Neighbourhood-Based Metaheuristics for the Set Covering Problem

1st Rafael Marques Claro
Doctoral Program in Electrical and Computers Engineering
Faculty of Engineering, University of Porto
up201504487@fe.up.pt

***Abstract*—The Set Covering Problem (SCP) is a well known optimization problem that belongs to the NP-Hard category. This problem consists in selecting subsets until all the attributes are covered, always looking for the solution with the minimum cost. This paper focuses on applying different heuristics for this problem. The adopted approach includes Constructive Heuristics (CH) and a redundancy elimination procedure (RE). To further improve the quality of the obtained solutions, two local search algorithms were implemented, based on First Improvement (FI) and on Best Improvement (BI). To escape local optima, the Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic was adopted. The effectiveness and efficiency of the algorithm were tested on a set of instances, comparing the obtained results with the best known solutions. The obtained average deviation was 0.7% for a hybrid GRASP approach, with 500 iterations. It took 228.6 seconds per instance to reach the final solution.**

***Index Terms*—SCP, Heuristics, Metaheuristics, Local Search, GRASP.**

I. Introduction

The Set Covering Problem (SCP) is a well known mathematical problem that represents the basis for various applications, such as scheduling, manufacturing, service planning, routing and resource allocation [1]. The SCP tries to cover all the n attributes recurring to a list of m subsets. Each subset fulfils a certain amount of attributes, having an associated cost. An attribute is covered when a subset that covers it is selected. A possible solution is composed by the selected subsets and it is considered feasible only if all the attributes are covered. The objective consists in minimizing the total cost of the final solution. This combinatorial problem is of NP-hard complexity. This means that the execution times rise exponentially with the increasing of the number of instances, being impracticable to perform a complete search among all possible possibilities. This work implements several methodologies to look for the best possible solution, recurring to heuristics and metaheuristics with neighbourhood based algorithms. Initially, Constructive Heuristics (CH) will be analysed, in order to generate an initial solution, recurring to Greedy methodologies. To remove redundant sets, a redundancy elimination procedure (RE) was implemented. Secondly, a local search was performed on the previous solution, by generating a neighbourhood through a single flip of each of the non selected subsets. Two main approaches were explored in this phase: the First Improvement and the Best Improvement algorithms. At last, a couple of methods were implemented to escape from local optima, like Greedy Randomized Adaptive Search Procedure (GRASP). For each of the implemented methodologies, the pseudo-code and the obtained results at each stage will be presented for a specific list of instances that are available in the repository[1].

II. Problem Formulation

The SCP consists in selecting subsets, of a total of m subsets, capable of covering all the n attributes, minimizing the total cost of the final solution. To solve this problem, the following representation was defined:

$$x_i = \begin{cases} 1, & \text{if subset } i \text{ is selected} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

$$y_j = \begin{cases} 1, & \text{if attribute } j \text{ is covered} \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

$$a_{ji} = \begin{cases} 1, & \text{if attribute } j \text{ is covered by subset } i \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

$$b_{ij} = \begin{cases} 1, & \text{if subset } i \text{ covers attribute } j \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

The objective function the is following:

$$min \sum_{i=1}^{m} x_i * w_i \tag{5}$$

Where w_i represents the cost of subset i.
Subject to:

$$\sum_{i=1}^{m} b_{ij} \geq 1, \forall j = 1, \cdots, n \tag{6}$$

In this way, it is possible to guarantee that the final solution is complete, because for each attribute j, at least one subset i that covers it has to be selected.

III. Constructive Heuristics

A Constructive Heuristic allows to obtain an initial solution to a problem by iteratively adding subsets to the solution, until all the attributes are covered. Usually, it applies Greedy approaches, that choose at each step the subset that optimizes certain criteria. Although it is unlikely to reach the optimal solution of the problem, it provides a feasible and valid solution in a short computational time. The CH can be deterministic or stochastic. A deterministic heuristic would always return

[1] https://github.com/RafaelClaro97/SCP

the same solution. A stochastic heuristic can produce different solutions, as it has a level of randomness in its algorithm [1]. Three heuristics (CH1, CH2 and CH3) were implemented for the SCP. All of the algorithms follow the same loop, shown in Algorithm 1.

Algorithm 1: Main loop

```
while Uncovered attributes > 0 do
    Dispatching rule (CH1, CH2 or CH3) – Select a subset;
    Update the vector of attributes covered by the selected subset;
    Update the vector of subsets that cover each attribute;
    Erase the already covered attributes from the attribute vector of each of the non used subsets;
Remove redundant subsets;
Calculate the total cost of the selected subsets;
```

Each of the CHs only differs on the dispatching rules, shown in the next subsections.

A. CH1

Dispatching Rule: At each stage, choose the subset that contains the highest coefficient of the number of uncovered attributes by the cost of the subset.

The first CH implements a greedy deterministic heuristics that chooses iteratively a subset that better fulfills the defined dispatching rule.

B. CH2

Dispatching Rule: At each stage, select the subset with the lowest cost that serves the attribute that is covered by the fewest number of subsets.

The second CH represents a different greedy deterministic heuristics, that will choose a subset at each step that better fulfills the defined dispatching rule.

C. CH3

Dispatching Rule: At each stage, randomly select one of the top three subsets that have the highest coefficient of the number of uncovered attributes by the cost of the subset.

The last CH has a similar dispatching rule to CH1, but it includes a random procedure, by selecting randomly one of the three best subsets. This means that this greedy approach is stochastic.

D. Results

This subsection will present some of the results, that were obtained in a computer with the following specifications:

- **CPU** - Intel® Core™ i7 7700HQ Quad-Core, 2.80 GHz;
- **RAM** - 16GB DDR4.

The previous algorithms were tested with the given instances. Table I shows the average deviation of the obtained results compared with the best known solutions so far. It also shows the total elapsed time of each CH.

TABLE I: Constructive Heuristics Results (Before RE).

	Average Deviation (%)	Total Elapsed Time (ms)
CH1	13.62	5.3
CH2	17.39	6.0
CH3	20.78	5.4

Given the objective of a CH, the obtained deviation values were satisfactory, specially considering that they are obtained in a very short computation time.

IV. Redundancy Elimination

The algorithm presented in the previous section guarantees that no redundant subset is selected at each iteration. However, some subsets may became redundant with the selecting of following subsets [1]. That means that some unnecessary subsets can be removed from the final solution, reducing its total cost. This procedure was implemented based on the pseudo-code shown in Algorithm 2.

Algorithm 2: Redundancy Elimination

```
Approval list: Subsets that are the only ones covering at least one attribute – cannot be removed;
Rejection list: Subsets that are never the only ones covering an attribute – redundant subsets;
while subsets in the rejection list > 0 do
    Update approval list and rejection list;
    Select the subset from the rejection list with the highest cost;
    Remove the selected subset from the selected subsets vector;
    Update the vector of which subsets cover each attribute;
```

A. Results

The application of this redundancy elimination procedure reduced significantly the average deviation from the best known solutions, resulting in an average improvement of 9.18%. Also, the fraction of instances that profit from RE was 100%, regardless of the used CH. Table I exhibits the obtained values after RE, and the total elapsed time as well.

TABLE II: Constructive Heuristics Results (After RE).

	Average Deviation (%)	Total Elapsed Time (ms)
CH1	5.57	5.4
CH2	9.03	6.1
CH3	9.93	5.5

From these data, it is also possible to conclude that the total elapsed time of the RE procedure is, approximately, of

0.1 ms. This allows to conclude that the RE represents a highly profitable procedure, considering the improvement that it applies, and the reduced execution time that it adds. Figures 1, 2 and 3 show graphically the effect of the RE procedure for each of the instances and CHs.

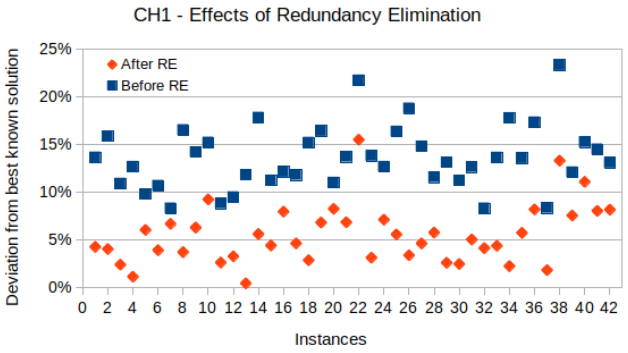

Fig. 1: Redundancy elimination effect in deviation from best known solutions (CH1).

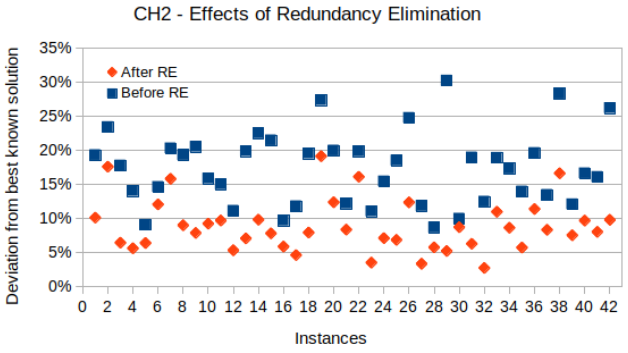

Fig. 2: Redundancy elimination effect in deviation from best known solutions (CH2).

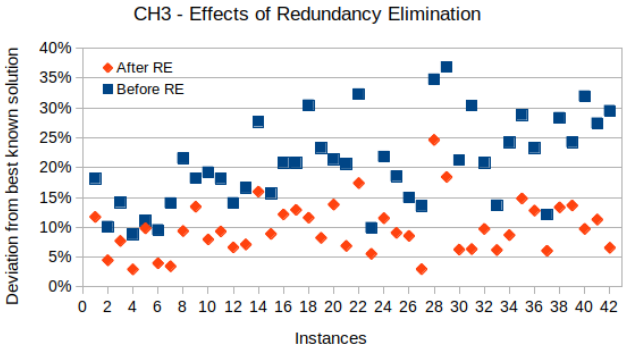

Fig. 3: Redundancy elimination effect in deviation from best known solutions (CH3).

V. Improvement Heuristics

Once a feasible solution is found by the CHs, it is possible to improve that solution by performing a local search in its neighbourhood. The neighbourhood of a solution corresponds to all the solutions (neighbours) that are generated from the initial one, by making small and simple changes to it. This search is performed until it founds a local optimum, that may be or not be the optimal solution. To generate the neighbourhood, one simple flip was executed to each of the non selected subsets at a time. To search through this neighbourhood two approaches were defined: the First Improvement (FI) and the Best Improvement (BI). In the BI, all the neighbours are generated and the best one is selected at each iteration. The FI generates the neighbours in a random sequence and moves to one of them as soon as it finds a better solution than the current one. Therefore, BI is a deterministic method and FI is stochastic. In the next subsections the pseudo-code of each of the algorithms will be presented, as well as the obtained results. From this procedure, the RE criterion changed from iteratively eliminating the redundant subset with the highest cost, to eliminating the redundant subset that presents the highest cost per number of attributes served.

A. First Improvement

As previously said, the FI searches through the neighbourhood until it find a neighbour that corresponds to a better solution than the current one. The implementation of this procedure is based on the pseudo-code presented in the Algorithm 3.

Algorithm 3: Local Search - First Improvement

```
Neighbourhood Search (N=1);
while new improved solution found do
    Neighbour: Randomly add a not used subset to
     current best solution;
    Update the vector of subsets that cover each
     attribute;
    Remove redundant subsets;
    Compare the total cost of this neighbour with the
     best solution;
    if it represents a new best solution then
        Stop searching;
    else
        Continue searching for a better neighbour;
Update current best solution with the first best
 neighbour found;
```

B. Best Improvement

The BI searches through the neighbourhood in order to select the best neighbour among all possibilities. To implement this procedure, an algorithm was implemented based in the pseudo-code presented in the Algorithm 4.

Algorithm 4: Local Search - Best Improvement

```
Neighbourhood Search (N=1);
while new improved solution found do
    Neighbour: Randomly add a not used subset to
     current best solution;
    Update the vector of subsets that cover each
     attribute;
    Remove redundant subsets;
    Compare the total cost of this neighbour with the
     best solution;
    if it represents a new best solution then
        Save that solution;
    else
        Continue searching for a better neighbour;
Select the best solution found (among all possibilities);
Update current best solution with the best neighbour
 found;
```

C. Results

The efficiency and effectiveness of these two different approaches were tested in the previously mentioned instances with two types of initial solutions. The first one corresponds to the solution found after the CHs application (before RE), while the second one corresponds to the solution obtained after the RE procedure. Note that for the FI method, as it contains a stochastic element, the algorithm was executed 5 times with a different $seed$, and the average of the obtained results was calculated. The difference between the average deviation of each execution of this algorithm is low, which can be confirmed through the calculating of variance, that is equal to 1.68%.

Table III shows the average percentage deviation from best known solutions for FI and BI for each of the CH, after and before RE.

TABLE III: Average percentage deviation from best known solutions.

Initial Solution	Local Search **FI**	**BI**
CH1	4.18%	3.74%
CH1+RE	4.16%	4.08%
CH2	6.00%	4.91%
CH2+RE	5.96%	5.40%
CH3	6.34%	5.49%
CH3+RE	6.35%	5.98%

It can be observed that the BI obtained better results than the FI. This result is explained by the fact that BI iterates through all the neighbourhood, selecting the best available solution, while FI stops as soon as it finds the first improved solution. Also, it is possible to observe that the initial solution influences the obtained results. When the initial solution used is the one before RE, it has a wider search space, producing better results in the BI case. For the FI case, this behaviour did not have the same influence.

Table IV shows the fraction of instances that profit from the local search.

TABLE IV: Fraction of instances that profit from local search.

Initial Solution	Local Search **FI**	**BI**
CH1	100%	100%
CH1+RE	66.67%	66.67%
CH2	100%	100%
CH2+RE	90.48%	90.48%
CH3	100%	100%
CH3+RE	85.71%	85.71%

By analysing the results, it is noticeable that when the initial solution is the one before RE, all instances profit from the local search, as these solutions contain redundant subsets. The difference in the fraction of instances that profit from local search, when the initial solution is the one obtained after RE, between CH2/3 and CH1 is explained by the fact that the CH1+RE solution already represents a better quality solution, being harder to improve it.

Table V exhibits the average processing time of both BI and FI, depending on the initial solution.

TABLE V: Average Elapsed Time (ms).

Initial Solution	Local Search **FI**	**BI**
CH1	142.5	499.2
CH1+RE	141.4	195.9
CH2	187.6	553.0
CH2+RE	188.1	289.0
CH3	183.4	618.8
CH3+RE	183.9	336.2

The results from the previous table show that the average elapsed time of FI is significantly lower than the average elapsed time of BI. It is explained by the fact that FI stops at the first best neighbour, while BI searches through all the neighbours. In the BI, the initial solution proved to have great influence on the performance. When it is the one after RE, it has a smaller search space, resulting in shorter computing

times. This implementation resulted in a total elapsed time for all the 42 instances of, approximately, 138 s.

Figures 4, 5 and 6 show graphically the effect of FI and BI procedures in each of the instances when the CH1, CH2 and CH3 (before RE) are performed, respectively.

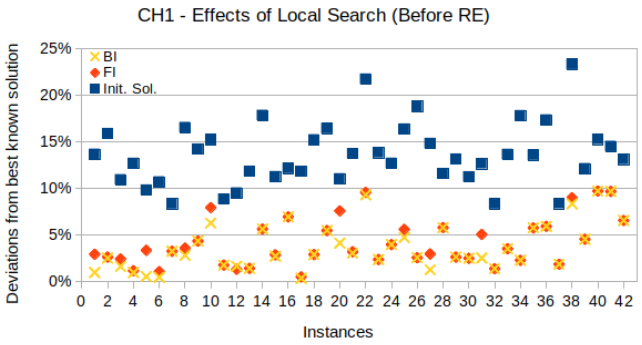

Fig. 4: Local Search effect in deviation from best known solution (CH1).

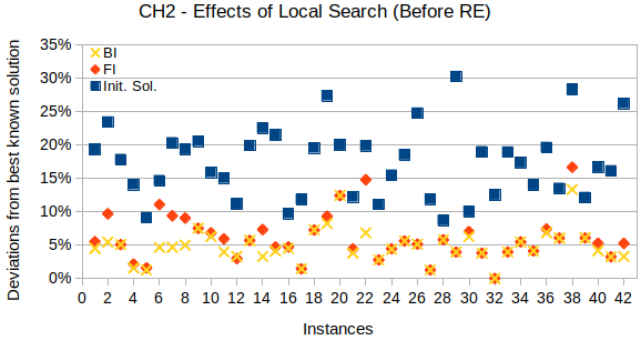

Fig. 5: Local Search effect in deviation from best known solution (CH2).

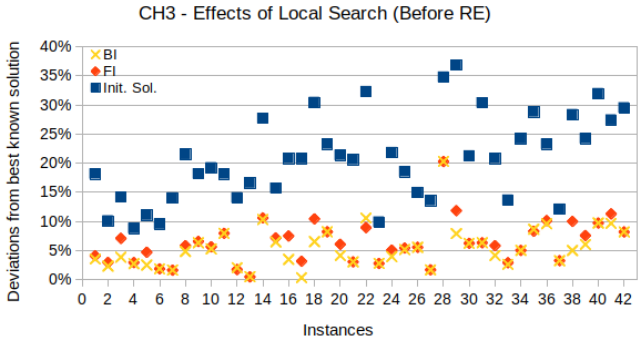

Fig. 6: Local Search effect in deviation from best known solution (CH3).

Figures 7, 8 and 9 show graphically the effect of the FI and BI procedures in each of the instances when CH1, CH2 and CH3 (after RE) are performed, respectively.

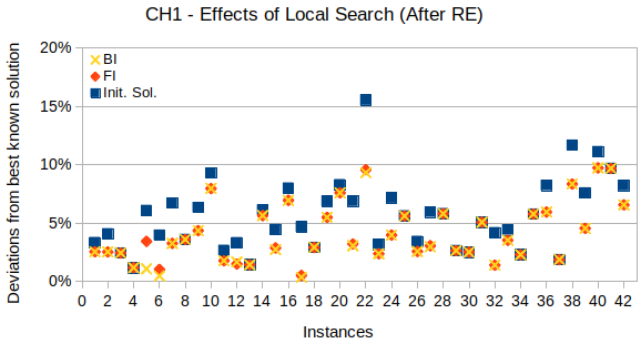

Fig. 7: Local Search effect in deviation from best known solution (CH1+RE).

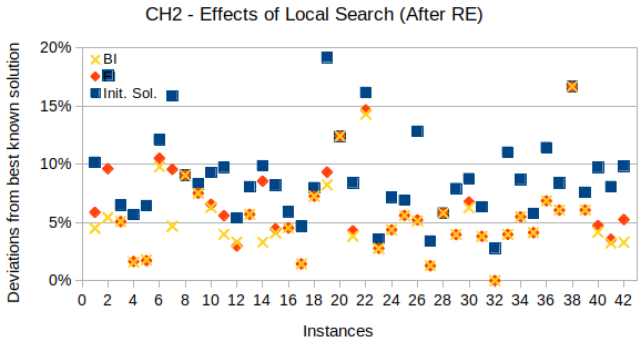

Fig. 8: Local Search effect in deviation from best known solution (CH2+RE).

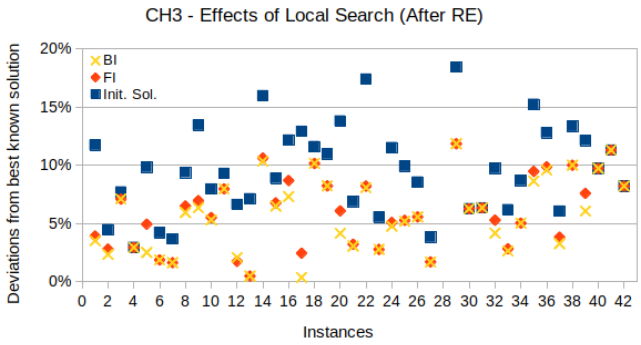

Fig. 9: Local Search effect in deviation from best known solution (CH3+RE).

VI. Greedy Randomized Adaptive Search Procedure

As the methods explained in the previous sections tend to be blocked in local optima, this part of the work will explore other methods based on Greedy Randomized Adaptive Search Procedure (GRASP), that allows to diversify the initial solutions by also accepting solutions that may not represent an improvement compared to the current one. GRASP is

a metaheuristics that has been proved to function well in multiple applications that involve combinatorial problems [2]. It consists in a stochastic method composed of two phases [3]. It first generates some initial solutions recurring to a greedy constructive heuristics with some randomness implied. The second phase consists in performing a local search, usually based on BI, for each of the initial solutions generated. The final solution corresponds to the best result among all the iterations [3]. Algorithm 5 shows the pseudo-code of the GRASP metaheuristics, based on [4].

Algorithm 5: GRASP

```
while it < max_iterations do
    Generate initial solution (with randomness);
    Perform Local Search (BI);
    Apply Redundancy Elimination;
    if it represents a new best solution then
        Save that solution;
    else
        Continue searching for a better solution;
    Increment it;
```

The CH implemented in this GRASP metaheuristics follows the main loop presented in Algorithm 1 and the dispatching rule is very similar to the CH3, presented in III-C, which is: at each stage, randomly select one of the top three subsets that have the highest coefficient of the number of uncovered attributes by its cost. The main difference is that, instead of selecting randomly one of the top three subsets that better satisfy the dispatching rule, it selects randomly one of the $\alpha \cdot m$ subsets, being α a parameter such that $\alpha \leq 1$, and m corresponds to the number of subsets. This dispatching rule was selected because the CH that implements this criterion (CH1) is the one that presents better results. For that reason, a control group based on CH1, that always selects the subset that has the highest coefficient of the number of uncovered attributes by its cost, was maintained. This strategy allows to perform an objective comparison between GRASP and Constructive Heuristics results. Also, the initial solution that will be subject to the local search is the one obtained after RE. Although it shows slightly worse results than the one obtained before RE, it showed to be significantly faster than the previous one. As multiple iterations will be performed in this algorithm, the efficiency was the priority.

Both the RE and BI procedures applied are the same as those previously mentioned in section V.

In the section VI-A, the behaviour of this algorithm will be analysed with different values for the total iterations and α.

A. Results

The results analysed in this section were obtained under the same circumstances as described in III-D. The behaviour of the GRASP algorithm was tested with α taking values belonging to different ranges. The results presented in this report represent the ones obtained when α is equal to 2% and 0.2%. These values were selected based on the fact that for α 5× or 10× larger, not only the results were considerably worse, but also much more time consuming because the search space is considerably larger. Besides testing for different α values, the number of total iterations was also a object of study, changing from 50 to 500 iterations. As this algorithm has stochastic elements, all the executions were performed 5 times, with a different $seed$, for each of the α values and number of iterations.

Tables VI and VII exhibit the obtained values for the average deviation from best known solutions and for the number of optimal solutions met, respectively, when the number of iterations is equal to 50.

TABLE VI: Average deviation results, with number of iterations equal to 50.

$\alpha = 2\%$ $it = 50$	**Average Deviation**	**Maximum Deviation**	**Minimum Deviation**
Control	4.08%	4.09%	4.08%
GRASP	16.27%	16.79%	15.79%
$\alpha = 0.2\%$ $it = 50$	**Average Deviation**	**Maximum Deviation**	**Minimum Deviation**
Control	4.08%	4.09%	4.08%
GRASP	1.75%	1.97%	1.55%

TABLE VII: Number of optimal solutions met, with number of iterations equal to 50.

$\alpha = 2\%$ $it = 50$	**Avg. Opt. Solution**	**Max. Opt. Solution**	**Min. Opt. Solution**
Control	0	0	0
GRASP	0.2	1	0
$\alpha = 0.2\%$ $it = 50$	**Avg. Opt. Solution**	**Max. Opt. Solution**	**Min. Opt. Solution**
Control	0	0	0
GRASP	3.8	6	1

By analysing these results, it is possible to conclude that, when α is equal to 2% and the number of iterations is equal to 50, the average deviation from best known solution is almost 4× larger when compared with the control method. For α values of this order of magnitude the results tend to be significantly worse, specially when the execution time (exhibited in table X) is considered. Regarding the number of optimal solutions met, this combination of parameters achieves one optimal solution once, which can be explained by the random element of this algorithm, resulting in one sporadic event.

Focusing now on the obtained results when α is equal to 0.2%, with the same 50 iterations, a significant improvement is obvious. Not only the average deviation from best known solution decreases almost 2× when compared to the control method, but also the number of optimal solutions met is almost 4, on average. These results show that for a small value of α the obtained solutions are more likely to be close to the optimal one. Besides that fact, the lower execution time of this algorithm for α equal to 0.2% (exhibited in table X) is another advantage.

Figure 10 shows the difference of the deviation from best known solutions for the two values of α, when the number of iterations is equal to 50.

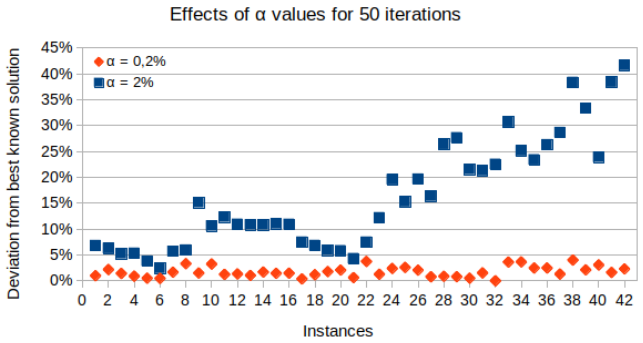

Fig. 10: α values effect in deviation from best known solutions, with number of iterations equal to 50.

Tables VIII and IX exhibit the obtained values for the average deviation from best known solutions and for the number of optimal solutions met, respectively, when the number of iterations is equal to 500.

TABLE VIII: Average deviation results, with number of iterations equal to 500.

$\alpha = 2\%$ $it = 500$	**Average Deviation**	**Maximum Deviation**	**Minimum Deviation**
Control	4.08%	4.09%	4.08%
GRASP	9.35%	9.54%	9.01%
$\alpha = 0.2\%$ $it = 500$	**Average Deviation**	**Maximum Deviation**	**Minimum Deviation**
Control	4.08%	4.09%	4.08%
GRASP	0.88%	0.97%	0.79%

By analysing these results, it is possible to conclude that, when α is equal to 2%, the increase of the number of iterations to 500 did not change the number of optimal solutions met. Although it decreased the average deviation from best known solutions, it continues to represent a value significantly larger (almost 2×) than the control method. As previously concluded, the large computational time for α equal to 2% (exhibited in table X) increases the limitations of this approach.

TABLE IX: Number of optimal solutions met, with number of iterations equal to 500.

$\alpha = 2\%$ $it = 500$	**Avg. Opt. Solution**	**Max. Opt. Solution**	**Min. Opt. Solution**
Control	0	0	0
GRASP	0.2	1	0
$\alpha = 0.2\%$ $it = 500$	**Avg. Opt. Solution**	**Max. Opt. Solution**	**Min. Opt. Solution**
Control	0	0	0
GRASP	10.4	13	8

Regarding the obtained values when α is equal to 0.2%, with the same 500 iterations, it not only decreases the average deviation from best known solutions to half of the value obtained for 50 iterations, but also achieves an average of more than 10 optimal solutions. The increasing of the number of iterations resulted in a considerable improvement of the solutions generated.

Figure 11 shows the difference of the deviation from best known solutions with the two values of α, when the number of iterations is equal to 500.

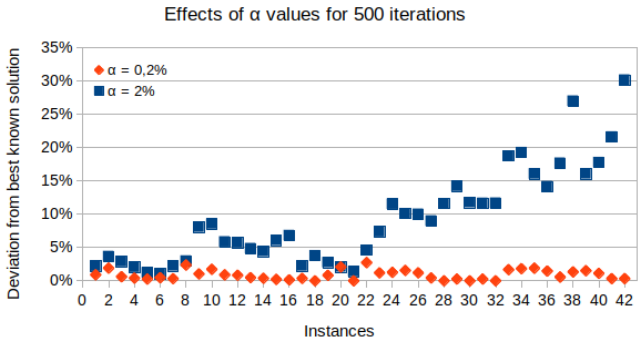

Fig. 11: α values effect in deviation from best known solutions, with number of iterations equal to 500.

Figures 12 and 13 exhibit the improvement obtained when the number of iterations is increased, from 50 to 500 iterations. Figure 14 shows the number of optimal solutions met, for α equal to 0.2%, when the number of iterations is increased from 50 to 500 iterations.

As already mentioned, table X shows the average elapsed time per instance, depending on the number of iterations and on the value that α takes.

As expected, the computation time increased with the increase of the number of iterations. Also, the execution time is smaller for smaller values of α, which is explained by the

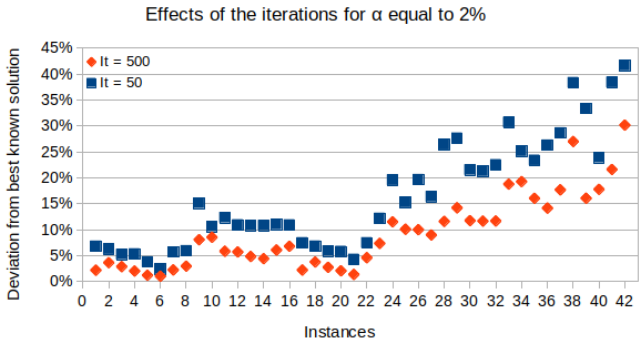

Fig. 12: Number of iterations effect in deviation from best known solution, with α equal to 2%.

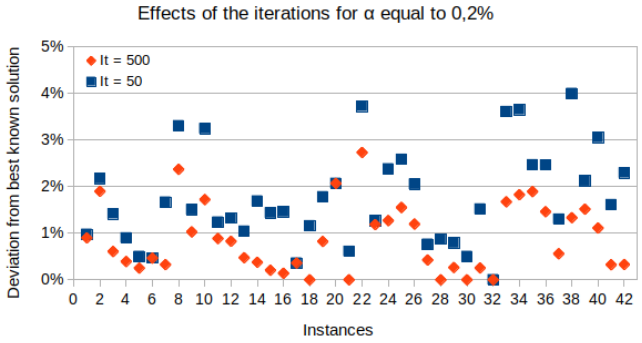

Fig. 13: Number of iterations effect in deviation from best known solution, with α equal to 0.2%.

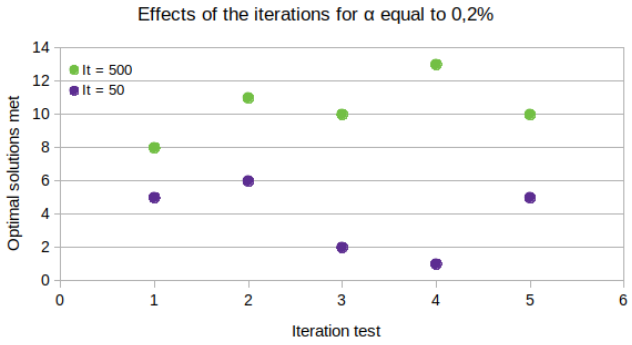

Fig. 14: Number of iterations effect in number of optimal solutions met, with α equal to 0.2%.

TABLE X: Average elapsed time per instance (s).

		iterations (it)	
		50	**500**
α	**2%**	57.0	601.8
	0.2%	19.8	181.8

fact that a smaller α results in a reduced search space, which is faster to compute.

VII. Hybrid GRASP Implementation

In this subsection a hybrid GRASP implementation, that was developed, will be explored. The main difference between the original GRASP implementation and this one relies on an algorithm that expands the neighbourhood of the local search when no better neighbour is found. This procedure is only performed once, and it adds a set of random unused subsets to the current solution, trying to escape from a local optimum, allowing to continue the local search. Algorithm 6 shows the pseudo-code of this hybrid GRASP approach.

Algorithm 6: Hybrid GRASP implementation

```
while it < max_iterations do
    Generate initial solution (with randomness);
    while new improved solution found do
        Perform Local Search (BI);
        Apply Redundancy Elimination;
        if it represents a new best solution then
            Save that solution;
        else
            while unused subsets can be selected do
                Add a random set of unused subsets to
                 the current solution;
                Apply Redundancy Elimination;
                if it represents a new best solution then
                    Save that solution;
                else
                    Continues searching for a better
                     solution;
    increment it;
```

Another change applied to this algorithm consists in the values that the parameter α takes. Instead of being a static value, as in the original implementation of GRASP, it assumes dynamic values, changing according to a hyperbolic function, as the following:

$$\alpha(t) = \frac{coeff}{t^p} + offset \tag{7}$$

Where t corresponds to the current iteration of the constructive heuristics, p to the polynomial degree of the hyperbolic function, $coeff$ to the starting value that α takes, and $offset$ is the value that α will tend for. By applying this hyperbolic function to obtain a dynamic value of α, it allows to add a larger number of subsets in the beginning of the solution formulation, becoming more restricted with the increasing of the number of the current iteration. In this way, a larger set of possible solutions is explored, increasing the possibility of achieving an optimal solution.

In section VII-A the behaviour of this algorithm will be analysed and compared to the original GRASP implementation, with different values for the total iterations.

A. Results

The results analysed in this section were obtained under the same circumstances as described in III-D. The behaviour of this hybrid GRASP algorithm was tested with α changing according to equation VII. The results obtained in this section had $coeff$ equal to 20%, $offset$ equal to 0.2% and p equal to 4. Resulting in:

$$\alpha(t) = \frac{0.2}{t^4} + 0.002 \tag{8}$$

The selected value for $offset$ was based in the value used for α in the previous section that got better results. The value of $coeff$ that was chosen allows to select a considerable range of the best subsets that better fulfill the requirements of the dispatching rule, minimizing the risk of selecting an expensive subset. For the polynomial degree, p, the chosen value allows to select a subset from a large range in the first iterations, reducing the selecting space to the value of $offset$ with the increase of the number of iterations. Besides testing for different values of α, the number of total iterations has also been object of study, changing from 50 to 500 iterations. As this algorithm has stochastic elements, all the executions were performed 5 times, with a different $seed$, for each of the different number of iterations.

Table XI exhibits the obtained values for the average deviation from best known solutions and for the number of optimal solutions met when the number of iterations is equal to 50.

TABLE XI: Average deviation and optimal solutions met, with number of iterations equal to 50.

$it = 50$	**Average Deviation**	**Maximum Deviation**	**Minimum Deviation**
Control	4.08%	4.09%	4.08%
H. GRASP	1.71%	1.80%	1.52%
$it = 50$	**Avg. Opt. Solution**	**Max. Opt. Solution**	**Min. Opt. Solution**
Control	0	0	0
H. GRASP	4.8	6	3

By analysing these results, it can be verified that the average deviation from the best known solutions did not change significantly, when compared with the simple GRASP implementation for 50 iterations. The most expressive improvement relies on the number of optimal solutions met, which increased by 1, on average.

Table XII exhibits the obtained values for the average deviation from best known solutions and for the number of optimal solutions met when the number of iterations is equal to 500.

Comparing the obtained results with the best ones from the simple GRASP implementation (section VI) for 500 iterations,

TABLE XII: Average deviation and optimal solutions met, with number of iterations equal to 500.

$it = 500$	**Average Deviation**	**Maximum Deviation**	**Minimum Deviation**
Control	4.08%	4.09%	4.08%
H. GRASP	0.69%	0.75%	0.60%
$it = 500$	**Avg. Opt. Solution**	**Max. Opt. Solution**	**Min. Opt. Solution**
Control	0	0	0
H. GRASP	16	17	15

the improvement is notorious. The average percentage deviation decreases by almost 0.2% points, but the most significant improvement is in the average number of optimal solutions met, which increased more than 50%.

These results allow to conclude that the general improvement that this hybrid GRASP approach applies in the average deviation from best known solutions is reduced, while being greatly profitable in the number of optimal solutions met.

Figures 15 and 16 exhibit the improvement obtained when the number of iterations is increased, from 50 to 500 iterations.

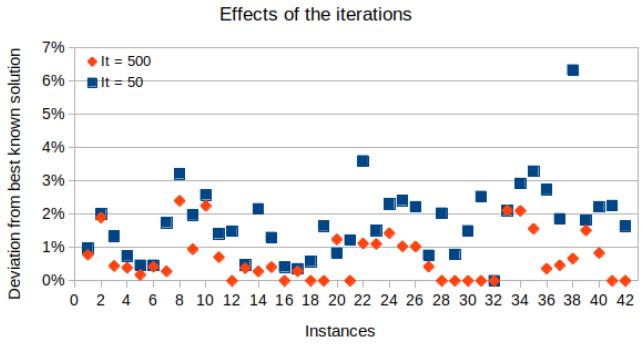

Fig. 15: Number of iterations effect in deviation from best known solutions.

Figures 17 and 18, and figures 19 and 20 exhibit the improvement obtained when the simple and hybrid GRASP approaches are compared, for 50 and 500 iterations, respectively.

Table XIII shows the average elapsed time per instance, depending on the number of iterations. As expected, the total elapsed time per instance increased almost 47 seconds, when compared with the simple GRASP implementation. Although it takes longer to obtain a final solution, the improvement that this hybrid GRASP approach presents justifies its use.

Fig. 16: Number of iterations effect in number of optimal solutions met.

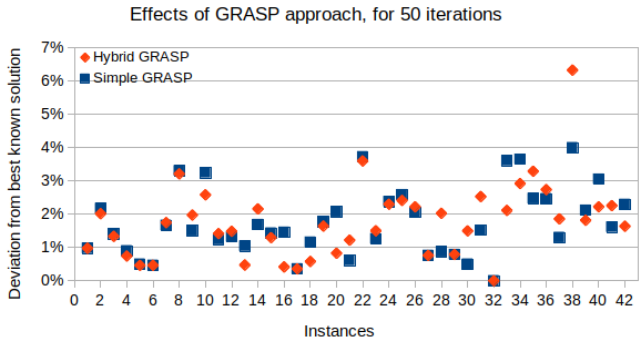

Fig. 17: GRASP approach effect in deviation from best known solutions, for 50 iterations.

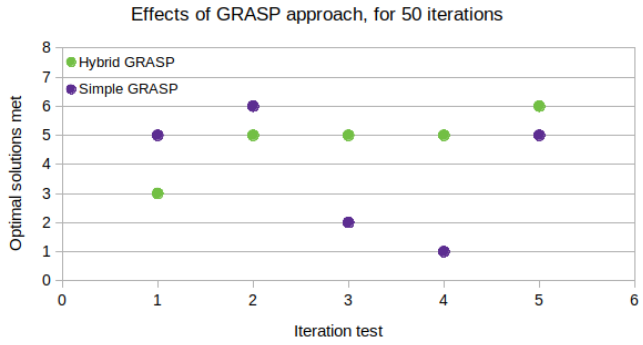

Fig. 18: GRASP approach effect in number of optimal solutions met, for 50 iterations.

TABLE XIII: Average elapsed time per instance (s).

	iterations (it)	
	50	**500**
H. GRASP	24.0	228.6

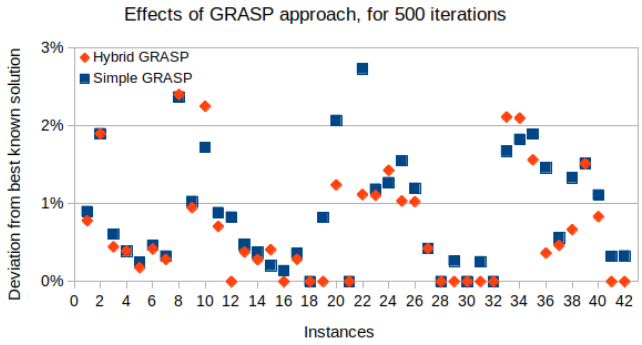

Fig. 19: GRASP approach effect in deviation from best known solutions, for 500 iterations.

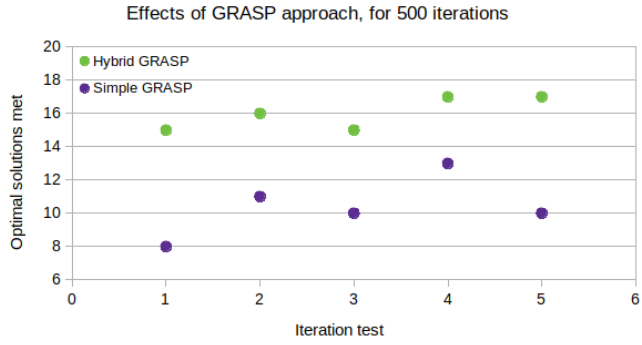

Fig. 20: GRASP approach effect in number of optimal solutions met, for 500 iterations.

VIII. Conclusion

The Set Covering Problem is a combinatorial problem that can be solved recurring to heuristics and metaheuristics. These methods allow to reach a feasible solution that may or may not be the optimal one. The quality of each approach is defined by the average deviation from the best known solutions and the number of optimal solutions met, as well as the time it takes to reach the final solution.

This paper presents two different Constructive Heuristics with deterministic dispatching rules, while the third CH has a stochastic one. The minimum average deviation from best known solutions obtained was of 13.62%. These CH reaches a feasible solution. However, they had redundant subsets in their generated solutions. To remove them, a Redundancy Elimination procedure was implemented, resulting in an average improvement of 9.18% in all the given instances.

To further improve the obtained solutions, a local search was performed. The two different approaches consist in First Improvement and Best Improvement. Neighbours were generated based on simple flips of non used subsets. BI obtained better results (the best case scenario for average deviation from best known solutions was 4.08%), given that it searches for a better neighbour through all the neighbourhood.

The last part of the developed work consists in studying a Greedy Randomized Adaptive Search Procedure to minimize the cost of the final solution. This metaheuristics may search through worse solutions, in order to allow it to reach better solutions. This algorithm has only two parameters: the maximum number of iterations (it) and α, that defines how greedy the solution construction phase will be. The behaviour of this GRASP implementation was tested for α varying from 2% to 0.2%, and the number of iterations from 50 to 500. As this method has stochastic elements in it, multiple tests were run. The best case scenario ($\alpha = 0.2\%$ and $it = 500$) obtained the value of 0.88% for the average deviation from the best known solutions, having met 10.4 optimal solutions, on average. The average elapsed time of this approach was of 181.8 seconds per instance.

Another developed approach consisted in a hybrid GRASP based procedure. Not only the values that α takes are dynamic, changing according to a hyperbolic function, but also an enlargement of the neighbourhood is performed when no improvement is obtained from the local search. This implementation resulted in an average deviation from best known solutions of 0.69% and an average of 16 optimal solutions met, for the best case scenario. The average elapsed time of this approach was 228.6 seconds per instance.

In conclusion, all the steps presented in this paper resulted in an iterative improvement of the final solution, reaching high precision results, in a short computational time.

REFERENCES

[1] N. Bilal, P. Galinier, and F. Guibault, "A New Formulation of the Set Covering Problem for Metaheuristic Approaches", ISRN Operations Research, vol. 2013, Article ID 203032, 10 pages, 2013.

[2] S. Binato, W.J. Hery, D.M. Loewenstern and M.G.C. Resende, "A Grasp for Job Shop Scheduling. In: Essays and Surveys in Metaheuristics", Operations Research/Computer Science Interfaces Series, vol 15. Springer, Boston, MA, 2002.

[3] L. Guimarães, R. Santos and B. Almada-Lobo, "Scheduling wafer slicing by multi-wire saw manufacturing in photovoltaic industry: a case study", International Journal of Advanced Manufacturing Technology 53(9):1129-1139, 2011.

[4] T. A. Feo and M. G. C. Resende, "Greedy Randomized Adaptive Search Procedures", Journal of Global Optimization, vol 6(2), pp. 109-133, 1995.