Searching for Maximum Cliques with Ant Colony Optimization

Serge Fenet and Christine Solnon

LIRIS, Nautibus, University Lyon I
43 Bd du 11 novembre, 69622 Villeurbanne cedex, France
{sfenet,csolnon}@bat710.univ-lyon1.fr

Abstract. In this paper, we investigate the capabilities of Ant Colony Optimization (ACO) for solving the maximum clique problem. We describe Ant-Clique, an algorithm that successively generates maximal cliques through the repeated addition of vertices into partial cliques. ACO is used to choose, at each step, the vertex to add. We illustrate this algorithm on two representative benchmark instances, and we study the impact of pheromone on the solving process, showing that ACO, without any other local heuristic nor local search, strongly improves it. We experimentally compare Ant-Clique with GLS, a Genetic Local Search approach, and we show that Ant-Clique finds larger cliques, on average, on a majority of DIMACS benchmark instances, eventhough it does not reach the best known results on some instances.

1 Introduction

The maximum clique problem is a classical combinatorial optimization problem that has important applications in different domains, such as coding theory, fault diagnosis or computer vision. Given a non-oriented graph $G = (V, E)$, such that V is a set of vertices, and $E \subseteq V \times V$ is a set of edges, a *clique* is a set of vertices $\mathcal{C} \subseteq V$ such that every couple of distinct vertices of $\mathcal{C}$ is connected with an edge in G, i.e., the subgraph induced by $\mathcal{C}$ is complete. A clique is *partial* if it is strictly included in another clique; otherwise it is *maximal*. The goal of the *maximum clique problem* is to find a clique of maximum cardinality.

This problem is one of the first problems shown to be NP-complete, and moreover, it does not admit a polynomial-time approximation algorithm (unless P=NP) [2]. Hence, complete approaches —usually based on a branch-and-bound tree search— become intractable when the number of vertices increases, and much effort has recently been directed on heuristic incomplete approaches. These approaches leave out exhaustivity and use heuristics to guide the search towards promising areas of the search space. In particular, [6] describes three heuristic algorithms obtained as instances of an evolutionary algorithm scheme, called GLS, that combines a genetic approach with local search.

In this paper, we investigate the capabilities of another evolutionary metaheuristic —Ant Colony Optimization (ACO)— for solving the maximum clique problem. In Section 2, we describe Ant-Clique, an ACO algorithm for the maximum clique problem. Basically, this algorithm uses a sequential greedy heuristic, and generates maximal cliques through the repeated addition of vertices into partial cliques. ACO is introduced as a heuristic for choosing, at each step, the vertex to enter the clique: this vertex is chosen with respect to a probability that depends on pheromone trails laying between it and the clique under construction, while pheromone trails are deposited by ants proportionally to the quality of the previously computed cliques. In Section 3, we illustrate this algorithm on two representative benchmark instances and we study the impact of pheromone on the solving process, showing that ACO, without any other local heuristic nor local search, strongly improves it. In Section 4, we experimentally compare Ant-Clique with GLS [6], a genetic local search approach, and we show that Ant-Clique finds larger cliques, on average, on a wide majority of benchmark instances of the DIMACS library, eventhough it does not reach the best known results on some benchmark instances.

2 Description of Ant-Clique

The Ant Colony Optimization (ACO) metaheuristic is an evolutionary approach that has been used to solve different hard combinatorial optimization problems ([3, 4]). The main idea of ACO is to model the problem as the search for a minimum cost path in a graph. Artificial ants walk through this graph, looking for good paths. Each ant has a rather simple behaviour so that it will typically only find rather poor quality paths on its own. Better paths are found as the emergent result of the global cooperation among ants in the colony. This cooperation is performed in an indirect way through pheromone laying.

The proposed ACO algorithm for searching for maximum cliques, called Ant-Clique is sketched below:

procedure Ant-Clique
    Initialize pheromone trails
    repeat
        for k **in** $1..nbAnts$ **do**: construct a clique $\mathcal{C}_k$
        Update pheromone trails w.r.t. $\{\mathcal{C}_1, \ldots, \mathcal{C}_{nbAnts}\}$
    until max cycles reached **or** optimal solution found

Pheromone trail initialization: Ants communicate by laying pheromone on the graph edges. The amount of pheromone on edge (v_i, v_j) is noted $\tau(v_i, v_j)$ and represents the learnt desirability for v_i and v_j to belong to a same clique.

As proposed in [9], we explicitly impose lower and upper bounds τ_{min} and τ_{max} on pheromone trails (with $0 < \tau_{min} < \tau_{max}$). The goal is to favor a larger exploration of the search space by preventing the relative differences between pheromone trails from becoming too extreme during processing.

We have considered two different ways of initializing pheromone trails:

1. Constant initialization to τ_{max}: such an initialization achieves a higher exploration of the search space during the first cycles, as all edges nearly have the same amount of pheromone during the first cycles [9].
2. Initialization with respect to a preprocessing step: the idea is to first compute a representative set of maximal cliques without using pheromone —thus constituting a kind of sampling of the search space— and then to select from this sample set the best cliques and use them to initialize pheromone trails. This preprocessing step introduces two new parameters: ϵ, the limit rate on the quality improvement of the sample set of cliques, and n_{Best}, the number of best cliques that are selected from the sample set in order to initialize pheromone trails. More details can be found in [8].

Construction of cliques by ants: ants first randomly select an initial vertex, and then iteratively choose vertices to be added to the clique —within a set of candidates that contains all vertices that are connected to every clique vertex:

choose randomly a first vertex $v_f \in V$
$\mathcal{C} \leftarrow \{v_f\}$
$Candidates \leftarrow \{v_i/(v_f, v_i) \in E\}$
while $Candidates \neq \emptyset$ **do**
  Choose a vertex $v_i \in Candidates$ with probability $p(v_i) = \frac{[\tau_{\mathcal{C}}(v_i)]^\alpha}{\sum_{v_j \in Candidates}[\tau_{\mathcal{C}}(v_j)]^\alpha}$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{v_i\}$
  $Candidates \leftarrow Candidates \cap \{v_j/(v_i, v_j) \in E\}$
end while

The choice of a vertex v_i within the set of candidates is made with respect to a probability that depends on a pheromone factor $\tau_{\mathcal{C}}(v_i)$. A main difference with many ACO algorithms is that this factor does not only depend on the pheromone trail between the last added vertex in $\mathcal{C}$ and the candidate vertex v_i, but on all pheromone trails between $\mathcal{C}$ and v_i[1], i.e., $\tau_{\mathcal{C}}(v_i) = \sum_{v_j \in \mathcal{C}} \tau(v_i, v_j)$.

One should remark that the probability of choosing a vertex v_i only depends on a pheromone factor, and not on some heuristic factor that locally evaluates the quality of the candidate vertex, as usually in ACO algorithms. Actually, we have experimented a heuristic proposed in [1], the idea of which being to favor vertices with larger degrees in the subgraph induced by the partial clique under construction. The underlying motivation is that a larger degree implies a larger number of candidates after the vertex is added to the current clique. When using no pheromone (or at the beginning of the search, when all pheromone trails have the same value), this heuristic actually allows ants to construct larger cliques than a random choice. However, when combining it with pheromone learning, we have noticed that after a hundred or so cycles, we obtain larger cliques without using the heuristic than when using it.

[1] This pheromone factor $\tau_{\mathcal{C}}(v_i)$ is computed in an incremental way: at the beginning, when the clique $\mathcal{C}$ only contains one vertex v_f, $\tau_{\mathcal{C}}(v_i)$ is initialized to $\tau(v_f, v_i)$; then, each time a new vertex v_k is added to the clique, $\tau_{\mathcal{C}}(v_i)$ is incremented with $\tau(v_k, v_i)$.

Updating pheromone trails: After each ant has constructed a clique, pheromone trails are updated according to ACO: first, all pheromone trails are decreased in order to simulate evaporation, i.e., for each edge $(v_i, v_j) \in E$, the quantity of pheromone laying on it, $\tau(v_i, v_j)$, is multiplied by an evaporation parameter ρ such that $0 \leq \rho \leq 1$; then, the best ant of the cycle deposits pheromone, i.e., let $\mathcal{C}_k$ be the largest clique built during the cycle, and $\mathcal{C}_{best}$ be the largest clique built since the beginning of the run, for each couple of vertices $(v_i, v_j) \in \mathcal{C}_k$, we increment the quantity of pheromone laying on it, $\tau(v_i, v_j)$, by $1/(1+|\mathcal{C}_{\text{best}}|-|\mathcal{C}_k|)$.

3 Experimental study of Ant-Clique

As usually in ACO algorithms, the behaviour of Ant-Clique depends on its parameters, and more particularly on α, the pheromone factor weight, and ρ, the evaporation rate. Indeed, diversification can be emphasized both by decreasing α, so that ants become less sensitive to pheromone trails, and by increasing ρ, so that pheromone evaporates more slowly. When increasing the exploratory ability of ants in this way, one usually finds better solutions, but as a counterpart it takes longer time to find them. This is illustrated in figure 1 on two DIMACS benchmark instances.

On this figure, one can first note that, for both instances, when $\alpha = 0$ and ρ=1, cliques are much smaller: in this case, pheromone is totally ignored and the resulting search process performs as a random one so that after 500 or so cycles, the size of the largest clique nearly stops increasing. This shows that pheromone actually improves the solution process with respect to a pure random algorithm.

When considering instance *gen_400_P0.9_75*, we can observe that, when α increases or ρ decreases, ants converge quicker towards the optimal solution: it is found around cycle 650 when α=1 and ρ=0.995, around cycle 400 when α=2 and ρ=0.995, around cycle 260 when α=2 and ρ=0.99, and around cycle 200 when α=3 and ρ=0.985. Actually, this instance is relatively easy to solve, regardless of the parameter tuning. In this case, one has better choose parameter values that favor a quick convergence such as α=3 and ρ=0.985.

However, when considering instance *C500.9*, which is more difficult, one can see that the setting of α and ρ let us balance between two main tendencies. On one hand, when favoring exploration, the quality of the final solution is better, but the time needed to converge on this value is also higher. On the other hand, when favoring convergence, ants find better solutions during the first cycles, but after 500 or so cycles, too many of them have converged on over-marked pheromone trails and the exploration behavior is inhibited. To summarize, one has to choose parameters depending on the availability of time for solving: for 300 cycles, one has to choose α=3 and ρ=0.985; for 800 cycles one has to choose α=2 and ρ=0.990; and for more available time one has to chose α=2 and ρ=0.995. Note that with α=1, pheromone's influence is too weak and convergence is to slow: the system won't reach optimal solution in acceptable time.

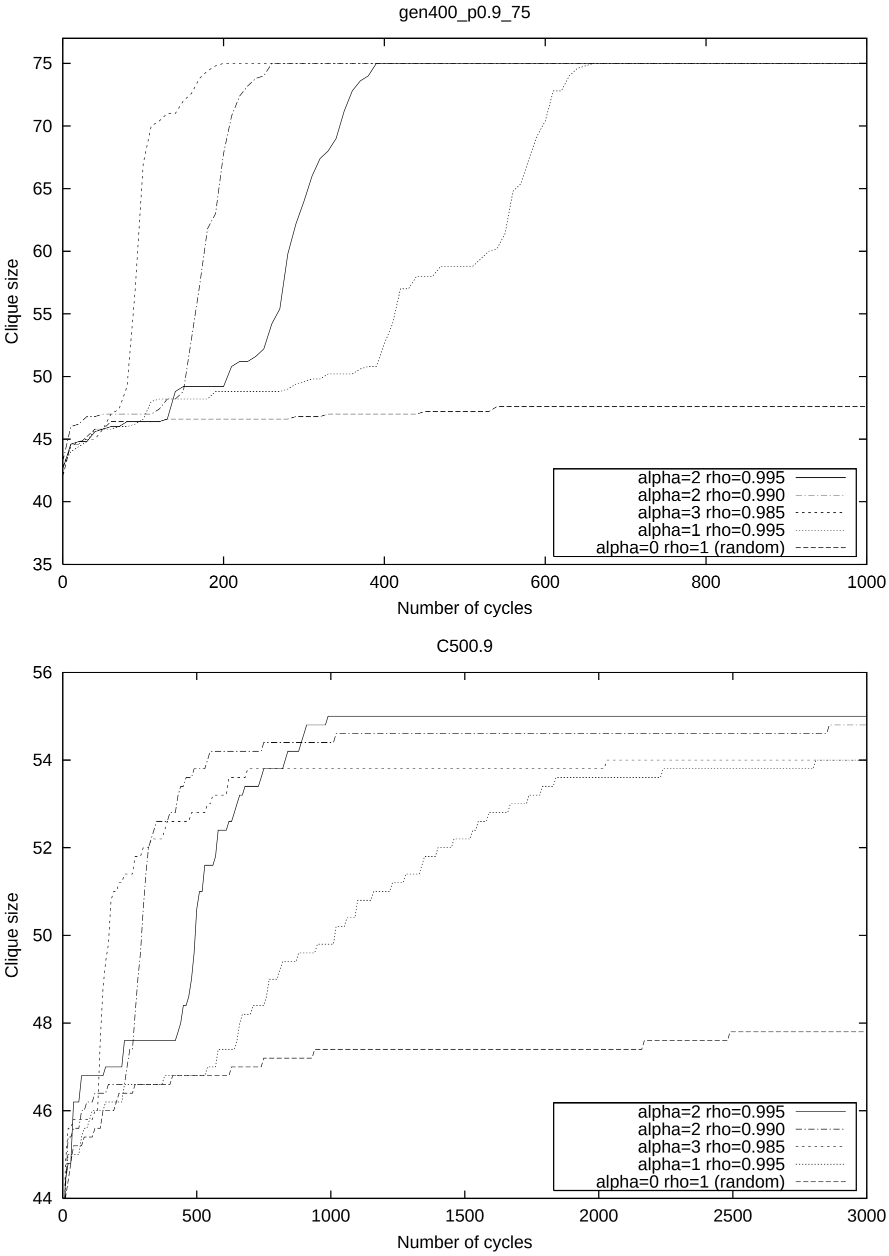

Fig. 1. Influence of pheromone on the solving process: each curve plots the evolution of the size of the largest clique (average on 5 runs), when the number of cycles increases, for a given setting of α and ρ. The other parameters have been set to nbAnts=15, $\tau_{min} = 0.01$, $\tau_{max} = 4$, and no preprocessing. The upper curves correspond to the DIMACS instance *gen_400_P0.9_75* —that is a rather easy one— and the lower curves to *C500.9* — that is more difficult.

4 Experimental comparison of Ant-Clique with GLS

Genetic Local Search: We compare Ant-Clique with genetic local search (GLS) [6], an evolutionary approach that combines a genetic approach with local search. GLS generates successive populations of maximal cliques from an initial one by repeatedly selecting two parent cliques from the current population, recombining them to generate two children cliques, applying local search on children to obtain maximal cliques, and adding to the new population the best two cliques of parents and children. GLS can be instanciated to different algorithms by modifying its parameters. In particular, [6] compares results obtained by the three following instances of GLS: GENE performs genetic local search (the population size is set to 10, the number of generations to 2000, and mutation and crossover rates to 0.1 and 0.9); ITER performs iterated local search, starting from one random point (the population size is set to 1, the number of generations to 20000, and mutation and crossover rates are null); and MULT performs multi-start local search, starting from a new random point at each time (the population size is set to 20000, the number of generations to 0, and mutation and crossover rates are null).

Experimental Setup: Ant-Clique has been implemented in C and run on a 350 MHz G4 processor. In all experiments, we have set α to 2, ρ to 0.995, τ_{min} to 0.01, τ_{max} to 4, and the maximum number of cycles to 3000. We first report results obtained with 7 ants and no preprocessing. In this case, Ant-Clique constructs 21 000 cliques so that this allow a fair comparison with GLS that constructs 20 000 cliques. We also report results obtained when increasing the number of ants to 30, and when using a preprocessing step to initialize pheromone trails (as described in section 2). When a preprocessing step is used, we have set ϵ to 0.001 and n_{Best} to 300.

Test Suite: We consider the 37 benchmark instances proposed by the DIMACS organisers and available at `http://dimacs.rutgers.edu/Challenge`. C$n.p$ and DSJC$n.p$ graphs have n vertices; Mann_a 27, 45 and 81 graphs respectively have 378, 1035 and 3321 vertices; brock$n$_m graphs have n vertices; genxxx_p0.9yy graphs have xxx vertices and maximum cliques of size yy; *hamming8-4* and *hamming10-4* respectively have 256 and 1024 vertices; *keller* 4, 5 and 6 graphs respectively have 171, 776 and 3361 vertices; and p_hat$n-m$ graphs have n vertices.

Comparison of Clique Sizes: Table 1 compares the sizes of the cliques found by Ant-Clique and GLS (note that for GLS, we report the results given in [6] for the GLS instance that obtained the best average results). On this table, one can first note that GLS clearly outperforms Ant-Clique on the three MANN instances: on these instances, ACO deteriorates the solution process, and the average size of the constructed cliques at each cycle decreases when the number of cycles increases. Further work will concern the study of the fitness landscape of these instances, in order to understand these bad results.

When comparing average results of Ant-Clique with 7 ants and no preprocessing with GLS, we can note that Ant-Clique outperforms GLS on 23 instances,

Graph	Ant-Clique						GLS		
	with preproc		without preprocessing						
	7 ants		7 ants		30 ants				
	best	avg(std)	best	avg(std)	best	avg(std)	best	avg(std)	
C125.9	34	34.0(0.0)	34	34.0(0.0)	34	34.0(0.0)	34	34.0(0.0)	I
C250.9	44	44.0(0.0)	44	44.0(0.2)	44	44.0(0.0)	44	43.0(0.6)	I
C500.9	56	54.3(1.1)	56	55.0(0.8)	56	55.5(0.7)	55	52.7(1.4)	I
C1000.9	65	63.5(1.2)	66	63.6(1.4)	67	65.0(1.0)	66	61.6(2.1)	G
C2000.9	70	68.5(1.2)	72	68.5(1.9)	73	70.0(1.7)	70	68.7(1.2)	I
DSJC500.5	13	12.9(0.3)	13	12.7(0.5)	13	12.9(0.3)	13	12.2(0.4)	G
DSJC1000.5	15	14.0(0.5)	15	13.9(0.6)	15	14.0(0.2)	14	13.5(0.5)	I
C2000.5	15	14.9(0.3)	16	14.6(0.6)	16	15.0(0.3)	15	14.2(0.4)	I
C4000.5	17	15.8(0.6)	16	15.5(0.5)	17	15.9(0.4)	16	15.6(0.5)	I
MANN_a27	126	125.6(0.3)	126	124.8(0.4)	126	125.5(0.5)	126	126.0(0.0)	I
MANN_a45	340	338.9(0.7)	338	336.4(0.6)	341	339.2(0.7)	345	343.1(0.8)	I
MANN_a81	1092	1090.6(0.7)	1087	1086.3(0.5)	1092	1089.4(0.7)	1098	1097.0(0.4)	I
brock200_2	12	11.9(0.3)	12	11.9(0.3)	12	12.0(0.0)	12	12.0(0.0)	M
brock200_4	17	16.4(0.5)	16	16.0(0.0)	17	16.6(0.5)	17	15.7(0.9)	M
brock400_2	25	24.5(0.5)	25	24.4(0.5)	29	24.7(1.0)	25	23.2(0.7)	I
brock400_4	33	25.3(3.0)	25	24.1(0.3)	33	25.6(3.3)	25	23.6(0.8)	G
brock800_2	21	19.7(0.6)	21	19.6(0.6)	21	19.8(0.5)	20	19.3(0.6)	G
brock800_4	20	19.3(0.5)	21	19.6(0.6)	20	19.7(0.5)	20	19.0(0.4)	I
gen200_p0.9_44	44	44.0(0.0)	44	42.6(1.9)	44	43.8(0.8)	44	39.7(1.6)	G
gen200_p0.9_55	55	55.0(0.0)	55	55.0(0.0)	55	55.0(0.0)	55	50.8(6.4)	G
gen400_p0.9_55	52	51.6(0.5)	52	51.5(0.5)	52	51.7(0.5)	55	49.7(1.2)	G
gen400_p0.9_65	65	65.0(0.0)	65	65.0(0.0)	65	65.0(0.0)	65	53.7(7.4)	G
gen400_p0.9_75	75	75.0(0.0)	75	75.0(0.0)	75	75.0(0.0)	75	62.7(12.3)	I
hamming8-4	16	16.0(0.0)	16	16.0(0.0)	16	16.0(0.0)	16	16.0(0.0)	G
hamming10-4	40	38.3(1.0)	40	38.4(1.5)	40	38.3(1.7)	40	38.2(1.2)	I
keller4	11	11.0(0.0)	11	11.0(0.0)	11	11.0(0.0)	11	11.0(0.0)	G
keller5	27	26.9(0.3)	27	26.6(0.5)	27	26.9(0.3)	27	26.3(0.6)	I
keller6	54	52.0(1.2)	53	51.3(1.2)	55	53.1(1.2)	56	52.7(1.8)	I
p_hat300-1	8	8.0(0.0)	8	8.0(0.0)	8	8.0(0.0)	8	8.0(0.0)	G
p_hat300-2	25	25.0(0.0)	25	25.0(0.0)	25	25.0(0.0)	25	25.0(0.0)	I
p_hat300-3	36	36.0(0.0)	36	36.0(0.2)	36	36.0(0.0)	36	35.1(0.8)	I
p_hat700-1	11	10.8(0.4)	11	10.8(0.4)	11	10.9(0.3)	11	9.9(0.7)	I
p_hat700-2	44	44.0(0.0)	44	44.0(0.0)	44	44.0(0.0)	44	43.6(0.7)	I
p_hat700-3	62	61.6(0.5)	62	61.5(0.5)	62	62.0(0.0)	62	61.8(0.6)	I
p_hat1500-1	11	10.9(0.3)	12	11.2(0.4)	12	11.0(0.2)	11	10.8(0.4)	G
p_hat1500-2	65	64.9(0.3)	65	64.8(0.4)	65	64.9(0.3)	65	63.9(2.0)	I
p_hat1500-3	94	92.9(0.4)	94	92.8(0.5)	94	93.1(0.3)	94	93.0(0.8)	I

Table 1. Results for DIMACS benchmarks. For each instance, the table first displays the results obtained by Ant-Clique (with preprocessing and 7 ants, without preprocessing and 7 ants, and without preprocessing and 30 ants) and reports the best, average and standard deviation of the maximum size cliques constructed over 25 runs. Finally, the table reports results given for GLS in [6]: we display the best, average and standard deviation for the GLS algorithm that obtained the best average results (I for Iter, G for Gene and M for Mult).

whereas GLS outperforms Ant-Clique on 8 instances. Then, when increasing the number of ants from 7 to 30, the average results found by Ant-Clique are improved for 24 instances, whereas they are deteriorated for 2 instances. When further increasing the number of ants, Ant-Clique can still improve its performances on the hardest instances. For example, for *keller6* and with 100 ants, Ant-Clique found a clique of size 59 (which is the best known result), and in average found cliques of size 54.1 instead of 53.1 with 30 ants and 51.3 with 7 ants. Finally, one can also note that, when fixing the number of ants to 7, the preprocessing step improves average solutions' quality for 18 instances, whereas it deteriorates it for 4 instances.

CPU times: Table 2 reports the number of cycles and the time spent to find best solutions. One can first note that, within a same set of instances, time grows quadratically with respect to the number of vertices of the graph. For instance, with 7 ants and no preprocessing, it respectively takes 0.4, 34.9 and 689.3 seconds to solve *C125.9*, *C500.9* and *C2000.9* that respectively have 125, 500 and 2000 vertices. We cannot precisely compare with GLS, as runs were made on different computers. However, within a same set of instances, times for GLS increase more slowly and, for example, ITER respectively spent 0.5, 2.7 and 24.8 seconds to solve *C125.9*, *C500.9* and *C2000.9*.

Note also that when increasing the number of ants from 7 to 30, the number of cycles is nearly always decreased (for all instances but 3) as the quality of the best computed clique at each cycle is improved. However, as the time needed to compute one cycle also increases, Ant-Clique with 30 ants is quicker than Ant-Clique with 7 ants for 15 instances only (those for which the number of cycles is strongly decreased), whereas it is slowler for 21 instances.

Finally, note that when initializing pheromone trails with a preprocessing step, instead of a constant initialization to τ_{max}, the number of cycles is nearly always strongly decreased (for all instances but 2). Indeed, this preprocessing step shorten the exploration phase, so that ants start converging much sooner. One can notice that on very easy instances, the best solution is found during preprocessing, or just after during the first cycles of the pheromone-oriented solving step. Also, on the 3 *MANN* instances, the best solution is always found during preprocessing as, on these instances, ACO deteriorates solutions quality. However, if the preprocessing step reduces the number of cycles, it is time consuming. Hence, Ant-Clique with preprocessing is quicker than without preprocessing for 24 instances, whereas it is slowler for 13. This result must be considered with the fact that preprocessing also allowed to improve solutions' quality for 18 instances, whereas it deteriorates them for 4 instances. Hence, on the maximum clique problem, preprocessing often boosts the resolution.

5 Conclusion

We have described Ant-Clique, an ACO algorithm for searching for maximum cliques. We have shown through experiments that this metaheuristic is actually well suited to this problem: without using any local heuristic to guide them

	Ant-Clique						
Graph	with preprocessing			without preprocessing			
	7 ants			7 ants		30 ants	
Graph	nbCy(std)	t-P+t-S=	time	nbCy(std)	time	nbCy(std)	time
C125.9	93(61)	1.0+0.2=	1.2	219(63)	0.4	140(49)	0.7
C250.9	356(144)	2.2+2.5=	4.7	888(523)	6.1	467(98)	6.7
C500.9	974(614)	4.7+24.5=	29.3	1488(654)	34.9	1211(603)	49.9
C1000.9	1730(672)	12.9+154.6=	167.6	1972(571)	161.2	1712(602)	202.2
C2000.9	2065(648)	23.8+672.9=	696.7	2281(498)	689.3	2041(487)	749.5
DSJC500.5	325(614)	0.9+4.5=	5.4	1049(952)	12.9	605(375)	8.8
DSJC1000.5	495(612)	1.8+24.4=	26.2	1084(910)	50.6	990(645)	51.3
C2000.5	576(741)	4.2+108.3=	112.5	1130(919)	203.1	869(615)	159.5
C4000.5	505(745)	7.6+377.2=	384.8	783(826)	551.0	1097(728)	772.3
MANN_a27	0(0)	10.1+0.0=	10.1	187(630)	7.8	171(455)	21.3
MANN_a45	0(0)	67.7+0.0=	67.7	14(14)	4.4	233(508)	231.3
MANN_a81	0(0)	84.6+0.0=	84.6	25(23)	97.0	7(0)	93.6
brock200_2	322(718)	0.1+0.6=	0.7	265(492)	0.4	80(113)	0.2
brock200_4	67(197)	0.3+0.2=	0.5	424(124)	0.9	622(748)	2.3
brock400_2	813(768)	2.2+10.2=	12.3	955(427)	10.6	954(640)	15.1
brock400_4	208(151)	2.1+2.7=	4.8	945(468)	10.4	806(605)	12.7
brock800_2	820(986)	1.4+32.3=	33.7	1437(626)	52.1	1227(766)	51.3
brock800_4	620(512)	1.4+25.4=	26.8	1437(576)	51.9	1270(776)	53.4
gen200_p0.9_44	364(165)	1.3+1.7=	3.0	1065(531)	4.7	376(146)	4.3
gen200_p0.9_55	4(5)	2.0+0.0=	2.0	203(31)	0.9	134(25)	1.5
gen400_p0.9_55	1423(898)	3.3+24.3=	27.7	1382(742)	22.3	1086(503)	35.8
gen400_p0.9_65	281(137)	3.8+4.7=	8.5	718(158)	11.1	412(40)	13.2
gen400_p0.9_75	49(42)	4.3+0.8=	5.1	555(443)	8.8	289(22)	9.2
hamming8-4	0(1)	0.7+0.0=	0.7	134(93)	0.5	59(41)	0.3
hamming10-4	806(493)	7.9+66.8=	74.6	1750(632)	129.9	1167(323)	103.2
keller4	1(1)	0.5+0.0=	0.5	9(8)	0.0	3(2)	0.0
keller5	936(775)	4.0+40.1=	44.1	1417(500)	55.7	1063(598)	50.2
keller6	1489(768)	23.6+1180.0=	1203.6	2242(375)	1682.4	2112(498)	1731.3
p_hat300-1	4(4)	0.2+0.0=	0.2	173(164)	0.4	34(31)	0.1
p_hat300-2	25(25)	0.8+0.1=	0.9	300(48)	1.2	190(47)	1.1
p_hat300-3	402(248)	1.9+3.1=	5.0	718(228)	4.9	486(96)	5.5
p_hat700-1	206(274)	0.4+3.0=	3.4	995(375)	14.1	619(310)	9.4
p_hat700-2	206(127)	3.1+5.5=	8.6	551(86)	12.7	452(99)	13.0
p_hat700-3	919(664)	5.9+35.8=	41.7	1106(438)	38.0	1030(347)	50.3
p_hat1500-1	461(745)	1.1+31.7=	32.8	1488(640)	97.0	700(374)	46.3
p_hat1500-2	791(419)	7.0+92.2=	99.2	1181(580)	125.8	790(220)	94.9
p_hat1500-3	945(776)	14.9+152.8=	167.8	1360(596)	214.7	1071(422)	195.1

Table 2. Time results for DIMACS benchmarks. For each instance, the table displays the results obtained by Ant-Clique (with preprocessing and 7 ants, without preprocessing and 7 ants, and without preprocessing and 30 ants) and reports the number of cycles (average over 25 runs and standard deviation) and the average CPU time needed to find the best solution. For ant-Clique with preprocessing, we successively report the time spent for the preprocessing step, the time spent for ACO solving, and the total time. Benchmarks were run on a 350MHz G4 processor.

neither local search technics to improve solutions quality, artificial ants are able to find optimal solutions on many difficult benchmark instances.

However, Ant-Clique does not reach the best known results on all benchmark instances. The best current approach for the maximum clique problem is, to our knowledge, Reactive Local Search (RLS) [1]. This approach combines local search with prohibition-based diversification techniques where the amount of diversification is determined in an automated way through a feedback scheme. This mechanism allows RLS to escape from local maxima during resolution. In the experiments reported here, Ant-Clique has been able to find the best solution found by RLS on 30 instances over 37.

Hence, further work will first explore the integration within Ant-Clique of some local search technics such as the one used by RLS. Actually, the best performing ACO algorithms for many combinatorial problems are hybrid algorithms that combine probabilistic solution construction by a colony of ants with local search [5, 9, 7]. Also, Ant-Clique performances may be enhanced by introducing a local heuristic to guide ants. Indeed, as pointed out in Section 2, we already have experimented a first heuristic, borrowed from [1], that has not given any improvement. However, we shall further investigate other heuristics for this problem.

References

1. R. Battiti and M. Protasi. Reactive local search for the maximum clique problem. *Algorithmica*, 29(4):610–637, 2001.
2. I. Bomze, M. Budinich, P. Pardalos, and M. Pelillo. The maximum clique problem. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 4. Kluwer Academic Publishers, Boston, MA, 1999.
3. M. Dorigo. *Optimization, Learning and Natural Algorithms* (in Italian). PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992.
4. M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999.
5. M. Dorigo and L.M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.
6. E. Marchiori. Genetic, iterated and multistart local search for the maximum clique problem. In Stefano Cagnoni, Jens Gottlieb, Emma Hart, Martin Middendorf, and G"unther Raidl, editors, *Applications of Evolutionary Computing, Proceedings of EvoWorkshops2002: EvoCOP, EvoIASP, EvoSTim*, volume 2279, pages 112–121, Kinsale, Ireland, 3-4 2002. Springer-Verlag.
7. C. Solnon. Ants can solve constraint satisfaction problems. *IEEE Transactions on Evolutionary Computation*, 6(4):347–357, 2002.
8. C. Solnon. Boosting ACO with a preprocessing step. In Stefano Cagnoni, Jens Gottlieb, Emma Hart, Martin Middendorf, and G"unther Raidl, editors, *Applications of Evolutionary Computing, Proceedings of EvoWorkshops2002: EvoCOP, EvoIASP, EvoSTim*, volume 2279, pages 161–170, Kinsale, Ireland, 3-4 2002. Springer-Verlag.
9. T. Stützle and H.H. Hoos. $\mathcal{MAX}-\mathcal{MIN}$ Ant System. *Journal of Future Generation Computer Systems*, 16:889–914, 2000.