Contents lists available at SciVerse ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Production, Manufacturing and Logistics

Design and analysis of hybrid metaheuristics for the Reliability p-Median Problem

Javier Alcaraz *, Mercedes Landete, Juan F. Monge

Institute Center of Operations Research, Universidad Miguel Hernández, Elche (Alicante), Spain

ARTICLE INFO

Article history:
Received 27 February 2011
Accepted 15 April 2012
Available online 28 April 2012

Keywords:
p-Median problem
Metaheuristics
Genetic algorithms
Scatter search

ABSTRACT

In the p-Median Problem, it is assumed that, once the facilities are opened, they may not fail. In practice some of the facilities may become unavailable due to several factors. In the Reliability p-Median Problem some of the facilities may not be operative during certain periods. The objective now is to find facility locations that are both inexpensive and also reliable. We present different configurations of two hybrid metaheuristics to solve the problem, a genetic algorithm and a scatter search approach. We have carried out an extensive computational experiment to study the performance of the algorithms and compare its efficiency solving well-known benchmark instances.

1. Introduction

The p-Median Problem (pMP) is one of the problems to have received most attention from researchers in discrete location theory. It was first introduced by [23]. In general, we have to decide which p plants or facilities should be opened in order to serve a set of customers or demand points. The objective is to minimize the total day-to-day transportation costs. It is a combinatorial optimization NP-Hard problem, as was shown by [28]. The first formulation of the problem, and the most commonly used, was given by [46], and some other more recent formulations can be found in [6,10,11]. To solve this problem, several exact, heuristic and metaheuristic techniques have been proposed.

Reese [39] gives an extensive bibliography on solution methods for the uncapacitated and capacitated p-median problem and a good survey of metaheuristic approaches for the p-median problem is presented in [35]. More recently, different exact, heuristic and metaheuristic methods have been proposed to solve this problem [10,13,17,11,4,38,14].

In the pMP, the facilities will always operate as planned, i.e., the facilities may not fail. In real life, one or more facilities or services may become unavailable for some time, for different reasons. In this case, the customers assigned to those facilities should be re-assigned to some other different facilities, with the corresponding increase in costs. The optimal solution could result in excessive transportation costs as customers previously served by these facilities must now be served by others further away. If we consider the possibility of facility failures, we might prefer a more expensive but also more reliable solution. This reliability approach was introduced in the facility location literature by [42]. In the Reliability p-Median Problem (RpMP), not only the transportation cost but also the expected failure cost is considered and should be minimized. Different authors have approached this facility location problem from three different points of view: network reliability, backup covering models and supply chains. In [42] we can find a review of the literature with a description of these approaches.

Let I be the set of clients and J the set of potential facility locations. Let NF be the subset of J which may not fail and F the subset which may fail, $J = F \cup NF$ and $F \cap NF = \emptyset$. For each $i \in I$ a demand h_i is required and the service of each unit from $j \in J$ implies a cost d_{ij} and do not serve client $i \in I$ implies cost θ_i. Finally, q is the probability that each facility in F has of failing. Thus, introducing variables $X_j \in \{0,1\}$, 1 if f_j is paid and $Y_{ijr} \in \{0,1\}$, 1 if client i is served by facility j when exactly r opened plants fail. The (RpMP) can be formulated as follows (see [42]):

$$\text{(RpMP) minimize} \qquad \alpha w_1 + (1-\alpha) w_2 \tag{1}$$

$$\sum_{j\in J} Y_{ijr} + \sum_{j\in NF}\sum_{s=0}^{r-1} Y_{ijs} = 1 \qquad \forall i \in I, r \in R \tag{2}$$

$$Y_{ijr} \leqslant X_j \qquad \forall i \in I, \quad j \in J, \quad r \in R \tag{3}$$

$$\sum_{j\in J} X_j = p \tag{4}$$

$$\sum_{r\in R} Y_{ijr} \leqslant 1 \qquad \forall i \in I, \quad j \in J \tag{5}$$

$$X_u = 1 \tag{6}$$

$$X_j \in \{0,1\} \qquad \forall j \in J \tag{7}$$

$$Y_{ijr} \in \{0,1\} \qquad \forall i \in I, \quad j \in J, \quad r \in R \tag{8}$$

where α is a value in [0, 1], $R = \{0, \ldots, p-1\}$ and

* Corresponding author. Tel.: +34 966658540.
E-mail addresses: jalcaraz@umh.es (J. Alcaraz), landete@umh.es (M. Landete), monge@umh.es (J.F. Monge).

0377-2217/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ejor.2012.04.016

$$w_1 = \sum_{i\in I}\sum_{j\in J} h_i d_{ij} Y_{ij0}$$

$$w_2 = \sum_{i\in I} h_i \left[\sum_{j\in NF}\sum_{r\in R} d_{ij} q^r Y_{ijr} + \sum_{j\in F}\sum_{r\in R} d_{ij} q^r (1-q) Y_{ijr}\right]$$

Variable X_u represents a fictitious non-failable facility which has transportation cost $d_{iu} = \theta_i$, i.e., non-serving the client i is equivalent to assigning i to plant u. Component w_1 in the objective function computes the cost of serving clients from their preferred facilities among those opened. Component w_2 in the objective function computes the expected failure cost: each client i is served by his rth preferred facility j if the closer are closed or if they have failed (probability q^r) and j itself has not failed (probability $(1-q)$ for $j \in F$ and 1 for $j \in NF$).

To the best of our knowledge, the only technique proposed to solve the Reliability p-Median Problem is an exact method based on a Lagrangean relaxation of constraints (2), see [42]. Despite the combinatorial nature of the problem, this method needs a reasonable computational effort and fails when solving some instances of medium size, for example, 150 customers and 30 facilities opened. As far as we know, no heuristic or metaheuristic method has been developed to solve this combinatorial optimization problem. In this work, we compare different configurations of two hybrid metaheuristics to solve the problem.

The remainder of the paper is organized as follows: in Section 2 a description of the common procedures employed by both algorithms is given. The new metahueristics are explained in detail in Section 3. The computational experiment and the results are described in Section 4 and Section 5 summarizes our conclusions and gives some final remarks.

2. Common features of the metaheuristics

As previously mentioned, several metaheuristics have been developed to solve the classical pMP, but as far as we know, none of them has been applied to the RpMP. We present two hybrid metaheuristics to solve the RpMP, a hybrid genetic algorithm and a hybrid scatter technique. In the next sections, the common features and mechanisms employed by the metaheuristics are described. We will describe in detail later how these metaheuristics make use of these mechanisms.

2.1. Encoding

Although the metaheuristics developed to solve the p-median problem have employed different encodings for the solutions, such as binary encoding [27], the facility list of p genes, where each gene represents the index of an opened plant seems to be the most appropriate [35]. Therefore, in both metaheuristics, we have employed this simple encoding. The solutions to the RpMP are then represented by a permutation of p facility locations. The same permutation but with the facilities sorted in different ways represent the same solution. In the implementation of our metaheuristics, we do not impose the condition of ordered lists. For example, $(2,5,7,9)$ and $(5,9,2,7)$ are valid chromosomes, compute the same fitness value and represent the same solution. The fitness value of a chromosome is directly the total cost of the solution that it represents. Therefore, the optimal chromosome will be that which has the lowest possible fitness value.

2.2. Initial population

Generally speaking, two different procedures can be used in order to generate a solution to be included in the initial population. The first method consists of applying a heuristic in order to obtain a feasible solution to the problem. The second way is to create a solution in a random way, without applying any heuristic algorithm to solve the problem. The first has the advantage of creating good solutions, but needs more computational effort. The second is faster but the quality of the solutions is poorer. A feasible solution for the Reliability p-Median Problem is represented by any permutation of p facility locations. So, generating a solution consists of deciding which p facilities should appear in the permutation. Any method to select p facilities among the set of these could be employed. We have employed a pure random mechanism to generate each individual in the initial population. This mechanism does not require an excessive computational effort. If the size of the population is large enough, each facility will appear with approximately the same frequency. One problem which could be presented with this procedure is when one or more facilities could not appear in the initial population. To avoid this problem Alp et al. [2] designed a procedure to generate the initial population of their genetic algorithm to solve the pMP. They designed this procedure in order to guarantee that every facility should be present in the initial population and with approximately the same frequency. However, this does not present a problem in our algorithms, because we incorporate a local search procedure that will include in the population any "good" facility not present in the initial population.

2.3. Selection of parents

The selection mechanism employed in the standard genetic algorithms is an artificial version of the natural phenomenon called the survival of the fittest. It is used to select which individuals will appear in the next generation and its frequency. Our hybrid genetic algorithm does not incorporate this mechanism. However, in both algorithms we use a selection mechanism to select the parents to undergo the crossover operation (employed by both algorithms) or the path relinking procedure (used only by the scatter search technique). This mechanism is an adaptation of the n-tournament, employed as a selection procedure in different genetic algorithms [21]. Two couples of individuals are randomly chosen from the population and compete to participate in the crossover. The best of each couple (the one with the best fitness value) will undergo the crossover or the path relinking operation.

2.4. Crossover

The crossover mechanism combines the features of two parent chromosomes to produce offspring which inherit their characteristics. The crossover is one of the most important genetic operators and must be correctly designed. Crossover must combine solutions to produce new ones. Combining means not only to use recombination, but also that the recombination is indeed beneficial. In the standard genetic algorithms two parents are combined to generate two offspring that replace them. We have employed the crossover mechanism designed by Alp et al. in their genetic algorithm [2], proposed to solve the pMP, in which two parents produce one offspring. Given two parents, we first take the union of their genes, obtaining a feasible or infeasible solution, a chromosome with a number of genes, greater or equal than p. Notice that the union of the parents could result in a feasible solution with length p if both parents are the same permutation of facility locations, which is not very probable. Then, the genes that appeared in both parents are fixed in the new solution. Finally, some genes that appeared only in one of the parents are removed: the length is reduced until p by deleting each time the facility whose discarding increases the fitness value in less quantity. This process requires an important computational effort but is very efficient and obtains offspring that inherit the good characteristics of the parents.

2.5. Interchange

The interchange method was introduced by Teitz and Bar [45] and is one of the most frequently used classical heuristics to solve the pMP. The idea of this method has been employed either alone [7,47] or as a subroutine of other more complex methods or within metaheuristics (see for example [15,24,25,29–31,33,34,40,44]) to solve this problem. The method consists of interchanging one solution attribute that is in the solution with one that is not. An extension of this version is the so called k-interchange in which k solution attributes are interchanged. Mladenovic et al. [34] use this idea in their chain-interchange heuristic with $k = 2$. We use a variant of this heuristic, based on the idea of the LK-Neighborhood proposed by Kochetov et al. [30] where a 1-interchange procedure is repeated several times. The basic idea is, given a solution, to randomly select a facility which does not appear in the current permutation and replace the one in the chromosome which computes the maximum fitness value decrease, if there is at least one interchange that improves the fitness value. Otherwise, the solution remains unaltered, that is, no interchange is carried out. This movement is repeated m times. In this way, a facility which has been discarded from a solution in the movement n, could be reintroduced in the solution in a further movement $n + j$. Therefore, any possible permutation could be generated using this mechanism. We use this procedure as an improvement mechanism (the resulting solution will always have a better or equal fitness value) but it also allows us to introduce new genetic material into the population or genetic material which may have disappeared during the evolution process. Therefore this mechanism helps to avoid the algorithm being trapped in local optima. This procedure, as crossover, requires an important computational effort. In the metaheuristics we present, two variants of this procedure have been implemented, depending on the number of times it is repeated, 1 or $N - p$, and are referred to as interchange (1) and interchange $(N - p)$ respectively. In the interchange $(N - p)$, all the facilities not belonging to the solution have the possibility of being included in it.

2.6. Insertion of a new member

This mechanism is responsible for deciding if a new individual created during the evolution process is included or not in the current population. In order to preserve the diversity of the population no clones are allowed, i.e., the new individual will be discarded if it is identical (the same permutation) to an existing member. If the new individual does not exist and its fitness value is better than the worst fitness value in the population, it will replace the member with the worst fitness value. It is necessary to point out that this procedure is not applied to those members created to form the initial population. Therefore, in the initial population clones may co-exist.

3. Metaheuristic algorithms

In this section we describe in detail the metaheuristics proposed to solve the Reliability p-Median Problem. The first is a hybrid genetic algorithm which makes use of a local search mechanism which improves the quality of the solutions and helps to avoid being trapped in local optima. The second algorithm proposed is a hybridization of a scatter search method. It also employs the improvement technique to outperform the solutions and to maintain population diversity.

3.1. Hybrid genetic algorithm

Genetic algorithms (GA) were first introduced by Holland [26] in 1975. They are based on the mechanisms of evolution and natural genetics (see Goldberg [21]). Individuals in a population compete among themselves in order to match with other individuals and create offspring which inherit their characteristics. Some individuals may suffer mutations on their genetic material. Such mutations can introduce new characteristics into the population, which may or not be beneficial, or reintroduce genetic material lost during the evolution. This basic template has been widely used to solve many different optimization problems with considerable success. However, several authors have incorporated improvement mechanisms and other features in order to obtain quality solutions in a reasonable computational time. As far as we know, no genetic algorithms have been developed to solve the RpMP, although some authors have applied them to other versions of the median problem (see for example [2,5,12,27,36,43,17,4]). We propose a new genetic algorithm which makes use of problem-specific knowledge and incorporates mechanisms that improve the quality of the solutions. Fig. 1 shows the basic scheme of the algorithm.

The first step is to generate the initial population, that is, to create the set of individuals that will initially participate in the evolution process. We have used the random mechanism described in Section 2.2 which is fast and, if the population size is large enough, gives a population with an appropriate level of diversity. Then, the population is evaluated, each individual in the population is assigned a fitness value given by the total cost of the solution that it represents. As previously mentioned, the total cost is a linear combination of the sum of the day-to-day operating cost and the expected failure cost. Once the population has been evaluated, the evolution process begins and continues until the stopping criterion is satisfied. The conditions to finish use to be related to the elapsed computational time, the number of solutions evaluated or the number of iterations of the genetic process. The selection of parents is carried out. We randomly select two pairs of individuals. Each member of a pair compete with the other to win and mate with the winner of the other pair, forming a couple to undergo the crossover operation described in Section 2.4. The resulting member of the crossover operation is a feasible solution of length p that is evaluated. After the crossover, no mutation mechanism is applied to the candidate solution. The mutation procedure is important in the genetic process because it can introduce in the individual new characteristics, not present in parents or in the population. This helps to maintain diversity in the population and avoids being trapped in local optima and the algorithm to present a premature convergence. However, we have incorporated the Interchange mechanism, which could be applied or not to the current solution, after the crossover, depending on the version of the algorithm used. In this way, every facility not belonging to the

```
Generate_initial_population()
Evaluate_population()
while not(stopping criterion) do
    Selection_parents()
    candidate=Crossover_parents()
    Possible_Interchange(candidate)
    Insertion(candidate)
Possible_Interchange(population)
```

Fig. 1. Procedure genetic algorithm.

solution has the opportunity to be introduced into it. The main objective of this procedure is to improve the fitness value of the solution to which it is applied, but this local search procedure also helps to maintain the necessary variability of the population in order to avoid a premature convergence. Although this mechanism needs a very important computational effort, preliminary results showed a good performance. After this procedure, the solution candidate is inserted into the population replacing the worst member if the insertion conditions described in Section 2.6 are satisfied. After this, the genetic process is repeated if the stopping criterion is not satisfied. To finish, the Interchange mechanism could be applied to every solution in the current population, and this also depends on the version of the genetic algorithm implemented.

3.2. Hybrid scatter search

Scatter Search (SS) is an evolutionary technique which has been used to solve several hard optimization problems. It was first introduced by Glover [18] who described it as a method that uses a succession of coordinated initializations to generate solutions. The original proposal did not provide certain implementation details and later Glover [19] provided such details and expanded the scope of application of the method to nonlinear, binary and permutation problems. In [20], Glover gives the scatter search template. The basic idea of the method is to generate a systematically dispersed set of points from a chosen set of reference points in order to maintain a certain diversity level among the members of this set. The original proposal has later been transformed by several authors incorporating advanced designs and procedures obtaining hybrid scatter search algorithms that have been successfully applied to several combinatorial optimization problems. A description of some of these methods and its applications are described in [32]. As far as we know only two works have applied scatter search methods to p-median problems [8,16] and none to the RpMP. The scatter search method that we present is a hybridization of the original proposal and uses some of the procedures employed by the hybrid genetic algorithm described before. Fig. 2 shows the basic scheme of the algorithm.

The action carried out by the scatter search algorithm is based on a set of good solutions called reference set (*RefSet*). This set consists of a group of solutions chosen from the initial population in such a way that a certain level of diversity is guaranteed. A solution may be added to the reference set if the diversity of the set is improved even when the objective value of such solution is worse than other solutions which could also be added. The first steps consist of generating the initial population and evaluating it, and are carried out in exactly the same way as the genetic algorithm. After that, we have to build the reference set. One half of the *RefSet* is formed with the best members of the population, those with the best fitness value. The second half is fulfilled with the best individuals of the population for a given function of diversity. To assess the diversity between two solutions we propose the following distance function:

$$d(i, RefSet) = \min\{d(i,j), \quad \forall j \in RefSet\} \quad \text{where}$$
$$d(i,j) = \frac{diff(i,j)}{P} + \frac{ofv_j}{ofv_i};$$

$diff(i,j)$, is the number of differences between the two solutions, and ofv_i, ofv_j are the objective value of solution i and j, respectively. Under this diversity function, great values of this distance are related to solutions with a great number of different chromosomes to each solution in the reference set and with reasonable objective values. So, we select from the initial population, the solution with the greatest distance to the *RefSet* and this solution is included in this set. We repeat this procedure until the *RefSet* is completed. Then, the evolution process is started and repeated until the stopping criterion is satisfied. The first step in this evolution process consists of the selection of parents in an identical way to the genetic algorithm. The resulting couple of the selection process undergoes the crossover mechanism or the path relinking procedure in a random way with the same probability. The crossover mechanism is the same as that employed in the genetic algorithm and has been detailed in Section 2.4.

The path relinking mechanism can be considered as an extension of the crossover mechanism, producing a new solution when combining two existing solutions. The path relinking method generates paths between the selected solutions i and j in the neighborhood space. Starting from one solution i (set of p facilities), the neighbor of i is a solution i' with $p-1$ facilities of i and 1 facility of j. Note that the number of steps in the path from i to j is the number of differences minus one between both solutions. The path relinking method concludes with the evaluation of the generated neighborhood solutions, and if the best solution is better than the worst solution of the reference set, this solution is incorporated into the reference set, replacing the worst individual of the set.

After the crossover operation or the path relinking mechanism, the resulting solution, could, depending on the version implemented, undergo the Interchange procedure and the insertion conditions described in Section 2.6 are evaluated to determine if this solution is included or not in the current population. Then, the stopping criterion is evaluated, and if it is not reached, another iteration of the evolution process is repeated. Finally, the Interchange mechanism could be applied to the entire population.

```
 Generate_initial_population()
 Evaluate_population()
 Generate_RefSet()
 while not (stopping criterion) do
     Selection_parents()
     if rand() ≤ 0.5 then
         candidate=Crossover_parents()
     else
         candidate=Path_Relinking()
     Possible_Interchange(candidate)
     Insertion(candidate)
Possible_Interchange(population)
```

Fig. 2. Procedure scatter search algorithm.

4. Computational experience: results

We have carried out an extensive computational experience in order to analyze the performance of the different versions of both algorithms and compare its efficiency. We have used two standard libraries. The first is the Discrete Location Problems Library (DLPL), which is available at [9]. On this page we have 30 instances, and in all of them the set of customers and the set of facilities are equal, i.e., all the cities act as customers, all of which may become a facility and all of which may fail. The number of customers/facilities in the dataset is $N = 100$, and for each instance we have generated another with $N = 50$ customers/facilities, removing the facilities from 51 to 100. We have used three different p values, $p = 5$, 10 and 15, and two different values of the parameter α (0.2 and 0.8) for each instance, having a total of 360 different instances. We have also used the well-known OR library, introduced by Beasley [3] and is available at [37]. In the uncapacitated p-median section there are 40 instances available. These instances have been widely used to compare the computational efficiency of exact and heuristic algorithms for solving the p-Median Problem (see [22]) and the references within). In all the instances, every node is both a potential customer and a facility. The instances have between 100 and 900 customers and p varies from 5 to 90. Each instance has been generated with two different α values (0.2 and 0.8). So, we have a total of 80 different problems to solve from this library.

All the instances (from both libraries) have been solved independently three times with each tested algorithm. In all the instances, we have considered the same failure probability $q = 0.05$. Based on some preliminary experiments, for all the instances, we have considered an initial population size of 50 individuals and a reference set of 10 solutions. The algorithms have been coded in C++ and tested on a PC with a 2.33 GHz Intel Xeon dual core processor, 8.5 Gb of RAM and the operating system was LINUX Debian 4.0. In all the executions performed, we have imposed the time limit, T_{max}, which depends on the size of the problem to be solved, and has been calculated as $T_{max}(N,p) = 2 * ln\binom{N}{p}$.

We have implemented four different configurations of the genetic algorithm (GA1, GA2, GA3 and GA4) and four different scatter search versions (SS1, SS2, SS3, and SS4), having a total of eight algorithms to analyze and compare. GA1, GA2, SS1 and SS2 only apply the Interchange mechanism during the evolution process after the crossover or the path relinking operation. On the other hand, GA3, GA4, SS3 and SS4 do not apply the Interchange mechanism during the evolution process, but the entire population undergo the Interchange procedure at the end of the execution. The even versions (GA2, GA4, SS2 and SS4) differ from the corresponding odd versions (GA1, GA3, SS1 and SS3) in the number of movements carried out by the Interchange mechanism each time it is applied. The odd configurations apply the interchange (1) mechanism and the even versions use the interchange $(N - p)$.

To compare the efficiency of the different configurations and analyze its performance we have solved all the problems with CPLEX (version 11.0) in order to obtain the optimal solution and calculate the deviation from it, imposing a cpu time limit of two hours. For the hardest instances of the DLP Library, those with 100 facilities and 10 or 15 opened facilities, the optimal solution has not been reached in most cases with a cpu time limit of two hours. So for those cases, we compare the deviation from the best solution found by CPLEX. In the OR Library, CPLEX has reported the optimal solution in 28.75% of the problems. Moreover, in 52.5% of the cases CPLEX has not been able to finding a feasible solution in two hours. When a feasible solution has been reached, we have compared the deviation from it. When no feasible solution is found, we compare the deviation from the best solution found by any of the techniques compared.

The total computational time needed to perform all the experiments conducted in this study has been of 1125.83 h.

We have first compared the performance of the genetic algorithm developed by Alp with the first version of the genetic algorithm (GA1) in order to determine if the new features incorporated are useful or not. The main problem presented by the Alp's algorithm is its premature convergence. The algorithm does not incorporate a selection mechanism. The parents to undergo the crossover operation are selected in a purely random way. Moreover, the algorithm does not present a mutation procedure, because, as the authors explain, they tried several mechanisms and they did not improve their results. However, we think this is an important drawback of the algorithm which is responsible for its premature convergence. To solve this problem, in this version of the genetic algorithm, parents to undergo the crossover are selected by binary tournament and the interchange (1) mechanism is applied to the solution after the crossover. Although this mechanism requires an important computational effort, it introduces variability and is able to improve the solution quality.

Table 1 shows the results given by the Alp's algorithm (Alp_GA) and GA1 when solving the benchmark instances of DLPL. Columns 3 and 4 represent the average deviation obtained by these algorithms from the optimal/best solution obtained by CPLEX. Let us remember that each instance has been solved three independent times with the corresponding time limit and we have measured the average deviation for each instance. The last two columns show the average cpu time needed by the algorithm to reach the solution, in seconds. If we look at the computational times employed by the algorithms to obtain the solutions and the deviations from the optimal/best solutions we can deduce that the Alp's algorithm converges prematurely. For example, in the largest instances, those with $N = 100$ and $p = 15$, the algorithm uses on average only 3.07 s of a total of 80 s, obtaining an average deviation from the best solution of 9.63%. By contrast GA1 uses a greater amount of the available time, 18.52 s on average, and reduces the deviation from 9.63 to only 2.89%. Something similar can be observed in the remaining cases.

Regarding the deviation from the optimal/best solution, presented in columns 3 and 4, we can observe the large differences between both algorithms. The average total deviation given by Alp_GA is 15.44% and GA1 reduces this deviation to 2.31%, that is, more than 5 times smaller. The largest differences are obtained when $N = 50$ and $p = 5$, where GA1 gets a deviation of 27.31% compared to 1.29% of GA1, more than 21 times better. In the hardest instances, GA1 obtains a deviation of 3.29% compared to 9.63% given by Alp_GA. On average, when solving the instances with 50 facilities GA1 is more than 10 times better and when solving the instances with 100 facilities, nearly 7 times better. These results demonstrate the superior performance of GA1 and the efficiency of the mechanisms incorporated into the algorithm.

The next step is to compare and analyze the different configurations of the algorithms. We have presented the results of the four configurations of the genetic algorithm in Tables 2 and 3 and the different configurations of the scatter search algorithm in Tables 4 and 5. Tables 2 and 4 report the results of the algorithms when solving the instances in DLP Library and Tables ,spstab43 and 5 the results of problems of OR Library. In all these tables columns 3 to 6 show the average deviation from the optimal/best solution and columns 7–10 the cpu time needed by the algorithm to reach the solution reported when solving the different sets of instances, grouped by N and p.

A quick look at Table 2 tell us that GA1 and GA2 report much better results than GA3 and GA4. The first conclusion is that the Interchange mechanism is indeed beneficial when applied during the evolution process, after the crossover, and not only to the final population. The considerably reduced CPU time needed by GA3 and GA4 to obtain the solution means that the evolution process

Table 1
Comparison of Alp and GA1 algorithms.

N	*p*	%DEV		CPU time	
		Alp_GA	*GA1*	*Alp_GA*	*GA1*
50	5	27.31	1.29	0.03	0.59
50	10	18.56	3.40	0.09	12.73
50	15	12.66	0.87	0.24	2.87
Total 50		19.51	1.85	0.12	5.40
100	5	12.18	1.37	0.15	1.79
100	10	12.27	3.65	1.15	13.14
100	15	9.63	3.29	3.07	18.52
Total 100		11.36	2.77	1.46	11.15
Total		15.44	2.31	0.79	8.27

Table 2
Results given by different configurations of the genetic algorithm (DLP Library).

N	p	%DEV				CPU time			
		GA1	GA2	GA3	GA4	GA1	GA2	GA3	GA4
50	5	1.29	0.75	32.54	6.31	0.59	0.33	0.02	0.11
50	10	3.40	2.06	19.34	15.97	12.73	8.21	0.12	0.41
50	15	0.87	0.36	12.54	4.65	2.87	2.27	0.27	0.84
Total 50		1.85	1.06	21.47	8.98	5.40	3.61	0.14	0.45
100	5	1.37	0.37	24.94	6.28	1.79	1.82	0.05	0.44
100	10	3.65	−0.43	30.53	12.79	13.14	12.46	0.44	1.84
100	15	3.29	−1.23	29.39	13.13	18.52	20.47	1.10	3.64
Total 100		2.77	−0.43	28.29	10.74	11.15	11.59	0.53	1.97
Total		2.31	0.32	24.88	9.86	8.27	7.60	0.34	1.21

has converged prematurely and the Interchange mechanism, applied after this, has not been capable of improving the solutions. When solving the instances with $N = 100$, GA3 need on average 0.53 s, compared with more than 11 s needed by GA1 or GA2. GA3 always uses the lowest quantity of cpu time and always reports the worst deviation. GA4 gets better results than GA3, but they are quite far from the results reported by GA1 and GA2. It is of interest to point out that, although the computational effort employed by GA2 on each set of instances is considerably greater than GA1 (GA1 repeats interchange procedure only once after the crossover and GA2 repeats this operation $(N - p)$ times), the total computational times needed by both algorithms are very similar. GA2 needs, on average 11.59 s to reach the best solution when solving the instances with 100 facilities and GA1 needs 11.15. This implies that GA2 needs less iterations than GA1 to find a solution of similar quality, because each of these iterations needs greater computational time. However, the deviations are not similar between both algorithms. GA2 obtains the best deviation in all cases. The negative deviations in the instances with 100 facilities and $p = 10$ or $p = 15$ means that this algorithm, gets in less than 20 s better results than CPLEX in two hours of computation time. Regarding the total deviations, GA2 is more than 7 times better than GA1, nearly 31 times better than GA3 and nearly 78 times better than GA3.

Table 3, which reports the results given by the different configurations of the genetic algorithm when solving the OR instances, has been divided into two Tables 3a and 3b. In the first, Table 3a, instances of the OR Library for which CPLEX has reported a feasible solution in 2 hours of computation time are shown. Therefore, the column %DEV shows the deviation from the solution reported by CPLEX. Table 3b shows the cases where CPLEX has not reported a feasible solution in two hours. For these instances, we have calculated the average deviation from the best solution given by any of the eight metaheuristics compared. It is interesting to point out that CPLEX, for example, does not reach a feasible solution when $N = 400$ and $p = 40$ or $p = 80$, but reaches a feasible one in the same cpu time when $p = 133$, which seems to be more difficult.

Looking at the computation times employed by the different GA versions reported in Table 3a, we can observe that the percentage of the available time employed by them increases as the difficulty of the problem becomes greater. For example, when $N = 300$, GA1 uses only 0.7 s to reach the best solution (of a total of 47 s, which is near 1.5% of the total time) and when $p = 100$ uses 316.36 of a total of 376 s. This happens in a similar way in the rest of the cases for all the configurations. This means that the time needed by the metaheuristic to reach the best solution increases in a faster way than the available computation time. So, the performance of the four metaheuristic is similar in this sense. Looking at the deviations from the solution given by CPLEX, we can observe that there are several instances where all the metaheuristics obtain a large negative deviation, which means that they are much easier to solve for the metahueristic than for CPLEX. Regarding the average total deviation from the solution given by CPLEX in two hours, the conclusion is the same as when solving the DLP Library, the best result is reported by GA2, although the performance of GA1 is rather similar. Once more, GA3 and GA4 obtain worse results, but there are no large differences between them.

Table 3b shows the deviation, from the best solution obtained by any of the eight metaheuristics presented, obtained by any of the four configurations of the GA and the computation times needed, when solving the hard instances of OR, that is, the instances for which CPLEX has not reported a feasible solution in two hours. Therefore, no negative deviations are possible in this table and the deviations are not comparable with the deviations shown in Table 3a. The computational times used by the different configurations are, in general terms, similar. The algorithm that in most cases uses all the disposable cpu time to find the best solution is GA4, which means that, perhaps, it could obtain much better results with more cpu time. Looking at the deviations, if we compare first GA3 and GA4, the results seem to show that GA3 performs better than GA4 when p is large in the majority of the cases. When p increases, the computational time needed by the Interchange mechanism implemented in GA4 increases exponentially when compared with GA3, and this time is not employed in the evolution process, which leads to bad solutions. Looking at the total deviation, GA3 is, as in the previous tables, better than GA4, and the differences are now largest. GA1 and GA2 present again a similar performance, although GA2 is better again.

The results of Tables 2 and 3 indicate that applying the Interchange mechanism combined with the evolution process is more efficient than applying it in a separate way, and that the best configuration of the four genetic algorithms compared is GA2, with independence on the library used.

The following step is to compare the different configurations of the scatter search algorithm, exactly in the same way as for the genetic algorithm. Therefore, as mentioned before, Tables 4 and 5 report the same results as Tables 2 and 3, but referred to the scatter search algorithm. After analyzing separately both approaches, scatter and genetic, we will make a comparison between them. Comparing first the cpu time used by the different configurations to reach the best solution, reported in Table 4 for the instances of the DLP Library, we can observe that, as happened in the genetic algorithm, versions 3 and 4 employ a small proportion of the time, compared to versions 1 and 2. This would suggest a good performance of the algorithms if the quality of the solutions were good, but this is not the case, as we can observe in the corresponding deviation. So, we can also conclude that both algorithms converge prematurely when solving these instances. Looking at the deviation from the best solution reported by CPLEX, we can observe the extremely large differences between the different versions of the scatter search method. Comparing the versions 1 and 4, the differences between them are smaller than between versions 1 and 4 of the genetic algorithm. It will be interesting to compare them in the larger instances solved, the OR instances. The best configuration when solving the DLP instances is SS2, which obtains a total average deviation of 0.02% compared with the worst, SS3, with a total average of 40.27% of deviation. Moreover, SS2 is the best in all the cases, followed far behind by SS1.

Table 5 shows the results when solving, with the scatter search method, the instances in the DLP Library. We have divided this table in two: the first show the results for the instances for which CPLEX has obtained a feasible solution in two hours of computation time. For those instances, we have measured the deviation from that solution. The second Table 5b shows the results obtained when solving the instances where CPLEX has not reached a feasible solution, and the deviation from the best solution reached by any

Table 3a
Results given by different configurations of the genetic algorithm (OR Library: Instances where CPLEX has reached a feasible solution).

N	*p*	%DEV				CPU time			
		*GA*1	*GA*2	*GA*3	*GA*4	*GA*1	*GA*2	*GA*3	*GA*4
100	5	0.00	0.00	0.23	0.00	0.15	0.08	0.18	1.41
100	10	0.00	0.00	0.13	0.00	0.74	2.30	0.85	4.23
100	20	1.88	0.00	4.88	1.19	27.92	45.42	4.61	16.37
100	20	0.70	0.99	1.88	1.57	12.34	45.62	5.77	16.16
100	33	0.38	0.14	0.55	0.37	27.53	33.09	23.70	38.63
Total 100		0.59	0.22	1.53	0.63	13.73	25.30	7.02	15.36
200	5	0.00	0.00	1.49	0.00	0.65	0.42	0.33	5.37
200	10	0.00	0.00	0.61	0.23	16.71	18.93	2.44	17.66
200	20	−9.51	−9.74	−8.06	−8.06	42.57	103.36	13.42	64.49
200	40	1.93	6.22	1.45	1.79	133.29	174.70	83.36	179.75
200	67	0.58	0.90	0.64	2.87	221.49	206.21	200.66	236.81
Total 200		−1.40	−0.52	−0.78	−0.63	82.94	100.72	60.04	100.81
300	5	0.00	0.00	1.34	0.00	0.70	0.76	0.64	12.11
300	10	0.00	0.00	0.93	0.64	33.40	28.69	4.06	41.63
300	30	−84.69	−83.74	−83.95	−84.46	151.46	146.60	74.63	179.98
300	60	−29.29	−21.40	−27.87	−23.70	278.56	238.90	280.66	283.85
300	100	10.85	14.08	12.76	17.39	316.36	334.84	299.43	365.29
Total 300		−20.63	−18.21	−19.36	−18.03	156.10	149.96	131.88	176.57
400	5	0.00	0.00	3.52	0.00	7.37	3.45	0.81	21.06
400	10	−0.27	−0.27	1.75	−0.12	31.40	37.10	5.03	56.74
400	133	4.93	−9.98	8.59	−4.21	399.21	349.40	388.01	453.29
Total 400		1.56	−3.42	4.62	−1.44	145.99	129.98	131.28	177.03
500	5	0.00	0.00	1.31	0.00	6.79	3.66	1.04	30.27
Total		−5.40	−5.41	−4.10	−4.97	89.93	93.34	73.14	106.58

Table 3b
Results given by different configurations of the genetic algorithm (OR Library: Instances where CPLEX has not reached a feasible solution).

N	*p*	%DEV				CPU time			
		*GA*1	*GA*2	*GA*3	*GA*4	*GA*1	*GA*2	*GA*3	*GA*4
400	40	2.40	23.57	3.10	16.04	229.33	202.77	234.26	247.32
400	80	15.47	28.83	16.38	28.31	367.25	281.02	300.09	368.64
Total 400		8.93	26.20	9.74	22.17	298.29	241.90	267.17	307.98
500	10	0.10	0.13	1.69	0.37	51.09	69.35	8.39	57.11
500	50	10.77	10.07	9.84	15.63	287.52	260.82	307.63	319.00
500	100	20.96	27.25	22.75	35.48	379.11	286.73	420.54	494.00
500	167	34.04	6.99	23.10	13.37	603.91	561.21	547.10	630.00
Total 500		16.47	11.11	14.35	16.21	330.40	294.53	320.92	375.03
600	5	0.00	0.00	0.84	0.00	4.52	3.48	1.35	31.54
600	10	0.48	0.46	2.62	0.56	50.74	58.20	8.64	62.79
600	60	13.80	14.37	12.94	18.21	361.55	264.49	344.33	371.56
600	120	26.94	25.53	25.00	40.85	388.36	525.04	492.65	594.00
600	200	31.74	24.91	33.79	45.39	574.92	757.00	533.40	757.00
Total 600		14.59	13.05	15.04	21.00	276.01	321.64	276.07	363.38
700	5	0.00	0.00	1.41	0.00	6.27	5.48	2.08	33.10
700	10	0.10	0.20	2.25	0.61	29.89	72.63	9.75	63.01
700	70	24.21	11.38	26.25	16.48	369.54	366.20	394.98	414.51
700	140	25.77	30.47	32.72	40.29	515.62	694.00	357.08	694.00
Total 700		12.52	10.51	15.66	14.34	230.33	284.58	190.97	301.15
800	5	0.00	0.00	1.40	0.07	22.33	9.13	2.03	33.35
800	10	0.52	0.34	2.03	0.84	52.35	72.55	10.57	73.56
800	80	22.07	23.69	19.82	30.54	390.60	370.13	388.33	514.00
Total 800		7.53	8.01	7.75	10.48	155.09	150.60	133.64	206.97
900	5	0.00	0.00	1.15	0.01	19.12	14.54	2.69	35.62
900	10	0.35	0.45	2.51	0.86	75.45	58.91	12.95	75.33
900	90	22.81	6.53	25.13	8.94	437.75	453.55	408.34	579.00
Total 900		7.72	2.33	9.59	3.27	177.44	175.66	141.33	229.98
Total		12.03	11.20	12.70	14.90	248.44	256.53	227.96	307.07

of the eight metaheuristics has been calculated. Looking at this deviation in Table 5a, we can observe that the worst algorithm is SS3, which obtains the worst deviation in all the cases. This algorithm is that which uses less quantity of cpu time, which seems to denote that it converges prematurely and is not capable of making use of the available time in order to search efficiently in the search space. For example, when solving the instances with $N = 400$ and $p = 133$, SS3 finds the best solution in only 292.07 s of the 502 s available. The other three configurations use nearly all the available time. We can also observe that the best

Table 4
Results given by different configurations of the Scatter Search Algorithm (DLP Library).

N	*p*	%DEV				CPU time			
		SS1	*SS2*	*SS3*	*SS4*	*SS1*	*SS2*	*SS3*	*SS4*
50	5	3.79	1.01	42.88	3.63	0.11	0.29	0.01	0.07
50	10	4.03	0.48	30.57	4.82	12.76	0.56	0.06	0.27
50	15	3.01	0.36	44.51	4.64	0.85	1.10	0.10	0.52
Total 50		3.61	0.62	39.32	4.36	4.57	0.65	0.06	0.29
100	5	2.78	0.46	30.48	3.49	0.78	1.54	0.03	0.32
100	10	3.73	−1.22	44.37	5.88	9.16	6.60	0.18	1.23
100	15	5.40	−0.95	48.82	9.56	8.95	18.40	0.45	2.73
Total 100		3.97	−0.57	41.22	6.31	6.29	8.85	0.22	1.42
Total		3.79	0.02	40.27	5.34	5.43	4.75	0.14	0.86

configuration is again SS2, which gets a negative deviation of −7.06% in 74.84 s on average compared to CPLEX in two hours of computation time. Moreover, SS2 is the best in all the cases. It is interesting to point out that SS4 now performs better than SS1, getting a lower deviation in the majority of the instances solved. SS4 is now the second best configuration with a total average deviation of −6,36% followed by SS1, with −5.26% and SS3, with 1.44%.

Results given in Table 5b ratify the tendency observed in the previous table, where SS4 obtained better results than SS1. For the hardest instances of the OR Library, SS4 obtains a total average deviation of 1.93% compared with the 5.54% obtained by SS1, which is a large difference. So, we can state that SS4 performs better than SS1 in large instances, which means that Interchange mechanism performs better included into the scatter search mechanism when repeated $N - p$ times on each solution. The difference between SS2 and SS4 has been reduced and is slightly less than 1%. The worst in all the cases is SS3 with a total average deviation of 11.79%. With respect to the cpu time employed by the algorithms to reach the best solution, SS3 is again that which uses less time; the other three use similar times. SS2 and SS4, that is the versions which apply the interchange $(N - p)$ are the algorithms which require more cpu time to reach the solution and those which use in most cases all the available time to reach that solution. So, with respect to the scatter search method analyzed, we can conclude that SS2 is the best, with independence on the size of the instances or the Library solved. The second best algorithm is SS1 when the instances are medium sized, but for the large size instances, SS4 performs better.

In the last two tables, Tables 6 and 7, we compare the performance of the best configuration of the genetic algorithm (GA2), the best scatter search version (SS2) and CPLEX. To do this, we have also solved all the instances (DLP and OR Libraries) with CPLEX imposing the same stopping criterion as the metaheuristics, the maximum CPU time, given by the formula presented at the beginning of this section. Looking at the CPU time needed by the algorithms to reach the best solution in Table 6, we can observe the enormous differences between CPLEX and the metaheuristics. With respect to the small instances, CPLEX uses on average 23.62 s to solve these instances, and the metaheuristics, 3.61 and 4.75 s respectively. However, the deviation obtained by the different techniques when solving the small instances are not so great. CPLEX performs very well in this group of instances, showing a deviation of 0.25%, compared to the 1.06 % obtained by GA2 and the 0.62% given by SS2. We would like to add that, if the available CPU time had been much more reduced, the results given by CPLEX would have been worse than the metaheuristics. In the large instances, those with 100 facilities, the differences are very large between CPLEX and the metaheuristics. SS2 presents an average deviation of 0.02% with respect to the solution given by CPLEX in two hours, but SS2 needed only 4.75 s. The deviation given by CPLEX is of 2872.69% on average in approximately 41 s on average. Therefore, the metaheuristics are much more appropriate than CPLEX in order to solve this kind of instances.

With respect to the instances of OR Library, we have solved only the instances where CPLEX had previously reached a feasible solution in two hours of CPU time. A dash in the CPLEX deviation column of Table 7 indicates that CPLEX has not reached a feasible solution in the corresponding CPU time. The results are similar to those reported in the previous table, that is, CPLEX needs much more time than the metaheuristics and reaches a much worse

Table 5a
Results given by different configurations of the Scatter Search Algorithm (OR Library: Instances where CPLEX has reached a feasible solution.

N	*p*	%DEV				CPU time			
		SS1	*SS2*	*SS3*	*SS4*	*SS1*	*SS2*	*SS3*	*SS4*
100	5	0.00	0.00	0.99	0.00	0.10	0.13	0.02	0.22
100	10	0.20	0.03	4.65	0.00	4.55	0.52	0.08	0.65
100	20	10.55	0.43	23.83	2.80	39.26	13.62	0.39	2.21
100	20	3.78	0.91	15.91	2.74	32.47	4.36	0.32	2.33
100	33	1.72	0.46	8.24	0.78	4.46	10.18	1.10	4.77
Total 100		3.25	0.37	10.72	1.26	16.17	5.76	0.38	2.04
200	5	0.00	0.00	2.38	0.00	0.18	0.50	0.04	0.85
200	10	0.24	0.04	4.13	0.23	14.27	6.59	0.14	2.82
200	20	−6.90	−9.88	7.12	−8.98	52.11	47.32	0.91	9.59
200	40	7.55	1.78	20.14	5.34	161.84	92.76	5.55	34.48
200	67	0.77	0.29	7.71	1.05	75.91	90.94	18.13	83.19
Total 200		0.33	−1.55	8.29	−0.47	60.86	47.62	4.95	26.18
300	5	0.00	0.00	3.04	0.00	1.13	1.11	0.08	1.91
300	10	0.81	0.00	6.72	0.04	10.07	4.98	0.27	6.08
300	30	−83.67	−84.75	−79.52	−84.03	82.99	135.67	4.38	46.89
300	60	−27.16	−31.67	−20.36	−30.65	261.30	227.32	29.42	186.82
300	100	5.59	0.38	16.42	1.89	296.22	339.61	85.06	346.98
Total 300		−20.88	−23.21	−14.74	−22.55	130.34	141.73	23.84	117.73
400	5	0.00	0.00	3.81	0.02	1.45	2.05	0.10	3.40
400	10	0.08	−0.25	5.55	−0.19	12.31	15.10	0.37	11.02
400	133	−13.49	−11.89	−5.75	−11.79	493.45	502.00	292.07	502.00
Total 400		−4.47	−4.05	1.20	−3.99	169.07	173.05	97.51	172.14
500	5	0.00	0.00	2.29	0.00	4.09	3.20	0.12	5.34
Total		−5.26	−7.06	1.44	−6.36	81.48	74.84	23.08	65.87

Table 5b
Results given by different configurations of the Scatter Search Algorithm (OR Library: Instances where CPLEX has not reached a feasible solution).

N	p	%DEV				CPU time			
		*SS*1	*SS*2	*SS*3	*SS*4	*SS*1	*SS*2	*SS*3	*SS*4
400	40	13.03	2.59	27.75	4.19	182.27	211.21	15.81	149.66
400	80	12.45	1.18	15.99	3.80	336.73	375.76	89.41	362.47
Total 400		12.74	1.88	21.87	3.99	259.50	293.48	52.61	256.06
500	10	0.25	0.00	7.17	0.21	29.81	43.37	0.45	16.70
500	50	3.10	1.52	19.44	6.67	287.20	311.99	37.06	276.99
500	100	7.93	0.45	8.08	1.35	471.86	494.00	248.82	494.00
500	167	7.29	1.82	14.59	2.74	630.00	630.00	607.33	630.00
Total 500		4.65	0.95	12.32	2.74	354.72	369.84	223.41	354.42
600	5	0.00	0.00	2.48	0.00	2.05	4.44	0.17	7.60
600	10	0.61	0.00	6.56	0.17	10.98	19.87	0.57	24.25
600	60	0.85	3.13	13.94	2.06	331.45	384.00	91.62	384.00
600	120	11.09	1.94	12.15	1.76	568.10	594.00	431.88	594.00
600	200	24.23	1.02	22.18	7.17	679.16	757.00	809.98	757.00
Total 600		7.36	1.22	11.46	2.23	318.35	326.46	256.25	353.37
700	5	0.00	0.00	2.88	0.00	7.43	6.09	0.21	10.44
700	10	0.38	0.00	5.89	0.20	29.24	23.66	0.81	33.05
700	70	3.14	4.76	19.88	5.35	439.54	449.00	145.75	449.00
700	140	17.59	0.82	18.81	1.43	665.12	694.00	612.21	694.00
Total 700		5.28	1.39	11.87	1.75	285.33	293.19	189.74	296.62
800	5	0.15	0.00	2.44	0.15	14.53	7.86	0.27	13.52
800	10	0.64	0.01	5.92	0.18	33.02	38.89	0.85	43.25
800	80	6.94	0.90	13.42	1.80	482.70	514.00	256.66	514.00
Total 800		2.58	0.30	7.26	0.71	176.75	186.91	85.92	190.25
900	5	0.00	0.00	2.86	0.03	13.37	11.13	0.35	17.96
900	10	0.33	0.13	7.51	0.13	47.41	70.84	1.02	55.02
900	90	6.23	1.21	17.69	1.11	575.70	579.00	489.04	533.45
Total 900		2.19	0.45	9.35	0.42	212.16	220.32	163.47	202.14
Total		5.54	1.02	11.79	1.93	279.11	290.15	180.34	288.59

Table 6
Comparison of the best configuration of the metaheuristics and CPLEX (DLP Library).

N	p	%DEV			CPU time		
		*GA*2	*SS*2	*CPLEX*	*GA*2	*SS*2	*CPLEX*
50	5	0.75	1.01	0.00	0.33	0.29	19.62
50	10	2.06	0.48	0.74	8.21	0.56	32.67
50	15	0.36	0.36	0.00	2.27	1.10	18.57
Total 50		1.06	0.62	0.25	3.61	0.65	23.62
100	5	0.37	0.46	582.72	1.82	1.54	36.00
100	10	−0.43	−1.22	11533.69	12.46	6.60	61.00
100	15	−1.23	−0.95	5118.98	20.47	18.40	80.00
Total 100		−0.43	−0.57	5745.13	11.59	8.85	59.00
Total		0.32	0.02	2872.69	7.60	4.75	41.31

solution. CPLEX performs well in the smallest instances of OR Library and gets a deviation of 1.85% compared to less than 0.4% given by the metaheuristics. For the instances with 200 or 300 facilities the differences are very large. CPLEX gets a deviation of more than 300% compared to the negative deviations reached by the metaheuristics. For example, in the largest instances with 300 facilities solved by CPLEX, those with $p = 60$, CPLEX reports a deviation of 1118.69%, compared with −23.21% given by SS2 and − 18.21% obtained by GA2. The results presented in the last two tables demonstrate the excellent performance of the metahueristics when compared with CPLEX and present them as the only alternative in large sized instances.

Table 7
Comparison of the best configuration of the metaheuristics and CPLEX (OR LIbrary: Instances where CPLEX has reached a feasible solution).

N	p	%DEV			CPU time		
		*GA*2	*SS*2	*CPLEX*	*GA*2	*SS*2	*CPLEX*
100	5	0.00	0.00	0.00	0.08	0.13	4.91
100	10	0.00	0.03	0.00	2.30	0.52	21.56
100	20	0.00	0.43	0.76	45.42	13.62	56.36
100	20	0.99	0.91	8.48	45.62	4.36	66.45
100	33	0.14	0.46	0.00	33.09	10.18	30.33
Total 100		0.22	0.37	1.85	25.30	5.76	35.92
200	5	0.00	0.00	0.00	0.42	0.50	30.28
200	10	0.00	0.04	47.70	18.93	6.59	81.38
200	20	−9.74	−9.88	259.46	103.36	47.32	125.00
200	40	6.22	1.78	1185.10	174.70	92.76	195.00
200	67	0.90	0.29	75.60	206.21	90.94	249.00
Total 200		−0.52	−1.55	313.57	100.72	47.62	136.13
300	5	0.00	0.00	36.63	0.76	1.11	47.00
300	10	0.00	0.00	82.70	28.69	4.98	84.00
300	30	−83.74	−84.75	0.00	146.60	135.67	190.00
300	60	−21.40	−31.67	1118.69	238.90	227.32	295.00
300	100	14.08	0.38	–	334.84	339.61	376.00
Total 300		−18.21	−23.21	309.51	149.96	141.73	198.40
400	5	0.00	0.00	36.32	3.45	2.05	50.00
400	10	−0.27	−0.25	76.64	37.10	15.10	89.00
400	133	−9.98	−11.89	–	349.40	502.00	502.00
Total 400		−3.42	−4.05	56.48	129.98	192.12	213.67
500	5	0.00	0.00	32.90	3.66	3.20	53.00
Total		−5.41	−7.06	145.31	93.34	78.84	134.01

5. Final remarks

The results presented in the previous section show the excellent performance of both paradigms (genetic algorithms and scatter search) to solve the Reliability *p*-Median Problem. With the incorporation of the Interchange mechanism we have managed to avoid the metaheuristics being trapped in local optima, a problem which was presented by other metaheuristics. The application of this mechanism is more efficient when carried out during the evolution process than when applied in a separate way to the final population.

Although the computational effort employed by the application of the interchange $(N - p)$ during the evolution process on each solution is considerably higher than the interchange (1), it has proved to be more efficient in both metaheuristics, and the total computational times are not higher. Each generation needs more cpu time when using the interchange $(N - p)$, but the number of iterations needed to find the best solution is lower. The best configuration of both algorithms use this version of the Interchange mechanism.

The interchange $(N - p)$ is also more efficient than the interchange (1) in both metaheuristics when only applied to the final population. We would like to emphasize that, in this case, as the available cpu time is fixed and scant, the interchange $(N - p)$ will be applied to fewer individuals than interchange (1), but the first gives much better results. In the case of the scatter search mechanism, when solving the hardest instances, the results obtained by SS4 are very near those of SS2.

It is important to point out that, in most cases, both metaheuristics make use of only a small part of the available computation time, but it has not presented a problem in comparing its efficiency. For example, when solving the instances with 50 facilities and $p = 15$, the metaheuristics find the best solution in an average of less than 3 s, when the available cpu time is 57 s. If, in a future work, the efficiency of these algorithms were to be compared with new metaheuristics developed to solve this problem, both algorithms could be redesigned, including new mechanisms (for example, the random replacement procedure employed by Alcaraz et al. [1] in their genetic algorithm to solve a project scheduling problem or the restart scheme employed by Ruiz et al. [41] in their metaheuristics to solve a flowshop scheduling problem) in order to make use of the total available time and improve the quality of the solutions.

Both metaheuristic algorithms have demonstrated their efficiency when compared with the performance of CPLEX when solving the different sets of instances. CPLEX performs well when solving the easiest instances, those with 50 facilities, but the performance when solving larger instances is very poor when compared with the metaheuristics. For example, in the hardest set of instances of the DLP Library, those with 100 facilities and N = 15, GA2 and SS2 have obtained much better solutions in an average of less than one third of a minute, than CPLEX in 2 hours of computational time. Moreover, metaheuristics are the only alternative for solving large instances of the problem, because as has been shown, CPLEX fails when solving instances of more than 400 facilities and the proposed metaheuristics have also demonstrated an excellent performance in these cases.

Although the general results seem to show a slightly better performance of SS2 compared to GA2, we do not come to the conclusion that SS2 is more efficient because the total differences are not marked and it is not better in all cases. The conclusion is that both offer an excellent and efficient performance.

Acknowledgments

This research was partially supported by "Ministerio de Ciencia y Tecnologı́a", Spain (Research Projects EC02008-05895-C02-01/ECON and MTM2009-14039-C06-04) and "Fundación Séneca" (Research Project 08716/PI/08).

References

[1] J. Alcaraz, C. Maroto, R. Ruiz, Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms, Journal of the Operational Research Society 54 (2003) 614–626.
[2] O. Alp, E. Erkut, Z. Drezner, An efficient genetic algorithm for the p-median problem, Annals of Operations Research 122 (2003) 21–42.
[3] J.E. Beasley, A note on solving large p-median problems, European Journal of Operational Research 21 (1985) 270–273.
[4] J.M. Cadenas, M.J. Canós, M.C. Garrido, C. Ivorra, V. Liern, Soft-computing based heuristics for location on networks: the p-median problem, Applied Soft Computing 11 (2011) 1540–1547.
[5] S.S. Chaudhry, S. He, P.E. Chaudhry, Solving a class of facility location problems using genetic algorithm, Expert Systems 20 (2003) 86–91.
[6] G. Cornuejols, G.L. Nemhauser, A. Laurence, A. Wolsey, A canonical representation of simple plant location Pproblems and its applications, SIAM Journal on Algebraic and Discrete Methods 1 (1980) 261–272.
[7] P.J. Densham, G. Rushton, A more efficient heuristic for solving large p-median problems, Papers in Regional Science 71 (3) (1992) 307–329.
[8] J.A. Díaz, E. Fernández, Hybrid scatter search and path relinking for the capacitated p-median problem, European Journal of Operational Research 169 (2) (2006) 570–585.
[9] Discrete Location Problems Library. <http://math.nsc.ru/AP/benchmarks/UFLP>.
[10] E. Domínguez, J. Muñoz, A neural model for the p-median problem, Computers and Operations Research 35 (2008) 404–416.
[11] S. Elloumi, A tighter formulation of the p-median problem, Journal of Combinatorial Optimization 19 (2010) 69–83.
[12] J. Fathali, A genetic algorithm for the p-median problem with pos/neg weights, Applied Mathematics and Computation 183 (2006) 1071–1083.
[13] K. Fleszar, An effective VNS for the capacitated p-median problem, European Journal of Operational Research 191 (2008) 612–622.
[14] S. García, M. Labbé y A. Marín, Solving large p-median problems with a radius formulation, INFORMS Journal on Computing, in press.
[15] F. Garciá-Loṕez, B. Melián Batista, J.A. Moreno-Pérez, J.M. Moreno-Vega, The parallel variable neighborhood search for the p-median problem, Journal of Heuristics 8 (2002) 375–388.
[16] F. Garciá-Loṕez, B. Melián Batista, J.A. Moreno-Pérez, J.M. Moreno-Vega, Parallelization of the scatter search for the p-median problem, Parallel Computing 29 (5) (2003) 575–589.
[17] K. Ghoseiri, S.F. Ghannadpour, An efficient heuristic method for capacitated p-median problem, International Journal of Management Science and Engineering Mamagement 4 (1) (2009) 72–80.
[18] F. Glover, Heuristics for integer programming using surrogate constraints, Decision Sciences 8 (1977) 156–166.
[19] F. Glover, Genetic algorithms and scatter search: unsuspected potentials, Statistics and Computing 4 (1994) 131–140.
[20] F. Glover, A template for scatter search and path relinking, in: J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, D. Snyers (Eds.), Artificial Evolution, Lecture Notes in Computer Science, vol. 1363, Springer, 1998, pp. 13–54.
[21] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley, 1989.
[22] B. Goldengorin, D. Krushinsky, Complexity evaluation of benchmark instances for the p-median problem, Mathematical and Computer Modelling 53 (2011) 1719–1736.
[23] S.L. Hakimi, Optimum locations of switching centers and the absolute centers and medians of a graph, Operations Research 12 (1964) 450–459.
[24] P. Hansen, N. Mladenović, Variable neighborhood search for the p-median, Location Science 5 (1997) 207–226.
[25] P. Hansen, N. Mladenovic, Variable neighborhood search: principles and applications, European Journal of Operational Research 130 (2001) 449–467.
[26] H.J. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, 1975.
[27] C.M. Hosage, M.F. Goodchild, Discrete space location–allocation solutions from genetic algorithms, Annals of Operations Research 6 (1986) 35–46.
[28] O. Kariv, L. Hakimi, An algorithmic approach to network location problems. Part II: The p-medians, Journal of Applied Mathematics 37 (3) (1979) 539–560.
[29] Y. Kochetov, Probabilistic local search algorithms for the discrete optimization problems. In: Discrete Mathematics and Applications, Moscow, MSU, 2001, pp. 84–117.
[30] Y. Kochetov, E. Alekseeva, T. Levanova, N. Loresh, Large neighborhood search for the p-median problem, Yugoslav Journal of Operations Research 15 (1) (2005) 53–63.
[31] T. Levanova, M.A. Loresh, Algorithms of ant system and simulated annealing for the p-median problem, Automation and Remote Control 65 (2004) 431–438.
[32] M. Laguna, R. Martí, Scatter search, in: C.R. Sharda, Stefan Voss (Eds.), Methodology and Implementations, Kluwer, 2003.
[33] N. Mladenović, J.A. Moreno-Pérez, J.M. Moreno-Vega, Tabu search in solving p-facility location–allocation problems, Les Cahiers du GERAD, G-95-38, Montreal, 1995.
[34] N. Mladenović, J.A. Moreno-Pérez, J.M. Moreno-Vega, A chain-interchange heuristic method, Yugoslav Journal of Operations Research 6 (1996) 41–54.
[35] N. Mladenović, J. Brimberg, P. Hansen, J.A. Moreno-Pérez, The p-median problem: a survey of metaheuristic approaches, European Journal of Operational Research 179 (2007) 927–939.
[36] J.A. Moreno-Pérez, J.L. García-Roda, J.M. Moreno-Vega, A parallel genetic algorithm for the discrete p-median problem, Studies in Location Analysis 7 (1994) 131–141.
[37] O.R. Library. <http://mscmga.ms.ic.ac.uk/info.html>.

[38] A. Plastino, R. Fuchshuber, S.deL. Martins, A.A. Freitas, S. Salhi, Statistical Analysis and Data Mining 4 (3) (2011) 313–335.
[39] J. Reese, Solution methods for the p-median problem: an annotated bibliography, Networks 48 (2006) 125–142.
[40] M. Resende, R.F. Werneck, On the implementation of a swap-based local search procedure for the p-median problem, in: R. Ladner (Ed.), Proceedings of the 5th Workshop on Algorithm Engineering and Experiments, SIAM, Philadelphia, 2003, pp. 119–127.
[41] R. Ruiz, C. Maroto, J. Alcaraz, Solving the flowshop scheduling problem with sequence sependent setup times using advanced metaheuristics, European Journal of Operational Research 165 (2005) 34–54.
[42] L.V. Snyder, M.S. Daskin, Reliability models for facility location: the expected failure cost case, Transportation Science 39 (3) (2005) 416–440.
[43] M.T. Steiner, A.A. Freitas, E.S. Correa, A genetic algorithm for solving a capacitated p-median problem, Numerical Algorithms 35 (2) (2004) 373–388.
[44] É.D. Taillard, Heuristic methods for large centroid clustering problems, Journal of Heuristics 9 (1) (2003) 51–73.
[45] M.B. Teitz, P. Bart, Heuristic methods for estimating the generalized vertex median of a weighted graph, Operations Research 16 (1968) 955–961.
[46] C.S. ReVelle, R. Swain, Central facilities location, Geographical Analysis 2 (1970) 30–42.
[47] R. Whitaker, A fast algorithm for the greedy interchange for large-scale clustering and median location problems, INFOR 21 (1983) 95–108.