Genetic Local Search for the Maximum Clique Problem

Elena Marchiori
Free University
Department of Mathematics and Computer Science
de Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
E-mail:elena@cs.vu.nl

Abstract

Genetic local search (GLS) consists of the application of genetic operators to a population of local optima produced by a local search procedure. The process is iterated until either a solution is found or a maximal number of generations is reached. This paper introduces a GLS algorithm for solving the Maximum Clique problem (MC). Computational tests indicate that GLS outperforms previous approaches based on genetic algorithms, and yields results comparable to those of the best heuristic algorithms tested at the second DIMACS implementation challenge. However, GLS is outperformed by the reactive local search algorithm by Battiti and Protasi [2] which uses information about past events to guide the search process.

Keywords: maximum clique problem, local search, genetic algorithms.

1 Introduction

A clique of a (undirected) graph is a subgraph s.t. 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 is a paradigmatic combinatorial optimization problem which is encountered in many different real life applications, such as cluster analysis, information retrieval, mobile networks, and computer vision (see, e.g., the survey in [3]).

MC is highly untractable: it is one of the first problems which has been proven to be NP-complete [14]. Moreover, even its approximations within a constant factor are NP-hard [7]. In particular, in [10] it is proven that if NP $\neq$ ZPP then no polynomial time algorithm can approximate the maximum clique to within a factor of $N^{1-\epsilon}$ 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 MC, many researchers have concentrated their effort to the design of efficient heuristics yielding sub-optimal solutions of satisfactory quality (e.g., [2, 3, 12]).

Genetic algorithms have been successfully applied to many hard combinatorial optimization problems (cf. [1]). However, the MC problem seems to be a particularly hard problem for genetic algorithms. On this problem pure genetic algorithms have scarse performance when compared with other local search techniques [6, 20]. The situation improves when problem knowledge is incorporated in the algorithm in the form of ad-hoc genetic operators and/or local optimization techniques [5, 8, 9, 18, 23].

It is not yet clear which graph properties render the MC problem a hard problem for GAs. Experimental investigations [11, 24] indicate that neither graph density, size, relative maximum clique size, relative number of cliques, nor Davidor's epistasis are good hardness measures.

In this paper we propose a GLS approach for the maximum clique problem, which combines a simple GA with a local optimizer for generating maximal cliques.

The effectiveness of this approach is tested on standard benchmark instances for MC collected at the DIMACS Center. GLS has worse performance than the reactive local search algorithm by Battiti and Protasi [2], in which information about past events is collected and used in the future part of the search process. Nevertheless GLS outperforms all previous proposals based on genetic algorithms we are aware of, and is competitive with the most effective heuristic algorithms for MC proposed at the DIMACS Implementation Challenge for Maximum Clique, Graph Coloring, and Satisfiability [12].

The rest of the paper is organized as follows. The next section introduces the GLS algorithm for MC. In Section 3 three different GLS instances - iterated, multistart and population based - are experimentally investigated. In Section 4 GLS is compared to other effective genetic and local search algorithms. Finally Section 5 contains some conclusive observations.

A preliminary version of this paper appeared in [15]. Here we use a slightly different local search procedure, we perform a more thorough investigation of the GLS algorithm, and extend the experimental comparison to other GAs.

Notation and Terminology

The following notation and terminology will be used throughout the paper.

A graph is denoted by G, its nodes by $m, n, \ldots$. Nodes of a graph of size N are supposed to be indexed with integers from 1 to N.

A subgraph C_G of G is a *clique* if every two distinct nodes n, n' in C_G are connected with an edge.

A *maximal clique* C_G of G is a clique which is not properly contained in any other clique of G. A *maximum clique* C_G of G is a clique of maximum size. Note that a

maximum clique is maximal but not vice versa.

2 Genetic Local Search for the Maximum Clique

Genetic Local Search has been applied with success to various paradigmatic combinatorial optimization problems (e.g., [16]). The GLS scheme used in our algorithm is illustrated below, where $|P(t)|$ denotes the cardinality of the set $P(t)$.

```
   BEGIN
    t := 0;
    initialize P(t);
(*) apply local search to P(t);
    evaluate P(t);
    WHILE (NOT termination-condition) DO
     BEGIN
      t := t+1;
      WHILE (|P(t)| < |P(t-1)|) DO
       BEGIN
        select parents from P(t-1);
        recombine parents;
        mutate children;
    (*) apply local search to children;
        evaluate children;
        insert in P(t) best two of parents and children;
       END
     END
   END
```

Observe that GLS is a stochastic algorithm because recombination and mutation are probabilistic operators (applied with probability specified by the crossover and mutation rate, respectively).

2.1 LMC: The Local Search Module

Local optimization of the chromosomes is realized using a local search algorithm called LMC.

LMC builds a maximal clique by starting with a random subgraph of the given graph: first, the graph is enlarged by deleting and adding some nodes randomly selected; next it is reduced to a clique, and finally it is extended to a maximal clique using a sequential greedy heuristic.

Given a subgraph S_G of the input graph G, the algorithm transforms S_G to a maximal clique by applying the following three steps in sequential order.

1. `Perturb:` (perturb the subgraph)

   (a) For every node n, $1 \leq n \leq N/2$, if n is in S_G then with small probability (typical value 0.1) remove n from S_G. (In this step nodes are supposed to be sorted in increasing order with respect to their degree.)

   (b) Add the set $s, s+1, \ldots, s+e$ of nodes to S_G, with s and e randomly chosen, where $3 \leq e \leq BK/2$ and s in $[1, N-e]$ (BK is the size of the largest known clique G).

2. `Repair:` (extract a clique)

   Let V be the set of nodes of S_G.

   Repeat the following steps until V becomes empty.

   (a) Choose randomly a node n in V.

   (b) With low probability (typical value 0.01) remove n from S_G; otherwise delete from S_G and from V each node of S_G that is not connected with n, except n itself.

   (c) Remove n from V.

3. `Extend:` (enlarge to a maximal clique)

   Let V be the set of nodes of $G \setminus S_G$. Repeat the following steps until V becomes empty.

   (a) Choose a random node n in V:

   (b) if $\{n\} \cup S_G$ is a clique then add n to S_G.

   (c) remove n from V.

It is easy to prove that the output of LMC is a maximal clique.

In our implementation of LMC a subgraph S_G of G is represented by a bit string x of lenght N, with $x_i = 1$ if and only if i is in S_G. In this way, the search space consists of the set $\{0,1\}^N$. Operators on graphs are translated to operators on binary strings in the expected way.

2.2 The Genetic Algorithm

The main features of the GA module can be summarized as follows.

We employ a generational genetic algorithm with fitness proportional selection rule (roulette-wheel, cf. [17]) and elitism [13] where the two best individuals of a population are copied to the population of the next generation. We use the *keep-two-best* replacement mechanism [26] which selects the two best chromosomes among the set consisting of the two parents and their two offsprings.

The representation is the same as the one used in (the implementation of) LMC, hence the set *Chrm* of chromosomes is $\{0,1\}^N$. For simplicity, in the sequel a chromosome is identified with the subgraph it represents.

The fitness function $f : Chrm \to [0, N]$ is defined by $f(x) = |Nodes(x)|$ (the number of nodes of x) if x is a clique; $f(x) = 0$ otherwise.

Finally, we use classical blind genetic operators: uniform crossover [25], and swap mutation which swaps the values of two randomly selected genes.

As expected the simple GA without local optimization produces solutions of scarce quality [6].

2.3 Analysis of GLS

The role of LMC in the GLS algorithm is that of a repair procedure for 'correcting' infeasible solutions generated by the application of the genetic operators (cf. [17]). However, LMC 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.

Using the technique reported in [22] for studying the limit behaviour of a GA one can prove the following result.

Theorem 2.1 *The GLS algorithm visits the global optimum after a finite number of iterations with probability one. Moreover it converges completely and in mean to the global optimum regardless of the initialization.*

This result is more of academic interest than of practical use: in fact, it remains valid even if a blind random search algorithm is used instead of LMC. It is much more difficult to provide bounds on the quality of the solution found after a finite number of generations. Very few results on this topic have been given (cf. [4, 19, 21]).

Concerning the computational complexity of the algorithm, it can be easily shown that the worst case complexity per iteration is $O(N^2)$.

In terms of memory usage the algorithm needs to store the input graph and the GA population, which consists of $|P(0)|$ chromosomes of length N.

3 Evaluation of GLS

GLS has been implemented in C++ and run on a Sun Ultra 250, UltraSPARC-II 400MHz.

We consider the following three instances of GLS, obtained by using different parameters settings:

- Population based (**GLMC**). It is obtained by setting population size to 10, mutation rate to 0.1, crossover rate to 0.9 and termination-condition to 2000 generations.
- Multistart (**MLMC**). This is obtained by setting population size to 5000 and termination-condition to 0 generations (hence no genetic operators are used).

- Iterated (**ILMC**). It is obtained by setting population size to 1, mutation rate to 0 (hence no genetic operators are used), and termination-condition to 20000 generations.

The initial population is generated randomly, where each gene of a chromosome is set to 1 with probability 0.2, otherwise it is set to 0. Parameters have been set to values experimentally determined after a small number of trials.

In order to test and compare the three algorithms we consider the benchmark instances employed in the International Implementation Challenge on Maximum Clique, Graph Coloring, and Satisfiability organized by the Center for Discrete Mathematics and Theoretical Computer Science (DIMACS). The results obtained by the partecipants have been published in [12]. These instances are available from the DIMACS archive using ftp to `dimacs.rutgers.edu` directory `pub/challenge` or at URL `http://dimacs.rutgers.edu/Challenge/`. The 37 instances here considered include random graphs (`Cx.y` and `DSJCx.y` of size x and density 0.y), Steiner Triple Graphs (`MANNx` with up to 3321 nodes and 5506380 edges), Brockington Graphs (`brockx_2` and `brockx_4` of size x), Sanchis graphs (`genx_p0.9_z` of size x), Hamming graphs (`hamming8-4` and `hamming10-4`), Keller graphs (`keller4, keller5, keller6` with up to 3361 nodes and 4619898 edges), and P-hat graphs (`p_hatx-z` of size x).

MLMC, ILMC and GLMC are run 10 times on each instance using different random seeds. The results of the experiments are summarized in Table 1 which reports the best, average and standard deviation of clique size, and Table 2 which contains average and standard deviation of time (in seconds).

The multistart variant MLMC has worst performance: MLMC outperforms ILMC and GLMC only on one instance (`brock200_4`), while on all other instances it yields in general results of scarce quality. ILMC and GLMC find maximal cliques of comparable quality: ILMC outperforms GLMC on five instances while GLMC outperforms ILMC on four instances (entries in bold style). The variance of the results obtained by ILMC and GLMC is also comparable, indicated by almost equal standard deviations.

Concerning the running time (results of Table 2) ILMC is the fastest of the three algorithms: for example on the dense graphs `MANN_a81` and `keller6` ILMC is about three and two times faster than GLMC, respectively.

The results indicate that there is no "added value" of GLMC with respect to ILMC. Even when larger population sizes are used - we experimented with population sizes up to 100 - GLMC does not find solutions of significantly better quality while its running time increases prohibitively.

4 Comparison with Other Approaches

In order to assess the performance of GLS, we compare experimentally GLMC to two hybrid genetic algorithms, and ILMC to the DIMACS heuristic algorithms for MC and

Graph	MLMC		GLMC		ILMC	
	Avg(Stdv)	Best	Avg(Stdv)	Best	Avg(Stdv)	Best
C125.9	32.6(0.5)	33	33.8(0.4)	34	34(0.0)	34
C250.9	39.1(0.7)	40	42.8(0.7)	44	43.0(0.6)	44
C500.9	46.7(0.6)	48	52.2(1.6)	**56**	52.7(1.4)	55
C1000.9	53.5(1.2)	56	61.6(2.1)	**66**	61.6(1.6)	64
C2000.9	59.7(0.9)	62	68.2(2.4)	**72**	68.7(1.2)	70
DSJC500.5	12.0(0.0)	12	12.2(0.4)	13	12.1(0.3)	13
DSJC1000.5	13.1(0.3)	14	13.3(0.5)	14	13.5(0.5)	14
C2000.5	14.1(0.3)	15	14.2(0.4)	15	14.2(0.4)	15
C4000.5	15.2(0.4)	16	15.4(0.5)	16	15.6(0.5)	16
MANN_a27	124.8(0.4)	125	125.6(0.5)	126	126.0(0.0)	126
MANN_a45	339.7(0.5)	340	342.4(0.5)	343	343.1(0.8)	**345**
MANN_a81	1091.2(0.7)	1092	1096.3(0.6)	1097	1097.0(0.4)	**1098**
brock200_2	12(0.0)	12	10.5(0.7)	12	10.5(0.8)	12
brock200_4	15.7(0.9)	**17**	15.4(0.5)	16	15.5(0.5)	16
brock400_2	21.7(0.6)	23	22.5(0.7)	24	23.2(0.7)	**25**
brock400_4	21.8(0.4)	22	23.6(0.8)	**25**	23.1(0.5)	24
brock800_2	18.0(0.6)	19	19.3(0.6)	20	19.1(0.8)	**21**
brock800_4	18.0(0.0)	18	18.9(0.5)	20	19.0(0.4)	20
gen200_P0.9_44	36.3(0.5)	37	39.7(1.6)	44	39.5(1.6)	44
gen200_P0.9_55	43.4(2.0)	46	50.8(6.4)	55	48.8(7.6)	55
gen400_P0.9_55	43.7(0.6)	45	49.7(1.2)	55	49.1(1.0)	51
gen400_P0.9_65	44.6(0.9)	47	53.7(7.4)	65	51.2(4.7)	65
gen400_P0.9_75	47.7(1.7)	52	60.2(12.1)	75	62.7(12.3)	75
hamming8-4	15.7(0.9)	16	16.0(0.0)	16	16.0(0.0)	16
hamming10-4	32.0(0.4)	33	37.7(1.9)	40	38.8(1.2)	40
keller4	11.0(0.0)	11	11.0(0.0)	11	11.0(0.0)	11
keller5	23.9(0.9)	25	26.0(0.8)	27	26.3(0.6)	27
keller6	45.3(1.1)	48	51.8(1.5)	55	52.7(1.8)	**56**
p_hat300-1	8.0(0.0)	8	8.0 (0.0)	8.0	8.0 (0.0)	8.0
p_hat300-2	22.9(0.9)	25	25(0.0)	25	25(0.0)	25
p_hat300-3	31.0(0.4)	32	34.6(0.9)	36	35.1(0.8)	36
p_hat700-1	9.1(0.3)	10	9.8(0.9)	11	9.9(0.7)	11
p_hat700-2	35.5(0.9)	37	43.5(0.8)	44	43.6(0.7)	44
p_hat700-3	49.5(1.4)	52	60.4(1.0)	62	61.8(0.6)	62
p_hat1500-1	10.2(0.4)	11	10.8(0.4)	11	10.4(0.5)	11
p_hat1500-2	46.9(1.0)	48	63.8(1.0)	65	63.9(2.0)	65
p_hat1500-3	64.3(1.3)	67	92.4(1.3)	94	93.0(0.8)	94

Table 1: Experimental Results for DIMACS 'snapshot': clique size

Graph	MLMC	ILMC	GLMC
	Avg(Stdv)	Avg(Stdv)	Avg(Stdv)
C125.9	2.4(0.1)	0.5(0.6)	0.1(0.1)
C250.9	4.4(0.1)	2.4(2.0)	3.7(3.5)
C500.9	8.8(0.1)	2.7(2.6)	5.7(5.5)
C1000.9	18.2(0.1)	8.6(6.5)	12.4(13.4)
C2000.9	46.3(0.3)	24.8(23.8)	33.8(34.2)
DSJC500.5	6.6(0.1)	0.4(0.5)	2.1(4.9)
DSJC1000.5	13.4(0.1)	2.3(3.8)	2.2(1.9)
C2000.5	29.3(0.1)	2.3(2.0)	7.2(12)
C4000.5	63.2(0.3)	15.7(8.0)	13.5(14.0)
MANN_a27	14.4(0.1)	15.6(10.1)	3.7(4.0)
MANN_a45	92.7(0.4)	54.4(44.3)	135.2(106.4)
MANN_a81	1203.3(39.5)	693.9(922.5)	2773.8(1158.3)
brock200_2	2.7(0.0)	0.2(0.2)	1.3(1.8)
brock200_4	2.8(0.1)	0.5(0.6)	0.9(1.6)
brock400_2	5.7(0.1)	2.0(2.3)	1.9(3.3)
brock400_4	5.7(0.1)	1.3(1.4)	1.3(1.4)
brock800_2	11.0(0.1)	3.9(4.9)	5.2(7.4)
brock800_4	11.0(0.1)	4.1(3.4)	7.8(7.7)
gen200_P0.9_44	3.5(0.0)	1.7(1.5)	1.3(1.3)
gen200_P0.9_55	3.5(0.0)	1.3(1.5)	1.4(2.9)
gen400_P0.9_55	6.8(0.0)	2.8(3.0)	3.8(6.5)
gen400_P0.9_65	6.8(0.0)	2.7(3.3)	4.3(4.3)
gen400_P0.9_75	6.8(0.1)	4.6(3.2)	3.7(6.3)
hamming8-4	3.5(0.0)	0.0(0.0)	0.0(0.0)
hamming10-4	15.5(0.1)	5.3(4.4)	5.8(5.6)
keller4	2.4(0.0)	0.0(0.0)	0.0(0.0)
keller5	10.9(0.1)	4.0(3.9)	9.1(10.2)
keller6	69.2(0.1)	36.2(23.8)	60.2(55.0)
p_hat300-1	3.7(0.0)	0.9(0.8)	0.4(0.5)
p_hat300-2	4.1(0.0)	0.5(0.5)	0.5(0.6)
p_hat300-3	4.4(0.0)	1.5(1.7)	3.6(4.5)
p_hat700-1	8.7(0.1)	2.6(3.8)	5.8(6.8)
p_hat700-2	9.5(0.0)	1.2(1.2)	1.6(1.9)
p_hat700-3	10.5(0.0)	4.5(4.4)	5.6(6.3)
p_hat1500-1	19.5(0.1)	2.1(3.1)	14.2(14.7)
p_hat1500-2	21.6(0.1)	12.2(9.2)	9.1(7.8)
p_hat1500-3	24.5(0.0)	7.1(11.4)	14.6(18.2)

Table 2: Experimental Results for DIMACS 'snapshot': time

to the best (to the best of our knowledge) heuristic algorithm for MC.

4.1 Comparison with Hybrid Genetic Algorithms

We are aware of two hybrid genetic algorithms for MC which have being tested on DIMACS benchmarks.

The algorithm by Bui and Eppley [5], called GMCA, is a hybrid GA with local optimization for improving the chromosomes. Moreover pre-processing is used for reordering the nodes of the graph in such a way that nodes which are likely to belong to a clique (e.g. with high degree) occur near each other. The fitness function is the weighted sum of density and size of the subgraph represented by the chromosome.

In [23] Sakamoto et al. introduce a GA for the maximum independent set problem, here called SLS. Recall that an independent set of a graph G is a clique of the complement of G (consisting of the nodes of G and all the edges which are not in G). Clearly MC and the maximum independent problem are equivalent since a maximum clique of G is a maximum independent set of the complement of G. In SLS a chromosome is a permutation of the nodes of the graph. A greedy decoding method is used which constructs a independent set using the nodes in the order in which they appear in the chromosome. The chromosome is then replaced with the sequence of nodes of the independent set followed by the remaining nodes of the graph. The fitness function is the size of the independent set minus the minimum of the sizes of the independent sets in the actual population.

Both GMCA and SLS use a population of size 50 and are run for 50 generations.

In Table 3 the results from [23] are compared to GLMC. Obserev that the DIMACS benchmarks of the table are slighty different from the DIMACS Challenge 'snapshot' we used in the previous section. In order to compare running times - GMCA and SLS are run on a Sun SPARC LX - we converted the running time of GLMC to Sun SPARC LX time using the Dhrystone score. It turns out that our computer is about 17 times faster than a Sun SPARC LX.

On 12 instances (in bold style) GLMC finds a best solution better than the one of GMCA and SLS; on the other instances SLS and GLMC yield equal best results and outperform GMCA on 10 instances. Concerning the running time, GLMC is faster than GMCA and SLS, except on the `MANN*` instances: in particular on `MANN_a45` and `MANN_a81` GLMC is about ten times slower than SLS, but is able to find the best known result.

The results indicate that instances from the `Cfat`, `Johnson` and `Hamming` classes can be regarded as GA easy.

4.2 Comparison with Local Search Algorithms

Table 4 reportes the size of the best clique found by ILMC, by the reactive local search algorithm by Battiti and Protasi [2](column labelled RLS), and by the best of the fifteen heuristic algorithms for MC presented at the DIMACS Challenge (column labelled

Graph	GMCA		SLS		GLMC	
	Avg Time	Best	Avg Time	Best	Avg Time	Best
c-fat200-1	8.2	12	12.3	12	0.0	12
c-fat500-1	33.2	14	60.7	14	0.0	14
johnson16-2-4	6.0	8	4.5	8	0.0	8
johnson32-2-4	187.4	16	63.2	16	1.7	16
keller4	13.3	11	9.1	11	0.0	11
keller5	438.1	18	256.7	27	63.7	27
keller6	-	-	4798.6	50	1023.4	**55**
hamming10-2	886.6	512	351.3	512	37.2	512
hamming8-2	53.0	128	21.2	128	1.7	128
san200_0.7_1	51.7	30	16.6	30	10.2	30
san400_0.5_1	411.2	7	43.1	13	42.5	13
san400_0.9_1	128.6	50	70.5	100	47.6	100
sanr200_0.7	21.5	17	14.6	18	10.5	18
sanr400_0.5	69.6	12	45.8	13	42.5	13
san1000	704.3	8	242.8	10	15.3	10
brock200_1	27.9	20	14.9	20	15.3	20
brock200_2	-	-	12.1	10	22.1	**12**
brock200_4	-	-	14.0	16	15.3	16
brock400_1	118.8	20	50.9	23	44.2	**24**
brock400_2	-	-	52.5	24	32.3	24
brock400_4	-	-	56.8	23	22.1	**25**
brock800_1	460.8	18	172.1	18	27.2	**19**
brock800_2	-	-	188.8	19	66.3	**20**
brock800_4	-	-	171.8	20	69.7	20
p_hat300_1	20.0	8	24.1	8	6.8	8
p_hat300_2	-	-	30.4	25	8.5	25
p_hat300_3	-	-	36.1	35	61.2	**36**
p_hat500_1	49.1	9	62.0	9	3.5	9
p_hat700_1	310.1	8	117.7	11	105.4	11
p_hat700_2	-	-	165.0	44	27.2	44
p_hat700_3	-	-	192.7	61	95.2	**62**
p_hat1000_1	671.0	8	247.6	10	54.4	10
p_hat1500_1	1580.3	10	573.2	11	249.9	11
p_hat1500_2	-	-	952	64	154.7	**65**
p_hat1500_3	-	-	1079.5	92	248.2	**94**
MANN_a27	121.8	125	52.4	126	62.9	126
MANN_a45	916.9	337	341	339.5	2298.4	**345**
MANN_a81	-	-	1094	6249.0	47154.6	**1098**

Table 3: Results for DIMACS benchmark graphs: GMCA, SLS, GLMC

Graph	DIMACS	RLS	ILMC
C125.9	34	34	34*
C250.9	44	44	44*
C500.9	57	57	55
C1000.9	67	68	64
C2000.9	75	78	70
DSJC500.5	15*	14	13
DSJC1000.5	15*	15	14
C2000.5	16	16	15
C4000.5	18	18	16
MANN_a27	126	126	126*
MANN_a45	345	345	345*
MANN_a81	1098	1098	1098
brock200_2	12	12	12*
brock200_4	17*	17	16
brock400_2	25	29*	25
brock400_4	24	33*	24
brock800_2	21	21	21
brock800_4	21	21	20
gen200_P0.9_44	44	44	44*
gen200_P0.9_55	55	55	55*
gen400_P0.9_55	55	55	51
gen400_P0.9_65	65	65	65
gen400_P0.9_75	75	75	75
hamming8-4	16	16	16*
hamming10-4	40	40	40*
keller4	11	11	11*
keller5	27	27	27
keller6	59	59	56
p_hat300-1	8	8	8*
p_hat300-2	25	25	25*
p_hat300-3	36	36	36*
p_hat700-1	11	11	11*
p_hat700-2	44	44	44*
p_hat700-3	62	62	62
p_hat1500-1	12	12*	11
p_hat1500-2	65	65	65
p_hat1500-3	94	94	94

Table 4: Results on DIMACS 'snapshot' (* indicates global optimality).

DIMACS). These latter algorithms are based on various approaches, like tabu search, simulated annealing, and neural networks.

On 23 of the 37 instances ILMC is able to find the best value found by DIMACS and RLS. In the other cases ILMC is able to find maximal cliques of good quality.

The best performance is obtained by RLS which outperforms DIMACS on 4 instances and ILMC on 14 instances, while it is outperformed by DIMACS on just 2 instances (`DISJ500.5` and `DISJ1000.5`)

Concerning running time, ILMC is in general ten times slower than RLS except on `MANN_a45` and `MANN_a81` where ILMC is about four times faster than RLS. Observe that the code of ILMC is not optimized, hence we are confident that more specialized data structures can possibly improve the efficiency of ILMC.

5 Conclusion

We have proposed a simple heuristic algorithm based on genetic local search, and considered three variants of the algorithm obtained by choosing different parameters settings: multistart (MLMC), iterated (ILMC) and population based (GLMC).

Experiments on the DIMACS benchmark graphs indicate that ILMC is the most effective of the three variants and that the genetic operators do not seem to contribute in guiding the search towards promising regions, yet their disruptive effect is in some cases useful for escaping from local optima (e.g. on random graphs `C500.9`, `C1000.9` and `C2000.9`).

The inferior performance of ILMC with respect to RLS can be justified by the different approaches they use: RLS uses memory to collect information about past events as done in tabu search; moreover, a reactive strategy is incorporated which regulates the search diversification and an explicit memory-influenced restart is activated periodically. In constrast, ILMC is a Markov (i.e., memory-less) process, where the next state depends only on the current one. An interesting issue for future research is to investigate whether the integration of memory into ILMC- as in reactive local search - would be beneficial for its performance.

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. *Algorithmica*, 2000. In press.

[3] I.M. Bomze, M. Budinich, P.M. Pardalos, and M. Pelillo. The maximum clique problem. *Handbook of Combinatorial Optimization*, 4, 1999.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] T. Haynes. Clique detection as a royal road function. In *Genetic Programming*, 1998.

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

[13] 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.

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

[15] E. Marchiori. A simple heuristic based genetic algorithm for the maximum clique problem. In J. Carroll et al., editor, *ACM Symposium on Applied Computing*, pages 366–373. ACM Press, 1998.

[16] P. Merz and B. Freisleben. Genetic local search for the TSP: New results. In *IEEE International Conference on Evolutionary Computation*, pages 159–164. IEEE Press, 1997.

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

[18] 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.

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

[20] 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.

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

[22] G. Rudolph. Finite Markov chain results in evolutionary computation: A tour d'horizon. *Fundamenta Informaticae*, 35(1-4):67–89, 1998.

[23] A. Sakamoto, X. Liu, and T. Shimamoto. A genetic approach for maximum independent set problems. *IEICE Trans. Fundamentals*, E80-A(3):551–556, 1997.

[24] 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.

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

[26] D. Thierens and D. Goldberg. Elitist recombination: an integrated selection recombination GA. In *Proceedings of the 1st IEEE Conference on Evolutionary Computation*, pages 508–518. IEEE Press, 1994.