INT J COMPUT COMMUN, ISSN 1841-9836
Vol.7 (2012), No. 2 (June), pp. 377-387

# A 2-level Metaheuristic for the Set Covering Problem

C. Valenzuela, B. Crawford, R. Soto, E. Monfroy, F. Paredes

**Claudio Valenzuela, Broderick Crawford**
Pontificia Universidad Católica de Valparaíso
Valparaíso, Chile
{claudio.valenzuela, broderick.crawford}@ucv.cl

**Ricardo Soto**
1. Pontificia Universidad Católica de Valparaíso
Valparaíso, Chile, and
2. Universidad Autónoma de Chile
ricardo.soto@ucv.cl

**Eric Monfroy**
Universidad Técnica Federico Santa María
Valparaíso, Chile
eric.monfroy@inf.utfsm.cl

**Fernando Paredes**
Escuela de Ingeniería Industrial
Universidad Diego Portales
Santiago, Chile
fernando.paredes@udp.cl

**Abstract:** Metaheuristics are solution methods which combine local improvement procedures and higher level strategies for solving combinatorial and nonlinear optimization problems. In general, metaheuristics require an important amount of effort focused on parameter setting to improve its performance. In this work a 2-level metaheuristic approach is proposed so that Scatter Search and Ant Colony Optimization act as "low level" metaheuristics, whose parameters are set by a "higher level" Genetic Algorithm during execution, seeking to improve the performance and to reduce the maintenance. The Set Covering Problem is taken as reference since is one of the most important optimization problems, serving as basis for facility location problems, airline crew scheduling, nurse scheduling, and resource allocation.
**Keywords:** metaheuristics, genetic algorithm, scatter search, ant colony optimization, set covering problem.

## 1 Introduction

The Set Covering Problem (SCP) is a classical problem in computer science and one of the most important discrete optimization problems since it can model conveniently real world problems. Some of these problems — which can be modeled as a set covering problem — include facility location, airline crew scheduling, nurse scheduling, resource allocation, assembly line balancing, vehicle routing, among others [6]. There are several studies which have implemented a SCP solution using metaheuristics [1,2,12]. Depending on the algorithm that has been used, the quality of the solution wanted and the complexity of the SCP chosen, it is defined the amount of customization efforts required. Conveniently, this work proposes transferring part of this customization effort to another metaheuristic (a "high level" metaheuristic) which can handle

the task of parameters adjustment for a low level metaheuristic. This approach is considered as a multilevel metaheuristic since there are two metaheuristics covering tasks of parameter setting, for the former, and problem solving, for the latter [9].

The main design of the implementation proposed considers a Genetic Algorithm (GA) [10] at online (Control) and offline (Tuning) parameter setting for a low level metaheuristic (Ant Colony Optimization (ACO) or Scatter Search (SS)) using a Reactive Search approach and an Automatic Parameter Tuning approach. In Reactive Search, feedback mechanisms are able to modify the search parameters according to the efficiency of the search process, i.e. the balance between intensification and diversification can be automated by exploiting the recent past of the search process through dedicated learning techniques [13]. The Automatic Parameter Tuning is carried by an external algorithm which searches for the best parameters in the parameter space in order to tune the solver automatically. Ant Colony Optimization and Scatter Search techniques [11] have shown interesting results at solving SCP [6] and similar problems [5]. For the purpose of this work, the former is selected by its constructional approach for generating solutions, plus its stochastic-based operators. The latter is considered as an evolutionary (population-based) algorithm which uses, essentially, deterministic operators. Both of them provide good reference metaheuristics in terms of their foundations, their problem solving approaches, their design maturity, and in terms of how different one is from the other, making them highly suitable to the development of this work.

## 2 Set Covering Problem

A general mathematical model of the Set Covering Problem can be formulated as follows:

$$(1)\ \ Minimize\, Z = \sum_{j=1}^{n} c_j x_j \qquad j = \{1, 2, 3, ..., n\}$$

Subject to:

$$(2)\ \ \sum_{j=1}^{n} a_{ij} x_j \geq 1 \qquad i = \{1, 2, 3, ..., m\}$$

$$(3)\ \ x_j = \{0, 1\}$$

Equation (1) is the objective function of set covering problem, where $c_j$ is the cost of $j$-column, and $x_j$ is decision variable. Equation (2) is a constraint to ensure that each row is covered by at least one column where $a_{ij}$ is a constraint coefficient matrix of size m x n whose elements comprise of either "1" or "0". Finally, equation (3) is the integrality constraint in which the value $x_j$ can be "1" if column $j$ is activated (selected) or "0" otherwise. Different solving methods have been proposed in the literature for the SCP. There exist examples using exact methods [8], linear programing and heuristic methods [7], and metaheuristic methods [2]. Has being pointed out, that one of the most relevant applications of SCP is given by crew scheduling problems in mass transportation companies where a given set of trips has to be covered by a minimum-cost set of pairings, a pairing being a sequence of trips that can be performed by a single crew.

## 3 Multilevel Metaheuristics and Parameter Setting

Metaheuristics, in their original definition, are solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable

of escaping from local optima and performing a robust search of a solution space [3]. Over time, these methods have also come to include any procedures that employ strategies for overcoming the trap of local optimality in complex solution spaces, especially those procedures that utilize one or more neighborhood structures as a means of defining admissible moves to transition from one solution to another, or to build or destroy solutions in constructive and destructive processes. A number of the tools and mechanisms that have emerged from the creation of metaheuristic methods have proved to be remarkably effective, so much that metaheuristics have moved into the spotlight in recent years as the preferred line of attack for solving many types of complex problems, particularly those of a combinatorial nature.

Multilevel Metaheuristics can be considered as two or more metaheuristics where a higher level metaheuristic controls de parameters of a lower level one, which is at charge of dealing more directly to the problem. Therefore, parameter Setting is a key factor in the design of Metaheuristics and Multilevel Metaheuristics [4], since they improve their solving performance by modifying and adjusting themselves to the problem at hand, either by self-adaptation or supervised adaptation.

### 3.1 Parameter Setting

Many parameters have to be set for any metaheuristic. Parameter setting may allow a larger flexibility and robustness, but requires a careful initialization. Those parameters may have a great influence on the efficiency and effectiveness of the search. It is not obvious to define *a priori* which parameter setting should be used. The optimal values for the parameters depend mainly on the problem and even the instance to deal with and on the search time that the user wants to spend in solving the problem. A universally optimal parameter values set for a given metaheuristic does not exist.

### 3.2 Tuning Before Solving

Also know as "Offline Parameter Initialization". As previously mentioned, metaheuristics have a major drawback; they need some parameter tuning that is not easy to perform in a thorough manner. Those parameters are not only numerical values but may also involve the use of search components [15]. Usually, metaheuristic designers tune one parameter at a time, and its optimal value is determined empirically. In this case, no interaction between parameters is studied. This sequential optimization strategy (i.e., one-by-one parameter) do not guarantee to find the optimal setting even if an exact optimization setting is performed. Main flavors of "tuning before solving" techniques include: Parameter Tuning on Preliminary Experiments, Empirical Manual Tuning and Automatic Parameter Tuning by an External Algorithm.

### 3.3 Control During Solving

Also know as "Online Parameter Initialization". The drawback of the offline parameter setting approaches is their high computational cost, particularly if this approach is used for each input instance of the problem. Indeed, the optimal values of the parameters depend on the problem at hand and even on the various instances to solve. Then, to improve the effectiveness and the robustness of offline approaches, they must be applied to any instance (or class of instances) of a given problem. The control of the solverÕs behavior during the run can be achieved by either modifying its components and/or its parameters. This corresponds, for instance, to an online adjustment of the parameters or heuristics. Such control can be achieved by means of supervised control schemes or by self adaptive rules. Of course, such approaches often rely on a learning process that tries to benefit from previously encountered problems along the search or even

during the solving of other problems. At this section are considered the approaches that change the parameters during the search with respect to the current state and other parameters. Of course, these parameters have a direct influence on the heuristics functions, but also these latter functions stay the same during the solving process. Some remarkable approaches are introduced as follows: Hyperheuristics and Reactive Search [16].

# 4 Implementation Details

## 4.1 GA-SS Design

The design proposed for the multilevel implementation is based on documented standards proposed for each metaheuristic. Both metaheuristics are looking to be as close as they can to its origins. Obviously, a multilevel implementation with adaptive parameter control approach forces some changes to the "high levelÓ metaheuristic, in this case, GA. This case will be similar to both combinations: GA-SS and GA-ACO. Hence, the design of GA seems particularly faithful to its basic design.

[ *Size of P , BestSet , DiverseSet , EnhanceTrials , MaxSol* ]

Figure 1: GA Chromosome Representation for SS.

The chromosome representation shows that the first gene, for this case the "*Size of P*" is the number of initial solutions for SS, will take values between 1 and $n/2$ where $n$ number of variables. Same way for the second gene "*BestSet*", which is the size of the Best Solutions Reference Set. Similarly, "*DiverseSet*" is the size of the Most Diverse Solutions Set. "*EnhanceTrials*", represents the number $e$ of trials for the Improvement Method to try to enhance a solution, where $e = \{1...n\}$. Finally, "*MaxSol*" is the limit of solutions generated by a call to the Scatter Search algorithm.

## 4.2 GA components

The main components of GA components are:

*Initialization function*: Initializes the values of genes within the variables bounds. It also initializes (to zero) all fitness values for each member of the population. It takes upper and lower bounds of each variable from user defined parameters. It randomly generates values between these bounds for each gene of each genotype in the population.

*Evaluation function*: The upper level metaheuristic uses two criterions; (1) the same evaluation function of the lower level metaheuristic, this is the corresponding *objective function* of SCP, and (2) a *Objective Function* of SCP penalized by the processing effort.

*"Keep the best" function*: This function keeps track of the best member of the population.

*Elitist function*: The best member of the previous generation is stored.

*Selection function*: Standard proportional selection for maximization/minimization problems incorporating elitist model — makes sure that the best member survives.

*Crossover selection*: selects two parents that take part in the crossover. This work implements a single point crossover.

*Mutation*: Random uniform mutation. A variable selected for mutation is replaced by a random value between lower and upper bounds of this variable.

### 4.3 SS components

A description of SS components included in GA-SS design:

*Diversification Generation Method:* This method generates diverse binary initial solutions, it starts with an all-zero arbitrary solution, and then begins adding "1" in every position with a jump of $k$ bits, where $k = \{1...n\}$ and $n < (number\_of\_vars) - 1$. In example, for $k = 1$ it will generate a solution vector $[1, 0, 1, 0, 1, 0, 1, 0...]$. Should be noticed that this method always starts placing a bit "1" at the first position. Together with the previously generated solution, the method generates the complement of that solution, which will be $[0, 1, 0, 1, 0, 1, 0, 1, 0, 1...]$. The quantity of solutions generated by this method is a parameter controlled by the GA.

*Improvement Method:* transforms a trial solution into a enhanced trial solution. If the trial solution is not feasible, should be fixed until it turns feasible. The input solutions are not required to be feasible. If the input trial solution is not improved as a result of the application of this method, the "enhanced" solution is considered to be the same as the input solution. The limit of enhancement trials (not the fix trials) is controlled by the GA. The duties of fixing and enhancing are improved since a vector of ratios is calculated (the length of the vector is the number of variables, or *columns*). This vector is then sorted from min to max ratio. Each element represents a variable at the objective function. Then, when trying to fix a solution should be followed in-order, so the first item will represent the column which covers more rows of the incidence matrix. If this position is "0" at the solution, should be turned to "1", because the method is wanting to *add* "1" for covering rows and turn the solution to feasibility. This process continues until the solution is feasible. When trying to enhance a solution, the vector of ratios should be accessed in post-order, trying to turn as many "1" to "0" as it can while the solution keeps feasible. The method will try to enhanced until the limit $l$ is reached, where $l$ is a parameter controlled by the GA.

*Reference Set Update Method:* builds and maintain two reference sets, consisting of the $b$ "best" solutions and the $d$ "*most* diverse" solutions found (where the value of $b$ and $d$ is set by the GA), organized to provide efficient accessing by other parts of the solution procedure. The criteria for adding the "best" solutions to its set is the cost at the objective function, also, the criteria for adding solutions at the "most diverse" set is the Hamming distance to all the solutions at the "best set" and the "diverse set".

*Subset Generation Method:* operates on the reference set, to produce a subset of its solutions as a basis for creating combined solutions. The most common subset generation method will be used, which generates all pairs of reference solutions (i.e., all subsets of size 2) from the "best set" and the "diverse set" together.

*Solution Combination Method:* transforms a given subset of solutions produced by the Subset Generation Method into one combined solution. The combination method is based in the *cost at objective function/ rows covered* ratio, which is calculated per column. When combining two solutions, the following procedure is used:

```
if (Sol_1[x] = Sol_2[x]) then
     newSol[x] := Sol_1[x]
elseif (Ratio[x] < median) then
     newSol[x] := 1
else newSol[x] := 0
```

where $x$ is the position of the bit being evaluated with $x = \{1...maxColumns\}$, and the *median* is the median of the vector of ratios for a given SCP instance.

### 4.4 GA-ACO Design

For this implementation, the same structure of GA components presented at 4.2 is used. Added to this, the implementation of ACO for SCP is very straightforward. The components used by ACO are:

*Pheromone.* Denoted by $t = \tau_i$, the matrix of pheromones will be used to consolidate information obtained by each ant, i.e. the amount of pheromone stored in each column $i$. $\tau_i(t)$ specifies the intensity of the pheromone at column $i$ in time $t$, and updates in a local way (according to the path of the ants), and in a global way (the pheromone evaporates in every matrix column). The matrix is initialized with a value $t_0$ which will be $10^{-6}$ for this implementation.

*Transition.* For apply a transition between two columns, this should be done according to the list of candidates. The list of candidates of a column contains the $c$ more attracting columns, and they are sorted in a sequential manner. The transition is carried according to:

If exists at least one column $j \in$ candidate list, then, choose the next column $j \in J_i^k$, between the $c$ columns in the candidate list according:

$$j = \begin{cases} max_u \in J_i^k[\tau_u(t)][\eta_u]^\beta & \text{if } q \leq q_0 \\ J & \text{if } q > q_0 \end{cases}$$

where $\eta_u$ represents the heuristic information, and $J$ is chosen with the probability:

$$p_j^k(t) = \frac{[\tau_j(t)][\eta_j]^\beta}{\sum_{l \in N^k}[\tau_l(t)][\eta_l]^\beta}$$

where $N^k$ is a possible neighbor of ant $k$.

After each selection of a column $j$, occurs a local modification of the level of pheromone of that column, given by the equation:

$$\tau_j(t) = (1 - \rho)\tau_j(t) + \rho\tau_0$$

This evaporation is done so the visited column will not be interesting for the following ants, stimulating in this way the exploration of solutions. At the equation above, $\rho\tau_0$ is a stabilization factor of the pheromone modification, used with the intention of not having less attractive columns so quickly, allowing the exploration of a number even bigger of new solutions. When every ant ended an iteration (a solution has been found), the pheromone level of the best solution found is updated globally, loading pheromone to each column of the solution according to:

$$\tau_j(t) = (1 - \rho)\tau_j(t) + \rho\triangle\tau_j(t)$$

where $\triangle\tau_j(t)$ is the variation of pheromone left on column $j$ by the best ant. This variation is calculated as the frequency of column $j$ in the routing of each ant, i.e. the number of times where column $j$ is in the solutions found by the ants. To operate, the following parameters controlled by the GA will be defined: *Number of Ants*, $\rho$ (the evaporation factor), $\beta$ (the importance in the choice of the next column, *Length of the List* (which will be used to limit the number of columns that can be visited next), $q_0$ (parameter which indicates if the exploration is supported at the moment of next column election) and *MaxIter* (max number of iterations).

### 4.5 Penalizing function

An aid to the Parameter setting function is needed so the Genetic Algorithm can handle a trade-off between keep improving a solution — ergo more resources applied — or to quit a solution and try another search. A penalizing function is a very good solution to this problem. The direct comparison is between a Parameter Tuning version and a Parameter Control one. As a penalizing function we propose:

$$fitness(sol) = ObjFuncValue(sol) + TimeTaken(sol) * FCT$$

where *sol* is a solution and $ObjFuncValue$ is the value — or cost — of the solution generated evaluated at the SCP Objective Function of the corresponding benchmark. Also, $TimeTaken$ is the amount of time taken to generate that solution. FCT is a correction factor which makes it possible to compare the time with the cost.

### 4.6 Parameter Control Considerations

Designs introduced in sections 4.1 and 4.4 were directly coded into the Parameter Tuning versions. To obtain the counterparts to the Parameter Control versions, some changes were introduced between transitions of the metaheuristics. GA was modified to enabling an intermediate results memory taking feedback of the performance of the lower level metaheuristic.

## 5 Experimental Results

### 5.1 Performance Measurement

The performance measures evaluated will are:

- Best Solution Found (value at SCP's Objective Function)
- Time Taken to find the Best Solution (best represented by "Calls to Objective Function made before best solution is found")
- Average Fitness (penalized) and its Standard Deviation

The quality of the solutions is taken measuring its closeness to the Optimal value of a certain instance (OR-Library SCP file). The average fitness is also an aid to evaluating quality in combination with the Standard Deviation. The computational effort is measured by: the number of calls to objective function needed to generate the best solution.

### 5.2 Reference Benchmarks

Each implementation — i.e. GA-SS and GA-ACO — are tested using the benchmarks provided by [17] which are widely used by Operation and Optimization Researchers. The files used are Set Covering Problem instances denoted by series SCP4x, SCP5x and SCP6x. To obtain more experimental results — and to compare the performance of each implementation — such benchmarks will be tested against both versions of parameter setting configurations. The benchmarks presented are minimization problems, where the idea is to find the lowest cost at the Objective Function. Each benchmark is based on a matrix of 200 rows and 1000 columns, which makes them a fairly large set of problems.

### 5.3 Environment

All the algorithms required for testing will be coded under standard C language and compiled with GNU GCC. The Xcode 3.2 IDE was selected for coding and depuration task, and as reference computer an Intel Core 2 Duo CPU with 4Gb of DDR3 RAM was used. To notice, testing implementation code does not allow the use of multiples cores as is the case of current CPU.

### 5.4 Robustness based on GA's parameters

The first table of this section shows the results for GA-SS with Parameter Tuning. These results are important since accomplish two objectives: to allow to evaluate the measures at this offline parameter setting configuration and to represent the execution of several single SS metaheuristic executions. As was explained before, a Parameter Tuning configuration runs a whole instance of SS metaheuristic with parameters defined — by GA — before its execution. When the SS execution finish, the best result is given as feedback to GA, which might continue its process of experimenting with other SS parameters. Hence, the results obtained consider a neutral parameter configuration — a random one — representing a human trial-and-error testing, where best results might be as good as the ones with GA-SS, but surely requiring more Human work in terms of defining those parameters and to experiment with them.

| PXOVER | PMUT | BestFound | CallsOF | Average | Fit-StDev |
|---|---|---|---|---|---|
| 0.25 | 0.25 | 1009 | 5778578 | 3017 | 433 |
| 0.5 | 0.5 | 1009 | 4698886 | 2915 | 675 |
| 0.75 | 0.75 | 1007 | 5750974 | 3104 | 571 |
| 0.25 | 0.75 | 1008 | 4545205 | 2803 | 458 |
| 0.75 | 0.25 | 1009 | 4698886 | 2644 | 578 |

Table 1: Different GA-SS parameters configurations using Parameter Tuning. Benchmark: SCP41.

First results (Table 1) show a considerable variability. The overall look represent what a human can expect when experimenting — no good luck considerations. As will be seen in the next tables, *PXOVER* and *PMUT* tend to lose its impact, being this instance the most unexpected.

| PXOVER | PMUT | BestFound | CallsOF | Average | Fit-StDev |
|---|---|---|---|---|---|
| 0.25 | 0.25 | 509 | 3446206 | 2235 | 1075 |
| 0.5 | 0.5 | 509 | 3462702 | 2329 | 1078 |
| 0.75 | 0.75 | 509 | 3481599 | 2002 | 954 |
| 0.25 | 0.75 | 509 | 3502315 | 1735 | 797 |
| 0.75 | 0.25 | 509 | 3308283 | 1980 | 890 |

Table 2: Different GA-SS parameters configurations using Parameter Control. Benchmark: SCP41.

At the second table (Table 2) best results found show no variation, mainly due to SS Improvement Function. The other measures show a low standard deviation specially in the overall calls to Objective Function. As the average solution results are the ones with more variation, there is logic in thinking that there was a good level of exploration between solution spaces.

In the next table (Table 3) the performance with ACO is evaluated, using Parameter Tuning. Aside of the better results obtained with ACO — and its best resources management — the same fashion as in GA-SS is seen: very low variation at each measure.

| PXOVER | PMUT | BestFound | CallsOF | Average | Fit-StDev |
|---|---|---|---|---|---|
| 0.25 | 0.25 | 449 | 1477894 | 781 | 236 |
| 0.5 | 0.5 | 434 | 1095816 | 768 | 240 |
| 0.75 | 0.75 | 449 | 1659306 | 896 | 274 |
| 0.25 | 0.75 | 449 | 1599972 | 862 | 260 |
| 0.75 | 0.25 | 449 | 1435961 | 762 | 220 |

Table 3: Different GA-ACO parameters configurations using Parameter Tuning. Benchmark: SCP41.

Using an online parameter setting criteria (Table 4) shows no more variation at the results; actually, a very low variation is obtained. Is clearer that the results under this configuration are the best of all. This will be reviewed in following experiments show down.

| PXOVER | PMUT | BestFound | CallsOF | Average | Fit-StDev |
|---|---|---|---|---|---|
| 0.25 | 0.25 | 436 | 673804 | 442 | 7 |
| 0.5 | 0.5 | 436 | 691756 | 442 | 8 |
| 0.75 | 0.75 | 436 | 743647 | 444 | 8 |
| 0.25 | 0.75 | 436 | 671342 | 444 | 8 |
| 0.75 | 0.25 | 436 | 703318 | 441 | 7 |

Table 4: Different GA-ACO parameters configurations using Parameter Control. Benchmark: SCP41.

### 5.5 Convergence to the best solution

This experiments were selected between the most interesting results: which presented a noticeable convergence through time. At first look, the temptation is to compare the obvious; GA-ACO performed better than its counterparts. After a careful review, the comparison is unfair fundamentally because ACO uses a more "intelligent" way of building solutions whereby SS is more blind. A preliminary conclusion is that the constructive process of ACO is better than the evolutive of SS. Anyway, the purpose of this graphics is to allow to observe how each version converges through time to a better solution.

The plot is based on benchmark SCP48 and the clearest idea is that Parameter Control allows to get better results in lesser time, which is mainly due to the intensification of the harvest of certain solution space where "best looking" results are found, and to the saved time when quitting to explore "bad looking" results. Also, a common item in each plot is that GA-SS (Tuning) tent to obtain a better result than GA-SS (Control) when a great quantity of solutions are searched.

### 5.6 Wide comparison of best results found

Following table introduces the best results found through the various instances executed. Faster convergence of Parameter Control versions has been considered in previous section — which is not represented in this table. Same results obtained by both versions of ACO hide evidently the efficiency factor. Every instance executed to obtain this results was using $PXOVER =$

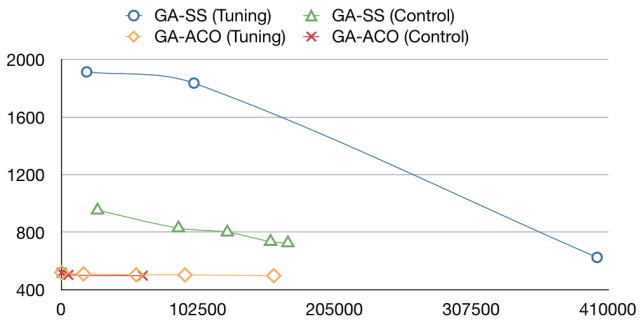


Figure 2: Convergence analysis to a better solution. Benchmark: SCP48

0.5 and $PMUT = 0.5$ which seems a fair tradeoff between crossover and mutation at GA. Under this circumstances, another criteria of robustness is handled: to perform well in a variety of instances. The worse performer through all experiments was GA-SS with Parameter Tuning, but, even with this issue, it was not an extremely bad perform.

| | SCP41 | SCP42 | SCP48 | SCP61 | SCP62 | SCP63 |
|---|---|---|---|---|---|---|
| Optimum | 429 | 512 | 492 | 138 | 146 | 145 |
| GA-SS-Offline | 1007 | 981 | 561 | 154 | 155 | 166 |
| GA-SS-Online | 509 | 603 | 642 | 164 | 165 | 176 |
| GA-ACS-Offline | 434 | 529 | 497 | 142 | 154 | 148 |
| GA-ACS-Online | 434 | 529 | 497 | 142 | 154 | 148 |

Table 5: Table of best results.

## 6 Conclusions

A 2-level metaheuristic has been tested on different SCP benchmarks showing to be very effective. One of the main goals of a multilevel approach is to provide an unattended solving method, for quickly producing solutions of a good quality for different instances. The overall work was extended on several relevant issues and quality measures: quality of solutions, robustness in terms of the instances and insensitivity to small deviation at parameters (GA parameters) solving large-scale problems, easiness of implementation, easiness to combine with other algorithms, automatic setting of parameters. All of them provided a fair approach to the core problem, and many of them have already shown lights of been covered, producing a renovated energy for keep working and admiring the synergy produced. Benchmarks have shown interesting results in terms of robustness of every approach being the Parameter Control the most robust in terms of a good performance in several instances using same parameters and in terms of stable results when small deviations are made to parameters. Also, Parameter Control shown to converge faster to better results than Parameter Tuning, but, when long running times are taken, both seem to obtain equal results. Considering the particular performance of a Constructive approach and an Evolutive one for the SCP, it seems to be more effective to "construct" intelligently a solution rather than "blindly" get one and then evolve it. An interesting fact is that SS with Parameter Control might get a performance closer to ACO. The overall implementation of a multilevel metaheuristic is pretty straightforward, no big obstacles were found. Also, the implementation of the GA using real numbers was a direct representation of what parameters a human user might

choose to operate, therefore achieving an implementation which accomplished our requirements.

## Bibliography

[1] B. Crawford, C. Lagos, C. Castro, F. Paredes, A Evolutionary Approach to Solve Set Covering, *ICEIS 2007 - Proceedings of the Ninth International Conference on Enterprise Information Systems, Volume AIDSS, Funchal, Madeira, Portugal, June 12-16, 2007 (2)*, pp.356-363, 2007

[2] U. Aickelin, An Indirect Genetic Algorithm for Set Covering Problems, *Journal of the Operational Research Society*, Vol.53, pp.1118-1126, 2002

[3] F. Tangour, P. Borne, Presentation of Some Metaheuristics for the Optimization of Complex Systems, *Studies in Informatics and Control*, Vol.17, No.2, pp.169-180, 2008

[4] C-M. Pintea, D. Dumitrescu, The importance of parameters in Ant Systems, INT J COMPUT COMMUN, ISSN 1841-9836, 1(S):376-380, 2006

[5] R. Martí, M. Laguna, Scatter Search: Diseño Básico y Estrategias, *Revista Iberoamericana de Inteligencia*, Vol.19, pp.123-130, 2003

[6] D. Gouwanda, S. G. Ponnambalam, Evolutionary Search Techniques to Solve Set Covering Problems, *World Academy of Science, Engineering and Technology*, Vol.39, pp.20-25, 2008

[7] A. Caprara, M. Fischetti, P. Toth, Algorithms for the Set Covering Problem, *Annals of Operations Research*, Vol.98, 1998

[8] J. E. Beasley, K. Jornsten, Enhancing an algorithm for set covering problems, *European Journal of Operational Research*, Vol.58, pp.293-300, 1992

[9] C. Cotta, M. Sevaux, K. Sörensen, *Adaptive and Multilevel Metaheuristics*, Springer, 2008

[10] Z. Michalewicz, *Genetic algorithms + data structures = evolution programs*, Springer, 1996.

[11] F. Glover, G. A. Kochenberger, *Handbook of metaheuristics*, Springer, 2003

[12] B. Crawford, C. Castro, Integrating Lookahead and Post Processing Procedures with ACO for Solving Set Partitioning and Covering Problems, *Proceedings of ICAISC*, pp.1082-1090, 2006

[13] Y. Hamadi, E. Monfroy, F. Saubion, What is Autonomous Search?, *Technical Report MSR-TR-2008-80*, 2008

[14] L. Lessing, I. Dumitrescu, T. Stützle, A Comparison Between ACO Algorithms for the Set Covering Problem, it Proceedings of ANTS, pp.1-12, 2004

[15] E. Talbi, *Metaheuristics: From Design to Implementation*, Wiley Publishing, 2009

[16] R. Battiti, M. Brunato, F. Mascia, *Reactive Search and Intelligent Optimization*, Springer Verlag, 2008

[17] J. E. Beasley, *OR Library, http://people.brunel.ac.uk/ mastjjb/jeb/info.html*