Effective Heuristics for the Set Covering with Pairs Problem

Luciana Brugiolo Gonçalves[1], **Simone de Lima Martins**[1] **and Luiz Satoru Ochi**[1]

[1]*Universidade Federal Fluminense, Instituto de Computação, Passo da Pátria 156 - Bloco E / 350 - 24210-240. Niterói, Brasil.* {lgoncalves,simone, satoru}@ic.uff.br

Keywords: Set Covering with Pairs, Heuristics, Combinatorial Optimization.

Abstract. This paper deals with the Set Covering with Pairs Problem (SCPP). This problem is a generalization of the Set Covering Problem (SCP), which is known to be NP-hard. The difference between both problems is that, in the SCPP, one element of the universe is admitted to be covered if there are at least two specific objects chosen to cover it. In this context, three constructive heuristics, one local search algorithm and a GRASP are proposed. The algorithms developed were tested in 560 instances available in the literature and they were capable of achieving optimal solutions for several instances.

1 Introduction

This paper presents heuristics to treat the Set Covering with Pairs Problem (SCPP) that was proposed by Hassin and Segev (Hassin and Segev, 2005). The SCPP is considered a generalization of the Set Covering Problem, in which the elements are covered with specific pairs of objects, rather than for a single object. The objective is to find a minimum cost subset of objects that induces a collection of pairs covering all elements.

Formally, let U be a ground set of elements and let A be a set of objects, where each object $i \in A$ has a non-negative cost w_i. For every pair $\{i, j\} \subseteq A$, with $i \neq j$, let $\mathcal{C}(i, j)$ be the subset of elements in U covered with the pair $\{i, j\}$. The Set Covering with Pairs problem asks to find a subset $S \subseteq A$ such that $\bigcup_{\{i,j\}\subseteq S} \mathcal{C}(i, j) = U$ and such that $\sum_{i\in S} w_i$ is minimized.

Let us consider, for example, an instance I composed by the set with three elements to be covered $U = \{u_1, u_2, u_3\}$ and four possible objects $A = \{a_1, a_2, a_3, a_4\}$ with costs $w_{a_1} = 3$, $w_{a_2} = 2$, $w_{a_3} = 5$ and $w_{a_4} = 10$. As shown in Figure 1, the covered elements for the following pairs are: $\mathcal{C}(a_1, a_2) = \{u_1, u_2\}$, $\mathcal{C}(a_1, a_4) = \{u_1\}$, $\mathcal{C}(a_2, a_3) = \{u_1, u_2\}$ and $\mathcal{C}(a_3, a_4) = \{u_3\}$. For the remaining pairs $\mathcal{C}(a_i, a_j) = \emptyset$. It is easy to see that it is not possible to build a solution without the pair $\{a_3, a_4\}$, because this pair is the only option to cover u_3. To cover the other remaining elements, the pair $\{a_2, a_3\}$ or the object a_2 should be inserted in the solution. The minimum cost is obtained by inserting the object a_2. Thus, $\mathcal{C}(a_3, a_4) \cup \mathcal{C}(a_2, a_3) = U$ and the best solution is $S = \{a_2, a_3, a_4\}$ with cost equal to 17.

A set covering instance, with $U = \{e_1, \ldots, e_n\}$ and $A = \{A_1, \ldots, A_m\}$, where A_l is a subset of U for $l = \{1, \ldots, m\}$, can be interpreted as Set Covering with Pairs instance by setting $\mathcal{C}(A_i, A_j) = A_i \cup A_j$ for every $i \neq j$. Therefore, the Set Covering with Pairs Problem is a generalization of the Set Covering Problem (SCP) and the hardness results regarding SCP extend to SCPP, as shown in (Hassin and Segev, 2005).

Another problem that can be seen as a SCPP instance is the k-Cover Problem when $k = 2$. In this case, U and A are considered as in the previous paragraph, and $\mathcal{C}(A_i, A_j) = A_i \cap A_j$ for every $i \neq j$.

The first algorithm proposed for the Set Covering with Pairs Problem (Hassin and Segev, 2005) was a greedy heuristic obtained from the algorithm developed in (Chvátal, 1979) for the Set Covering Problem. In this approach the solution is built iteratively and, in each iteration,

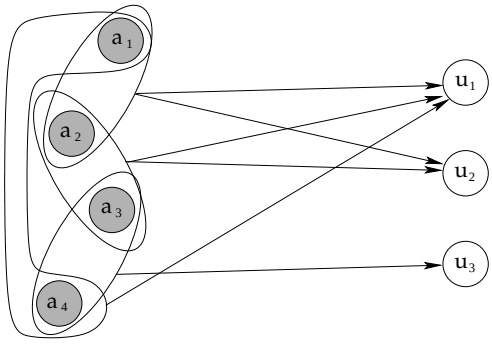

Figure 1: Example of an instance of the Set Covering with Pairs Problem

each object and all pairs of objects are evaluated. Despite generating good results, this heuristic requires much computational time.

Some recent problems in computational biology can be treated as SCPP. The Pure Parsimony Haplotyping Problem (PPHP) is a problem where the objective is to determine a set of pairs of haplotypes H such that each genotype of a given set of genotypes G is resolved with a pair from H. This problem can be treated as a SCPP and some exact and non-exact algorithms were developed for solving it (Catanzaro and Labbé, 2009). Another biochemical problem that can be modeled as SCPP is the Polymerase Chain Reaction (PCR) Primer Design. PCR requires the presence of two single-stranded DNA sequences called primers, which complement specific parts of either the forward or reverse strand of the double-stranded DNA and enable duplication of the region in between. Heuristics and approximation algorithms for this problem were developed in (Fernandes and Skiena, 2002) and (Hajiaghayi et al., 2006).

Hermelin et al. (Hermelin et al., 2008) developed approximation algorithms for the Minimum Substring Cover Problem, where the elements to be covered are strings in S and the elements to cover them are their substrings. If each string in S can be written as a concatenation of at most l strings in $\mathcal{C}$, $\mathcal{C}$ is defined as an l-cover of S. This problem is closely related to SCPP when $l = 2$.

In this work, we propose three constructive algorithms and a local search algorithm to SCPP. As the metaheuristic GRASP (Greedy Randomized Adaptive Search Procedures) proposed by Feo and Resende (Feo and Resende, 1995) has shown good results for several optimization problems (Resende and Ribeiro, 2005), this approach is also evaluated to treat the SCPP.

The remainder of the paper is organized as follows. A mathematical model which describes the SCPP is presented in Section 2. Section 3 contains the proposed algorithms to treat the problem. To evaluate these algorithms, 560 instances introduced by (Resende, 2007) were used. Section 4 shows the results obtained for these instances. Finally, Section 5 presents the concluding remarks of this work.

2 Set Covering with Pairs Problem

For the SCPP, let U be a ground set of elements and A be a set of objects, where each object $i \in A$ has a non-negative cost w_i. For an element $e \in U$, the set P_e contains the set of pairs $\{i, j\} \subseteq A$ where $\mathcal{C}(i, j)$ contains e. An element $e \in U$ is said to be partly covered if an object i of a pair $\{i, j\} \in P_e$ is present in the partial solution S, but the object j which completes this pair does not yet belong to S. For the element to be covered with a pair, it is necessary that two objects that form a pair in P_e are in the solution set S.

Hassin and Segev (Hassin and Segev, 2005) modeled the SCPP as a simple mixed integer

linear program (MIP), based on the model presented in (Gusfield, 2003). This MIP has one boolean variable x_v for each $v \in A$, where $x_v = 1$ if v is chosen and $x_v = 0$ otherwise. In addition, for each pair $\{u, v\} \subset A$, with $u < v$, the variable $y_{u,v}$ is equal to 1 if both u and v are in the chosen set. Hereafter, the expressions that compose the model are presented.

$$\min \sum_{i \in A} w_i \cdot x_i \tag{1}$$

Subject to:

$$\sum_{\{u,v\} \in P_e} y_{u,v} \geq 1, \qquad \forall e \in U; \tag{2}$$

$$y_{u,v} \leq x_u, \qquad \forall u, v \in A | u < v; \tag{3}$$

$$y_{u,v} \leq x_v, \qquad \forall u, v \in A | u < v; \tag{4}$$

$$x_v \in \{0, 1\}, \qquad \forall v \in A; \tag{5}$$

$$y_{u,v} \in \{0, 1\}, \qquad \forall u, v \in A. \tag{6}$$

The constraints (2) state that for each element $e \in U$, there should be at least one pair (u, v) that covers e. And, constraints (3) and (4) guarantee that $y_{u,v}$ is positive only if x_u and x_v are in the chosen set. The objective function (1) minimizes the sum of of the costs the selected objects.

3 Proposed heuristics

3.1 Constructive heuristics

In this section, we describe the three constructive heuristics developed for SCPP. The first one is based on the coverage of one element at every iteration and the other two are based on the selection of one object at each construction step.

3.1.1 The Best Pair Heuristic (BPH)

This heuristic is based on the cheapest insertion strategy. At each iteration, the algorithm chooses the best pair of objects to cover a particular element.

As shown in Algorithm 1, the algorithm starts with an empty set of objects S. Next, the elements to be covered are sorted in ascending order according to the number of possible pairs of objects that may cover them, and are stored in the element list EL (line 2).

The loop from lines 3 to 9 is performed while S does not cover all elements $e \in U$. In line 4, an uncovered element e is selected from EL. In line 5, a Restricted Candidate List (RCL) is created. The possible pairs of objects that cover e are evaluated according to the function shown in Equation (7), where $coverage(i, j)$ represents the number of new elements that will be covered by placing the pair $\{i, j\}$ in S and the factor $increase\_solution(i, j)$ is the increase in the cost solution by inserting the pair (i, j).

$$f(i, j) = coverage(i, j)/increase\_solution(i, j) \tag{7}$$

If just one of the nodes of the pair (i, j) already belongs to S, $increase\_solution(i, j)$ is the cost of the object that is not yet in the solution and when both nodes do not belong to the

Algorithm 1: Best_Pair_Heuristic (α)

```
1  S ← { };
2  EL ← Sort_Elements (U);
3  while (S not covers all elements e ∈ U) do
4      e ← Select_Next_Uncovered_Element (EL);
5      RCL ← Create_List_Pairs (e, α);
6      {i, j} ← Choose_Pair_Randomly (RCL);
7      S ← S ∪ {i, j};
8      Update_Element_List (EL);
9  end
10 return S;
```

solution, $increase\_solution(i, j)$ is set to $w_i + w_j$. Next, given the parameter α as a number in $[0, 1]$, a fraction α of the pairs better evaluated are inserted in the RCL.

In line 6, a pair of objects $\{i, j\}$ which covers e is randomly selected from the RCL and inserted into the solution S (line 7). The EL list is updated in line 8, where e and the new covered elements are removed. The algorithm stops when S cover all elements in U.

3.1.2 The Best Object Heuristic (BOH)

This algorithm is a simplification of the greedy algorithm introduced in (Hassin and Segev, 2005). While in the original algorithm the elements are evaluated individually and in pairs, in this proposed heuristic only the insertion of one element at a time is considered. Another difference is the function used for evaluating the cost of inserting an object in the solution.

In each iteration, every object $a \in A \backslash S$ is evaluated according to the number of new pairs that are completed by inserting it, as well as the number of new elements that can have a as one element of a cover pair, but do not have the other pair element associated with it already inserted in S.

Let U_a be the set of uncovered elements where a appears in at least one of the pairs that can cover it. In Equation (8), $f(a)$ is calculated according to the number of elements of the sets U' and U'', where $U' \cup U'' = U_a$, such that U' is the set of nodes covered with $S \cup \{a\}$ and U'' is the set composed by the other elements of the set U_a partially covered with $S \cup \{a\}$. The weights p_1 and p_2 are associated with each set.

$$f(a) = p_1 \times |U'| + p_2 \times |U''| \tag{8}$$

Algorithm 2: Best_Object_Heuristic (α)

```
1  S ← { };
2  while (S not covers all nodes e ∈ U) do
3      RCL ← Create_Object_List (A \ S, α);
4      a ← Choose_Object_Randomly ( RCL );
5      S ← S ∪ {a};
6  end
7  return S;
```

Algorithm 2 starts with an empty set of objects S. Then, as long as the set solution S does not cover all elements of U, at each iteration a node $a \in A \backslash S$ is inserted in S. In order to select the next element a to be inserted, the candidates are sorted in descending order according to the function f described in the Equation (8). Then, the first fraction α of the candidates are chosen to form a Restricted Candidate List (line 3). In line 4, an object a is randomly selected from this list and it is included in the solution S in line 5.

3.1.3 The Modified Best Object Heuristic (MBOH)

The structure of this heuristic is similar to that of the Best Object Heuristic. The difference is the function that evaluates each object at the time to create the Restricted Candidate List.

Let U_a be the set of not yet covered elements where the object $a \in A$ appears in at least one of the pairs that can cover it. To evaluate the cost of inserting node a in the partial constructed solution, we consider how a can contribute to cover the elements of U_a. The Equation (9) is used to evaluate each object $a \in A \backslash S$ as follows:

$$f(a) = \frac{\sum_{e \in U_a} \textit{percentage of participation}(a, e)}{w_a} \tag{9}$$

For example, consider the subset $U_a = \{e_5, e_7\}$. If an object a appears in half of the possible pairs that can cover e_5 and in $1/4$ of those that can cover e_7, then the value of the function for this object is $f(a) = (0.50 + 0.25)/w_a$.

The objects are sorted in descending order according to this function. Then, an object is randomly selected from the best candidates, using the same parameter α, and is included in the solution.

3.2 Local Search

In order to improve the solution obtained by the constructive algorithms, a local search heuristic was developed. The local search algorithm starts with a solution obtained by one of the constructive algorithms described in the previous section. Then, in each iteration, some objects are removed from the solution and a procedure is executed to make this new solution feasible. The algorithm is executed until no improvement in the current solution is achieved.

Algorithm 3 describes this heuristic. The parameter S corresponds to the initial solution, $percent \in [0, 1]$ indicates the percentage of objects that will be removed from the solution and $withoutImproves$ is the number of iterations without improvement that are allowed by the algorithm.

In the first stage of the algorithm (line 4), $|S| \times percent$ nodes of the best solution are randomly selected and removed. In this step, an infeasible solution S' is created. Then, to make the solution feasible, the reconstruction is carried out inserting nodes of the set $A \backslash S'$ (line 5). The constructive heuristics presented in the previous section can be used to reconstruct the solution.

If the new solution has a lower cost than the best known solution (*bestSolution*), then *bestSolution* is updated in line 7 and the number of no improvements is set to 0 in line 8. Otherwise the number of no improvements is incremented by one in line 10. The algorithm ends when $withoutImproves$ iterations are carried out without any improvement.

Algorithm 3: Local_Search (S, $percent$, $withoutImproves$)

```
1  bestSolution ← S;
2  countImproves ← 0;
3  while countImproves < withoutImproves do
4      S' ← Remove_Elements_from_Solution (bestSolution, percent);
5      S' ← Rebuild_Solution (S');
6      if ( cost(S') < cost(bestSolution) ) then
7          bestSolution ← S';
8          countImproves ← 0;
9      else
10         countImproves ← countImproves + 1;
11     end
12 end
13 return bestSolution
```

3.3 GRASP

We propose a GRASP heuristic (Feo and Resende, 1995) to deal with the SCPP. GRASP is an iterative process, where each iteration consists of two phases: construction and local search. These two steps are performed while the stop criterion is not reached.

The construction phase of GRASP is an iterative process where, at each iteration, the elements that do not belong to the partial solution are evaluated by a greedy function, which estimates the gain of including it in the partial solution. They are ordered by their estimated value in a list called Restricted Candidate List (RCL) and one of them is randomly chosen and included in the solution. The size of the list is limited by a parameter α. In the construction phase a feasible solution is built, and its neighborhood is explored by a local search. The result is the best solution found over all iterations.

The GRASP heuristic proposed in this work uses the three construction heuristics presented in the previous section, and is showed in Algorithm 4. In each of the three initial interactions, a solution is obtained by using each constructive heuristic in greedy mode ($\alpha = 0$) followed by the local search (lines 1 to 6). The variables $costBPH$, $costBOH$ and $costMBOH$ store the cost of the best solution found by each constructive heuristic during the execution of iterations. The variable S^* stores the best solution found by the algorithm, and is the result returned by GRASP.

From line 12 to 18, the remaining GRASP iterations are performed. In line 13, one of the constructive heuristics is selected to be used to construct a solution in line 14. The constructive heuristic is selected randomly according to the quality of the best solution obtained in previous iterations using the heuristic (variables $costBPH$, $costBOH$ and $costMBOH$). Heuristics which provided better solutions receive greater probability to be selected. The constructive procedure is executed using the selected heuristic (line 14) and the local search described in Section 3.2 is applied to this solution in line 15.

If the new solution S obtained in line 15 presents a lower cost than the cost of solution S^*, S^* is updated in line 16. In line 17, the best cost found by the chosen constructive heuristic is updated if the solution S presents better cost than the last solution found using the same heuristic in previous iterations. The stop criterion adopted in the experiments is the maximum number of iterations.

The parameters $percent$ and $withoutImproves$, used in the Local Search procedure, were

Algorithm 4: SCPP_GRASP (α, $maxIteration$)

$S_1 \leftarrow$ Best_Pair_Heuristic (0);
$S'_1 \leftarrow$ Local_Search (S_1, $percent$, $withoutImproves$);
$S_2 \leftarrow$ Best_Object_Heuristic (0);
$S'_2 \leftarrow$ Local_Search (S_2, $percent$, $withoutImproves$);
$S_3 \leftarrow$ Modified_Best_Object_Heuristic (0);
$S'_3 \leftarrow$ Local_Search (S_3, $percent$, $withoutImproves$);
$costBPH \leftarrow$ cost(S'_1);
$costBOH \leftarrow$ cost(S'_2);
$costMBOH \leftarrow$ cost(S'_3);
$S^* \leftarrow$ Best_Solution(S'_1, S'_2, S'_3);
$c^* \leftarrow$ cost(S^*) ;
for *countIteraction = 4* **to** *maxIteration* **do**
    $Alg \leftarrow$ Select_Constructive_Heuristic($costBPH$, $costBOH$, $costMBOH$);
    $S \leftarrow$ Constructive_Greedy_Randomized(Alg, α);
    $S \leftarrow$ Local_Search(S, $percent$, $withoutImproves$);
    Update_Solution (S, S^*);
    Update_Best_Solution_Alg ($S, Alg, costBPH, costBOH, costMBOH$);
end
return S^*

defined in Section 3.2. The parameter α was defined in Section 3.1.

4 Computational Results

The proposed algorithms were implemented in C++, compiled in g++ version $4.1.3$ and executed in a PC Intel(R) Core(TM) 2 Duo T7250 with 2GHz CPU and 2GB RAM.

To evaluate the heuristics, we used 560 instances obtained from (Resende, 2007). These instances were generated by randomly selecting from a set $N = \{1, \ldots, n\}$ the elements to form the set U and the objects to form the set A. Thus, the maximum number of elements in U and the objects in A are restricted to $|N|$. The instances were generated with $|N|$ equal to 26, 50, 100, 190, 220, 250 or 300, and considering that each object of A has the same cost.

In four of these groups, it was assumed that the set of objects A is equal to the entire set N and different element sets U were generated by randomly selecting 12.5%, 25% , 50% and 100% of the elements in N. The other three groups have $|U| = |A|$, where U is a set consisting of 12.5%, 25% and 50% of elements randomly chosen from N. To identify the groups, for positive integers (a, u) the notation (Aa_Uu) is used to refer to classes of instances which the objects set forms a fraction $1/a$ of the set N, and the elements set forms a fraction $1/u$ of the set N.

In the local search algorithm, the value of parameter $withoutImproves$ was set to 5 and the parameter $percent$ was set to 0.3. These parameters were adjusted empirically after preliminary tests. In the same way, in the Best Object Heuristic the parameter p_1 was set to 2 and p_2 to 1, favoring complete coverage of elements.

In all experiments, the constructive heuristic used in local search is BPH with $\alpha = 0.3$ because it generates good quality solutions in much less computational time than the other constructive heuristics.

The first conducted experiment was executed using greedy constructive procedures. For

the three constructive heuristics, the better element is selected to be inserted in the solution, i.e the parameter $\alpha = 0$. For most instances, the performance of the algorithms were similar. Therefore, we only report a subset of 37 instances where it is possible to see differences between the results of the algorithms.

Table 1 shows a comparison between the solutions obtained by each greedy heuristic (GBPH-Greedy Best Pair Heuristic, GMBOH-Greedy Modified Best Object Heuristic and GBOH-Greedy Best Object Heuristic) using or not using the Local Search-LS.

For each group, the number of instances where the heuristic reached the best known solution is presented. The column labeled $Group$ indicates the group of instances, $\#Inst.$ is the number of instances in each group and IP is the number of instances where the optimal solution is known. In the remaining columns, the number of instances where the best known solution was reached by each heuristic is presented.

From Table 1, it is possible to see that the pure heuristics (without local search) GBPH, GMBOH and GBOH obtain the best solution in 25, 32 and 20 instances respectively. The use of the local search algorithm was not effective when applied to heuristic GMBOH, but was able to significantly increase the number of best solutions achieved by heuristics GBPH and GBOH. GBPH+LS, GMBOH+LS and GBOH+LS obtain the best solution in 33, 32 and 27 instances respectively.

Group	#Inst	IP	GBPH	GBPH+LS	GMBOH	GMBOH+LS	GBOH	GBOH+LS
A1-U1	16	2	7	13	12	12	5	12
A1-U2	5	2	5	5	5	5	4	4
A1-U4	6	2	6	6	6	6	4	4
A1-U8	4	2	4	4	4	4	2	2
A2-U2	4	1	1	3	3	3	3	3
A4-U4	1	1	1	1	1	1	1	1
A8-U8	1	1	1	1	1	1	1	1
Sum	37	11	25	33	32	32	20	27

Table 1: Computational results for greedy algorithms

Regarding the computational effort, the average time of the heuristics GBPH, GMBOH and GBOH for the instances shown in Table 1 are respectively 302.16, 330.85 and 226.57 seconds. The use of local search increases the running time by only 85 seconds, as the heuristics generated good initial solutions.

Additional experiments were performed using $\alpha = 0.3$ for the constructive heuristics. Each proposed algorithm was executed ten times with ten different random number seeds.

In Table 2, the best results obtained are presented by group of instances. There are 560 instances, but we show the results only for 176 instances for which we obtained different results among the proposed algorithms. For 384 instances, all algorithms found the same result.

In this table, the column labeled by $Group$ indicates the group of instances, $\#Inst.$ is the number of instances in each group for which the algorithms present different results. The other columns show, for each algorithm, the number of instances where the algorithm found the best value among all algorithms.

We can see that for best values the pure heuristics BPH, MBOH and BOH did not obtain the best solution in 13, seven and five instances respectively. The use of the local search algorithm was effective when applied to all constructive heuristics. Both BPH+LS and MBOH+LS did not obtain the best solution in only three instances and BOH+LS obtained all best results.

In Table 3, we present another comparison of the proposed algorithms by showing the average results obtained in ten runs for the same 176 instances.

Group	#Inst.	BPH	BPH + LS	MBOH	MBOH + LS	BOH	BOH + LS
A1-U1	56	46	54	50	54	52	56
A1-U2	34	34	34	34	34	34	34
A1-U4	28	28	28	28	28	28	28
A1-U8	11	11	11	11	11	10	11
A2-U2	32	29	31	31	31	32	32
A4-U4	11	11	11	11	11	11	11
A8-U8	4	4	4	4	4	4	4
Sum	176	163	173	169	173	171	176

Table 2: Best results for random algorithms

In this table, the column labeled by $Group$ indicates the group of instances, $\#Inst.$ is the number of instances in this group for which the algorithms present different results and the other columns show, for each algorithm, the number of instances where the algorithm found the best average value among all algorithms.

Group	#Inst.	BPH	BPH + LS	MBOH	MBOH + LS	BOH	BOH + LS
A1-U1	56	46	45	52	53	2	28
A1-U2	34	34	34	34	34	2	29
A1-U4	28	28	28	28	28	2	19
A1-U8	11	11	11	11	11	2	8
A2-U2	32	29	29	31	31	1	25
A4-U4	11	11	11	11	11	1	10
A8-U8	4	4	4	4	4	1	3
Sum	176	163	162	171	172	11	122

Table 3: Average results for random algorithms

The pure heuristics BPH, MBOH and BOH did not obtain the best average solution in 13, 5 and 165 instances respectively, so we can see that the MBOH heuristic obtained the best results. The use of the local search algorithm showed to be effective only when applied to BOH heuristic. BPH+LS, MBOH+LS and BOH+LS did not obtain the best average solution in 14, 4 and 54 instances respectively.

We can see that the heuristics based on BPH and MBOH show a similar behavior regarding best and average values, while heuristics based on BOH present quite different results. The pure BOH heuristic obtained 171 best results and only 11 best average results, and BOH+LS obtained 176 best results and 122 best average results. This discrepancy can be explained by the Standard Deviation (SD) values presented by the heuristics. BPH and MBOH presented null values for SD for all executions, BPH+LS presented values different from null in 11 instances and MBOH+LS in only 2 instances, while BOH presented values different from null in 166 instances and BOH+LS in 57 instances. The average execution time of the constructive heuristics BPH, MBOH and BOH for the results presented in Table 2 are respectively 74.48, 78.23 and 54.30 seconds. The use of local search increases the running time by only 30 seconds, as the heuristics generated good initial solutions.

The presented execution times are different for the greedy and the random algorithms because we calculated the average execution time using the results obtained only for the instances in which the algorithms presented different results. As the greedy algorithms presented different results in instances that demand more computational time, their average execution time is higher than the time obtained by the random algorithms.

A comparison was made between the algorithms using the greedy constructive heuristics and local search and the algorithms using the random constructive heuristics ($\alpha = 0.3$) and local search. GBPH+LS found 549 best solutions and BPH+LS found 557; GMBOH+LS found 555 best solutions and MBOH+LS found 557 and GBOH+LS found 550 while BOH+LS found 560.

As expected, when using α greater than zero the quality of the solution for some instances was improved. When using α different from 0, we introduce more diversification in the algorithm which may enlarge the possibility of obtaining good solutions (Feo and Resende, 1995).

In order to verify whether or not the differences of mean values obtained by the proposed heuristics using $\alpha = 0.3$ are statistically significant, we employed the unpaired Students t-test technique.

Table 4 presents, for each pair of heuristics and for each group of instances, the number of solutions that presents a p-value less than 0.01, which means that the probability of the difference of performance being due to random chance alone is less than 0.01. Between parentheses, there is a pair showing, for these solutions that are statistically significant, the number of better solutions found by the heuristic on the left side of the pair and by the heuristic on the right side.

Group	#Inst.	BPH+LS / MBOH+LS	BPH+LS / BOH+LS	MBOH+LS / BOH+LS
A1-U1	56	5(2/3)	2(2/0)	4(4/0)
A1-U2	34	0	0	0
A1-U4	28	0	0	0
A1-U8	11	0	0	0
A2-U2	32	2(0/2)	1(0/1)	2(1/1)
A4-U4	11	0	0	0
A8-U8	4	0	0	0
Total	176	7(2/5)	3(2/1)	6(5/1)

Table 4: t-test results

We can see that for the statistically significant results, the MBOH+LS heuristic shows better results than the other two algorithms.

Tests were performed with the proposed GRASP executing 30 iterations and the α parameter set to 0.3. For 494 instances from 560, we found the optimal solution using CPLEX. The constructive heuristics and local search algorithm presented in this paper were able to find all the known optimal solutions. So, we decided to run the GRASP heuristic for the 66 instances where the optimal solution is not known attempting to improve the solutions obtained so far. GRASP was executed ten times with ten different random number seeds.

Table 5 presents, for each heuristics and for each group of instances, the percentage difference between the best solution found by the heuristic and the best known solution.

For all 66 instances, only MBOH+LS and GRASP obtained the best solution among all known solutions.

Groups	#Inst	BPH(%)	BPH+LS(%)	BOH(%)	BOH+LS(%)	MBOH(%)	MBOH+LS(%)	GRASP(%)
A1_U1	19	0.56	0.10	0.19	0.10	0.19	0.00	0.00
A1_U2	16	0.00	0.00	0.00	0.00	0.00	0.00	0.00
A1_U4	7	0.00	0.00	0.00	0.00	0.00	0.00	0.00
A1_U8	7	0.00	0.00	0.00	0.00	0.37	0.00	0.00
A2_U2	12	0.21	0.11	0.11	0.11	0.00	0.00	0.00
A4_U4	5	0.00	0.00	0.00	0.00	0.00	0.00	0.00
A8_U8	0	-	-	-	-	-	-	-
Avg	66	0.13	0.03	0.05	0.03	0.09	0.00	0.00

Table 5: Comparison of best solution costs

Table 6 presents, for each heuristic and for each group of instances, the percentage difference between the average solution found by the heuristic and the best known solution.

For all 66 instances, only GRASP obtained the best solution among all known solutions.

The average time of GRASP for the tested instances is 745.9 seconds.

Groups	#Inst	BPH(%)	BPH+LS(%)	BOH(%)	BOH+LS(%)	MBOH(%)	MBOH+LS(%)	GRASP(%)
A1_U1	19	0.56	0.34	0.19	0.17	1.09	0.39	0.00
A1_U2	16	0.00	0.00	0.00	0.00	0.17	0.02	0.00
A1_U4	7	0.00	0.00	0.00	0.00	0.29	0.05	0.00
A1_U8	7	0.00	0.00	0.00	0.00	0.40	0.00	0.00
A2_U2	12	0.21	0.20	0.11	0.11	0.17	0.09	0.00
A4_U4	5	0.00	0.00	0.00	0.00	0.07	0.00	0.00
A8_U8	0	-	-	-	-	-	-	-
Avg	66	0.13	0.09	0.05	0.05	0.36	0.09	0.00

Table 6: Comparison of average solution costs

In the experiments, we observe that the GRASP solution was obtained in one of the three initial iterations. So, we can consider that, for this group of instances, a simple combination between construction and local search heuristics is sufficient to obtain good results.

5 Conclusions

In this work, three constructive heuristics, a local search algorithm and a GRASP strategy were presented for the Set Covering with Pairs Problem (SCPP). To analyze the performance of the proposed algorithms, one set of instances was used composed by 560 unit cost instances (Resende, 2007).

Computational results showed that the three proposed constructive greedy algorithms were able to find many optimal solutions. The heuristic GMBOH+LS obtained the best results.

The experiments conducted using the random constructive algorithms showed that these algorithms were able to find better results than the greedy algorithms and that the local search was able to improve the results for all three random constructive heuristics.

Results obtained for average values showed that the heuristics based on Best Pair Heuristic (BPH) and on Modified Best Object Heuristic (MBOH) presented more robust results than the ones based on Best Object Heuristic (BOH). Statistics tests also showed that MBOH+LS heuristic presents more statistically significant results than the other heuristics.

We developed a GRASP heuristic combining the proposed constructive heuristics and the local search algorithm. The GRASP and MBOH+LS heuristics found the best cost solutions for all instances and GRASP found all best average solutions.

References

Catanzaro, D. and Labbé, M.: 2009, The pure parsimony haplotyping problem: overview and computational advances, *International Transactions in Operational Research* **16**(5), 561 – 584.

Chvátal, V.: 1979, A greedy heuristic for the set-covering problem, *Mathematics of Operations Research* **4**, 233–235.

Feo, T. and Resende, M.: 1995, Greedy randomized adaptive search procedures, *J. of Global Optimization* **6**, 109–133.

Fernandes, R. J. and Skiena, S. S.: 2002, Microarray synthesis through multiple-use pcr primer design, **18**, S128 – S135.

Gusfield, D.: 2003, Haplotype inference by pure parsimony, *Combinatorial Pattern Matching*, Vol. 2676 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 144–155.

Hajiaghayi, M. T., Jain, K., Lau, L. C., Mandoiu, I. I., Russell, A. and Vazirani, V. V.: 2006, Minimum multicolored subgraph problem in multiplex pcr primer set selection and population haplotyping, *In Proceedings of the 6th International Conference on Computational Science (ICCS)*, pp. 758–766.

Hassin, R. and Segev, D.: 2005, The set cover with pairs problem, *FSTTCS 2005: Proceedings of the 25th International Conference on Foundations of Software Technology and Theoretical Computer Science*, Vol. 3821 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 164–176.

Hermelin, D., Rawitz, D., Rizzi, R. and Vialette, S.: 2008, The minimum substring cover problem, *Information and Computation* **206**, 1303–1312.

Resende, M. G. C.: 2007, Private commmunication.

Resende, M. and Ribeiro, C.: 2005, GRASP with path-relinking: Recent advances and applications, *in* T. Ibaraki, K. Nonobe and M. Yagiura (eds), *Metaheuristics: Progress as Real Problem Solvers*, Springer, pp. 29–63.