A Hierarchical Social Metaheuristic for the Max-Cut Problem

Abraham Duarte[1], Felipe Fernández[2], Ángel Sánchez[1] and Antonio Sanz[1]

[1] ESCET-URJC, Campus de Móstoles, 28933, Madrid Spain
{a.duarte, an.sanchez, a.sanz}@escet.urjc.es
[2] Dept. Tecnología Fotónica, FI-UPM, Campus de Montegancedo, 28860, Madrid, Spain
Felipe.Fernandez@es.bosch.com

Abstract. This paper introduces a new social metaheuristic for the Max-Cut problem applied to a weighted undirected graph. This problem consists in finding a partition of the nodes into two subsets, such that the sum of the weights of the edges having endpoints in different subsets is maximized. This NP-hard problem for non planar graphs has several application areas such as VLSI and ASICs CAD. The hierarchical social (HS) metaheuristic here proposed for solving the referred problem is tested and compared with other two metaheuristics: a greedy randomized adaptive search procedure (GRASP) and an hybrid memetic heuristic that combines a genetic algorithm (GA) and a local search. The computational results on a set of standard test problems show the suitability of the approach.

1 Introduction

An important graph bipartition problem is the Max-Cut problem defined for a weighted undirected graph $S = (V, E, W)$, where V is the ordered set of vertices or nodes, E is the ordered set of undirected arcs or edges and W is the ordered set of weights associated with each edge of the graph. This Max-Cut optimization problem consists in finding a partition of the set V into two disjoint subsets (C, C') such that the sum of the weights of edges with endpoints in different subsets is maximized. Every partition of vertices V into C and C' is usually called a cut or cutset and the sum of the weights of the edges is called the weight of the cut.

For the considered Max-Cut optimization problem, the cut value or weight of the cut given by

$$w(C,C') = \sum_{v\in C, u\in C'} w_{vu} \tag{1}$$

is maximized. In [9] is demonstrated that the decision version of Max-Cut is NP-Complete. This way, we need to use approximate algorithms for finding the solution in a reasonable time. Notice that for planar graphs exacts algorithms exist which solve the Max-Cut problem in polynomial time [10].

In this paper we propose a new hierarchical social (HS) metaheuristic for finding an approximate solution to Max-Cut problem. Additionally, two standard metaheruristics: GRASP and memetic algorithms (MA) are analyzed and compared with the HS algorithm.

The new introduced metaheuristic, hierarchical social algorithms [3,4], is inspired in the behaviour of some hierarchical societies. HS algorithms make use of local search heuristics with a group-population-based strategy. Their basic structure is a graph where a dynamical domain partition is performed. In each region of the considered partition a hierarchical group is settled. Each group consists of a core that determines the value of the corresponding group objective function and a periphery that defines the local search region of the involved group. Through a competitive group population strategy, the set of groups is optimized, the number of groups successively decreases and an approximate optimal solution is found.

A second metaheuristic used to compare with HS algorithms is a Greedy Randomized Adaptive Search Procedure (GRASP) [5,15,16], based on a construction phase and a local optimization phase.

A third metaheuristic also used to compare the experimental results, is based on a genetic algorithm [7,12,17] with an additional local search based on the problem domain knowledge (memetic algorithm [13,14]), which allows a remarkable improvement of the solution obtained.

2 Review of Max-Cut Problem Solution

The Max-Cut problem can be formulated as an integer quadratic program [18]. This program can not be efficiently solved because, in the general case, it is a NP-complete problem [9]. For planar graphs exact polynomial time algorithms exist [10]. Several continuous and semidefinite relaxations have been proposed in order to achieve a good solution in a reasonable running time. In [11,18] is described the semidefinite relaxation of the Max-Cut problem. Goemans et al. showed in [6] a randomized algorithm that guarantees a 0.878-approximation to the optimum and in addition an upper bound on the optimum. Based on this work, other approximation algorithms have been proposed [2].

There are other approaches to the Max-Cut problem relaxation [2]. Maybe, one of the most interesting is the 2-rank relaxation proposed in [1] that in practice produces better solutions than the mentioned randomized algorithm [6].

3 Hierarchical Social Algorithms for the Max-Cut Problem

This section presents the general features of a new metaheuristic called hierarchical social (HS) algorithms. In order to get a more general description of this metaheuristic, the reader is referred to [3][4].

HS algorithms are inspired in the hierarchical social behaviour observed in a great diversity of human organizations and biological systems.

This metaheuristic have been successfully applied to several problems such as critical circuit computation [4] and DFG scheduling with unlimited resources [3].
The key idea of HS algorithms is a simultaneous optimization of a set of feasible solutions. Each group of a society contains a feasible solution and these groups are initially random distributed through the solution space. By means of an evolution

strategy, where each group tries to improve itself or competes with the neighbour groups, better solutions are obtained through the corresponding social evolution.
In this social evolution, the groups with lower quality tend to disappear. As a result, the rest of the group objective functions are optimized. The process usually ends with only one group that contains the best solution found.

3.1 Metaheuristic structure

For the Max-Cut problem, the feasible society is modelled by the specified undirected weighted graph $S=(V,E,W)$ also called feasible society graph. The set of individuals are modelled by nodes V of the specified graph, and the set of feasible relations are modelled by arcs A of the specified graph. Figure 1.a shows an example of the feasible society graph for a particular Max-Cut problem.

The state of a society is modelled by a hierarchical policy graph. This hierarchical policy graph also specifies a society partition composed by a disjoint set of groups $\Pi=\{g_1, g_2,\dots,g_g\}$, where each individual or node is assigned to a group. Each group $g_i\subset\Pi$ is composed by a set of individuals and active relations, which are constrained by the feasible society. The individuals of all groups cover the individuals of the whole society.

The specification of the hierarchical policy graph is problem dependent. The initial society partition determines an arbitrary number of groups and assigns individuals to groups. Figure 1.b shows a society partition example formed by two groups.

A society partition can be classified into *single-group or monopoly partition*, in which there is only one group, and *multiple-group or competition partition*, in which there are more than one group. Obviously in example shown in Figure 1.b, the partition shown is a competition partition.

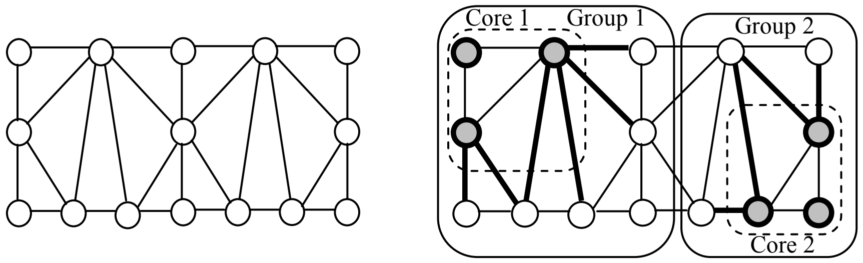

Fig. 1. (a) Feasible society graph. (b) Society partition and groups partition

Each individual of a society has its individual objective function *f1*. Each group has its group objective function *f2* that is shared by all individuals of a group. Furthermore each group g_i is divided into two disjoint parts: core and periphery. The core determines the value of the corresponding group objective function *f2* and the periphery defines the local search region of the involved group.

In the Max-Cut problem framework, the set of nodes V_i of each group g_i is divided into two disjoint parts: $g_i=(C_i, C_i')$ where C_i represents the *core* or subgroup of nodes that belongs to the considered cutset and C_i' is the complementary subgroup of nodes or *periphery*. The core edges are the arcs that have their endpoints in C_i and C_i'. Figure 1.b also shows an example of cores for the considered previous partition. The core edges of each group are shown in this figure by thick lines.

For each group $g_i = (C_i, C_i')$, the group objective function $f2(i)$ is given by the function $cut(i)$:

$$f2(i) = cut(i) \quad \text{where} \quad cut(i) = \sum_{v \in Ci, u \in Ci'} w_{vu} \tag{2}$$
$$\forall v \in g_i \quad f2(v,i) = f2(i) = cut(i)$$

where the weights w_{vu} are supposed to be null if the corresponding edges do not belong to the specified graph. Obviously, this value is determined by the core edges, because is the sum of the edges weights which have one endpoint in the core C_i, and the other endpoint in the periphery C_i' of the corresponding group g_i.

For each individual or node v, the individual objective function $f1(v,i)$ relative to each group $g_i = (Ci, Ci')$ is given by the function:

$$f1(v,i) = max(\sigma(v,i), \sigma'(v,i)) \quad \text{where} \tag{3}$$
$$\sigma(v,i) = \sum_{u \in Ci} w_{vu} \quad \text{and} \quad \sigma'(v,i) = \sum_{u \in Ci'} w_{vu}$$

The HS algorithms here considered, try to optimize one of their objective functions (*f1* or *f2*) depending on the operation phase. During an autonomous or winner phase, each group *i* tries to improve independently the group objective function *f2*. During a loser phase, each individual tries to improve the individual objective function *f1*, the original groups cohesion disappeared and the graph partition is modified in order to optimize the corresponding individual objective function. The strategy followed in a loser phase could be considered inspired in Adams Smith's "invisible hand" economic society mechanism described in his book "An Inquiry into the Nature and Causes of the Wealth of Nations".

3.2 Metaheuristic process

The algorithm starts from a random set of feasible solutions i.e. it obtains an initial partition that determines the corresponding groups structure. Additionally for each group, an initial random cutset is derived.

The groups are successively transformed through a set of competition phases based on local search strategies. For each group, there are two main strategies: *winner or autonomous strategy* and *loser strategy*. The groups named *winner groups*, which apply the winner strategy, are that which have the higher group objective function *f2*. The rest of groups apply loser strategy and are named *loser groups*. During optional *autonomous phases* in between competition phases, all groups behave like winner groups. These optional autonomous phases improve the capability of the search procedure.

The winner strategy can be considered as a local search procedure in which the quality of the solution contained in each group is improved by autonomously working with the individuals that belong to each group and the relations among these individuals.

The loser strategy is oriented to let the exchange of individuals among groups. In this way the groups with inferior quality (lower group objective function *f2*) tend to disappear, because their individuals move from these groups to another groups that grant their maximum individual promotion (highest individual objective function *f1*).

Winner and loser strategies are the basic search tools of HS algorithms. Individuals of loser groups, which have lower group objective functions, change of group during a loser strategy, in order to improve their individual objective function *f1*. This way, the loser groups tend to disappear in the corresponding evolution process. Individuals of winner groups, which have highest group objective functions, can move from the core to periphery or inversely, in order to improve the group objective function *f2*. These strategies produce dynamical groups populations, where group annexations and extinctions are possible.

3.3 High-level pseudo-code

A general high-level description of an HS metaheuristic is shown in Figure 2, where the main search procedures are: Winner Strategy and Loser Strategy.

In the Max-Cut problem, the Winner Strategy (Figure 3) is oriented to improve the cut value of a group. For a given group g_i, a nodes exchange, between core and periphery, allows to optimize the group objective function. The exchange is accomplished if there is an improvement of the cut weight or objective function $f2(i)$ of the corresponding group g_i. This procedure is based on the local search procedure presented in [15,16]. In our implementation, we also allow to move the nodes between C_i and C_i' in parallel and simultaneous way on a single iteration. The only restriction is the following: if one node u is gone out from C_i (or C_i'), none of its adjacent nodes can change their position in the same iteration. This restriction avoids cycling in the corresponding procedure.

```
Procedure Hierarchical_Social_Algorithm(S)
Var
 S=(V,E,W)=({v}{e}{w}):Initial_Society_graph;
 G={gi}:Groups_structure;
 F1={f1}:Individuals_objective_function_structure;
 F2={f2}:Groups_objective_function_structure;
 i,k:1..Number_of_groups /*Group indices*/
Begin /* Begin social evolution */
 G=Get_initial_random_partition_and_groups_structure();
 Repeat  /* group evolution*/
  Compute_F1()_and_F2(); /*Objective functions*/
  For each gi in G
   If f2(i)=max{f2(k)∀k} Or Autonomous_phase Then
    Winner_Strategy(i)
   Else
    Loser_Strategy(i);
  End For
  Update_groups_structure(G);
 Until termination_condition_met;   /*End of social evolution*/
 Return(G); /* Approximate optimal solution */
End Hierarchical_Social_Algorithm
```

Fig. 2. High-level pseudo-code for Hierarchical Social Algorithms

For each vertex v and each group $g_i = (C_i, C_i')$ the following function σ and σ' are considered:

$$\sigma(v,i) = \sum_{u \in Ci} w_{vu} \quad \sigma'(v,i) = \sum_{u \in Ci'} w_{vu} \tag{4}$$

These evaluation functions are used in the pseudo-code of Figure 3.

The Loser Strategy allows an individual to move to another group. The individuals belonging to groups with lower cut value *f2*(*i*) can change their group in order to increase the individual objective function *f1*(*v*,*i*). Each node searches for the group *j* that gives the maximum improvement in its individual objective function. Figure 4 shows the pseudo-code of Loser Strategy considering the functions σ and σ´ defined in (4).

```
Procedure Winner_Strategy(i)/* for Max_Cut problem*/
Var gi=(Ci,Ci´):Cutset_structure_of_considered_group
Begin
For v = 1 to Unmber_of_nodes_of_considered_group
  If v∈Ci and σ(v,i)>σ′(v,i) Then Ci´=Ci′∪{v}; Ci=Ci\{v}; /*v→Ci´*/
  If v∈Ci´and σ(v,i)<σ′(v,i) Then Ci′= Ci′\{v}; Ci=Ci∪{v};/*v→Ci */
End For
End Winner_Strategy
```

Fig. 3. Pseudo-code of Winner Strategy

```
Procedure Loser_Strategy(i)/* for Max_Cut problem*/
Var
 gi=(Ci,Ci´):Cutset_structure_of_considered_group
 G={g_i}={(C_i,C_i´)}:Cutset_structure_of_groups
 i,j,k:1..Number_of_groups /*Group indices*/
Begin
For v = 1 to Number_of_nodes_of_considered_group
  j= arg max {max(σ(v,k),σ′(v,k)),∀k} /*j=host group*/
  If (v∈Ci) Then Ci=Ci\{v} Else Ci´=Ci´\{v}; /*Remove v from gi*/
  If σ(v,j)>σ´(v,j) Then Cj′=Cj′∪{v} Else Cj=Cj ∪{v}; /*Add v to gj*/
End For
End Loser_Strategy
```

Fig. 4. Pseudo-code of Loser Strategy.

4 GRASP for MAX-CUT Problem

This section briefly summarizes the use of a Greedy Randomized Adaptive Search Procedure (GRASP) for solving the Max-Cut problem. For a more detailed description the reader is referred to references [5,15,16]. GRASP is a multi-start iterative method where in each iteration there are two phases, construction phase and local search phase. In the first phase, a greedy feasible solution is constructed and in the second phase, starting from this solution, a local optimal solution is derived. Figure 5 shows a high level pseudo-code of this metaheuristic, assuming that the objective function *f* is the same that the group objective function (*f2*) introduced in (2).

The construction phase makes of an adaptive greedy function that uses a restricted candidates list (RCL) and a probabilistic selection criterion [15,16]. In the Max-Cut problem, for each node $v \notin C \cup C'$, the following greedy function *f*(*v*) is considered:

$$\sigma(v) = \sum_{u \in C} w_{vu} \quad \sigma'(v) = \sum_{u \in C'} w_{vu} \tag{5}$$

$$f(v) = max\{\sigma(v), \sigma'(v)\} \qquad \forall v \notin C \cup C' \tag{6}$$

This greedy function is used to create an RCL based on the specified cut-off value α. Later on, a random procedure successively selects vertices from this RCL in order to obtain an initial feasible solution.

```
Procedure GRASP(S,Max_number_of_iterations)
Var g:Solution_structure;
Begin
 For i=1 to Max_number_of_iterations   /*Multi-start*/
  g= Build_a_greedy_randomized_solution(S);
  g= Local_Search(g);
  If i = 1 Then g* = g; /*Initial solution*/
  Else f(g)>f(g*) Then g* = g; /*Update solution*/
 End For
 Return(g*); /* Approximate optimal solution */
End GRASP;
```

Fig. 5. Pseudo-code of a generic GRASP

The local search phase is later applied, which is also based on the previous definitions of σ and σ', taking into account that all nodes have already been assigned to C or C'. If a node $v \in$ C and σ(v) > σ′(v) then it should be changed from C to C': C'=C'∪{v} and C= C\{v}, and conversely. This second phase stops when all possible movements have been computed and no improvement was found.

5 Memetic Algorithms for the MAX-CUT Problem

This section briefly summarizes the use of genetic algorithms for solving the Max-Cut problem. For a more detailed description the reader is referred to references [7,12,14,17].
The introduction of an additional local search phase on a genetic algorithm produces a memetic algorithm. This improves the quality of the solutions obtained. In Figure 6 is shown the high level pseudocode of a memetic algorithm.

```
Procedure_Memetic_Algorithm(S, Number_generations)
 Initialize_random_population(S);
 Local_Search(); /*Local search used in GRASP */
 For gen =1 to Number_generations
   Select_parents(gen);
   Produce_offspring_by_fixed_crossover_of_parents(gen);
   Local_Search(gen); /*Local search used in GRASP */
   Produce_mutation(gen);
   Evaluate_the_offspring(gen);
   Replacement_process(gen);
 End For
End Memetic_Algorithm
```

Fig. 6. High-level general pseudo-code of used memetic algorithm.

In order to use memetic algorithms for solving the Max-Cut problem, we need to code each feasible solution. Let V = {1,…,n} the nodes set of a given graph. The cuts of this graph can be coded by a Boolean n-vector $I = (i_1,...,i_n)$ such that the value of each $i_u \in \{0,1\}$ is given by the characteristic function: $i_u=\{(u\in C)\rightarrow 1;\ (u\in C')\rightarrow 0\}$:

In the evaluation step, the selection method used is the fitness roulette-wheel selection [12], which favours individuals with high fitness without suppressing the chance of selection of individuals with low fitness, avoiding premature convergence of the population.

This way, a subset on individuals is selected, introduced in the new population and recombined with a probability p_r. In this paper we consider the crossover method, called fixed crossover [2], which is dependent on the structure of the crossed individuals. In general, the use of structural information is an advantage providing more quality descendants.

The fixed crossover function considered f: $\{1,0\}\times\{1,0\}\rightarrow\{1,0\}$is specified by the following random Boolean function:

$$f((\alpha,\beta))=\{(0,0)\rightarrow 0;\ (0,1)\rightarrow \text{rand01};\ (1,0)\rightarrow \text{rand01};\ (1,1)\rightarrow 1\ \} \tag{7}$$

where rand01 is a random Boolean value. In this way, if both parents are in the same subset, the offspring node lies in this subset. Otherwise, the node is randomly assigned to one of the subsets. With this crossover function, each bit i_u of new offspring is given by:

$$i_u = f(father(i_u), mother(i_u)) \qquad \forall u \in V \tag{8}$$

To ends up with the evolution cycle, new individuals are subject to mutation (a random change of a node from C to C' or inversely) with probability p_m.

6 Experimental Results

In this section, we describe the experimental results for the three considered metaheuristics. The computational experiments were evaluated in an Intel Pentium 4 processor at 1.7 GHz, with 256 MB of RAM. All algorithms were coded in C++, without optimization and for the same programmer in order to have more comparable results.

The main objective of this section consists in comparing different types of evolutionary approaches. In some sense, the memetic algorithms are in one extreme (population based metaheuristic) and GRASP algorithms are in the other extreme (single constructive metaheuristic). HS metaheuristics can be considered in the middle of these two alternatives, because they start with a population of solutions but usually end with a single solution. Some main details of the metaheuristics implementation are the following:

1. HS Algorithms: we ran 100 independent iterations of HS Algorithms. The number of groups and autonomous iterations were randomly selected, with a uniform distribution, in the intervals [|V|/80, |V|/2] and [1, 11] respectively.
2. GRASP: we ran 100 independents iterations of GRASP. The cut-off value α used to construct the RCL is selected randomly, with a uniform distribution in [0,1].
3. Memetic Algorithms: we ran 100 independent iterations. Each iteration had a population of 8 individuals, the maximum number of generations was 20, the probability of crossover was 0.8 and finally the probability of mutation was 1/|V|.

All the metaheuristic were tested on the graphs Gxx shown in Table 1. These test problems were generated by Helmberg and Rendl using the graph generator described in [6]. They are *planar*, *toroidal* and *randomly* generated graphs of varying sparsity and size. The last two graphs types are in general non-planar. In these experiments, the graph sizes vary from 800 nodes to 3000 and their density from 0.17% to 6.12%.

The first three columns of Table 1 show respectively the *name* of the graph, the number of nodes *n* and the number of arcs *m*. The following three columns are the experiments for the three tested metaheuristic (Memetic algorithms, GRASP and HS algorithms). These columns are divided into three sub-columns: the maximum Max-Cut value found in the 100 independent iterations *cut*, the time spent in running these 100 independent iterations *T*(*s*), and the *mean* and the standard deviation *sd,* respectively in these 100 independent iterations, shown in the third sub-column.

Right column shows the SDP value [6] that can be considered as an upper bound.

The experimental comparison of these metaheuristics shows a superior behavior of HS algorithms in relation with the computational complexity and quality of the approximate solutions obtained. Moreover, the quality of the obtained solutions using the memetic approach is also remarkable.

Table 1. Results for Helmberg´s instances [8]: solutions found with 100 independent iterations for Memetic, GRASP and HS metaheuristics. In bold is the maximum value for each instance.

Problem			**Memetic Algorith**			**GRASP**			**HS Algorithm**			
Name	*n*	*m*	*Cut*	*T(s)*	*Mean (sd)*	*Cut*	*T(s)*	*Mean (sd)*	*Cut*	*T(s)*	*Mean (sd)*	*SDP*
G1	800	19176	11546	1356	11466.4 (34.0)	11475	248	11369.8(39.6)	**11549**	221	11444.8(33.7)	12078
G2			**11546**	1340	11464.9(39.9)	11506	246	11376.2(37.2)	11501	215	11446.2(29.4)	12084
G3			11545	1336	11461.6(36.6)	11467	317	11374.3(34.2)	**11550**	226	11442.8(35.1)	12077
G11	800	1600	380	770	353.2(13.1)	506	371	482.0(10.0)	**546**	256	532.5(6.78)	627
G12			380	285	340.2(15.5)	500	362	481.2(8.01)	**540**	241	525.6(6.82)	621
G13			398	284	364.0(15.6)	530	335	500.0(9.38)	**568**	268	551.7(6.33)	645
G14	800	4694	2990	768	2961.9(14.1)	2983	418	2962.9(9.13)	**3014**	205	2986.6(9.05)	3187
G15			2970	760	2941.4(13.1)	2964	412	2941.6(10.5)	**2993**	208	2969.4(9.38)	3169
G16			2971	762	2947.5(12.2)	2967	411	2946.9(10.4)	**2996**	199	2973.5(8.63)	3172
G22	2000	19990	**13089**	5709	12987.7(55.5)	12971	7578	12884.4(33.6)	13061	1975	12993.7(34.9)	14123
G23			**13110**	7232	12988.9(54.9)	12998	7598	12887.5(40.5)	13098	2035	12997.0(36.9)	14129
G24			**13125**	7295	12996.7(51.9)	12978	7606	12890.1(39.2)	13089	1996	12990.3(37.8)	14131
G32	2000	4000	916	3262	846.0(26.3)	1220	4777	1193.6(12.7)	**1360**	3275	1330.9(10.9)	1560
G33			900	3223	835.1(31.5)	1212	4761	1179.8(12.5)	**1324**	3289	1299.9(13.0)	1537
G34			888	3266	812.3(28.8)	1228	4765	1188.5(17.5)	**1334**	3269	1308.8(12.1)	1541
G35	2000	11778	7478	3819	7408.1(31.5)	7456	6270	7422.0(14.1)	**7548**	2350	7487.7(18.2)	8000
G36			7450	3965	7399.0(28.0)	7445	6290	7417.7(14.0)	**7524**	2389	7482.2(18.2)	7996
G37			7465	3947	7411.8(30.9)	7456	6295	7426.7(13.7)	**7548**	2415	7488.7(19.7)	8009
G43	1000	9990	6553	1239	6494.3(28.7)	6500	981	6438.0(27.1)	**6561**	655	6490.9(22.8)	7027
G44			6532	1220	6490.0(24.5)	6497	922	6432.7(24.5)	**6540**	721	6488.5(22.9)	7022
G45			6545	1200	6485.4(31.0)	6499	952	6435.7(26.6)	**6564**	683	6484.9(24.4)	7020
G48	3000	6000	5008	7007	4869.6(84.6)	5996	12653	5991.2(13.3)	**6000**	7420	5931.2(74.5)	6000
G49			5024	6906	4859.5(81.0)	5996	12521	5988.3(22.0)	**6000**	7525	5930.7(65.6)	6000
G50			4988	9276	4848.3(81.3)	**5880**	12607	5877.2(7.40)	**5880**	7389	5840.8(41.6)	5988

7 Conclusions

This paper has introduced a hierarchical social algorithm to efficiently solve the Max-Cut problem. We have converted the search process into a social evolution one. This social paradigm exploits the power of competition and cooperation among different groups to explore the solution space. We have experimentally shown that the proposed HS algorithm significantly reduces the number of iterations of the search procedure. The memetic approach described has also given high quality solutions.

We propose as a future work a deeper study about the correlation between the performance of the algorithms and the main characteristics of the graph (type, density, etc.).

8 References

1. Burer, S., Monteiro, R.D.C., Zhang X.: Rank-two Relaxation heuristic for the Max-Cut and other Binary Quadratic Programs. SIAM Journal of Optimization, 12:503-521, 2001.
2. Dolezal O., Hofmeister, T., Lefmann, H: A comparison of approximation algorithms for the MAXCUT-problem. Reihe CI 57/99, SFB 531, Universität Dortmund, 1999.
3. Fernández F., Duarte A., Sánchez A.: A Software Pipelining Method Based on Hierarchical Social Algorithms, Proceedings of MISTA Conference, Vol. 1, pp 382-385, Nottingham, UK, 2003.
4. Fernández F., Duarte A., Sánchez A.: Hierarchical Social Algorithms: A New Metaheuristic for Solving Discrete Bilevel Optimization Problems, Technical Report ESCET/URF - DTF/ UPM. HSA1 Universidad Politécnica de Madrid, 2003.
5. Festa P., P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro: Randomized heuristics for the MAX-CUT problem, Optimization Methods and Software, vol. 7, pp. 1033-1058, 2002.
6. Goemans, M. X., Williams, D.P.: Improved Approximation Algorithms for Max-Cut and Satisfiability Problems Using Semidefinite Programming. Journal of the ACM.42:1115-1142 , 1995.
7. Goldberg. D.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, 1989.
8. Helmberg, C., Rendl, F.: A Spectral Bundle Method for Semidefinite Programming. SIAM Journal of Computing, 10:673:696, 2000.
9. Karp, R.M.: Reducibility among Combinatorial Problems. In R. Miller J. Thatcher, editors, Complexity of Computers Computation, Prenum Press, New York, USA (1972).
10. Hadlock F. O: Finding a Maximum Cut of a Planar Graph in Polynomial Time. SIAM Journal on Computing 4 (1975) 221-225.
11. Lovàsz, L.: On the Shannon Capacity of a Graph, IEEE Trans. of Information Theory, IT-25:1-7, 1978.
12. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. 3rd edn. Springer-Verlag, Berlin Heidelberg New York, 1996.
13. Michalewicz, Z., Fogel. D. B.: How to Solve it: Modern Heuristics, Springer-Verlag, Berlin, 2000.
14. Moscato P., Cotta C.: A Gentle Introduction to Memetic Algorithms. In Handbook of Metaheuristic. F. Glover and G. A. Kochenberger, editors, Kluwer, Norwell, Massachusetts, USA, 2003.
15. Resende M.G., Ribeiro C.: Greedy Adaptive Search Procedures. In Handbook of Metaheuristic. F. Glover and G. A. Kochenberger, editors, Kluwer, Massachusetts, USA , 2003.
16. Resende M.G.: GRASP With Path Re-linking and VNS for MAXCUT, In Proceedings of 4th MIC, Porto, July 2001.
17. Spears, W. M.: Evolutionary Algorithms. The Role of Mutation and Recombination, Springer-Verlag, Berlin Heidelberg New York, 1998.
18. Shor, N. Z.: Quadratic Optimization Problems, Soviet Journal of Computing and System Science, 25:1-11, 1987.