the maximum clique problem: Empirical evidence on a larger scale. *Developments in Global Optimization*, pages 95–108, 1997.

[5] R. Boppana and J.H.M. Korst. Approximating maximum independent sets by excluding subgraphs. *BIT*, 32:180–196, 1992.

[6] T.N. Bui and P.H. Eppley. A hybrid genetic algorithm for the maximum clique problem. In L.J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms (ICGA)*, pages 478–484. Morgan Kaufmann, 1995.

[7] B. Carter and K. Park. How good are genetic algorithms at finding large cliques: an experimental study. Technical report, Boston University, Computer Science Department, MA, October 1993.

[8] A.L. Corcoran and R.L. Wainwright. Using LibGA to develop genetic algorithms for solving combinatorial optimization problems. In Lance Chambers, editor, *The Application Handbook of Genetic Algorithms*, pages 143–172. CRC Press, 1995.

[9] A.E. Eiben, E.H.L. Aarts, and K.M. Van Hee. Global convergence of genetic algorithms: a Markov chain analysis. In H.P. Schwefel and R. Manner, editors, *Parallel Problem Solving from Nature*, pages 4–12. Springer-Verlag, LNCS 496, 1991.

[10] U. Feige, S. Goldwasser, S. Safra, L. Lovász, and M. Szegedy. Approximating clique is almost NP-complete. In *Proc. 32nd Annual IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 2–12, 1991.

[11] C. Fleurent and J.A. Ferland. Object-Oriented imlementation of heuristic search methods for graph coloring, maximum clique, and satisfiability. In D. Johnson and M. Trick, editors, *Cliques, Coloring and Satisfiability.* AMS, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol 26, 1996.

[12] J.A. Foster and T. Soule. Using genetic algorithms to find maximum cliques. Technical report, Dept. of Computer Science, Univ. Idaho, 12 1995.

[13] L.E. Gibbons, D.W. Hearn, and P. M. Pardalos. A continuous based heuristic for the maximum clique problem. In D. Johnson and M. Trick, editors, *Cliques, Coloring and Satisfiability.* AMS, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol 26, 1996.

[14] J. Hastad. Clique is hard to approximate within $n^{1-\epsilon}$. In *Proc. 37th Annual IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 627–636, 1996.

[15] D. Johnson and M. Trick (Eds.). *Cliques, Coloring, and Satisfiability.* AMS, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol 26, 1996.

[16] K.A. De Jong. An analysis of the behaviour of a class of genetic adaptive systems. Doctoral Dissertation, University of Michigan, Dissertation Abstract International 36(10), 5140B, 1975.

[17] R.M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Plenum Press, NY, 1972.

[18] A. Kolen and E. Pesch. Genetic local search in combinatorial optimization. *Discrete Applied Mathematics*, 48:273–284, 1994.

[19] J.C. Lagarias and P.W. Shor. Keller's cube-tiling conjecture is false in high dimensions. *Bullettin AMS*, 27(2):279–283, 1992.

[20] D.W. Matula. The largest clique size in a random graph. Technical report, Dept. of Computer Science, Southern Methodist University, Dallas, Texas, 75275, 1976.

[21] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, Berlin, 1994.

[22] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer. Evolution algorithms in combinatorial optimization. *Parallel Computing*, 7:65–85, 1988.

[23] A.S. Murthy, G. Parthasarathy, and V.U.K. Sastry. Clique finding - a genetic approach. In *Proceedings of the 1st IEEE Conference on Evolutionary Computation*, pages 18–21. IEEE Press, 1994.

[24] P.M. Pardalos and J. Xue. The maximum clique problem. *Journal of Global Optimization*, 4:301–328, 1994.

[25] K. Park and B. Carter. On the effectiveness of genetic search in combinatorial optimization. In *Proceedings of the 10th ACM Symposium on Applied Computing.* ACM Press, 1995.

[26] M. Peinado. Improved lower bounds for the randomized Boppana-Halldorsson algorithm. In *First Annual Computing and Combinatorics Conference (COCOON'95).* Springer-Verlag, 1995.

[27] G. Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, 1994.

[28] N.J.A. Sloane. Unsolved problems in graph theory arising from the study of codes. Graph Theory Notes of New York XVIII, 1989.

[29] P. Soriano and M. Gendreau. Solving the maximum clique problem using a tabu search approach. *Annals of Operation Research*, 41:385–403, 1992.

[30] P. Soriano and M. Gendreau. Tabu search algorithms for the maximum clique problem. Technical report, CRT University of Montreal, Canada, CRT-968, 1994.

[31] T. Soule and J.A. Foster. Genetic algorithm hardness measures applied to the maximum clique problem. In T. Bäck, editor, *Seventh International Conference on Genetic Algorithms*, pages 81–88. Morgan Kaufmann, 1997.

[32] G. Syswerda. Uniform crossover in genetic algorithms. In J. Schaffer, editor, *Third International Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann, 1989.

Bibliographical Note

Elena Marchiori is researcher in Computer Science at the CWI in Amsterdam, and at the Department of Computer Science of Leiden University. Her main research interests include formal methods and semantics of programming languages, and artificial intelligence techniques.

Name	Nodes	Density	Time to Best		Iter to Best		Clique Size		Best
			Min	Avg	Min	Avg	Avg	Best	DIMACS
C125.9	125	0.898	0.2	0.3	6	8.7	34	34	34*
C250.9	250	0.899	1.1	3.6	9	31.6	42.6	44	44*
C500.9	500	0.900	9.8	17.3	31	66.5	52.9	55	57
C1000.9	1000	0.901	70.8	93.3	75	86.6	58	64	67
C2000.9	2000	0.900	225.2	329.5	63	99	67.1	69	75
C2000.5	2000	0.500	3.1	61.8	0	19	14.4	16	16
C4000.5	4000	0.500	56.6	306.8	4	23.7	15.4	16	18
DSJC500.5	500	0.501	0.6	3.2	2	13.9	12.3	13	14
DSJC1000.5	1000	0.500	6.3	24.9	7	30.7	13.7	15	15*
MANN_a27	378	0.990	1.4	3.0	2	5.1	125	125	126*
MANN_a45	1035	0.996	47.9	143.9	13	42	342	342	345*
MANN_a81	3321	0.998	511.4	1149.1	13	30.7	1096	1096	1098
brock200_2	200	0.496	0.1	1.4	1	30.4	11.6	12	12*
brock200_4	200	0.657	0.1	1.1	2	22.2	15.6	17	17*
brock400_2	400	0.749	0.5	3.3	2	19.2	23.5	29	29*
brock400_4	400	0.748	2.2	4.5	12	26.7	24.1	33	33*
brock800_2	800	0.651	1.1	15.5	1	27.4	18.8	21	21
brock800_4	800	0.649	2.3	23.8	2	27	18.7	20	21
gen200_p0.9_44	200	0.900	1.4	3	16	37.3	40.7	44	44*
gen200_p0.9_55	200	0.900	0.4	0.7	5	8.2	55	55	55*
gen400_p0.9_55	400	0.900	3.1	13.3	12	57.7	49	52	55
gen400_p0.9_65	400	0.900	4	13.8	16	59.5	55.8	65	65
gen400_p0.9_75	400	0.900	3.7	7.6	16	32.1	65	75	75
hamming8-4	256	0.639	0.07	0.08	0	0.7	16	16	16*
hamming10-4	1024	0.828	11.3	34.8	12	40.2	37.8	40	40*
keller4	171	0.649	0.03	0.05	0	0.7	11	11	11*
keller5	776	0.751	5.7	10.4	12	19.8	26.3	27	27
keller6	3361	0.818	246.6	370	28	50	51.4	53	59
p_hat300-1	300	0.243	0.1	1.3	0	13.9	8	8	8*
p_hat300-2	300	0.488	0.4	1.7	4	18.4	25	25	25*
p_hat300-3	300	0.744	1.2	3.6	10	31	35.2	36	36*
p_hat700-1	700	0.249	2.6	12.8	5	30	10.3	11	11*
p_hat700-2	700	0.497	4	5.7	10	15	43.9	44	44*
p_hat700-3	700	0.748	6.9	12.6	15	27.8	61.2	62	62
p_hat1500-1	1500	0.253	3.7	30.2	1	15.9	10.4	11	12*
p_hat1500-2	1500	0.506	24.9	44	18	32.1	64.7	65	65
p_hat1500-3	1500	0.753	69	98.4	44	61.5	91.4	94	94

Table 1: Results on DIMACS benchmark graphs: 'Best DIMACS' denotes the best clique size found by the fifteen heuristic algorithms presented at the second DIMACS challenge (* indicates global optimality).

Name	Nodes	Density	Time to Best		Iter to Best		Clique Size		SG
			Min	Avg	Min	Avg	Avg	Best	Range
c-fat200-1	200	0.077	0.03	0.04	0	0	12	12	12*
c-fat200-2	200	0.163	0.04	0.04	0	0	24	24	24*
c-fat200-5	200	0.426	0.07	0.07	0	0	58	58	58*
c-fat500-1	500	0.036	0.16	0.17	0	0	14	14	14*
c-fat500-2	500	0.073	0.13	0.15	0	0	26	26	26*
c-fat500-5	500	0.186	0.16	0.17	0	0	64	64	64*
c-fat500-10	500	0.374	0.31	0.31	0	0	126	126	126*
johnson8-2-4	28	0.555	0.00	0.00	0	0	4	4	4*
johnson8-4-4	70	0.768	0.01	0.01	0	0	14	14	14*
johnson16-2-4	120	0.764	0.03	0.03	0	0	8	8	8*
johnson32-2-4	496	0.878	0.2	0.2	0	0	16	16	16*
hamming6-2	64	0.904	0.01	0.01	0	0	32	32	32*
hamming6-4	64	0.349	0.01	0.01	0	0	4	4	4*
hamming8-2	256	0.968	0.4	0.5	1	1.6	128	128	128*
hamming10-2	1024	0.990	12.7	13.5	3	3.8	512	512	512*
san200_0.7_1	200	0.700	0.1	0.3	1	5.2	30	30	16−30*
san200_0.7_2	200	0.700	0.06	1.6	0	27.2	17	18	15−18*
san200_0.9_1	200	0.900	0.3	0.7	3	6.6	70	70	47−70*
san200_0.9_2	200	0.900	0.5	0.7	5	7.4	60	60	41−60*
san200_0.9_3	200	0.900	1.2	3.4	15	42.8	38	44	36−44*
san400_0.5_1	400	0.500	0.2	4.1	0	14.9	9.8	13	8−13*
san400_0.7_1	400	0.700	0.2	2.6	0	13.1	34.6	40	21−40*
san400_0.7_2	400	0.700	0.1	2.4	0	12.7	21.6	30	18−30*
san400_0.7_3	400	0.700	0.7	5	3	26.5	17.1	22	17−22*
san400_0.9_1	400	0.900	0.8	3.5	2	11.1	100	100	100*
san1000	1000	0.501	1.2	11.9	0	11.9	10.5	15*	10
sanr200_0.7	200	0.697	0.3	0.9	5	17.9	17.4	18	18*
sanr200_0.9	200	0.898	1.7	2.4	21	30.1	40.7	42	41−42*
sanr400_0.5	400	0.501	0.1	4.4	0	27.9	11.9	13	12−13*
sanr400_0.7	400	0.700	1.1	4	6	24.6	19.8	21	20−21
brock200_1	200	0.745	0.1	0.7	2	13.4	18.2	21	20−21*
brock200_3	200	0.605	0.0	0.9	0	15.7	13.9	15	14
brock400_1	400	0.748	1.7	6.2	9	34.7	23.6	25	24−25
brock400_3	400	0.748	1.2	4.7	6	26.7	23.3	25	24−25
brock800_1	800	0.649	8.5	19.2	17	37.5	19.2	21	20−21
brock800_3	800	0.651	5.1	20.5	8	35.7	19.1	20	20−21
p_hat500_1	500	0.253	1	5.2	4	22.3	9	9	9*
p_hat500_2	500	0.505	1.4	3.4	5	13.9	35.7	36	36*
p_hat500_3	500	0.752	2.8	8	9	28.4	49.6	50	49-50
p_hat1000_1	1000	0.245	2.3	25.6	2	29.7	9.9	10	10
p_hat1000_2	1000	0.490	9.4	21.5	12	28.3	45.6	46	46
p_hat1000_3	1000	0.744	32.7	38.7	21	43.1	65.5	66	66

Table 2: Results on DIMACS benchmark graphs: 'SG Range' denotes the range of the results obtained by the three algorithms by Soriano and Gendreau based on tabu search.

Name	GMCA Time (Avg)	GMCA Best	HGA Best	HGA Time (Avg)
c-fat200-1	8.2	12	12*	0.04
c-fat500-1	33.2	14	14*	0.17
johnson16-2-4	6.0	8	8*	0.03
johnson32-2-4	187.4	16	16*	0.2
keller4	13.3	11	11*	0.05
keller5	438.1	18	27*	10.4
hamming10-2	886.6	512	512*	13.5
hamming8-2	53.0	128	128*	0.5
san200_0.7_1	51.7	30	30*	0.3
san400_0.5_1	411.2	7	13*	4.1
san400_0.9_1	128.6	50	100*	3.5
sanr200_0.7	21.5	17	18*	0.9
sanr400_0.5	69.6	12	13*	4.4
san1000	704.3	8	15*	11.9
brock200_1	27.9	20	21*	0.7
brock400_1	118.8	20	25	6.2
brock800_1	460.8	18	21	19.2
p_hat300_1	20.0	8	8*	1.3
p_hat500_1	49.1	9	9*	5.2
p_hat700_1	310.1	8	11*	12.8
p_hat1000_1	671.0	8	10*	25.6
p_hat1500_1	1580.3	10	11	30.2
MANN_a27	121.8	125	125	3.0
MANN_a45	916.9	337	342	143.9

Table 3: Best Results on DIMACS benchmark graphs of the GMCA hybrid genetic algorithm (to facilitate a comparison, we report also the best result and the average time of HGA).