Optim Lett manuscript No.
(will be inserted by the editor)

Exact and Heuristic Approaches for the Set Cover with Pairs Problem

Luciana Brugiolo Gonçalves · Simone de Lima Martins · Luiz Satoru Ochi · Anand Subramanian

Received: date / Accepted: date

Abstract This work deals with the Set Cover with Pairs Problem (SCPP) which is a generalization of the Set Cover Problem (SCP). In the SCPP the elements have to be covered by specific pairs of objects, instead of a single object. We propose a new mathematical formulation using extended variables that is capable of consistently solve instances with up to 500 elements and 500 objects. We also developed an ILS heuristic which was capable of finding better solutions for several tested instances in less computational time.

Keywords Set Cover With Pairs · mathematical formulation · integer programming · ILS metaheuristic

1 Introduction

The Set Cover with Pairs Problem (SCPP) was proposed in [11] and it is considered to be a generalization of the well-known Set Cover Problem (SCP) [16, 18], in which the elements have to be covered by specific pairs of objects, rather than a single object.

The SCPP can be formally defined as follows. Let U be a ground set of elements to be covered and let A be a set of objects, where each object $i \in A$ has a non-negative cost c_i. For every pair $\{i, j\} \subseteq A$, let $\mathscr{C}(i, j)$ be the collection of elements in U covered by the pair $\{i, j\}$. The objective is to find a subset $S \subseteq A$ such that $\bigcup_{\{i,j\}\subseteq S} \mathscr{C}(i, j) = U$ and $\sum_{i\in S} c_i$ is minimized.

As presented in [11], considering an instance where $U = \{e_1, \ldots, e_n\}$ and $A = \{A_1, \ldots, A_m\}$, such that A_l is a subset of U for $l = \{1, \ldots, m\}$, the SCPP can be interpreted as an SCP instance by letting $\mathscr{C}(i, j) = A_i \cup A_j$ for every $i \neq j$. Therefore, the

L. B. Gonçalves E-mail: lgoncalves@ic.uff.br · S. L. Martins E-mail: simone@ic.uff.br · L. S. Ochi E-mail: satoru@ic.uff.br · A. Subramanian E-mail: asubramanian@ic.uff.br
Instituto de Computação - Universidade Federal Fluminense (UFF)
Rua: Passo da Pátria, 156 - Niterói, RJ - Brazil

SCPP is a generalization of the SCP. The hardness results regarding the SCP can be extended to the SCPP which, in turn, is known to be NP-hard [7]. Another problem that can be considered as an SCPP instance is the k-Cover Problem when $k = 2$. In this case, the cover function is written as $\mathscr{C}(i, j) = A_i \cap A_j$ for every $i \neq j$.

Some recent biological problems can also be treated as the SCPP. The Pure Parsimony Haplotyping Problem (PPHP) is a biological 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 by a pair from H. Some exact and heuristic algorithms for solving this problem were developed in [5].

Another biochemical problem that can be modeled as the 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. The PCR as well as heuristics and approximation algorithms for this problem were described in [6] and [10].

Hermelin et al [12] 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 can be written as a concatenation of at most l strings in $\mathscr{C}$, $\mathscr{C}$ is defined as an l-cover of S. This problem is closely related to SCPP when $l = 2$.

The first mathematical formulation used to describe the SCPP was presented in [9] by means of an example. This formulation was formally described in [4].

Another mathematical formulation for the SCPP was presented in [11]. The same was employed in [3] to represent the Minimum Monitoring Set Problem. Also, it had been already used [14] to describe the Haplotyping Inference Problem. Hence, this formulation is going to be hereafter referred as Lancia formulation [14] .

In this work we present a mathematical formulation for the SCPP, which can be seen as a special case of the one developed in [2] for the Haplotyping Inference Parsimony Problem (HIPP). However, due to the NP-hardness of SCPP, the use of the developed formulation is still restricted to instances with a limited number of elements and objects. Therefore, we also propose a heuristic algorithm, based on the Iterated Local Search (ILS) metaheuristic, in order to obtain good quality solutions in a reasonable computational time.

The remainder of the paper is organized as follows. Section 2 describes the mathematical formulation presented in [14]. Section 3 presents an improved extended formulation for the SCPP as well as the the results obtained by both formulations. Section 4 describes the proposed ILS heuristic and a comparison, in terms of solution quality and computational time, between the formulations and the heuristic approach. Section 5 presents the concluding remarks of this work.

2 The Lancia Formulation (F1)

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 c_i. For an element $e \in U$, define P_e as the set of pairs $\{i, j\} \subseteq A$ where $\mathscr{C}(i, j) \supset \{e\}$. Let x_i be 1 if $i \in A$ is chosen, 0 otherwise. In addition, for each

pair $\{i, j\} \subseteq A$, $i < j$, let y_{ij} be 1 if i and j are both chosen and 0 otherwise. The integer programming (IP) formulation F1 presented in [14] is as follows.

$$\text{minimize} \sum_{i \in A} c_i x_i, \tag{1}$$

Subject to:

$$\sum_{\{i,j\} \in P_e} y_{ij} \geq 1, \ \ e \in U \tag{2}$$

$$y_{ij} \leq x_i, \ \ \forall \{i, j\} \subseteq A \tag{3}$$

$$x_i \in \{0, 1\}, \ \ \forall i \in A \tag{4}$$

$$y_{ij} \in \{0, 1\}, \ \ \forall \{i, j\} \subseteq A \tag{5}$$

The objective function (1) minimizes the sum of the costs of the selected objects. Constraints (2) state that for each element $e \in U$ we select at least one pair of objects that covers it. Constraints (3) ensure that a pair of objects cannot cover any element unless $i \in A$ and $j \in A$ are in the solution set. Constraints (4) and (5) define the domain of the variables.

3 The Extended Formulation (F2)

In this section, we propose an improved mathematical formulation for the SCPP, which is based on the one proposed in [9] for the Haplotyping Inference Problem. Let the extended variables w_{ije} be 1, if the pair $\{i, j\} \in P_e$ is the one chosen to cover e and 0 otherwise. Also, consider $T_e \subseteq A$ as the set of objects such that $T_e = \cup_{\{i,j\} \in P_e} \{i, j\}$. The extended IP formulation of [9] can be expressed as follows.

$$\text{minimize} \sum_{i \in A} c_i x_i \tag{6}$$

Subject to:

$$\sum_{\{i,j\} \in P_e} w_{ije} = 1, \ \ \forall e \in U \tag{7}$$

$$w_{ije} \leq x_i, \ \ \forall e \in U, \ \forall \{i, j\} \subseteq P_e \tag{8}$$

$$w_{ije} \leq x_j, \ \ \forall e \in U, \ \forall \{i, j\} \subseteq P_e \tag{9}$$

$$x_i \in \{0, 1\}, \ \ \forall i \in A \tag{10}$$

$$w_{ije} \in \{0, 1\}, \ \ \forall e \in U, \ \forall \{i, j\} \subseteq P_e \tag{11}$$

The objective function (6) is the same of F1. Constraints (7) ensure that each element $e \in U$ must be covered by exactly one pair $\{i, j\} \in P_e$. Constraints (8)-(9)

state that if w_{ije} is set to one then x_i and x_j must also be set to one. Constraints (10) and (11) define the domain of the variables.

The formulation of [9] can be improved by replacing inequalities (8)-(9) with (12), i.e.:

$$\sum_{j \in T_e | i<j, \{i,j\} \in P_e} w_{ije} + \sum_{j \in T_e | j<i, \ \{i,j\} \in P_e} w_{jie} \leq x_i, \ \ \forall e \in U, \ \forall i \in T_e \tag{12}$$

It is clear that inequalities (12) dominate (8)-(9). Thus let F2 be the IP formulation composed by (6)-(7), (10)-(11) and (12).

It should be pointed out that the valid inequalities (13) can be added to both F1 and F2. These inequalities indicate that two objects $i \in T_e$ must be selected for each element $e \in U$.

$$\sum_{i \in T_e} x_i \geq 2 \ \ e \in U \tag{13}$$

Theorem 1 *The value obtained by the LP relaxation of F2 with constraints (13) is stronger than the one obtained by F1 with (13).*

Proof Given the solution (x^*, w^*) with cost z^*_{F2} obtained by the LP relaxation of F2 (LP-F2), it is possible to build a feasible solution of the LP relaxation of F1 with the same cost.

The variables x_i are common to both formulations and the relationship between y_{ij} and w_{ije} is defined in the Equation (14).

$$y_{ij} = \max_{e \in U} \{w_{ije}\} \tag{14}$$

Constraints (2) are satisfied because if $\sum_{\{i,j\} \in P_e} w_{ije} = 1, e \in U$ and, given the relationship defined in (14), we conclude that $\sum_{\{i,j\} \in P_e} y_{ij} \geq 1$.

Constraints (3) are also satisfied because given (8), which implies in $w_{ije} \leq x_i$, $\forall e \in U$ and $\forall \{i,j\} \subset P_e$ we obtain that $\max_{e \in U} \{w_{ije}\} \leq x_i$ and considering (14) one can verify that $y_{ij} \leq x_i$ (3).

On the other hand, it is not always possible to build a feasible solution for the LP relaxation of F2 in terms of (x, w) given the solution (x^*, y^*) with cost z^*_{F1} obtained by the LP relaxation of F1 (LP-F1). More precisely, $z_{F2} \geq z^*_{F1}$ for any feasible solution of LP-F2. Let us consider, for example, an instance I, which is composed by a single element to be covered $U = \{4\}$ and three possible objects $A = \{1,2,3\}$ with $P_4 = \{\{1,2\},\{1,3\}\}$, where the costs of the elements in A correspond to $c_1 = 10$, $c_2 = 4$ e $c_3 = 1$. For this instance, the formulations F1 and F2 can be written as follows.

$$\text{minimize } Z_{F1}\text{: } 10x_1 + 4x_2 + x_3 \tag{15}$$

Subject to

$$y_{12} + y_{13} \geq 1 \tag{16}$$

$$y_{12} \leq x_1 \tag{17}$$

$$y_{12} \leq x_2 \tag{18}$$
$$y_{13} \leq x_1 \tag{19}$$
$$y_{13} \leq x_3 \tag{20}$$
$$x_1 + x_2 + x_3 \geq 2 \tag{21}$$
$$x_i \in \{0,1\}, \forall i \in A \tag{22}$$
$$y_{ij} \in \{0,1\}, \forall \{i,j\} \subset A \tag{23}$$

$$\text{minimize } Z_{F2}\text{: } 10x_1 + 4x_2 + x_3 \tag{24}$$

Subject to

$$w_{124} + w_{134} = 1 \tag{25}$$
$$w_{124} + w_{134} \leq x_1 \tag{26}$$
$$w_{124} \leq x_2 \tag{27}$$
$$w_{134} \leq x_3 \tag{28}$$
$$x_1 + x_2 + x_3 \geq 2 \tag{29}$$
$$x_i \in \{0,1\}, \forall i \in S \tag{30}$$
$$w_{ije} \in \{0,1\}, \forall e \in U, \forall \{i,j\} \subset P_e \tag{31}$$

The solution obtained using LP-F1 is $x_1^* = 0.5$, $x_2^* = 0.5$, $x_3^* = 1$, $y_{1,2}^* = 0.5$ and $y_{1,3}^* = 0.5$ with $Z_{LP-F1}^* = 8$. Note that is not possible to build a solution with the same cost for the LP relaxation of F2 since from (25) and (26), we can verify that $x_1 \geq w_{124} + w_{134} = 1$, which necessary implies $x_1 = 1$ in F2. The cost of the object 1 is $c_1 = 10$ which indicates that $Z_{LP-F2} \geq 10$ and therefore $Z_{LP-F2} > Z_{LP-F1}^* = 8$.

The solution obtained using LP-F2 corresponds to $x_1^* = 1$, $x_2^* = 0$, $x_3^* = 1$, $w_{124}^* = 0$ and $w_{134}^* = 1$ with cost $Z_{LP-F2}^* = 11$. This is also the optimal solution for the instance I because the values of the variables are integer.

3.1 Computational Experiments

The mathematical formulations were tested using the solver CPLEX 11.2 [13] and executed in an Intel(R) Core(TM)2 Quad CPU Q9550 with 2.83GHz CPU and 8GB of RAM running under Linux 64 bits (kernel 2.6.27-16).

To evaluate the formulations, two set of instances were created. The first set G_1 is composed by 60 instances which were generated from 20 instances containing more than 500 elements and originally developed for the SCP (available in [1]). Ten instances were used to generate SCPP instances with 300 objects and the other 10 to generate SCPP instances with 500 objects. For each of these 20 instances, three instances were created with different element sets which were defined by randomly selecting 25% (I25 instances), 50% (I50 instances) and 100% (I100 instances) elements from the SCP instance.

The second group of instances G_2 is also derived from instances originally developed for the SCP available in [1], but they are generated in a different way to that used for generating G_1 instances. We used 53 SCP instances where for each line to be covered there is a list of columns L that can cover it. In order to generate an SCPP instance, all combinations of two columns from L are considered to be pair of objects which may cover an element associated to the line. For each line, the parameter $p \in [0,100]$ is used to select the percentage of the number of pairs that will not be included in the new instance. Each SCP instance originated three SCPP instances that were obtained using the following values for p: 25%, 50% and 75%. Hence, three sets of instances were generated: scp_25, scp_50 e scp_75, resulting in a total of 159 instances.

All instances are available at `http://labic.ic.uff.br/Instance`.

For the first group of instances, the Extended Formulation (F2) was executed until the optimal solution was found. Considering $\delta_{F2}(i)$ as the time spent using F2 to obtain the optimal solution of an instance i, the time limit established for the Lancia Formulation (F1) was set to be $\Delta_{F1}(i) = \lceil \delta_{F2}(i) \times 5 \rceil$.

In the following tables, **Group** denotes the group of instances, **LR** represents the average gap between the linear relaxation and the Best Known Solution (BKS), **R-LB** corresponds to the average gap between the root relaxation and the BKS and **B-LB** indicates the average gap between the IP solution and the BKS. Also, the CPU time in seconds is respectively presented for these three cases. Finally, we provide the average results obtained for the instances of each group.

The results presented in Table 1 show that the bounds obtained using F2 were considerably stronger than those obtained when using F1. In addition, when using F2, the optimal solution for all instances of this group were obtained, while, when using F1, no optimal solution was found and the given time limit was not even enough to process the root node. The computational time spent to solve the linear relaxation of F1 is much larger than the one spent to solve the linear relaxation of F2. Also, it can be observed that the average gap between the linear relaxation and the best know solution was considerably larger when using F1 (98.46%) when compared to the one obtained by F2 (0.08%).

Table 1 Results for G_1 instances

Group	Gusfield Model - F1						Extended Model - F2					
	gap (%)			time (s)			gap (%)			time (s)		
	LR	R-LB	B-LB	LR	Root	Total	LR	R-LB	B-LB	LR	Root	Total
I100	99.00	81.67	81.67	2600.09	666.13	666.13	0.08	0.01	0.00	3.86	123.70	125.06
I50	98.59	92.55	92.55	398.12	220.47	220.47	0.05	0.03	0.00	3.09	39.47	39.79
I25	97.78	96.58	96.58	164.42	104.99	104.99	0.11	0.06	0.00	4.58	18.63	18.83
Avg.	98.46	90.26	90.26	1054.21	330.53	330.53	0.08	0.03	0.00	3.84	60.60	61.23

Table 2 presents the results obtained for the G_2 instances. In order to obtain results in reasonable time, a time limit of 10 hours was established. Both formulations failed to conclude their execution due to lack of memory for 10 instances (5 from the scp_25

group and 5 from the scp_50 group). Therefore, we have only considered the results obtained for 149 instances.

The Extended Formulation (F2) was capable of finding the optimal solution for 85 instances, whereas the Lancia Formulation (F1) obtained the optimal solution for only 11 instances, within the time limit established. For the instances with higher amount of pairs (group scp_25), F1 was not successful to find the optimal solutions for all instances. From Table 2, it is clear that F2 is highly superior than F1, since it produced consistently better results in every measure considered by this study.

Table 2 Results for G_2 instances

Group	Gusfield Model - F1						Extended Model - F2					
	gap (%)			time (10^3s)			gap (%)			time (10^3s)		
	LR	R-LB	B-LB	LR	Root	Total	LR	R-LB	B-LB	LR	Root	Total
scp_25	14.83	12.36	11.75	7.37	11.54	36.00	9.64	9.08	7.19	1.29	2.81	15.02
scp_50	20.45	15.67	15.43	7.80	13.52	32.32	9.64	9.47	7.29	0.98	1.56	14.18
scp_75	24.78	21.39	21.07	9.58	12.88	31.92	13.74	13.66	11.58	2.88	3.97	18.62
Avg.	20.02	16.47	16.08	8.25	12.65	33.41	11.01	10.74	8.68	1.72	2.78	15.94

4 Iterated Local Search Heuristic

Although the developed extended formulation F2 was capable of finding some optimal solutions, several instances were not solved within the time limit of 10 hours. This was somehow expected since the SCPP is NP-Hard. Therefore, we have proposed a heuristic procedure in order to obtain good quality solutions in a reasonable computational time.

The developed solution approach is based on the ILS metaheuristic, which consists of a procedure that explores the neighborhood of a solution by applying perturbations in the incumbent solutions to escape from local optima. This metaheuristic has been successfully applied to solve many combinatorial optimization problems as pointed out in [15]. Recently, some very good results were obtained by our research group using ILS to solve the vehicle routing problem with simultaneous pickup and delivery [17].

There are four components to consider when implementing an ILS heuristic [15]: (*i*) a procedure to generate an initial solution; (*ii*) local search algorithms; (*iii*) a perturbation strategy; and (*iv*) an acceptance criterion.

Algorithm 1 shows the basic structure of ILS. An initial solution S_0 is obtained by a constructive algorithm (line 1), and a local search is applied to S_0 in line 2. While the termination condition is not met, in each iteration, a perturbation is applied in solution S^* (line 4) and the neighborhood of this new solution is explored by a local search algorithm (line 5). If the acceptance criterion is satisfied, the solution obtained after the local search is set as the new current solution.

A fundamental issue to obtain good solutions when implementing an ILS heuristic is the definition of the perturbation strategy. If the perturbation is too small, i.e.,

Algorithm 1 *Iterated Local Search*

```
1: S_0 = Constructive_Algorithm( );
2: S* = Local_Search (S_0);
3: repeat
4:     S' ← Perturbation(S*);
5:     S'* ← Local_Search (S');
6:     S* ← Acceptance_Criterion (S*, S'*);
7: until (termination condition met);
8: Return (S*);
```

it does not perform considerable changes in the current solution, the local search applied in this new solution may return to the previous solution, so few solutions of the search space are explored.

Algorithm 2 shows the ILS heuristic developed in this work.

Algorithm 2 *Iterated Local Search for SCPP*

```
1: S_0 = ADH( );
2: S* = DALS (S_0, γ, k);
3: repeat
4:     S' ← Perturbation(S*);
5:     S'* ← COLS(S');
6:     S* ← Acceptance_Criterion (S*, S'*);
7: until (best solution is not updated for impr_iter iterations);
8: Return (S*);
```

A greedy constructive procedure called Add Drop Heuristic (ADH) was developed for building the initial solution (line 1). In line 2, the procedure Drop Add Local Search (DALS) is performed after the initial solution is built. Then, a perturbation is carried on in line 4 and the neighborhood of this solution is explored by the procedure Change Object Local Search (COLS) in line 5. The acceptance criterion is evaluated in line 6.

Algorithm 3 describes the steps performed by ADH. In line 1, the solution S is initialized containing all objects of A. Next, the algorithm verifies the possibility of removing each element of S. In line 2, the objects $a \in A$ are sorted in ascending order according to their associated cost c_a and stored in the ordered set A'. In line 4, the first object a is selected from A' and removed from A'. Then, in line 5, the procedure Verify_feasibility checks if deleting object a from S will cause the uncovering of some elements from U. If all elements from U are still covered after removing a, then it is removed from S in line 7. The algorithm terminates when there are no more candidates to be removed in set A' (line 3).

Algorithm 4 presents the pseudocode of DALS. The algorithm receives three input parameters: the initial solution S obtained by ADH, a constant $\gamma \in [0,1]$ and a positive integer k. The first phase of the algorithm consists in randomly selecting $\gamma \times |S|$ objects to be removed from the current solution S, which are stored in Q. The elements which are deleted are stored in a tabu list lt to disable their removal from the solution in the next iterations (line 3). Then, the elements from Q are deleted

Algorithm 3 *ADH*

```
1: S ← A;
2: A′ ← Sort (A);
3: while (A′ ≠ ∅) do
4:     a ← Choose_object_drop(A′);
5:     answer ← Verify_feasibility(a);
6:     if (answer = true) then
7:         S ← S\{a};
8:     end if
9: end while
```

from S (line 4), and the solution becomes unfeasible, because some elements from U turn to be uncovered. To generate a feasible solution S'', a rebuilding procedure is performed considering just the uncovered elements from U (line 5). The rebuilding procedure selects the objects to be included in the solution so that they cover these uncovered elements with minimum cost. The best solution S^* found is updated in line 7, if the solution S'' presents a smaller cost. The iterations are performed until k iterations are performed without improving S^*. The perturbation strategy consists in

Algorithm 4 *DALS* (S, γ, k)

```
1: S* ← S;
2: while (num_iter_without_improvement < k) do
3:     Q ← Select_objects( S, γ, lt );
4:     S ← S\Q;
5:     S'' ← Rebuild_solution (S);
6:     if ( cost(S'') < cost(S*) ) then
7:         S* ← S'';
8:     end if
9:     S ← S*;
10: end while
```

removing $\alpha \times |S'|, \alpha \in [0,1]$ objects from the current solution S' and rebuilding it using two constructive heuristics developed in [8]. Each time one of them is randomly selected. The first one, called Best Pair Heuristic (BPH), is based on the cheapest insertion strategy. At each construction iteration, the algorithm chooses the best pair of objects to cover a particular element. The second constructive heuristic, called Modified Best Object Heuristic (MBOH), works as follows. In each construction iteration, the algorithm chooses the best object not yet inserted in the solution according to its contribution in covering the elements.

The Change Object Local Search (COLS) heuristic developed to be used during the ILS iterations is shown in Algorithm 5. The neighborhood evaluated by this algorithm is defined by deleting one object from the solution S and rebuilding a new solution not using this object. The best solution S^* obtained by the local search procedure is initialized in line 1. In each iteration, from lines 2 to 14, S'' stores the best solution found in neighborhood of S^* and S' represents a temporary solution. To find the best solution in the neighborhood of S^*, each object of the current solution is deleted (line 5) and a new solution is build using a constructive heuristic, which is

randomly selected between BPH and MBOH described before. A penalization is applied in the removed object so that it can not be reinserted in the rebuilt solution (line 6). This is a best-improving strategy, so at each iteration all neighborhood is evaluated (lines 4 to 10) and S'' stores the best solution found in the neighborhood of S^*. If a solution S'' presents smaller cost than the current best solution S^*, the best solution S^* is updated in line 12, and the local search is restarted in the neighborhood of the new solution S^*. The algorithm terminates when all neighborhood is searched and no better solution than S^* is found.

Algorithm 5 *COLS* (S)

```
 1: S* ← S;
 2: while (there is improvement in S*) do
 3:     S'' ← S*;
 4:     for all object j ∈ S* do
 5:         S' ← S*\{j};
 6:         S' ← Rebuild_with_penalization (S', j);
 7:         if (cost(S') < cost(S'')) then
 8:             S'' ← S';
 9:         end if
10:     end for
11:     if (cost(S'') < cost(S*)) then
12:         S* ← S'';
13:     end if
14: end while
```

The solution obtained after executing the procedure COLS is accepted as the new current solution if its cost is less than β times the current solution cost.

The ILS procedure terminates when the best solution is not updated for *impr_iter* iterations.

4.1 Computational Results

The ILS heuristic was coded in C++ and was implemented using the same machine described in Subsection 3.1. After executing some preliminary experiments for tuning parameters, the γ parameter which defines the percentage of objects to be removed in the perturbation strategy was set to 0.30 and the β parameter used in the acceptance criterion was set to 1.05.

Two experiments were performed adopting two different termination criteria. In the first one, the ILS procedure terminates when the best solution was not updated for 10 consecutive iterations and in the second one, the ILS procedure terminates after a time limit. For both experiments, the algorithm was executed 10 times for each instance using in each time a different seed for generating the random numbers.

Table 3 presents the results obtained for G_2 instances, for which the extended formulation F2 was not able to find all optimal solutions for all instances. The column Extended Model - F2 denotes the average gap between the solution found by F2 and the best solution found by F2 or ILS and the average computational time for finding

the solution using F2. The column ILS-iter corresponds to the average gap between the solution found by ILS adopting the first termination criterion and the best solution found by F2 or ILS and the average computational time and the column ILS-time represents the gap and computational time for the solution found by ILS using the second termination criterion. These results show that the ILS heuristic was capable of obtaining better solutions in less computational time.

Table 3 Results for the G_2 instances - F2 $\times$ ILS

Group	Extended Model - F2		ILS-iter		ILS-time	
	gap(%)	time(10^3s)	gap(%)	time(10^3s)	gap(%)	time(10^3s)
scp_25	2.16	15.02	4.30	0.91	1.34	1.90
scp_50	2.32	14.18	4.87	0.52	1.59	1.64
scp_75	1.04	18.62	4.04	0.26	1.37	1.36
Avg.	1.84	15.94	4.40	0.57	1.43	1.63

Another experiment was performed to compare the results obtained by formulations F1 and F2 and those found by the ILS heuristic. In this experiment, the execution time limit set for both formulations F1 and F2 was of 10 hours, while the one set for the ILS heuristic was of 1 hour. Figures 1 and 2 illustrate, respectively, the results obtained for instances scp410_25 and scpsclr10_75. The vertical axis corresponds to the solution cost whereas the horizontal axis represents the execution time.

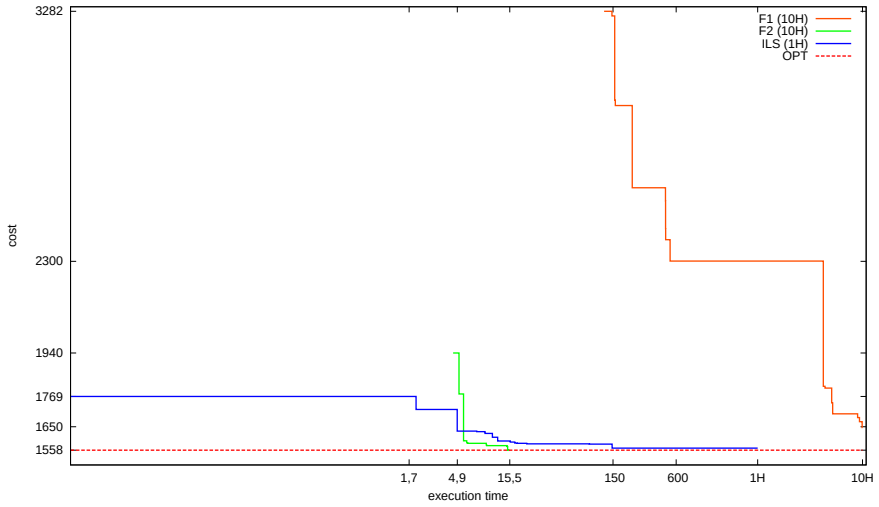

Fig. 1 Comparing results obtained by F1, F2 and ILS for instance scp410_25

The optimal solution of the instance scp410_25 is known and its value is 1558, as shown in Figure 1. This solution was obtained after 15.5 seconds using F2. Both F1 and the ILS heuristic were not able to find this solution in the given time limits. After 10 hours, the best solution found by F1 presented a gap of 5.71% from the optimal

solution. The ILS heuristic found a solution with a gap of 0.50% after 147 seconds but it was not able to further improve it within the established time limit of 1 hour.

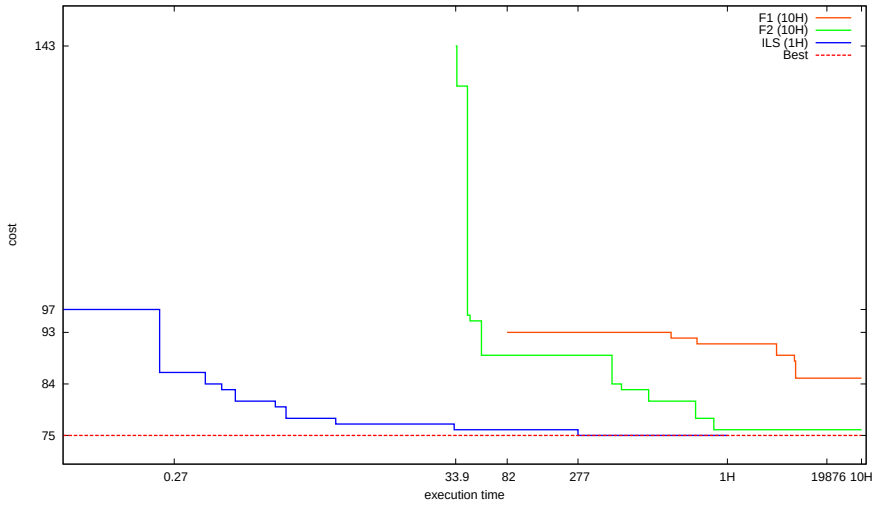

Fig. 2 Comparing results obtained by F1, F2 and ILS for instance scpsclr10_75

From Figure 2, it can be seen that we were not able to find an optimal solution for instance scpsclr10_75 when using both F1 and F2, in the time limit of 10 hours. The best solution cost obtained by F1 and F2 were respectively 85 and 76. The ILS heuristic obtained, in 277 seconds, the best solution with an associated cost of 75.

These results show that the proposed formulation F2 improves the results obtained by F1 as already presented in the previous section, and also that the ILS heuristic was able to improve the quality of the solutions obtained by F2 in much less computational time.

5 Concluding Remarks

This work proposed an extended mathematical formulation for the Set Cover with Pairs Problem (SCPP) which is stronger than the one proposed in [9]. Experimental tests were carried out to evaluate the performance of both formulations in instances with up to 500 objects. The results showed that the extended formulation consistently obtained more optimal solutions and consumed less computational time. We also developed an ILS heuristic which were able to find better quality results in less computational time when compared to the results obtained by the extended mathematical formulation.

Acknowledgements We would like to thank Artur Alves Pessoa for the valuable discussions concerning the extended mathematical formulation presented in this paper.

Research partially supported by Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro (FAPERJ: E-26/100.538/2008) and Conselho Nacional de Pesquisa (CNPQ).

References

1. Beasley J (1990) Or-library: Distributing test problems by electronic mail. Journal of the Operational Research Society 41(11):1069–1072
2. Bertolazzi P, Godi A, Labbé M, Tininini L (2008) Solving haplotyping inference parsimony problem using a new basic polynomial formulation. Comput Math Appl 55(5):900–911, DOI http://dx.doi.org/10.1016/j.camwa.2006.12.095
3. Breslau L, Diakonikolas I, Duffield NG, Gu Y, Hajiaghayi M, Johnson DS, Karloff H, Resende MGC, Sen S, Towsley D (2007) Optimal node placement for path disjoint network monitoring. Tech. rep., AT&T Labs Research, NJ, USA
4. Brown DG, Harrower IM (2006) Integer programming approaches to haplotype inference by pure parsimony. IEEE/ACM Transactions on Computational Biology and Bioinformatics 3:141–154, DOI http://doi.ieeecomputersociety.org/10.1109/TCBB.2006.24
5. Catanzaro D, Labbé M (2009) The pure parsimony haplotyping problem: overview and computational advances. International Transactions in Operational Research 16(5):561–584
6. Fernandes RJ, Skiena SS (2002) Microarray synthesis through multiple-use PCR primer design. Bioinformatics 18:S128–S135
7. Garey MR, Johnson DS (1979) Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA
8. Gonçalves LB, Martins SL, Ochi LS (2010) Effective heuristics for the set covering with pairs problem. International Transactions in Operational Research 17:739–751
9. Gusfield D (2003) Haplotype inference by pure parsimony. In: Combinatorial Pattern Matching, Lecture Notes in Computer Science, vol 2676, Springer Berlin / Heidelberg, pp 144–155
10. Hajiaghayi MT, Jain K, Lau LC, Mandoiu II, Russell A, Vazirani VV (2006) Minimum multicolored subgraph problem in multiplex PCR primer set selection and population haplotyping. In: In Proceedings of the 6th International Conference on Computational Science (ICCS), pp 758–766
11. Hassin R, Segev D (2005) The set cover with pairs problem. In: FSTTCS 2005: Proceedings of the 25th International Conference on Foundations of Software Technology and Theoretical Computer Science, Springer Berlin / Heidelberg, Lecture Notes in Computer Science, vol 3821, pp 164–176
12. Hermelin D, Rawitz D, Rizzi R, Vialette S (2008) The minimum substring cover problem. Information and Computation 206:1303–1312
13. Ilog, Inc (2009) Solver cplex 11.2, ilog concert technolog. URL http://www.ilog.com/products/cplex/, accessed 15 May 2009
14. Lancia G, Pinotti MC, Rizzi R (2004) Haplotyping populations by pure parsimony: Complexity of exact and approximation algorithms. INFORMS J on Computing 16(4):348–359, DOI http://dx.doi.org/10.1287/ijoc.1040.0085

15. Lourenço HR, Martin OC, Stützle T (2003) Iterated local search. In: GLOVER F, KOCHENBERGER G (eds) Handbook of Metaheuristics, vol 57, Springer - Kluwer Academic Publishers, New York, pp 320–353
16. Pardalos PM, Du D (eds) (1998) Network Design: Connectivity and Facilities Location, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol 40. American Mathematical Society
17. Subramanian A, LMDrummond, CBentes, LSOchi, RFarias (2010) A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. Computers & Operations Research 37(11):1899–1911
18. Vemuganti R (1998) Applications of set covering, set packing and set partitioning modes: a survey. In: Pardalos PM, Du D (eds) Handbook of Combinatorial Optimization, vol 1, Kluwer Academic Publishers, pp 573–746