An Ant Colony based Hyper-Heuristic Approach for the Set Covering Problem

Alexandre Silvestre FERREIRA[a*]; Aurora Trinidad R. POZO[a]; Richard Aderbal GONÇALVES[b]

[a] *Federal University of Paraná, Department of Computer Science, Curitiba, Brazil*
[b] *State University of Centro-Oeste, Department of Computer Science, Guarapuava, Brazil.*
[*] *asferreira@inf.ufpr.br*

KEYWORD

ant colony; hyper-heuristic; set covering problem; optimization

ABSTRACT

The Set Covering Problem (SCP) is a NP-hard combinatorial optimization problem that is challenging for meta-heuristic algorithms. In the optimization literature, several approaches using meta-heuristics have been developed to tackle the SCP and the quality of the results provided by these approaches highly depends on customized operators that demands high effort from researchers and practitioners. In order to alleviate the complexity of designing meta-heuristics, a methodology called hyper-heuristic has emerged as a possible solution. A hyper-heuristic is capable of dynamically selecting simple low-level heuristics accordingly to their performance, alleviating the design complexity of the problem solver and obtaining satisfactory results at the same time. In a previous study, we proposed a hyper-heuristic approach based on Ant Colony Optimization (ACO-HH) for solving the SCP. This paper extends our previous efforts, presenting better results and a deeper analysis of ACO-HH parameters and behavior, specially about the selection of low-level heuristics. The paper also presents a comparison with an ACO meta-heuristic customized for the SCP.

1. Introduction

The Set Covering Problem (SCP) is an important NP-hard combinatorial optimization problem (Cormen et al., 2009) that can be used to model several other problems such as: scheduling, manufacturing, service planning, information retrieval and delivery and routing problem (Lan et al., 2007). The SCP is described by a binary matrix *A* with *m x n* dimensions where each element can assume binary values and each column has an associated cost c_j. It is said that a column *j* covers a line *i* if the associated matrix element $a_{i,j} = 1$. The goal is to choose a subset of columns that simultaneously covers all lines and minimizes the total sum cost (Oliveira et al., 1999).

Mathematically, the SCP can be formulated as follows:

$$min = \sum_{i=1}^{n} c_j \cdot x_j \tag{1}$$

subject to:

$$\sum_{i=1}^{m} a_{ij} \cdot x_j \geq 1, j = 1..n \tag{2}$$

$$x_j \in 0, 1, j = 1..n \tag{3}$$

where $x_j = 1$ means that column *j* is in the solution. The restriction of Equation 2 defines that each line has to be covered by at least one column and Equation 3 defines that all variables are binary.

Several exact (Fisher and Kedia, 1990; Balas and Carrera, 1996; Caprara et al., 2000) and meta-heuristic (Jacobs and Brusco, 1995; Beasley and Chu, 1996; Lan et al., 2007; Ren et al., 2008; Mulati and Constantino, 2011) methods have been proposed to solve the SCP. Exact methods are recommended only for small instances due to their time complexities. Thus, meta-heuristics have emerged as the main focus for solving this problem. Despite their good results, meta-heuristics for the SCP are usually very complex to design (due to customized operators), imposing huge drawbacks for its practical use. Furthermore, the best meta-heuristics proposed for the SCP have to be greatly modified to solve similar problems (other scheduling or manufacturing problems, for example).

The design of simple and flexible approaches avoiding such drawbacks is the main goal of hyper-heuristics. Hyper-heuristics are commonly defined as heuristics to choose heuristics (Burke et al., 2013). They are capable of collecting and exploring the information about simple low-level heuristics and selecting the most appropriate one during its execution. A feature of hyper-heuristics is that they operate on the search space of (low-level) heuristics, rather than on the search space of solutions. So, the problem of designing a complex and efficient heuristic for a problem is transformed into the problem of choosing a good combination of simple heuristics, greatly alleviating the design of a solver.

In a previous study (Ferreira et al., 2014), we proposed a hyper-heuristic approach based on Ant Colony Optimization (ACO-HH) for solving the SCP, the approach presented promising results. This paper extends our previous work by investigating the hypothesis that the performance of ACO-HH can be further improved by an appropriate adjustment of its parameters. To investigate this hypothesis, firstly we conducted a study on the impact of ACO-HH parameters. Next, we compared the performance of ACO-HH with results reported in the literature by the Ant-Line algorithm (Mulati and Constantino, 2011). Results indicates that ACO-HH has a good compromise between design and time complexity versus quality results. It also indicates that the use of hyper-heuristic is a promising area, particularly for the SCP.

The remaining of this paper is organized as follows: the next section presents an overview of the researches on hyper-heuristics. Some background concepts are described in Section 3. Section 4 presents the ACO-HH algorithm. Experimental studies to calibrate the parameters of ACO-HH, as well as an empirical comparison with the Ant-Line are reported in Section 5. Finally, Section 6 presents the conclusions.

2. Hyper-Heuristics

Hyper-heuristics were proposed to simplify and generalize the design of meta-heuristics. They can be defined as heuristics for selecting or generating heuristics (Burke et al., 2010). A hyper-heuristic works with two types of heuristics: high-level and low-level ones. High-level heuristics are responsible for choosing which low-level heuristics to apply and what solutions to be accepted in the population (possibly composed of only

one individual) and are, theoretically, independent of the problem. On the other hand, low-level heuristics are related to the problem being solved as they modify or generate solutions. High-level heuristics can be meta-heuristics and such is the case in this work, where an ACO is used as a high-level heuristic. Low-level heuristics are composed of local search and perturbative operators.

Burke et al (Burke et al., 2010) introduced a classification scheme that divides hyper-heuristics into two categories: selection hyper-heuristics and generation hyper-heuristics. In the first category, the main idea is to select at each step, from a pool of low-level simple heuristics, a good heuristic to be used by the algorithm based on the historical information accumulated about each heuristic. In the second approach, a high-level heuristic to build low-level heuristics from a pool of existing components is used.

Furthermore, based on the learning paradigm employed, hyper-heuristics can be classified as on-line or off-line. In on-line hyper-heuristics, the learning process occurs during the optimization of the problem while in off-line hyper-heuristics a set of benchmark instances is used for the learning process, which generally results in an algorithm able to tackle different instances of the same problem. This paper focuses on on-line selection hyper-heuristics.

Briefly, a hyper-heuristic operates as follows: the high-level heuristic selects the low-level heuristics based on some measurement of the performance (e. g. solution improvement). Then, the selected low-level heuristic is applied to the solution and the resultant performance is returned to the hyper-heuristic that uses this result in an acceptance mechanism to decide whether to accept or not the new solution in the population.

Cowling (Cowling et al., 2001) proposed that a hyper-heuristic framework should have two dimensions:

1. The Hyper-heuristic: a high-level heuristic that evaluates and selects other heuristics and other functions that are independent of the problem;

2. Problem Domain: is composed by the low-level heuristics, the evaluation function and any other structure or operator, which needs information about the problem.

Between those dimensions, there is a conceptual barrier called 'Domain Barrier' to block the exchange of unnecessary information. Thus, the high-level heuristic (the core a hyper-heuristic) only has knowledge about the number of heuristics and their performance values. Hence hyper-heuristics can be employed in different problem domains without modifications, resulting in flexible approaches (Burke et al., 2013).

Overall, hyper-heuristics approaches have shown good results to solve optimizations problems and the study about hyper-heuristics is an emergent field for research (Burke et al., 2013). The following subsection briefly describes some studies on the area of selection hyper-heuristics with perturbative heuristics (the main focus of this work). A more in depth review of hyper-heuristic can be found in (Burke et al., 2013).

2.1 Hyper-heuristics to select perturbative heuristics

A perturbative heuristic modifies a complete solution generating another complete solution, so, instead of what happens in constructive heuristics, there is no partial solutions involved. Perturbative selection hyper-heuristics aim to automatically select one perturbative heuristic from a set of perturbative heuristics available and apply it to an existing candidate solution. Initial studies on this field considered only one solution and focused on the selection mechanism. The two best-known selection mechanisms are the Choice Function (Cowling et al., 2001) and Reinforcement Learning variants (Nareyek, 2004).

Recently, the interest on population-based hyper-heuristics have grown. Cowling et al (Cowling et al., 2002) proposed an indirect genetic algorithm for solving a personnel scheduling problem. The approach can be described as a hyper-heuristic that uses a GA as the heuristic selection mechanism (high-level heuristic). This study was extended on (Han and Kendall, 2003) by including adaptive length chromosomes and guided operators, producing promising results on the trainer scheduling problem when compared to both a direct encoding genetic algorithm and a memetic algorithm. ACO was used as a hyper-heuristic in (Burke et al., 2005) and (Chen et al., 2007) to address a personnel scheduling and a sports timetabling problem, respectively, with promising results. Similarly, Ren et al (Ren et al., 2010) discussed an ant-based hyper-heuristic for solving the p-median problem.

Moreover, different frameworks have been proposed to be used by multi-point-based search methods, for example, the AMALGAM approach for continuous multi-objective optimization proposed by Vrugt and Robinson (Vrugt and Robinson, 2007) that selects a set of population-based multi-objective algorithms. In this approach, the number of new solutions produced by each low-level heuristic (operator) is proportional to the percentage of previously created individuals that remains in the working population at each stage.

Grobler et al (Grobler et al., 2012) investigated the use of local search strategies to improve the performance of a hyper-heuristic algorithm. The authors mixed a set of meta-heuristics including a genetic algorithm, particle swarm optimization variants, CMA-ES and differential evolution variants under a hyper-heuristic framework with good results.

Biazzini et al (Biazzini et al., 2009) combined several algorithms for numerical optimization such as differential evolution and random search in a distributed framework based on the island model. Meignan et al (Meignan et al., 2010) presented a self-adaptive and distributed approach based on agents and hyper-heuristics. Several agents concurrently explored the search space using a set of operators. The approach was applied to the vehicle routing problem.

One of the main purposes of the hyper-heuristics is to make possible the creation of heuristics that can be applied on different domains with minimal modifications (preferably, none). In order to promote this purpose, in (Ochoa et al., 2012) the authors developed a framework to compare high-level hyper-heuristics that control a set of problem specific low-level heuristics. The set of low-level heuristics is different for each problem domain. This framework was used for the CheSCH 2011 challenge where different hyper-heuristics were compared. One hyper-heuristic that showed good results on the competition is the one proposed by Nunez and Ceballos (Núnez and Ceballos, 2011), which is based on Ant Colony Optimization.

Based on the work of Nunez and Ceballos, we proposed a hyper-heuristic called ACO-HH for the SCP (Ferreira et al., 2014). This problem, as far as we known, was not tackled by a hyper-heuristic approach before. So, in this work, we extend our previous work with empirical studies to better understand the influence of the different parameters on ACO-HH performance.

3. Ant Colony Optimization

Ant Colony Optimization (ACO) is an example of artificial swarm intelligence meta-heuristic, which is inspired by the collective behavior of social insects (Dorigo and Socha, 2006). ACO is mainly based on the concepts presented by the two bridge experiment conduced in (Deneubourg et al., 1990), that can be seen at Figure 1. In this experiment, ants leave their nest without information about the location of food sources,

move randomly at the start of the experiment, and deposit a substance called pheromone on the ground. Because ants probably prefer to follow a stronger pheromone trail, the bridge that has more pheromone will attract more ants. Then, the pheromone trail will grow until the colony of ants converges toward the use of the same bridge (shorter one). Therefore, the behavior that emerges is a group of relatively "not intelligent" ants that interact through simple rules and dynamically self-organize maintaining their positions around the shortest trails. So, the main idea is the cooperation between the agents through the environment using pheromone.

In the ACO algorithm, artificial ants simulate the pheromone trail following behavior of real ants to find the shortest route between a food source and their nest. The pheromone marks a trail, representing a solution for a problem, which will be positively increased to become more attractive in subsequent iterations. Therefore, the pheromone concentration indicates how useful a solution was, serving as a history of the best ants previous movements. Besides the pheromone concentration, ants can use heuristic functions with useful local information about the problem domain.

ACO is a constructive procedure where at each step the ant will select one component to be part of its solution using a probability that is calculated using the pheromone information (cooperation) and the heuristic information; At the end of each iteration the pheromone values are updated according to the quality of the solutions that were built. Two mechanisms are used to update the pheromone: (1) the evaporation which avoid the repetition of bad choices and (2) the pheromone deposit used to intensify (reinforce) the good choices. At the end of the algorithm, the best solution found is returned.

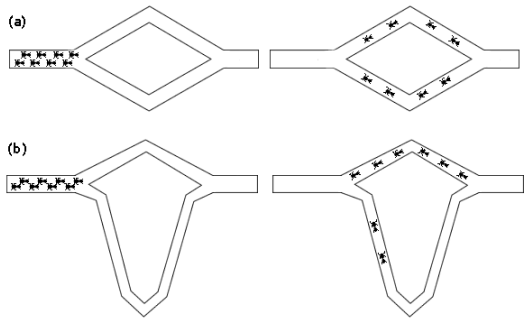

Figure 1: (a) Both paths have the same distance the ants choose them equally. (b) One path has a bigger distance than other, most of ants choose the shortest path.

4. ACO-HH

ACO-HH hyper-heuristic was proposed in (Núnez and Ceballos, 2011) for the CheSCH 2011 competition, and can be classified as an on-line learning selection hyper-heuristic. ACO-HH follows the general guidelines from the ACO meta-heuristic, but instead of generating solutions, ACO-HH creates a path of low-level heuristics, i.e., at each step of the iteration a low-level heuristic is selected accordingly to its accumulated pheromone.

In the ACO-HH approach, each ant has two structures: the problem solution *s* and the selection order

of heuristics *A*. Given an initial solution *s*, at each construction step, each ant chooses one of the available heuristics to compose their heuristic order *A* and apply it to its current solution. If the generated solution is the best found by an ant in the current iteration, it is set as the initial solution for the ant in the next iteration. Figure 2 presents an example of two ants building their paths.

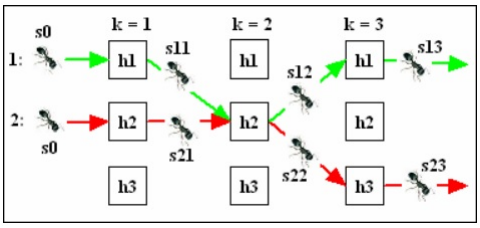

Figure 2: The ants build their paths by choosing heuristics and applying them to its solution (Núnez and Ceballos, 2011).

The probability that an ant *j* chooses a heuristic *h* at the construction step *k* is given by Equation 4.

$$p_j^k(h) = \frac{\tau_{kh}^{\alpha} \cdot \eta_{kh}^{\beta}}{\sum_{l=1}^{H} \tau_{kl}^{\alpha} \cdot \eta_{kl}^{\beta}} \tag{4}$$

Where *H* is the number of low-level heuristics, τ_{kh} is the pheromone value associated with the heuristic *h* at construction step *k* and η_{kh} is the heuristic information that is calculated by Equation 5.

$$\eta_{kh} = M_{kh} + 1 - MIN\{M_{kl} | l = 1..H\} \tag{5}$$

$$M_{kh} = \begin{cases} (\sum_{h \in s_{ij}} (f(s_{k-1}) - f(s_k))) & , \text{if } E > 0 \\ 0 & , \text{if } E = 0 \end{cases}$$

Where *E* is the number of ants that have the heuristic *h* in *A*, i.e. applied heuristic *h* in any step.

As on ACO, α and β control the influence of pheromone information and heuristic information, respectively. The heuristic information associated with a low-level heuristic is based on its average local improvement.

The pheromone update mechanism is different from the one used on the classic ACO. It does not use the evaporation and deposit mechanisms, building the pheromone matrix from the average of improvements obtained by the historical information about the generated solutions. The update mechanism is shown in Equation 6 (Núnez and Ceballos, 2011).

$$\tau_{kh} = MC_{kh} + 1 - MIN\{MC_{kl} | l = 1..H\} \tag{6}$$

$$MC_{kh} = \begin{cases} (\sum_{h \in s_{ij}} (f(s_0) - f(s_n))) & , \text{if } E > 0 \\ 0 & , \text{if } E = 0 \end{cases}$$

Where *n* is the size of *A*, $f(s_0)$ and $f(s_n)$ are the performances of the initial and final solution for an ant i at the iteration j.

Following, a description of some implementation details of ACO-HH to solve the Set Covering Problem is presented.

Algorithm 1: ACO-HH

Input: iterations, constructionSteps, antNumber, α, β, p_1, a set of heuristics H

$bestSolution \leftarrow \infty$;
foreach *ant in ants* **do**
    ant.setSolution(createInitalSolution()) ; /* using fitness proportionate selection */
    $ant.setBestIterationSolution(\infty)$;
end
for $i = 0;\ i < iterations;\ i++$ **do**
    for $k = 0;\ k < constructionSteps;\ k++$ **do**
        foreach *ant in ants* **do**
            $h \leftarrow selectHeuristic()$; /* probabilistically according to Equation 4 */
            $s' \leftarrow applyHeuristic(h, ant)$;
            $ant.addPath(h)$;
            $saveLocalImprovement(h, ant.getSolution(), s')$; /* local heuristic improvement used in heuristic information */
            $ant.setSolution(s')$;
            if $ant.getSolution() < ant.getBestIterationSolution()$ **then**
                $ant.setBestIterationSolution(ant.getSolution())$;
            end
        end
    end
    foreach *ant in ants* **do**
        if $ant.getBestIterationSolution() < bestSolution()$ **then**
            $bestSolution \leftarrow ant.getBestIterationSolution()$;
        end
        ant.setSolution(ant.getBestIterationSolution());
        $saveGlobalImprovment(ant)$; /* iteration ant improvement used in pheromone update */
    end
    foreach *h in H* **do**
        $updatePheromone(h)$; /* according to Equation 6 */
    end
end
return *bestSolution*

4.1 ACO-HH for SCP

The pseudo-code for the ACO-HH is presented at Algorithm 1.

The problem solution *s* is represented by a binary vector of size *columns* equals to the number of columns of the current problem instance; The value 1 for an index *j* of the vector means that the column *j* is part of the solution. The heuristic order *A* is represented by an integer vector of size *ant size*, this size is a parameter of the algorithm. The value h for an index *k* means that the heuristic h was selected by an ant at the construction step *k*.

Firstly, the algorithm creates initial solutions for each ant. The solutions are built using a fitness proportionate selection based on the heuristic information presented in Equation 7.

$$\eta_j = \frac{k_j}{c_j} \tag{7}$$

Where k_j is the cardinality of column *j* and c_j is its cost. The cardinality of a column *j* is calculated by

the number of lines that *j* covers that are not covered by any other column in the solution.

After the construction of the initial solutions, the main iterations of the algorithm begin. At each iteration, each ant builds its path by choosing a low-level heuristic (as show in Equation 4). The selected heuristic is applied to the solution associated with the ant. Next, the generated solution substitutes the current solution of the ant and the improvement obtained by the heuristic (difference between the new solution and the previous one) is saved.

ACO-HH for the SCP utilizes the following low-level heuristics:

1. One Bit Flip (OF): The One Bit Flip heuristic randomly selects a column *j* from the ant solution where *v[j]=1* and reverse its value to 0. The goal is to remove a redundant column from the solution.

2. Scattered Crossover (SC): Given two solutions s_1 and s_2 a random bit mask with the size of the solutions is created. If the value at position *i* of the mask equals 1 then the generated solution copies the i-th value of solution s_1 ($s'[i] = s_1[i]$), otherwise the value is copied from solution s_2 ($s'[i] = s_2[i]$).

3. Fusion Crossover (FC): The fusion crossover used in this work is defined in (Beasley and Chu, 1996) and is similar to SC but the number of ones in the mask is proportional to the relative quality between solution s_1 and s_2, i.e. the better s_1 is relative to s_2 the more ones will the mask contain.

4. JB2 (JB): The JB Local Search 2 was implemented as defined in (Mulati, 2009). This heuristic always generates a feasible solution and works as follows: given a solution the heuristic destroys a percentage of it (defined by the parameter p_1) and rebuild it using the heuristic information presented in Equation 7, but taking in account only the columns that have their cost less or equal to the max cost presented in the solution.

At the end of each iteration, the best solution found for each ant is stored. Moreover, the best solution is set as the initial solution for the next iteration. The improvement obtained in relation to the initial solution is also saved. Afterwards, the best solution found so far is possibly updated and the pheromone matrix is updated as show in Equation 6. Finally, after the maximum number of iterations is satisfied, the best solution is returned.

5. Experimental Results and Analysis

This section presents the methodology, the results obtained and the comparison between ACO-HH and the Ant-Line proposed in (Mulati and Constantino, 2011).

5.1 Methodology

The ACO-HH approach was developed using the C++ programming language and executed on an Intel Core i7 of 3.40Ghz with 6Gb of memory running the Ubuntu Linux operational system. 65 SCP instances from the OR-Lybrary were used (Beasley, 1990), these instances are presented at Table 1, where they are grouped by class (C.SCP), their quantities between parentheses after the class name, the size of the instances is shown by the (*m x n*) column of the table and the density of an instance is shown in column (De(%)).

C.SCP	m x n	De(%)
4(10)	200 x 1000	2
5(10)	200 x 2000	2
6(5)	200 x 1000	5
a(5)	300 x 3000	2
b(5)	300 x 3000	5
c(5)	400 x 4000	2
d(5)	400 x 4000	5
nre(5)	500 x 5000	10
nrf(5)	500 x 5000	20
nrg(5)	1000 x 10000	2
nrh(5)	1000 x 10000	5

Table 1: SCP Instances

For each instance 10 runs of the ACO-HH algorithm are performed, and the maximum, minimum and average values obtained are reported as percentage differences (PD) of these values in relation to the optimal. In order to statistically compare both algorithms the Friedman Rank Sum Test is performed on the average and best values. The test was executed using the R functional language with 95% confidence.

PD is calculated as shown in Equation 8 where *avg(value)* is the average between the *value* and the *optimal* and *dif(value)* is the difference between those values. The closer to 0.0 is the value of PD, the better is the result obtained.

$$pd(value) = \left(\frac{dif(value)}{avg(value)}\right) \cdot 100 \tag{8}$$

5.2 Parameters Setup

In order to find a good configuration for the ACO-HH parameters, an empirical experiment was realized starting with the parameters used in the previous paper (Ferreira et al., 2014): $iterations = 10$, $k = 30$, $antNumber = 15$, $\alpha = 1$, $\beta = 2$ and $p_1 = 0.3$, the results for this parameters can be seen in Table 2. The parameters were analyzed in order, one at a time.

During this parameters setup, five SCP instances were used, each one of a different class. The instances used were: *scp4.10*, *scpc.1*, *scpnre.1*, *scpnrf.1* and *scpnrg.1*. Again, for each instance 10 runs of the ACO-HH algorithm were performed.

Number of Iterations: This parameter defines the number of times that the ants will build their paths and the number of times that the pheromone is updated; we analyzed values for the *iterations* parameter ranging from 10 to 30. The results of this experiment are presented in Table 3. Even though comparable values were found for the smaller instances (*scp4.10* and *scpc.1*), an improvement could be noticed when the number of iteration is 30 on the best and average values for the bigger instances (like *nrf1* and *nrg.1*). So the *iterations* parameter was set to 30.

Parameter *k*: The *k* parameter defines the number of heuristics that will be selected by each ant on an iteration. The values 35 and 40 were tested for this parameter. The results of this experiment are presented

Base Parameters			
SCP	WS(%)	AS(5)	BS(%)
4.10	15.17	9.52	1.56
c.1	12.78	7.13	4.31
nre.1	9.83	6.22	0.0
nrf.1	13.33	6.22	0.0
nrg.1	10.75	7.01	3.89

Table 2: Results of the parameters used in the previous paper. WS(%) is the PD of the worst solution, AV(%) is the PD of the average of the solutions and BS(%) is the PD of the best solution found. All the PDs are in relation to the optimal solution.

i = 20			
SCP	WS(%)	AS(5)	BS(%)
4.10	13.43	7.79	4.19
c.1	11.22	7.13	3.46
nre.1	6.66	3.05	0.0
nrf.1	6.89	5.55	0.0
nrg.1	10.85	7.01	2.80
i = 30			
4.10	13.43	7.65	0.96
c.1	9.64	7.21	4.73
nre.1	6.66	2.04	0.0
nrf.1	6.89	4.19	0.0
nrg.1	11.76	6.64	2.80

Table 3: Result for parameter i.

in Table 4. The results suggest that ACO-HH has little sensitivity to the parameter k as there were little differences between the results obtained with different parameters. Thus, we set $k = 30$ based on the better results obtained by this value on our original work (Ferreira et al., 2014).

The *antNumber* parameter: this parameter defines how many solutions will be built on each iteration. The values investigated for this parameter are 20 and 25. The results obtained are presented in Table 5. As the performance didn't improve, as the number of ants was incremented, this parameter was set to 25 in order to have more solutions.

The parameter p_1: defines the percentage of a solution that will be destroyed by the JB2 heuristic. This parameter may have a great influence on the results because it is tailored to avoid local minimum. Because this heuristic rebuilds the solution using heuristic information, in general, it produces good solutions. The values used for the p_1 parameter were 0.6 and 0.9. The results of the experiment are presented in Table 6. The results showed that a high value for p_1 is detrimental to the performance of the algorithm, rebuilding too much of a solution; while a low value is not enough to escape local minima. So, this parameter was set to 0.6.

The tests for the α parameter ranges from 2 to 4 and the results obtained by the tests are presented in

k = 35			
SCP	WS(%)	AS(5)	BS(%)
4.10	12.92	7.72	4.0
c.1	15.44	7.87	4.73
nre.1	6.66	2.38	0.0
nrf.1	13.33	6.22	0.0
nrg.1	10.72	6.64	2.84
k = 40			
4.10	12.92	7.83	2.87
c.1	14.69	7.99	1.76
nre.1	6.66	1.36	0.0
nrf.1	13.33	4.87	0.0
nrg.1	12.26	7.49	4.44

Table 4: Results for parameter k.

antNumber = 20			
SCP	WS(%)	AS(%)	BS(%)
4.10	23.67	12.68	4.76
c.1	13.93	8.07	3.88
nre.1	9.83	2.72	0.0
nrf.1	6.89	2.81	0.0
nrg.1	10.24	7.33	5.52
antNumber = 25			
4.10	12.75	7.67	3.25
c.1	16.93	7.66	2.17
nre.1	9.83	6.05	0.0
nrf.1	6.89	3.50	0.0
nrg.1	10.24	6.59	2.80

Table 5: Result for antNumber.

Table 7. There was no significant difference for the smaller instances but, for larger instances, bigger values of α produced worst results. Therefore, we set α to the minimum, 1.

β parameter: For the β parameter the values 2, 4 and 6 were tested. Table 8 shows the results obtained. No significant difference were found between the choices and the value of β was set to 2, as in the original version (Ferreira et al., 2014).

Overall, the majority of the parameters with exception to p_1, has no great influence on the results. This suggests that the low-level heuristics implemented are not very effective and get stuck on local optima pretty early. After the setup tests, the final parameters values were set as follows: $iterations = 30$, $k = 30$, $antNumber = 25$, $\alpha = 1$, $\beta = 2$ and $p_1 = 0.6$.

$p_1 = 0.6$			
SCP	WS(%)	AS(%)	BS(%)
4.10	9.80	5.90	2.68
c.1	7.62	6.35	2.60
nre.1	0.0	0.0	0.0
nrf.1	0.0	0.0	0.0
nrg.1	6.06	3.89	2.80
$p_1 = 0.9$			
4.10	8.56	3.85	1.92
c.1	8.02	3.03	4.47
nre.1	0.0	0.0	0.0
nrf.1	0.0	2.96	0.98
nrg.1	10.24	2.27	2.80

Table 6: Results for p_1.

$\alpha = 2$			
SCP	WS(%)	AS(%)	BS(%)
4.10	12.2	6.91	3.62
c.1	10.04	5.19	2.17
nre.1	9.83	1.02	0.0
nrf.1	6.89	0.71	0.0
nrg.1	7.12	4.50	2.80
$\alpha = 4$			
4.10	11.02	5.61	0.96
c.1	9.64	3.10	3.03
nre.1	3.38	1.02	0.0
nrf.1	6.89	2.12	0.0
nrg.1	5.52	3.80	0.0

Table 7: Results for parameter α.

$\beta = 4$			
SCP	WS(%)	AS(%)	BS(%)
4.10	10.68	6.40	1.92
c.1	9.24	4.85	2.60
nre.1	9.83	1.70	0.0
nrf.1	13.30	3.50	0.0
nrg.1	8.17	5.95	3.89
$\beta = 6$			
4.10	9.62	6.27	1.92
c.1	13.16	5.27	1.76
nre.1	6.6	1.36	0.0
nrf.1	13.33	6.22	0.0
nrg.1	7.65	5.25	2.80

Table 8: Results for parameter β.

5.3 Analysis of the Selection of low-Level Heuristics

The main operation of a selection hyper-heuristic is the mechanism to choose which low-level heuristic should be applied at each step of the algorithm. Therefore, an understanding of the behavior of this mechanism is important to have a better comprehension of the behavior of the hyper-heuristic as a whole.

One important point to be noted is that the overall behavior of the selection procedure was similar for all instances independent of its characteristics, i.e. small or large, dense or sparse has no great influence in the selection of the low-level heuristics. This fact came as a surprise, as one of the main characteristics of an on-line learning hyper-heuristic, such as ACO-HH, is its capability to adapt to different characteristics of the instances. This finding may also be important to the comprehension of the SCP, as it gives an approximate good usage of each operator in order to solve it efficiently.

As the behavior of the selection mechanism was similar for all instances, Figure 3 shows the average usage percentage on 10 runs for each low-level heuristic in ACO-HH per instance.

Accordingly to Figure 3, the ranking of usage for the low-level heuristics was JB, OF, FC and SC, from the more used to less used heuristic. Furthermore, there is two classes of usage: JB and OF were used around 60% of the time with almost equal probability while FC and SC were used the remaining time in almost equal proportion. FC and SC are very similar heuristics operating in a pair of solutions, the main difference between them is that the latter uses an uniform mask while the former uses a mask based on the relative fitness between the two solutions under operation. JB and OF operates in a single solution and while the first uses heuristic information about the problem and modifies a great percentage of each solution, the second heuristic is very simple and modifies only a single position. From these facts it is possible to infer that ACO-HH prefers low-level heuristics that operates on single solutions but is blind to the information used but the heuristics.

Another point that was observed during the experiments was the inability of the low-level heuristics to improve the solution during the final stages of the optimization process. This was expected, as the best solutions found for this problem were found by constructive heuristics and ACO-HH only employs perturbative heuristics due to the restriction that at each stage of the algorithm a complete solution has to be

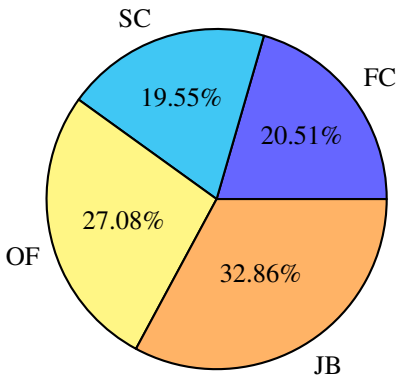

Figure 3: ACO-HH heuristic percentage selection. Fusion Crossover (FC), Scattered Crossover (SC), One Flip (OF) and JB local search (JB).

generated and evaluated in order to update the pheromone matrix. This constraint greatly limits the possible low-level heuristics employed by ACO-HH and will be the focus of future researches.

5.4 Results

The results obtained for all instances of all classes are presented in Tables 9, 10 and 11, where SCP is the instance used, AT(s) is the average time of the runs in seconds, WS(%) is the percentage difference between the worst solution found and the optimal, AS(%) is the percentage difference between the average of the solutions found and the optimal solution and BS(%) is the percentage difference between the best solution found and the optimal solution. These tables have special lines that summarize the behavior of ACO-HH through all instances of the same class.

The results for classes 4, 5 and 6 are presented in Table 9. The AS results for class 4 are around 6 while the BS are inferior to 2 in most instances with only 3 exceptions. For class 5 most AS results are less than 6 and 7 of 10 BS results are less than 2. All BS results for class 6 are inferior to 2 with the only exception of instance 6.3 (BS = 2.04) and all AS results are inferior to 5.

Table 10 shows and summarizes the results for classes a, b, c and d. ACO-HH obtained similar results for classes a and c, with average AS of more than 5 and average BS of more than 1.80. For classes b and d the results obtained by ACO-HH were very good: average BS of less than 0.60 and average AS of less than 5 in both cases.

The results for classes nre, nrf, nrg and nrh are presented in Table 11. The results for classes nre and nrf were very good with average BS of 0.00 for both classes and average AS of 2.48 and 3.96, respectively. For class nrg the average AS was 4.15 and the average BS was 2.27 while for class nrh the average AS and BS were 4.68 and 1.94, respectively.

5.5 Comparison

This subsection presents a comparison between the ACO-HH and the Ant-Line proposed in (Mulati and Constantino, 2011) using the Friedman Rank Sum Test. Table 12 presents the PD results for both algorithms,

SCP	AT(s)	WS(%)	AS(%)	BS(%)
4.1	40.30	13.47	5.64	0.92
4.2	32.63	12.98	7.41	1.74
4.3	33.49	10.65	5.68	2.48
4.4	44.50	12.16	5.97	1.00
4.5	32.20	8.59	4.19	1.93
4.6	34.53	13.22	6.77	2.99
4.7	30.21	12.84	4.83	1.15
4.8	33.39	14.33	6.92	0.40
4.9	43.66	8.37	6.37	3.82
4.10	39.39	12.57	5.92	0.00
4(10)	36.43	11.92	5.97	1.64
5.1	34.89	12.24	7.23	1.95
5.2	32.44	14.35	9.43	4.21
5.3	31.03	11.27	6.09	1.31
5.4	27.43	13.22	5.97	1.23
5.5	27.30	11.18	5.97	1.41
5.6	27.76	14.37	4.09	0.00
5.7	35.21	9.43	5.92	2.69
5.8	47.76	11.45	5.63	1.03
5.9	31.75	8.24	4.03	0.00
5.10	38.41	11.72	8.45	3.70
5(10)	33.40	11.75	6.28	1.75
6.1	27.04	10.92	4.39	1.43
6.2	21.00	5.98	3.69	1.36
6.3	22.89	12.29	4.25	2.04
6.4	22.26	7.35	3.89	1.51
6.5	25.40	9.46	4.78	1.84
6(5)	30.51	9.86	4.20	1.74

Table 9: Results for instances from class 4 to 6.

Table 13 and Table 14 present a statistical comparison in relation to the average and best values respectively, the table presents the SCP class that is being compared, the chi-square value, the p-value, the degree of freedom and if there is a statistical difference with 95% confidence.

For the majority of classes, ACO-HH was much faster than the Ant-Line while the Ant-Line obtained better results. Despite this, the results obtained for ACO-HH are promising, since there is no statistical difference for the classes *nre*, *nrf* and *nrh* in relation to the best values. It is important to note that ACO-HH is very similar to a classic ACO while the Ant-Line proposed in (Mulati and Constantino, 2011) has a good amount of modification in order to solve SCP instances. Furthermore, ACO-HH can be easily modified to solve other problems, the only change needed is the solution representation; and the use of a different set of low-level heuristics. These modifications are related only to the problem domain and isolated from the

SCP	AT(s)	WS(%)	AS(%)	BS(%)
a.1	56.20	9.41	5.23	2.34
a.2	54.52	9.81	5.92	2.73
a.3	66.08	7.06	4.30	1.70
a.4	57.51	12.45	7.24	1.27
a.5	52.55	9.67	3.45	1.26
a(5)	57.37	9.68	5.23	1.86
b.1	45.70	9.65	4.39	0.00
b.2	50.80	12.34	4.00	0.00
b.3	45.11	9.52	3.68	0.00
b.4	50.69	10.77	5.53	2.50
b.5	45.84	5.40	1.65	0.00
b(5)	47.63	9.54	3.85	0.50
c.1	78.38	9.24	5.19	0.88
c.2	72.67	8.31	4.81	1.36
c.3	86.06	11.62	6.49	4.00
c.4	69.61	10.79	6.57	1.36
c.5	73.67	10.98	5.42	1.38
c(5)	76.08	10.19	5.70	1.80
d.1	68.74	12.50	4.24	0.00
d.2	71.56	7.29	4.73	0.00
d.3	73.17	9.27	4.21	1.37
d.4	70.37	9.23	4.41	0.00
d.5	70.85	12.30	6.80	1.62
d(5)	70.94	10.12	4.88	0.60

Table 10: Results for instances from class a to d.

high-level ACO heuristic used to select the low-level heuristics. On the other hand, the Ant-Line would have to be redesigned in order to be applied to other problems. So, it is possible to infer that ACO-HH is more flexible than the other algorithm.

6. Conclusion

Hyper-heuristics is a recent field of research that proposes techniques that are more flexible and, in many cases, more simple than state-of-the-art meta-heuristics. A recent trend in the hyper-heuristics area is the use of population-based meta-heuristics as a high-level heuristic. In this paper an ACO-based hyper-heuristic, called ACO-HH, is proposed for the Set Covering Problem. To the best of our knowledge, ACO-HH is the first hyper-heuristic approach for the SCP.

ACO-HH is very similar to a standard ACO, but it searches the space of the low-level heuristics instead of the space of solutions. In this work 4 perturbative low-level heuristics are used: One Bit Flip, Scattered

SCP	AT(s)	WS(%)	AS(%)	BS(%)
nre.1	158.23	0.00	0.00	0.00
nre.2	173.52	6.66	4.24	0.00
nre.3	165.34	3.63	2.55	0.00
nre.4	162.34	6.89	3.85	0.00
nre.5	166.32	3.50	1.76	0.00
nre(5)	165.15	4.14	2.48	0.00
nrf.1	339.75	0.00	0.00	0.00
nrf.2	337.16	12.50	1.32	0.00
nrf.3	337.23	13.30	7.56	0.00
nrf.4	338.57	13.30	2.81	0.00
nrf.5	350.60	14.28	8.11	0.00
nrf(5)	340.66	10.68	3.96	0.00
nrg.1	226.55	4.98	3.13	0.56
nrg.2	253.57	6.28	4.25	2.56
nrg.3	226.55	8.09	3.55	3.55
nrg.4	281.12	9.09	4.65	1.76
nrg.5	212.26	7.44	5.16	2.93
nrg(5)	240.01	7.18	4.15	2.27
nrh.1	381.98	6.15	4.97	3.12
nrh.2	403.20	6.15	4.19	1.57
nrh.3	374.53	8.13	5.44	3.33
nrh.4	406.49	6.66	4.71	1.70
nrh.5	375.05	8.65	4.09	0.00
nrh(5)	388.25	7.15	4.68	1.94

Table 11: Results for instances from class nrfe to nrfh.

Crossover, Fusion Crossover and JB Local Search 2. The first one can be characterized as a mutation, the following two are crossovers while the last one is a local search.

An experimental analysis of the sensitivity of the parameters was conducted in order to find good parameters values. This analysis pointed out that the algorithm has a low sensitivity to the majority of its parameters, which is a good characteristic for a hyper-heuristic.

The relative usage of each low-level heuristic was also investigated. It was concluded that, for the Set Covering Problem, ACO-HH has similar behavior with respect to the choice of low-level heuristics independently from the instance being solved. Furthermore, it was possible to infer that ACO-HH preferred single point low-level heuristics (One Bit Flip and JB Local Search 2).

ACO-HH obtained good results for some instances of various classes of the SCP. When compared to Ant-Line, it was considerably faster but obtained solutions with lower quality. So, the ACO-HH is recommend for finding solutions with good quality in a short time while the Ant-Line is recommend if the quality of the solutions obtained is the only important factor. Besides this, ACO-HH is more flexible: just the low-level heuristics and the problem representation have to be modified in order to solve other problems (combinatorial

	ACO-HH				Ant-Line			
C.SCP	AT(s)	WS(%)	AS(%)	BS(%)	AT(s)	WS(%)	AS(%)	BS(%)
4(10)	36.43	11.92	5.97	1.64	2.51	0.72	0.27	0.05
5(10)	33.40	11.75	6.28	1.75	13.73	1.06	0.41	0.03
6(5)	30.51	9.86	5.13	1.74	4.68	2.37	0.73	0.00
a(5)	57.37	9.68	5.23	1.86	101.16	1.06	0.57	0.34
b(5)	47.63	9.54	3.85	0.50	35.66	0.54	0.18	0.00
c(5)	76.08	10.19	5.70	1.80	243.00	1.75	0.64	0.00
d(5)	70.94	10.12	4.88	0.60	150.62	2.48	0.94	0.00
nre(5)	165.15	4.14	2.48	0.00	2166.76	2.12	0.99	0.00
nrf(5)	340.66	10.68	3.96	0.00	3377.26	4.39	3.24	1.43
nrg(5)	240.01	7.18	4.15	2.27	8578.34	2.53	1.36	0.73
nrh(5)	388.25	7.15	4.68	1.94	8640.48	4.34	2.51	0.98

Table 12: Comparison of the results obtained by the ACO-HH with the Ant-Line proposed in (Mulati and Constantino, 2011).

SCP Class	Chi^2	**P-value**	**Df**	**Difference**
4	10	0.0015	1	TRUE
5	10	0.0015	1	TRUE
6	5	0.0025	1	TRUE
a	5	0.0025	1	TRUE
b	5	0.0025	1	TRUE
c	5	0.0025	1	TRUE
d	5	0.0025	1	TRUE
nre	3	0.0830	1	FALSE
nrf	4	0.0430	1	TRUE
nrg	5	0.0025	1	TRUE
nrh	5	0.0025	1	TRUE

Table 13: Results of the Friedman Rank Sum Test comparing the average values obtained by the two algorithms per SCP class.

or not) using the ACO-HH algorithm. So, ACO-HH can be considered a promising approach.

The paper describes in details the project and evaluation of the algorithm. Four low-level heuristics were implemented: OF, SC, FC and JB. In a first step, the study presents experiments to understand how the different parameters affect the behavior of the algorithm.

Future work includes the investigation of ACO-HH in other problem domains, the use of other meta-heuristics as high-level heuristics and the proposal of generation hyper-heuristics for the SCP.

SCP Class	Chi^2	P-value	Df	Difference
4	9	0.0027	1	TRUE
5	10	0.0015	1	TRUE
6	5	0.0025	1	TRUE
a	5	0.0025	1	TRUE
b	3	0.0830	1	FALSE
c	5	0.0025	1	TRUE
d	2	0.1600	1	FALSE
nre	2	0.1600	1	FALSE
nrf	1	0.3200	1	FALSE
nrg	5	0.0025	1	TRUE
nrh	1	0.3200	1	FALSE

Table 14: Results of the Friedman Rank Sum Test comparing the best values obtained by the two algorithms per SCP class.

7. References

Balas, E. and Carrera, M. C., 1996. A dynamic subgradient-based branch-and-bound procedure for set covering. *Operations Research*, 44(6):875–890.

Beasley, J. E., 1990. OR-Library: distributing test problems by electronic mail. *Journal of the operational research society*, pages 1069–1072.

Beasley, J. E. and Chu, P. C., 1996. A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94(2):392–404.

Biazzini, M., Banhelyi, B., Montresor, A., and Jelasity, M., 2009. Distributed hyper-heuristics for real parameter optimization. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1339–1346. ACM, New York, NY, USA. ISBN 978-1-60558-325-9. doi:10.1145/1569901.1570081.

Burke, E., Kendall, G., Silva, D. L., O'Brien, R., L, D., Silva, A., and Soubeiga, E., 2005. An Ant Algorithm Hyperheuristic for the Project Presentation Scheduling Problem. In *In: Proceedings of the Congress on Evolutionary Computation 2005 (CEC 2005). Volume 3*, pages 2263–2270. IEEE press.

Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., and Qu, R., 2013. Hyper-heuristics. *J Oper Res Soc*, 64(12):1695–1724. ISSN 0160-5682.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R., 2010. A classification of hyper-heuristic approaches. In *Handbook of metaheuristics*, pages 449–468. Springer.

Caprara, A., Toth, P., and Fischetti, M., 2000. Algorithms for the set covering problem. *Annals of Operations Research*, 98(1-4):353–371.

Chen, P.-C., Kendall, G., and Berghe, G., 2007. An Ant Based Hyper-heuristic for the Travelling Tournament Problem. In *Computational Intelligence in Scheduling, 2007. SCIS '07. IEEE Symposium on*, pages 19–26. doi:10.1109/SCIS.2007.367665.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C., 2009. *Introduction to algorithms*. MIT press.

Cowling, P., Kendall, G., and Han, L., 2002. an investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In *Proceedings of the Congress on Evolutionary Computation 2002, CEC*

2002, pages 1185–1190.

Cowling, P., Kendall, G., and Soubeiga, E., 2001. A hyperheuristic approach to scheduling a sales summit. In *Practice and Theory of Automated Timetabling III*, pages 176–190. Springer.

Deneubourg, J.-L., Aron, S., Goss, S., and Pasteels, J. M., 1990. The self-organizing exploratory pattern of the argentine ant. *Journal of insect behavior*, 3(2):159–168.

Dorigo, M. and Socha, 2006. An Introduction to Ant Colony Optimization. *Technical Report*, No. TR/IRIDIA/2006-010.

Ferreira, A. S., Pozo, A. T., and Gonçalves, R. A., 2014. Aplicação do Algoritmo ACO-HH para o problema de cobertura de conjuntos. In *Encontro Nacional de Inteligência Artificial e Computacional - ENIAC*, pages 342–346. Sociedade Brasileira de Computação - SBC. In Portuguese.

Fisher, M. L. and Kedia, P., 1990. Optimal solution of set covering/partitioning problems using dual heuristics. *Management science*, 36(6):674–688.

Grobler, J., Engelbrecht, A., Kendall, G., and Yadavalli, V., 2012. Investigating the use of local search for improving meta-hyper-heuristic performance. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–8. doi:10.1109/CEC.2012.6252970.

Han, L. and Kendall, G., 2003. Guided Operators for a Hyper-Heuristic Genetic Algorithm. In *Proceedings of AI-2003: Advances in Artificial Intelligence. The 16th Australian Conference on Artificial Intelligence*, pages 807–820.

Jacobs, L. W. and Brusco, M. J., 1995. Note: A local-search heuristic for large set-covering problems. *Naval Research Logistics (NRL)*, 42(7):1129–1140.

Lan, G., DePuy, G. W., and Whitehouse, G. E., 2007. An effective and simple heuristic for the set covering problem. *European journal of operational research*, 176(3):1387–1403.

Meignan, D., Koukam, A., and CrÃl'put, J.-C., 2010. Coalition-based metaheuristic: a self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics*, 16(6):859–879. ISSN 1381-1231. doi:10.1007/s10732-009-9121-7.

Mulati, M. H., 2009. *Investigação da Meta-Heurística de Otimização por Colônia de Formigas Artificiais Aplicada ao Problema de Cobertura de Conjunto*. Master's thesis, Dissertação de mestrado, Universidade Estadual de Maringá, Departamento de Informática.

Mulati, M. H. and Constantino, A. A., 2011. Ant-Line: A Line-Oriented ACO Algorithm for the Set Covering Problem. In *Proceedings of the 2011 30th International Conference of the Chilean Computer Science Society*, pages 265–274. IEEE Computer Society.

Nareyek, A., 2004. chapter Choosing Search Heuristics by Non-stationary Reinforcement Learning, pages 523–544. Kluwer Academic Publishers, Norwell, MA, USA. ISBN 1-4020-7653-3.

Núnez, J. L. and Ceballos, A., 2011. A general purpose Hyper-Heuristic based on Ant colony optimization.

Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J., Walker, J., Gendreau, M., Kendall, G., McCollum, B., Parkes, A., Petrovic, S., and Burke, E., 2012. *HyFlex: A Benchmark Framework for Cross-domain Heuristic Search*, volume 7245 of *LNCS*, pages 136–147. Springer, Heidelberg.

Oliveira, N. V. d. et al., 1999. Problema de cobertura de conjuntos: uma comparação numérica de algoritmos heurísticos.

Ren, Z., Feng, Z., Ke, L., and Chang, H., 2008. A fast and efficient ant colony optimization approach for the set covering problem. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 1839–1844. IEEE.

Ren, Z., Jiang, H., Xuan, J., and Luo, Z., 2010. Ant Based Hyper Heuristics with Space Reduction: A Case Study of the P-median Problem. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature: Part I*, PPSN'10, pages 546–555. Springer-Verlag, Berlin, Heidelberg. ISBN 3-642-15843-9, 978-3-642-15843-8.

Vrugt, J. A. and Robinson, B. A., 2007. Improved evolutionary optimization from genetically adaptive multimethod search. *Proceedings of the National Academy of Sciences*, 104(3):708–711. doi:10.1073/pnas.0610471104.