Scatter search for an uncapacitated p-hub median problem

Rafael Martí, Ángel Corberán, and Juanjo Peiró*

Departament d'Estadística i Investigació Operativa, Universitat de València, Spain

Thursday 31st July, 2014

Abstract

Scatter search is a population-based method that has been shown to yield high-quality outcomes for combinatorial optimization problems. It uses strategies for combining solution vectors that have proved effective in a variety of problem settings. In this paper, we present a scatter search implementation for an $\mathcal{NP}$-hard variant of the classic p-hub median problem. Specifically, we tackle the uncapacitated r-allocation p-hub median problem, which consists of minimizing the cost of transporting the traffics between nodes of a network through special facilities that act as transshipment points. This problem has a significant number of applications in practice, such as the design of transportation and telecommunications networks.

As it is customary in metaheuristic implementations, we design specific methods to create an efficient solving procedure based on the scatter search methodology. In particular, we propose mechanisms to generate, combine, and improve solutions for this problem. Special mention deserves the use of path-relinking as an extension of the classical combination method. Extensive computational experiments establish the effectiveness of our procedure in relation to approaches previously identified to be best.

Keywords and phrases: scatter search, path-relinking, hub location, p-hub, r-allocation, combinatorial optimization.

1 Introduction

The aim of this paper is to expand the scatter search methodology by implementing its underlying framework when solving a variant of the well known hub location problem (HLP). Hubs are special facilities that act as transfer points, concentrating and redistributing products that must be delivered in systems

*Author for correspondance: Juanjo Peiró (juanjo.peiro@uv.es) with many origins and destinations. They appear in problems related to the design of transportation and telecommunications networks. Hub location problems have generated a considerable amount of research interest over the years, as documented in [1, 2, 3, 4, 7, 10, 14]. Although we target here a specific variant of the p-hub problem, our results (as compared with previous methods), clearly show that the evolutionary methods in general, and scatter search in particular, are well suited for solving location problems. Thus, we have two objectives: First, to obtain high quality solutions to the uncapacitated r-allocation p-hub median problem (UrApHMP), and, second, to show the effectiveness of the scatter search methodology to this type of location problems.

In the UrApHMP, one is given a network $G = (V, E)$ with a set of demand nodes V, and a set of edges E. For each pair of nodes i and $j \in V$, there is an amount of traffic t_{ij} (generally of people or goods) that needs to be transported. It is assumed that direct transportation between i and j is not possible. This well-known assumption is based on the empirical evidence that it is not physically and/or economically viable, for example, to schedule a flight between any two pairs of cities or to add a link between any two computers for data transmission, since this would require a large amount of resources. Therefore, the traffic t_{ij} is routed along a path $i \to k \to l \to j$, where nodes k and $l \in V$ are used as intermediate points for this transportation. The UrApHMP consists of choosing a set H of nodes, ($H \subseteq V$, $|H| = p$), that can be used as intermediate transfer points between any pair of nodes in G, in order to minimize the total transportation cost of all the traffics of the network. The nodes in H are commonly called *distribution centers* or *hub nodes*. The other nodes in the network are known as *terminal nodes*. For the sake of simplicity, we call them *hubs* and *terminals*, respectively.

Three optimization subproblems arise when solving the UrApHMP: a *location* problem to choose the best locations for the hubs, an *assignment* problem of each terminal to r of the hubs, and a *routing* problem to obtain the minimum cost route transporting the traffics between any given pair of nodes. Regarding the allocation strategy in the assignment process, the UrApHMP generalizes two extensively studied versions of the p-hub location problem: the *single* and *multiple* versions. In the single version ($r = 1$), each terminal is assigned to only one of the p hubs, thus allowing to send and receive the traffics through this hub. In contrast, in the multiple version ($r = p$), each terminal can send and receive traffics through any of the p hubs. In the UrApHMP, $1 \leq r \leq p$, each terminal is allowed to be allocated to r of the p hubs. The motivation of this variant comes from the fact that the single allocation version is too restricted for real-world situations, while the multiple allocation variant results in high fixed costs and complicated networks, which does not reflect the real models either. Yaman presented in [24] a study of allocation strategies and introduced the r-allocation version of the problem. We refer the reader to this paper for an excellent analysis of different mathematical programming formulations for this version.

From the standpoint of a metaheuristic classification, Scatter Search (SS) may be considered as a population-based algorithm that constructs solutions by combining others to efficiently solve $\mathcal{NP}$-hard optimization problems. It derives its foundations from strategies originally proposed for combining decision rules and constraints in the context of integer programming [11, 13, 15, 16]. Path-relinking (PR) was originally proposed in the context of tabu search [12, 21,

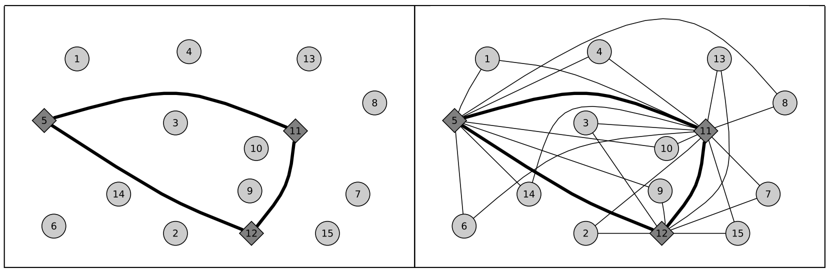

Figure 1: The image on the left shows a location example of $p = 3$ hubs (in nodes 5, 11 and 12) for a network G. The image on the right shows G with the p hubs and a possible assignment of all nodes to their respective hubs when $r = 2$.

23] as a method to find high quality solutions in the path between two good solutions.

Hybridizations of SS and PR methodologies have been successfully applied to other related facility location problems, such as the p-median [5] and the p-center [19] problems. In these papers, a GRASP constructive method, based on a partially randomizing greedy criterion, constructs the population of initial solutions of SS. This method is coupled with PR, which creates a path between two given good solutions to explore its intermediate solutions. The hybrid algorithms resulting from the combination of SS and PR methodologies have proved to produce high quality solutions in terms of its cost and computational effort. Despite this, we consider that the proposed procedures for the p-median and p-center cannot directly apply to solve the UrApHMP because of the differences among the three problems. Note, for example, that the p-median and p-center problems are not directly related to the delivery of goods between any two given nodes of a network, since they are more focused on the supply of products from facilities to clients. Moreover, in the p-median and p-center problems, each client is assigned to only one facility, and, in the p-center problem, the objective function is to minimize the maximum cost (distance) between the clients and their assigned facilities. It is well documented in the metaheuristic literature that, in order to obtain good solutions, the solving method has to be specifically designed for the problem in hand. In this line, a GRASP is specifically proposed in [20] for the UrApHMP. It is based on the idea of constructing a set of feasible solutions and improving them with three local searches. The authors also propose a filtering mechanism to discard low-quality solutions and to apply the local searches only to the promising solutions, with the aim of saving computing time.

In this paper we explore the adaptation of SS and PR for the UrApHMP. The good performance of this hybridization in related problems triggered our interest to implement it in this problem. In the following section we describe our proposals for the different elements that integrate this methodology. We finally present exhaustive computational experiments to first calibrate these elements and then compare our algorithm with the previous methods, including the GRASP referenced above.

2 Scatter search for the *p*-hub problem

If solutions of a combinatorial optimization problem are seen as points in a space, the exploration of this space is done in scatter search by evolving a set of reference points. These points define a set, known as *Reference Set* (*RefSet*), and typically consist of good solutions obtained by prior problem solving efforts.

A given iteration of the SS method generally consists of three main steps: to combine the solutions of *RefSet*, to improve the solutions obtained, and to update *RefSet* with the resulting solutions that are better than those currently in *RefSet*. In the next subsections we explain how these steps are adapted to the UrApHMP.

As shown in Figure 2, the method starts by generating a set *Pop* of diverse solutions with a selective good quality (see Subsection 2.1). Then, we select β of them to create *RefSet*. We follow the standard design of selecting the best $\frac{\beta}{2}$ solutions in *Pop*, and then the most diverse $\frac{\beta}{2}$ solutions w.r.t. the solutions already in *RefSet* (see Subsection 2.2). The main loop of the method consists of applying the combination method to all the pairs of solutions in *RefSet*. As it is described in Subsection 2.4, the path-relinking methodology is the method we use to create new solutions from each pair of solutions selected from *RefSet*.

Path-relinking is an intensification strategy to explore trajectories connecting good (also known as *elite*) solutions obtained by other heuristic methods. When hybridizing both methodologies, SS and PR, it seems natural to consider path-relinking as the method for combining the solutions in *RefSet*, generating paths between and beyond these reference points. Once a new solution has been obtained, we decide whether it is included or not in *RefSet*. We implement a standard *RefSet* update method, and we only include a solution in *RefSet* if it better than the worst solution and provides sufficient diversity according to a "distance" between the solution and *RefSet*. Finally, the improvement method we propose is described in Subsection 2.6.

2.1 The Diversification Generator Method

The diversification generator method (DGM) yields a population *Pop* of π initial feasible solutions for the problem. It seeks to balance quality, in terms of solutions cost, and diversity, in terms of solution attributes, and can be seen as the mechanism that creates a first generation of solutions. If the attributes of this first generation of solutions are good, there is some confidence of getting better solutions in the following iterations after strategically combining them.

To create *Pop*, we have developed and tested seven different DGM for the UrApHMP. Six of them are based on GRASP constructions [8, 9] that differ in the implementation designs, and in the different expressions for the evaluations of hubs that will be used in each of the solutions. The seventh is simply a random construction to provide diversity to *Pop*.

As it is well known, in the *semi-greedy* implementation of GRASP, each element of a solution is iteratively selected by evaluating all candidate elements with respect to a greedy function g that measures their attractiveness. Only the q elements with best g values are placed in a restricted candidate list (RCL), where q is a search parameter. Then, an element in the RCL is randomly selected, according to a uniform distribution, to become part of the solution.

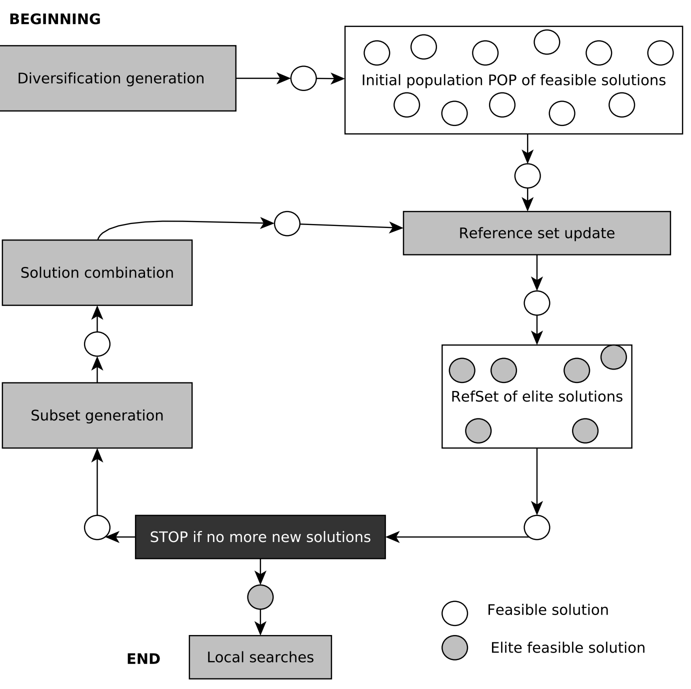
BEGINNING
Diversification generation
Initial population POP of feasible solutions
Reference set update
Solution combination
RefSet of elite solutions
Subset generation
STOP if no more new solutions
END
Local searches
Feasible solution
Elite feasible solution

Figure 2: Scheme of the proposed scatter search

The values of g are updated at each iteration to reflect the changes brought on by the selection of the previous element.

An interesting variant of the classic GRASP design described above was recently proposed in [22]. It is based on a *sampled greedy* that first builds a RCL by uniformly sampling q elements at random. Then, g is evaluated over these q elements. The RCL element with the best g value is added to the solution under construction. In both GRASP implementation, the value of q controls the trade-off between greediness and randomness. In the first design, lower q values favor greedy selection (w.r.t. randomization), while this is obtained with large q values in the second design.

In our implementation, we have based the g evaluation for the selection of the p hubs on a cost criterion. Let $h \in V$ be a candidate node to be used as hub. If h were a hub, it would be used for the transportation of the traffics among some terminals, possibly the φ terminals $i_1, \ldots, i_\varphi$ with lower assignment cost to h, where φ is a given parameter. We then compute $g(h)$ as

$$g(h) = \sum_{s=1}^{s=\varphi} cost(i_s, h), \quad \forall h \in V,$$

where $cost(i,h)$ represents the assignment cost of terminal i to hub h. We propose several ways for computing $cost(i,h)$. In all of them, let $\overrightarrow{T_i} = \sum_{j \in V} t_{ij}$ be the sum of all the traffics from i to all nodes j. Similarly, let $\overleftarrow{T_i} = \sum_{j \in V} t_{ji}$

be the sum of all the traffics from all nodes j to node i. The different alternatives for computing the *cost* functions are:

Type 1 Here, $cost(i,h)$ is computed as the cost of sending $\overrightarrow{T_i}$ from i to h, i.e. $cost(i,h) = d_{ih}\overrightarrow{T_i}$. This evaluation cost was proposed by Peiró, Corberán and Martí in [20] and produced good quality solutions.

Type 2 Now, we also consider the cost of receiving the incoming traffic $\overleftarrow{T_i}$ for i from h, $cost(i,h) = d_{ih}\overrightarrow{T_i} + d_{hi}\overleftarrow{T_i}$.

Type 3 In addition to the costs of transporting the incoming an outgoing traffics, we consider here some of the discounting factors that are usually present in the UrApHMP. To do this, we enrich the *cost* expression as $cost(i,h) = \chi d_{ih}\overrightarrow{T_i} + \frac{\alpha+\delta}{2} d_{hi}\overleftarrow{T_i}$, where χ, α and δ are the unit rates for collection (origin-hub), transfer (hub-hub), and distribution (hub-destination), respectively (see, for instance, [24]).

The three types of *cost* functions above, combined with the two GRASP designs (the semi-greedy and the sampled greedy), are applied to populate *Pop*. As it will be shown, they have proven to be effective to obtain a balanced *Pop* set of good quality and diverse solutions. We call these methods DGM1 to DGM6, as shown in Table 1. A last method, called DGM7, is based on the notion of constructing solutions at random by simply generating random sets of p hubs, helping to create a limited amount of solutions not guided by an evaluation function to just bring diversity to *Pop*.

	Semi-greedy	Sampled greedy
Type 1	DGM1	DGM4
Type 2	DGM2	DGM5
Type 3	DGM3	DGM6

Table 1: Classification of the different diversification generation methods.

Once the p hubs are selected for a solution, the following step is to allocate r of the p hubs to each terminal. Let $H^i \subseteq H$ be the set of the r hubs assigned to terminal i in a solution. For any terminal i we compute the following estimation of the assignment cost of i to a hub h as:

$$assignment(i,h) = d_{ih}\overrightarrow{T_i} + \sum_{j \in V} d_{hj} t_{ij}.$$

Then, we assign to i the hub h_a with the lowest *assignment* cost. Then, $h_a \in H^i$. For the remaining assignments to i, the above expression is updated to reflect the previous assignments:

$$assignment(i,h) = d_{ih}\overrightarrow{T_i} + \sum_{j \in V \setminus H^i} d_{hj} t_{ij} - \sum_{u \in H^i} d_{iu} t_{iu}, \quad \forall h \in H \setminus H^i.$$

This process is done in a greedy way, selecting at each iteration the lowest assignment cost. Note that the *assignment* expressions are an estimation, because they assume that there is only one hub h in the path between any pair of nodes i and j, which is not necessarily true.

Finally, we route all the traffics at their minimum cost. For each pair of nodes i and j we have to determine the hubs $k \in H^i$ and $l \in H^j$ minimizing the routing cost of the traffic sent from i to j. In mathematical terms, given $k \in H^i$ and $l \in H^j$, we denote as $c_{ij}(k,l)$ the cost of transporting the traffics from i to j through hubs k and l, i.e.

$$c_{ij}(k,l) = t_{ij}(\chi d_{ik} + \alpha d_{kl} + \delta d_{lj}).$$

The routing cost from i to j, c_{ij}, is then obtained by searching the hubs $k \in H^i$ and $l \in H^j$ minimizing the expression above, i.e.

$$c_{ij} = \min_{k \in H^i, l \in H^j} c_{ij}(k,l).$$

Note that the time complexity of evaluating the routing costs if $\chi \neq \delta$ is $\mathcal{O}(n^2r^2)$, since the path $i \to k \to l \to j$ does not necessarily involve the same hubs k and l in the opposite direction. Note also that even if H^i and H^j have common hubs, it cannot be ensured that the best route from i to j will be through a common hub.

Once the p hubs have been located, r hubs have been assigned to each node and all the traffics have been routed, we have a feasible solution for the UrApHMP. A solution is denoted by $s = (H, A)$ and its cost by $f(s)$, where $H = \{h_1, \ldots, h_p\} \subseteq V$ is the set of hubs in the solution and $A = [a_{ij}]_{i=1,\ldots,n; j=1,\ldots,r}, a_{ij} \in H^i$, is a matrix whose rows contain the r hubs assigned to each node.

2.2 The Reference Set Construction Method

Using methods DGM1 to DGM7, we have generated π feasible solutions, but only β of them, the *best* ones, will become part of *RefSet*. The notion of *best* solution is not limited here to its quality, as a measure given by the objective function, but also to the diversity that each solution brings to *RefSet* in terms of its attributes.

The process to select the β solutions of *Pop* that will define *RefSet* is done as follows: As the SS methodology specifies [15], we want to choose $\frac{\beta}{2}$ solutions attending their quality. To do this, we order all solutions in *Pop* by ascending order of their costs and introduce them, one by one, only if there is no other solution already introduced in *RefSet* with the same cost. We stop this selection process after examining the 50% of the solutions in *Pop*, even if they are less than $\frac{\beta}{2}$ solutions. The rest of the solutions of *RefSet* will be selected from *Pop* by a diversity criterion process that tries to choose those solutions of *Pop* that differ most from *RefSet*.

Given $s \notin \mathit{RefSet}$ and $t \in \mathit{RefSet}$, let $C = \{h : h \in H_s \cap H_t\}$ be the set of common hubs in solutions s and t. We define $d_H(s,t) = p - |C|$ as the number of hubs in s not present in t (or viceversa). We can consider the lower the value of $d_H(s,t)$ is, the closer s and t are. To select the solution $s \in Pop$ to be included in *RefSet*, we define $dist(s, \mathit{RefSet}) = \min_{t \in \mathit{RefSet}} d_H(s,t)$. This distance is computed for all solutions in *Pop*, and the solution s^* with maximum value of $dist(s, \mathit{RefSet})$ is introduced in *RefSet*. Another distance function, d_A, has been defined to break a possible tie when two or more solutions $s_1, \ldots, s_z \in \mathit{Pop}$ take the same *dist* value ($dist(s_1, \mathit{RefSet}) = \ldots = dist(s_z, \mathit{RefSet})$). New distance d_A is based on the assignments of terminals to the hubs in C. Given $s \notin \mathit{RefSet}$, $t \in \mathit{RefSet}$, and $h \in C$, let T_s^h be the set of terminals assigned to hub h in solution s. Then $|T_s^h \cap T_t^h|$ gives the number of common assignments to h in both solutions. We define $d_A(s,t) = \min_{h \in C} |T_s^h \cap T_t^h|$. If a tie occurs, we select the solution with the maximum d_A value.

2.3 The Subset Generation Method

Once the *RefSet* has been created, the subset generation method (SGM) consists of producing, at any iteration, different sets $X \subset \mathit{RefSet}$ to serve as a basis for the application of the combination method later on. The SGM we propose works by ordering the solutions of *RefSet* in ascending order of their cost (from best to worst) and then generating subsets defined by any two different solutions. To avoid the repetition of previously generated subsets at earlier iterations, each subset is generated only if at least one of its solutions was introduced in *RefSet* in the preceding iteration. Suppose that m subsets were not considered for this reason, then the number of resulting subsets at each iteration for which the combination method will be applied is $\frac{\beta^2 - \beta}{2} - m$.

2.4 The Solution Combination Method

The solution combination method (SCM) is an element of scatter search that is context-dependent. Although it is possible to design "generic" combination procedures, we thought that it would be more effective to design the SCM based on the characteristics of our problem. Our proposal for the SCM is a path-relinking implementation that is applied to each subset generated in the previous step. As it has been mentioned, path-relinking is an intensification procedure that explores *paths* (also called *trajectories*) in the neighborhood space of two good solutions, generating intermediate solutions that can eventually be better than the two being connected.

More formally, let $\mathcal{G} = (F, M)$ be the search space graph, where node set F is the set of feasible solutions of the problem, and M is the set of edges associated with the moves in the neighborhood structure. Given two solutions $s, t \in F$, a move is defined as $(s,t) \in M$ if and only if $s \in \mathcal{N}(t)$ and $t \in \mathcal{N}(s)$, where $\mathcal{N}$ is a given neighborhood structure. The path-relinking operator explores a path $\mathcal{P}(s,t)$ that connects s and t with the objective of finding solutions $s^* \in \mathcal{P}(s,t)$ for which $f(s^*) < min\{f(s), f(t)\}$. Let s be the "initial" solution of the path and t its "guiding" solution. This path is generically accomplished by swapping out elements selected in s with elements in t, generating a set of *intermediate* solutions. To obtain a first intermediate solution s' in the path $\mathcal{P}(s,t)$, we remove a hub v from H_s and replace it by a hub u in H_t, thus obtaining $H_{s'} = H_s \setminus \{v\} \cup \{u\}$. Let $\Delta(s,t)$ be the set of attributes present in t but not in s. In our hub location problem, we define $\Delta(s,t)$ as the set of nodes that are hubs in t but not in s. In mathematical terms, $\Delta(s,t) = \{u : u \in H_t, u \notin H_s\}$. Note that one or more first intermediate solutions can be obtained when $|\Delta(s,t)| > 1$.

To combine the solutions in the subsets X generated from *RefSet*, the procedure we have designed works as follows: For each subset $X = \{s, t\}$, we explore the path between its two solutions. To do this, we consider as initial solution the one with worse value, s. Then, we calculate $\Delta(s,t)$. If $|\Delta(s,t)| \geq 2$, it means that at least two hubs of H_t are not in H_s. If $|\Delta(s,t)| \leq 1$, no path-relinking is necessary as there are no intermediate solutions between s and t. Our PR generates $m = |\Delta(s,t)|^2$ different solutions, and each terminal node of these m solutions needs to be assigned to r of the hubs as a previous step to know its value. The best of these m solutions, s_δ, is saved and the remaining $m-1$ solutions are discarded as possible steps of this path. Replace s^* by s_δ if $f(s_\delta) < f(s^*)$, where s^* is the best solution found in $\mathcal{P}(s,t)$ so far. This procedure is repeated while $|\Delta(s,t)| \geq 2$, and returns s^* as the output. Note that $f(s^*)$ is not necessarily better than $f(s)$ or $f(t)$. All the generated solutions s^* are stored in a set called *Pool*.

2.5 The Reference Set Update Method

The reference set update method (RSUM) is associated with each application of the SGM. The update operation consists of maintaining a record of the β best solutions found so far by the procedure. The issues related to this updating function are straightforward: All the solutions in *RefSet* that are worse than those in the current *Pool* will be replaced by these ones, with the aim of keeping in *RefSet* the β best and most different solutions found so far. Let $s \in Pool$ and $w \in RefSet$ such that $f(s) < f(w)$. In this case, s will replace w if s is different from all other solutions in *RefSet*.

2.6 The Improvement Method

As mentioned in Section 1, three optimization subproblems arise when solving the UrApHMP. Since we solve the routing subproblem optimally, we propose two improvement procedures based on local search strategies for the other two subproblems: LS_H for the hub selection, and LS_A for the terminal allocations. Both are based on the local search procedures proposed in [20].

LS_H implements a classical exchange procedure in which a hub h_i is removed from H, and a non-hub node $h'_i \in N \setminus H$ replaces h_i, thus obtaining $H' = \{h_1, h_2, \ldots, h'_i, \ldots, h_p\}$. Peiró, Corberán and Martí describe in [20] a mechanism to determine the order of exploration of the hubs in this procedure, but we have empirically checked that LS_H performs better in the SS scheme without this mechanism. Note that any change in H affects the other components in s. Specifically, when hub h_i is replaced by h'_i, we need to re-evaluate the hub assignment of at least all the vertices assigned to h_i. Moreover, the routes for the traffics are not necessarily optimal for the new set of hubs H'. Hence, since the solution structure changes that much, we evaluate all the routes from scratch. As a classical local search procedure, LS_H performs moves as long as the cost improves. We have implemented here the so-called *first strategy*, in which the first improving move is performed, instead of scanning the entire neighborhood to determine the best move.

The local search procedure LS_A is similar to LS_H, but it considers changes in the assignment of terminals to hubs. In particular, for a node i with $H^i =$

$\{h_{i1}, \dots, h_{ia}, \dots, h_{ir}\}$, this procedure exchanges an assigned hub with a non-assigned one. In mathematical terms, we replace h_{ia} with $\bar{h}_{ia} \in H \setminus H^i$, thus obtaining $\bar{H}^i = \{h_{i1}, \dots, \bar{h}_{ia}, \dots, h_{ir}\}$.

3 Computational experiments

This section describes the computational experiments performed to test the efficiency of the scatter search with path-relinking method we propose to solve the UrApHMP. The procedure has been implemented in C using GCC 4.8.2 with optimization flags -O3, -march=corei7 and -m64. The results reported in this section have been obtained with an Intel core i7–3770 at 3.40GHz and 16GB of RAM, under Ubuntu 14.04 GNU/Linux − 64 bits operating system. The metrics that we use to measure the performance of the algorithms in a particular experiment are:

- Dev: Average percentage deviation with respect to the best solution found (or from the optimal solution, if available).
- # Best: Number of best solutions found.
- CPU: Average computing time in seconds.

3.1 Test instances

We have tested our algorithms on the following three sets of instances:

1. The **CAB** (Civil Aviation Board) data set, based on airline passenger flows among some important cities in the United States. It consists of a data file, presented by O'Kelly in [18], with the distances and flows of a 25 nodes network. From this original file, 23 instances with 25 nodes and $p = \{1, \dots, 5\}$ and $r = \{1, \dots, p\}$ have been generated by several authors. The following parameter values have been widely used: $\chi = \delta = 1$, $\alpha = \{0.2, 0.4, 0.6, 0.8, 1.0\}$.
2. The **AP** (Australian Post) data set, based on real data from the Australian postal service and presented by Ernst and Krishnamoorthy in [6]. The size of the original data file is 200 nodes. Smaller instances can be obtained using a code from ORLIB. As with CAB, many authors have generated different instances from the original file. We have extended this set of instances by generating 311 instances with $n = \{40, 50, 60, 65, 70, 75, 80, 85, 90, 95, 100, 150, 200\}$. For those with $40 \leq n \leq 95$, p varies from 1 to 10, and for those with $100 \leq n \leq 200$, p takes values between 1 and 20. In all the cases, $r \in \{1, \dots, p\}$. According with previous articles, cost parameter values have been chosen as $\chi = 3$, $\alpha = 0.75$ and $\delta = 2$. The flows between nodes are not symmetric in these instances (i.e., for a given pair of nodes i and j, $t_{ij} \neq t_{ji}$, in general). Moreover, flows from one node to itself can be positive.
3. The **USA423** data set. This family of instances was introduced by Peiró, Corberán and Martí in [20], and is based on real airline data. It consists of a data file concerning 423 cities in the United States, where real distances and passenger flows for an accumulated three months period are

considered. From the original data, 30 instances have been generated with $p \in \{3,4,5,6,7\}$ and $2 \leq r \leq p-1$. For each combination of parameters p and r, two different values for χ, α, δ have been used: 0.1, 0.07, 0.09, and 0.09, 0.075, 0.08, respectively.

The experiments are divided into two main blocks. The first block, described in Section 3.2, is devoted to study the behavior of the components of the solution procedure, as well as to determine the best values for the search parameters. The second block of experiments, in Section 3.3, has the goal of comparing our procedure with the best published methods. Regarding the size of the instances, they have been classified as small ($25 \leq n \leq 50$), medium ($55 \leq n \leq 100$), and large ($105 \leq n \leq 200$). The entire set of instances is available at `www.optsicom.es`.

3.2 Parameter calibration

The first set of experiments to calibrate our method is performed on a subset of 47 instances: five instances from the CAB set with $n = 25$, and 42 instances with $40 \leq n \leq 200$ from the AP set. We refer to these 47 instances as the *training set* and to the remaining instances as the *testing set*.

The values of π, q and φ: We first study the values for the parameters used in the diversification generator method: π (that determines how many solutions will be constructed in *Pop*), q (that defines the size of the RCL in the DGM1-DGM6 methods), and φ (that determines the number of elements in the evaluation for the selection of hubs).

First, we have given parameter π two possible values: 100 and 150. For each instance of the training set, we have constructed 0.15π solutions with each of the methods DGM1 to DGM6, to obtain a 90% of the initial solutions generated. The remaining solutions (up to π) are obtained with DGM7, the random generator. In order to evaluate the capacity of the DGM methods without taking into account the effect of the combination and improvement methods, this experiment only considers the constructive phase, not performing any of the subsequent elements of the scatter search procedure. The results on the training set are shown in Table 2.

As expected (see Table 2), the best solutions in terms of quality are obtained with $\pi = 150$. In this case, the algorithm obtains an average percentage deviation of 1.4% and 31 best known solutions. The CPU time for $\pi = 150$ is still reasonable, with virtually no difference in small and medium instances. Although for the large instances, the CPU values are slightly larger, we consider that the enhancement of the results worth the CPU effort, so we set $\pi = 150$ in the rest of the experiments.

In the second experiment, we study the value of the parameter q that defines the RCL size. To do this we have considered $q = \min\{\frac{n}{2}, \omega p\}$, where ω is another parameter whose impact is studied next. For each instance of the training set, we have constructed $\frac{1}{6}\pi$ solutions with each DGM1 to DGM6 method but not with DGM7, because it is a totally random procedure. As in the previous experiment, we have only considered the constructive phase. The results for $\omega = 2,3,4,5$ are presented in Table 3. Note that the best solutions in terms of quality are obtained with $\omega = 5$. With this value, we have obtained an average

		Dev (%)		**# Best**		**CPU**	
Size	**# inst**	**100**	**150**	**100**	**150**	**100**	**150**
s (small)	8	2.1	0.0	2	8	0.0	0.0
m (medium)	27	2.2	0.6	8	19	0.2	0.3
l (large)	12	2.0	4.1	8	4	1.2	1.8
	47	2.1	1.4	18	31	0.4	0.6

Table 2: Calibration of π for the DGM of SS.

percentage deviation of 1.4% and 26 best known solutions. In order to compare the results, we have performed the well-known non-parametric Friedman test. This test answers the question: Do the solutions obtained with the different methods represent different populations? The resulting probability value of 0.000006981 indicates that the compared values come from different methods. Moreover, we do not appreciate differences in the CPU times among the different values of ω, so we set $\omega = 5$ from now on.

		Dev (%)				**# Best**				**CPU**			
Size	**# inst**	**2**	**3**	**4**	**5**	**2**	**3**	**4**	**5**	**2**	**3**	**4**	**5**
s	8	2.1	1.7	1.8	0.6	5	1	3	4	0.0	0.0	0.0	0.0
m	27	7.4	5.0	3.6	1.3	3	4	4	16	0.2	0.2	0.2	0.2
l	12	8.9	3.3	3.4	2.4	1	2	3	6	1.1	1.1	1.2	1.2
	47	6.9	4.0	3.2	1.4	9	7	10	26	0.4	0.4	0.4	0.4

Table 3: Calibration of ω for the DGM of SS.

Now we study the value of φ, the number of terminals appearing in the computation of $g(h)$. This number is computed as $\varphi = \lfloor \lambda \frac{n}{p} \rfloor$, where $\lambda \in \{1, 1.2, 1.5, 1.7, 2\}$. Again, we have constructed $\frac{1}{6}\pi$ solutions with methods DGM1 to DGM6 for each instance of the training set, using only the constructive phase of SS. The results for the different values of λ are shown in Table 4. It seems there is no a clear value of λ outperforming the others, hence, we have compared the results using the Friedman test. The resulting probability value of 0.1461 confirms that the compared values do not present significant differences. Considering that the best average deviation was found for $\lambda = 1$, $\varphi = \lfloor \frac{n}{p} \rfloor$ is the value we have finally chosen for the rest of the experiments.

		Dev (%)					**# Best**					**CPU**				
Size	**# inst**	**1**	**1.2**	**1.5**	**1.7**	**2**	**1**	**1.2**	**1.5**	**1.7**	**2**	**1**	**1.2**	**1.5**	**1.7**	**2**
s	8	1.4	3.1	3.1	3.1	3.0	4	1	0	3	1	0.0	0.0	0.0	0.0	0.0
m	27	3.1	4.3	3.8	2.9	2.6	7	3	3	9	7	0.2	0.2	0.2	0.2	0.2
l	12	3.0	1.5	5.0	4.5	3.0	3	3	1	4	4	1.3	1.3	1.3	1.3	1.3
	47	2.7	3.4	4.0	3.3	2.8	14	7	4	16	12	0.5	0.4	0.4	0.4	0.4

Table 4: Calibration of φ through λ for the DGM of SS.

The value of β: In order to study the size of *RefSet* (β), we have constructed 0.15π solutions with DGM1 to DGM6 for each instance of the training set. The remaining solutions up to π are obtained with DGM7. In this experiment, we include in the algorithm all elements of scatter search except the local searches, to evaluate the power of the combination method (SCM) without the effects of the improvement procedures. As it can be seen in Table 5, the higher the value of β, the better are the results. However, this obviously implies higher computing times. We have compared the results using the Friedman test for all the values of β tested. The resulting probability value of 0.0000 indicates that the results obtained for the different values of β are significantly different. An important issue is to know if a similar deviation to the one obtained with $\beta = 10$ can be obtained in shorter times with a smaller size of β by using the improvement methods. This issue is the subject of the next experiment.

		Dev (%)						**# Best**						**CPU**					
Size	**# inst**	**5**	**6**	**7**	**8**	**9**	**10**	**5**	**6**	**7**	**8**	**9**	**10**	**5**	**6**	**7**	**8**	**9**	**10**
s	8	1.0	0.0	0.0	0.0	0.0	0.0	5	8	8	8	8	8	0.1	0.1	0.2	0.2	0.4	0.4
m	27	3.7	2.0	1.3	0.5	0.4	0.1	3	6	7	8	13	20	1.2	2.1	3.1	4.5	6.2	8.1
l	12	4.4	1.9	1.9	0.3	0.2	0.1	0	2	2	4	8	7	4.9	8.0	12.1	15.7	23.3	29.0
	47	3.4	1.6	1.2	0.4	0.3	0.1	8	16	17	20	29	35	2.0	3.3	4.9	6.6	9.6	12.2

Table 5: Calibration of β for the DGM of SS.

The effect of local searches: Now we study the effect of the improvement procedures described in Subsection 2.6. Recall that two local searches have been proposed, one for the hub selection (LS_H) and another for the terminal allocations (LS_A). Note that the standard SS design specifies to apply the improvement method to all the solutions resulting from the combination method. Some previous works have proposed a selective implementation of the local search procedures, reducing their application to only the best solutions obtained from the combination method in each global iteration. Since, due to the combinatorial nature of this problem, the local searches we propose are quite time consuming, in this paper we go a step further and limit the application of the improvement method to only the best solutions across all the global iterations. Specifically we have designed our procedure in such a way it applies LS_H and/or LS_A only to the solutions of *RefSet* just before the end of the algorithm. To evaluate the effect of the proposed local searches, we have studied the following four variants:

A: LS_H and LS_A are applied to all the solutions.

B: LS_H and LS_A are applied to the best solution only.

C: LS_A is applied to all the solutions.

D: LS_A is applied to the best solution only.

In what follows we compare each variant above for each value of β (from 5 to 10). Table 6 shows the results obtained on the 47 instances of the training set. Variants C and D exhibit the worst results in terms of the number of best solutions found and the deviation with respect to the best solutions found in this experiment. The results clearly indicate that applying LS_H to the solutions in *RefSet* substantially improves their value (from 2.5% to 4.5% on average). Variant A exhibits a larger number of best solutions found compared to variant B, what confirms that applying LS_H and LS_A to all the solutions of *RefSet* is useful to match the best known solutions. Nevertheless, note that this exhaustive application of the local search procedures to all the solutions in *RefSet* results in much higher CPU times (from 11.15 to 31.45 seconds in variant A versus 3.94 to 15.24 in variant B). Despite the fact that variant B is not able to get some of the best known solutions, the average deviation is very small (from 0.1% to 0.2%). This indicates that, for short computing times, variant B obtains very good solutions in terms of average deviation of the cost and, hence, is also a very good option. Both variants, A and B, are the ones chosen to compare in Subsection 3.3 the SS procedure with the previously proposed methods for the UrApHMP.

			Dev (%)						**# Best**						**CPU**					
Var	**Size**	**# inst**	**5**	**6**	**7**	**8**	**9**	**10**	**5**	**6**	**7**	**8**	**9**	**10**	**5**	**6**	**7**	**8**	**9**	**10**
	s	8	0.0	0.0	0.0	0.0	0.0	0.0	7	8	8	8	8	8	0.2	0.2	0.3	0.4	0.5	0.6
A	*m*	27	0.1	0.0	0.0	0.1	0.1	0.0	21	22	24	18	21	24	4.6	6.0	7.4	9.1	11.5	14.0
	l	12	0.0	0.0	0.2	0.0	0.0	0.1	8	10	10	11	10	7	33.3	43.7	51.8	64.8	81.4	91.3
		47	0.1	0.0	0.1	0.1	0.1	0.0	36	40	42	37	39	39	11.2	14.6	17.5	21.8	27.5	31.4
	s	8	0.1	0.1	0.1	0.1	0.1	0.1	7	7	7	7	7	7	0.1	0.1	0.2	0.2	0.3	0.4
B	*m*	27	0.2	0.2	0.2	0.3	0.2	0.0	14	16	18	12	13	19	2.0	2.7	3.9	5.1	6.7	8.3
	l	12	0.1	0.1	0.4	0.2	0.1	0.2	6	5	5	3	6	5	10.8	13.8	17.7	25.5	34.4	40.7
		47	0.2	0.2	0.2	0.2	0.1	0.1	27	28	30	22	26	31	3.9	5.1	6.8	9.4	12.7	15.2
	s	8	3.0	2.7	2.7	2.4	2.4	2.4	0	0	0	0	0	0	0.1	0.1	0.2	0.3	0.4	0.5
C	*m*	27	3.6	2.6	2.3	1.7	1.5	1.6	0	0	0	2	3	2	2.5	3.3	4.7	6.1	8.0	9.8
	l	12	7.6	6.7	5.7	5.2	4.9	4.6	0	0	0	0	0	0	16.1	22.9	27.3	37.8	52.1	61.7
		47	4.5	3.7	3.3	2.7	2.5	2.5	0	0	0	2	3	2	5.6	7.8	9.7	13.2	18.0	21.4
	s	8	3.0	2.7	2.7	2.4	2.4	2.4	0	0	0	0	0	0	0.1	0.1	0.2	0.2	0.3	0.4
D	*m*	27	3.6	2.6	2.3	1.7	1.5	1.6	0	0	0	2	3	2	1.6	2.3	3.4	4.7	6.4	8.0
	l	12	7.6	6.7	5.7	5.2	4.9	4.6	0	0	0	0	0	0	7.6	11.0	14.5	21.7	30.9	37.7
		47	4.5	3.7	3.3	2.7	2.5	2.5	0	0	0	2	3	2	2.9	4.1	5.7	8.3	11.6	14.3

Table 6: Computational results obtained with the four variants for the local search procedures.

Regarding the β parameter, and considering only variants A and B, it can be observed that the best results in terms of Dev are obtained with values $\beta = 6$ and $\beta = 10$. In order to compare both sets of results, we have performed two well-known non-parametric tests for pairwise comparisons: the Wilcoxon test and the Sign test. The Wilcoxon test answers the question: Do the two samples (in our case, the solutions obtained with $\beta = 6$ and $\beta = 10$) represent two different populations? The resulting probability value of 0.50 indicates that there are not statistical differences, meaning that the compared values do not come from different methods. The Sign test computes the number of instances on which an algorithm beats the other one. The resulting probability value of 1.0 corroborates the previous result. As the CPU effort is clearly lower with $\beta = 6$ than with $\beta = 10$, we select $\beta = 6$ from now on.

3.3 Comparing the proposed SS with previous methods

After the calibration process described before, we now compare the performance of our SS algorithm versus the GRASP procedure proposed in [20], which as far as we know is the best heuristic algorithm for the UrApHMP. All the methods under comparison are run in the same computer.

As a summary, our SS with PR procedure is set as follows: we generate 150 initial solutions; the size of the RCL when constructing solutions with methods DGM1 to DGM6 is defined by $\min\{\frac{n}{2}, 5p\}$; $\lfloor \frac{n}{p} \rfloor$ terminals are considered for the evaluation of each hub candidate; the size of *RefSet* is set to six solutions; and both local search procedures LS_H and LS_A are applied to the solutions of *RefSet*, using variants A and B, just before the end of the algorithm.

The results with the two versions of the algorithm corresponding to variants A and B, denoted SS^A and SS^B respectively, are shown in the Appendix and summarized in Table 7. This table clearly shows that, except in the small size instances, our SS procedures outperform the previously proposed GRASP method. In particular, although SS^A and SS^B match 16 out of 30 small size instances while GRASP is able to match 25, and their deviation is 0.10% versus 0.06% in GRASP, the CPU time used by the SS methods is much shorter. Regarding the medium size instances, we can observe that the behavior of SS^B and GRASP are similar, although the former uses 1/5 of the GRASP computing time. In these instances, SS^A performs better than GRASP (108 best solutions found and Dev 0.02% versus 79 best and Dev 0.18%) using less than 1/2 of the GRASP computing time. Finally, in what refers to the large size instances, both SS versions clearly outperform GRASP in terms of Dev (0.01% and 0.07% versus 0.23% on average). Moreover, SS^A is able to find a larger number of best solutions (40 against 26 of SS^B and 29 of GRASP, respectively), although at a bigger computing time.

			Dev (%)			**# Best**			**CPU**		
Size	n	**# inst**	**SS^A**	**SS^B**	**GRASP**	**SS^A**	**SS^B**	**GRASP**	**SS^A**	**SS^B**	**GRASP**
	25	18	0.1	0.1	0.1	11	11	15	0.1	0.1	0.4
s	40	6	0.1	0.1	0.0	1	1	5	0.3	0.2	3.2
	50	6	0.0	0.0	0.0	4	4	5	0.6	0.3	9.4
	summary	30	0.1	0.1	0.1	16	16	25	0.2	0.1	2.8
	60	21	0.0	0.2	0.4	19	11	5	2.7	1.4	4.6
	65	19	0.0	0.5	0.3	12	7	10	3.3	1.6	6.4
	70	19	0.0	0.2	0.3	14	10	15	4.2	2.0	11.0
	75	18	0.0	0.2	0.0	9	9	15	4.5	2.1	9.2
m	80	18	0.0	0.3	0.1	13	8	8	6.5	3.0	14.9
	85	18	0.0	0.1	0.1	13	12	9	6.7	3.0	15.6
	90	21	0.0	0.1	0.1	15	12	8	8.0	3.6	22.9
	95	21	0.0	0.0	0.1	13	10	9	9.8	4.1	24.3
	summary	155	0.0	0.2	0.2	108	79	79	5.8	2.6	13.8
	100	21	0.0	0.0	0.1	13	10	9	11.7	4.9	4.8
l	150	20	0.0	0.0	0.3	13	10	12	40.5	13.7	20.6
	200	21	0.0	0.2	0.2	14	6	8	86.1	27.5	58.9
	summary	62	0.0	0.1	0.2	40	26	29	46.2	15.4	28.2

Table 7: Computational results on the CAB and AP instances

In order to compare the results obtained with SS^A, SS^B, and GRASP from a statistical point of view, we have performed first the non-parametric Friedman test. The resulting probability value of 0.000 indicates that the results are significantly different. The ranks values produced by this test are 1.77 for SS^A, 2.09 for GRASP, and 2.15 for SS^B. Then, we have performed the non-parametric Wilcoxon and Sign tests for pairwise comparisons. When comparing SS^A with SS^B and SS^A with GRASP, the resulting probability values of 0.000 indicate that the compared results come from different methods. When comparing SS^B with GRASP, the resulting probability of 0.53 indicates that there are no significant differences between the two methods regarding the results obtained.

To complement this analysis, we have carried out a final comparison on very large size instances. Table 8 shows the result of SS^A, SS^B, and GRASP on the 30 USA423 instances described in Subsection 3.1. Procedures SS^A and SS^B have been run with the parameters specified at the beginning of this section, while GRASP parameters' values are specified in [20]. As shown in this table, SS^A and GRASP are run for the same running time. This table confirms the superiority of SS^A compared to GRASP, since the former exhibits an average percent deviation of 0.3% and GRASP only achieves a 4.5%. The resulting probability of 0.01 of the non-parametric pairwise Wilcoxon test confirms this conclusion. Additionally, SS^B is very competitive since it obtains slightly better results than GRASP in shorter computing times.

					Solution value			**Dev (%)**			**CPU**		
p	r	χ	α	δ	**SS^A**	**SS^B**	**GRASP**	**SS^A**	**SS^B**	**GRASP**	**SS^A**	**SS^B**	**GRASP**
3	2	0.1	0.07	0.09	33727412959	40576260062	43867093585	0.0	20.3	30.1	99.0	30.3	100
4	2	0.1	0.07	0.09	31555933763	31555933763	31540232352	0.0	0.0	0.0	169.1	47.0	170
4	3	0.1	0.07	0.09	31378516085	31378516085	41500538133	0.0	0.0	32.3	259.3	63.3	260
5	2	0.1	0.07	0.09	29361690398	29361690398	29428603263	0.0	0.0	0.2	227.4	73.9	230
5	3	0.1	0.07	0.09	29012500636	29012500636	29005134361	0.0	0.0	0.0	337.1	98.3	340
5	4	0.1	0.07	0.09	28993057993	28993057993	35768287695	0.0	0.0	23.4	443.5	145.5	445
6	2	0.1	0.07	0.09	28761333285	29648009936	29813468007	0.0	3.1	3.7	348.0	116.0	350
6	3	0.1	0.07	0.09	28140175764	28636803830	27999028069	0.5	2.3	0.0	474.8	118.9	475
6	4	0.1	0.07	0.09	27952573758	28449216973	27690657697	0.9	2.7	0.0	545.6	144.3	550
6	5	0.1	0.07	0.09	27937297018	27937297018	27688845102	0.9	0.9	0.0	660.6	212.5	665
7	2	0.1	0.07	0.09	28076360663	28076360663	30799812597	0.0	0.0	9.7	416.6	122.9	420
7	3	0.1	0.07	0.09	26692507210	27724315371	26751472180	0.0	3.9	0.2	561.4	179.4	565
7	4	0.1	0.07	0.09	26258768165	26258768165	27537936675	0.0	0.0	4.9	717.9	207.6	720
7	5	0.1	0.07	0.09	26255012774	26255012774	26989369645	0.0	0.0	2.8	944.1	247.9	945
7	6	0.1	0.07	0.09	26255010934	26255010934	26255010934	0.0	0.0	0.0	1115.1	309.1	1120
3	2	0.09	0.075	0.08	30176127832	36338055076	30176127832	0.0	20.4	0.0	112.1	34.2	115
4	2	0.09	0.075	0.08	28373652481	28373652481	30530137285	0.0	0.0	7.6	173.5	61.3	175
4	3	0.09	0.075	0.08	28125164296	28125164296	28125164296	0.0	0.0	0.0	297.7	70.3	300
5	2	0.09	0.075	0.08	26451114267	26451114267	26492913729	0.0	0.0	0.2	232.3	82.8	235
5	3	0.09	0.075	0.08	26088080203	26088080203	26079665626	0.0	0.0	0.0	360.6	111.8	365
5	4	0.09	0.075	0.08	26061806930	26061806930	27249791197	0.0	0.0	4.6	471.3	138.6	475
6	2	0.09	0.075	0.08	26525421601	26525421601	26016295685	2.0	2.0	0.0	318.7	118.7	320
6	3	0.09	0.075	0.08	24971043351	25449149342	25788838779	0.0	1.9	3.3	497.9	155.3	500
6	4	0.09	0.075	0.08	25265904571	25265904571	24828738291	1.8	1.8	0.0	631.6	203.9	635
6	5	0.09	0.075	0.08	25243965930	25243965930	25385565293	0.0	0.0	0.6	853.8	264.2	855
7	2	0.09	0.075	0.08	25906507439	29264746888	26592882079	0.0	13.0	2.6	413.9	130.5	415
7	3	0.09	0.075	0.08	24784471473	25578317735	24066130754	3.0	6.3	0.0	636.6	192.6	640
7	4	0.09	0.075	0.08	23586398612	24971605186	23619010159	0.0	5.9	0.1	765.5	220.9	770
7	5	0.09	0.075	0.08	23577554532	24905755621	24220406570	0.0	5.6	2.7	1085.47	326.35	1090
7	6	0.09	0.075	0.08	23577543900	24887707334	25017677542	0.0	5.6	6.1	1356.71	500.71	1360
								0.3	3.2	4.5	517.6	157.6	520.1

Table 8: Computational results on the USA423 instances

As a final test, we compare the performance of SS^A and SS^B with the results of the evolutionary approach recently proposed by Milanović [17] for the multiple version of the p-hub median problem. The proposed SS procedures are applied on 40 AP instances of different sizes, where $r = p$. Table 9 shows, in each row, the size and the value of p on each instance, and, for each method, the value of the objective function, the average percentage deviation with respect to the best known solution, and the CPU time. Results for the evolutionary method are directly taken from [17], and therefore running times are only indicative and cannot be directly compared with the SS computing times.

Results in Table 9 clearly show that both SS procedures are able to achieve state-of-the-art results for the multiple allocation problem, with a similar quality to those reported with the evolutionary method in [17], even though this evolutionary method is specifically designed for this particular version of the problem.

		Solution value			**Dev (%)**			**CPU**		
n	p	**SS**^{A}	**SS**^{B}	**Evo**	**SS**^{A}	**SS**^{B}	**Evo**	**SS**^{A}	**SS**^{B}	**Evo**
40	2	173415.5	173415.5	173415.5	0.0	0.0	0.0	0.1	0.0	0.1
	3	155458.1	155458.1	155458.1	0.0	0.0	0.0	0.2	0.1	0.0
	4	140682.2	140682.2	140682.2	0.0	0.0	0.0	0.4	0.2	0.4
	5	130384.1	130384.1	130384.1	0.0	0.0	0.0	0.6	0.4	0.5
	6	122170.2	122170.2	122170.2	0.0	0.0	0.0	1.1	0.6	5.9
	7	116035.9	116035.9	116035.9	0.0	0.0	0.0	1.5	0.9	7.1
	8	109971.1	109971.1	109971.1	0.0	0.0	0.0	2.1	1.5	8.4
	9	104211.5	104211.5	104211.5	0.0	0.0	0.0	2.6	1.6	9.6
	10	99451.8	99451.8	99451.8	0.0	0.0	0.0	4.2	2.6	12.5
50	2	174390.6	174390.6	174390.6	0.0	0.0	0.0	0.1	0.1	0.1
	3	156014.6	156014.6	156014.5	0.0	0.0	0.0	0.3	0.1	0.3
	4	141154.3	141154.3	141154.3	0.0	0.0	0.0	0.7	0.3	0.6
	5	129414.2	129507.4	129414.2	0.0	0.1	0.0	1.0	0.5	0.8
	6	121673.6	121673.6	121673.6	0.0	0.0	0.0	2.2	1.3	8.5
	7	115913.4	115913.4	115911.6	0.0	0.0	0.0	3.0	1.5	10.3
	8	111139.4	111139.4	109927.6	1.1	1.1	0.0	3.9	2.4	12.5
	9	104968.8	104968.8	104968.8	0.0	0.0	0.0	6.7	3.9	15.2
	10	100509.3	100645.4	100509.2	0.0	0.1	0.0	9.7	6.3	18.0
100	2	176246.8	176246.8	176246.8	0.0	0.0	0.0	0.7	0.3	5.1
	3	157870.9	157870.9	157870.9	0.0	0.0	0.0	2.2	0.8	13.3
	4	143004.4	143086.4	143004.3	0.0	0.1	0.0	3.8	1.3	18.5
	5	133483.0	133483.0	133483.0	0.0	0.0	0.0	7.6	2.6	23.8
	6	126107.9	126107.9	126107.9	0.0	0.0	0.0	16.0	6.3	31.3
	7	120164.6	120164.6	120164.6	0.0	0.0	0.0	24.5	10.9	41.3
	8	115144.7	115144.7	114295.9	0.7	0.7	0.0	25.7	14.4	57.0
	9	109449.1	109449.1	109449.1	0.0	0.0	0.0	38.7	20.2	68.7
	10	104800.8	104800.8	104794.3	0.0	0.0	0.0	71.6	45.3	87.2
	15	88882.5	89273.6	88882.5	0.0	0.4	0.0	226.9	148.0	167.1
	20	79453.7	79453.7	79191.6	0.3	0.3	0.0	564.9	385.1	233.9
200	2	178094.0	178094.0	178094.0	0.0	0.0	0.0	5.0	1.9	44.2
	3	159725.1	159725.1	159725.1	0.0	0.0	0.0	13.8	5.0	73.9
	4	144508.2	144508.2	144508.2	0.0	0.0	0.0	36.1	11.9	95.5
	5	136761.8	136761.8	136761.8	0.0	0.0	0.0	56.2	16.8	156.2
	6	129556.5	130739.0	129556.5	0.0	0.9	0.0	124.7	24.6	185.7
	7	123608.9	124132.2	123608.9	0.0	0.4	0.0	153.6	51.0	226.3
	8	117879.5	117879.5	117710.0	0.1	0.1	0.0	231.8	85.9	285.0
	9	112374.5	112374.5	112374.5	0.0	0.0	0.0	389.1	123.0	366.9
	10	107846.8	108913.6	107846.8	0.0	1.0	0.0	485.7	201.3	432.7
	15	92806.9	92920.7	92669.6	0.1	0.3	0.0	1566.4	965.0	816.4
	20	83385.9	83385.9	83385.9	0.0	0.0	0.0	3857.5	2422.7	1130.0
					0.1	0.1	0.0	198.6	114.2	116.8

Table 9: Computational results on AP instances for the multiple allocation version

4 Conclusions

In this article, we have proposed a new metaheuristic algorithm based on scatter search for the uncapacitated r-allocation p-hub median problem. This problem was introduced by Yaman [24] as a generalization of the classical single and multiple p-hub median problem. The proposed scatter search procedure incorporates several designs for the diversification generator method, a path-relinking procedure for combining solutions, and two local search procedures as the improvement method. The computational experiments on a large set of instances from the literature show that our algorithm is able to find high-quality solutions in short computing times, and outperforms a previously published GRASP procedure.

It is worth mentioning that our scatter search design only applies the improvement method at the end of the search. This selective application reduces the CPU time without sacrifying the final solution quality, making our method competitive in both quality of solutions and speed.

We hope that the good solutions obtained in this version of a hub location problem trigger the interest of researchers to apply the scatter search methodology to other problems in this family.

Acknowledgements

This work was supported by the Spanish Ministerio de Economía y Competitividad (projects TIN-2012-35632-C02, MTM-2012-36163-C06-02, and grant BES-2013-064245) and by the Generalitat Valenciana (project Prometeo 2013/049). This support is gratefully acknowledged.

References

[1] Alumur, S., and Kara, B. Y. Network hub location problems: The state of the art. *European Journal of Operational Research 190*, 1 (2008), 1–21.

[2] Campbell, J. F., and O'Kelly, M. E. Twenty-five years of hub location research. *Transportation Science 46*, 2 (2012), 153–169.

[3] Contreras, I., and Fernández, E. General network design: A unified view of combined location and network design problems. *European Journal of Operational Research 219*, 3 (2012), 680–697.

[4] Correia, I., Nickel, S., and Saldanha da Gama, F. Hub and spoke network design with single-assignment, capacity decisions and balancing requirements. *Applied Mathematical Modelling 35*, 10 (2011), 4841–4851.

[5] Díaz, J. A., and Fernández, E. Hybrid scatter search and path relinking for the capacitated p-median problem. *European Journal of Operational Research 169*, 2 (2006), 570–585.

[6] Ernst, A. T., and Krishnamoorthy, M. Efficient algorithms for the uncapacitated single allocation p-hub median problem. *Location Science 4*, 3 (1996), 139–154.

[7] Farahani, R. Z., Hekmatfar, M., Arabani, A. B., and Nikbakhsh, E. Hub location problems: A review of models, classification, solution techniques, and applications. *Computers & Industrial Engineering 64*, 4 (2013), 1096–1109.

[8] Feo, T. A., and Resende, M. G. C. Greedy randomized adaptive search procedures. *Journal of Global Optimization 6*, 2 (1995), 109–133.

[9] Festa, P., and Resende, M. G. C. Grasp: Basic components and enhancements. *Telecommunication Systems 46*, 3 (2011), 253–271.

[10] Gelareh, S., and Nickel, S. Hub location problems in transportation networks. *Transportation Research Part E: Logistics and Transportation Review 47*, 6 (2011), 1092–1111.

[11] Glover, F. A template for scatter search and path relinking. In *Artificial Evolution*, J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, Eds., Lecture Notes in Computer Science 1363, Springer (1998), pp. 13–54.

[12] Glover, F. Tabu search and adaptive memory programing – advances, applications and challenges. In *Interfaces in Computer Science and Operations Research* (1996), Kluwer, pp. 1–75.

[13] Glover, F., Laguna, M., and Martí, R. Fundamentals of scatter search and path relinking. *Control and Cybernetics 29*, 3 (2000), 652–684.

[14] Labbé, M., and Yaman, H. Solving the hub location problem in a star-star network. *Networks 51*, 1 (2008), 19–33.

[15] Laguna, M., and Martí, R. *Scatter Search: Methodology and Implementations in C.* Kluwer Academic Publishers (2002).

[16] Martí, R., Laguna, M., and Glover, F. Principles of scatter search. *European Journal of Operational Research 169*, 2 (2006), 359–372.

[17] Milanović, M. A new evolutionary based approach for solving the uncapacitated multiple allocation p-hub median problem. In *Soft Computing in Industrial Applications*, X. Z. Gao, A. Gaspar-Cunha, M. Köppen, G. Schaefer, and J. Wang, Eds., Advances in Intelligent and Soft Computing 75, Springer (2010), pp. 81–88.

[18] O'Kelly, M. E. A quadratic integer program for the location of interacting hub facilities. *European Journal of Operational Research 32*, 3 (1987), 393–404.

[19] Pacheco, J. A., and Casado, S. Solving two location models with few facilities by using a hybrid heuristic: A real health resources case. *Computers & Operations Research 32*, 12 (2005), 3075–3091.

[20] Peiró, J., Corberán, A., and Martí, R. GRASP for the uncapacitated r-allocation p-hub median problem. *Computers & Operations Research 43*, 1 (2014), 50–60.

[21] Resende, M. G. C., Ribeiro, C. C., Glover, F., and Martí, R. Scatter search and path-relinking: Fundamentals, advances, and applications. In *Handbook of Metaheuristics*, M. Gendreau and J. Y. Potvin, Eds., International Series in Operations Research & Management Science 146. Springer (2010), pp. 87–107.

[22] Resende, M. G. C., and Werneck, R. F. A hybrid heuristic for the p-median problem. *Journal of Heuristics 10*, 1 (2004), 59–88.

[23] Ribeiro, C. C., and Resende, M. G. C. Path-relinking intensification methods for stochastic local search algorithms. *Journal of Heuristics 18*, 2 (2012), 193–214.

[24] Yaman, H. Allocation strategies in hub networks. *European Journal of Operational Research 211*, 3 (2011), 442–451.

Appendix

Medium-sized hard instances												
						DEV (%)			CPU			
Inst	p	r	χ	α	δ	**SS**^{A}	**SS**^{B}	**GRASP**	**SS**^{A}	**SS**^{B}	**GRASP**	**Best known**
AP60	3	2	3	0.75	2	0.0	0.0	1.3	0.41	0.17	0.30	157493.38
AP60	4	2	3	0.75	2	0.0	0.3	1.8	0.62	0.27	0.48	142412.35
AP60	4	3	3	0.75	2	0.0	1.7	0.1	0.87	0.39	1.24	142268.41
AP60	5	2	3	0.75	2	0.0	0.2	0.0	0.90	0.44	1.06	130186.83
AP60	5	3	3	0.75	2	0.0	0.0	0.2	1.19	0.57	2.18	129680.76
AP60	5	4	3	0.75	2	0.0	0.0	0.1	1.52	0.72	2.86	129652.93
AP60	6	2	3	0.75	2	0.0	0.0	0.0	1.38	0.70	1.52	122365.30
AP60	6	3	3	0.75	2	0.0	0.3	0.3	1.93	0.96	1.63	122066.63
AP60	6	4	3	0.75	2	0.0	0.3	0.3	2.54	1.22	4.51	121979.40
AP60	6	5	3	0.75	2	0.0	0.3	0.0	3.11	1.51	4.55	121979.40
AP60	7	2	3	0.75	2	0.0	0.0	0.1	1.57	0.96	2.76	116380.78
AP60	7	3	3	0.75	2	0.0	0.2	0.5	2.45	1.24	3.22	116003.39
AP60	7	4	3	0.75	2	0.0	0.1	0.1	3.21	1.43	5.09	115959.96
AP60	7	5	3	0.75	2	0.0	0.1	0.1	4.20	1.89	7.31	115951.78
AP60	7	6	3	0.75	2	0.0	0.0	0.1	4.64	2.10	7.75	115951.78
AP60	8	2	3	0.75	2	0.0	0.0	1.1	2.05	1.30	2.98	110041.02
AP60	8	3	3	0.75	2	0.0	0.0	0.4	2.96	1.49	6.40	109888.11
AP60	8	4	3	0.75	2	0.0	0.0	0.5	3.65	2.06	6.82	109668.92
AP60	8	5	3	0.75	2	0.0	0.0	1.2	4.61	2.53	6.53	109651.38
AP60	8	6	3	0.75	2	0.0	0.0	0.0	6.73	3.62	11.39	109651.38
AP60	8	7	3	0.75	2	0.0	0.0	0.0	6.71	3.71	15.80	109651.38
AP65	3	2	3	0.75	2	0.0	2.1	0.0	0.43	0.21	0.69	157509.77
AP65	4	2	3	0.75	2	0.1	0.7	0.0	0.82	0.40	0.82	142702.65
AP65	4	3	3	0.75	2	0.0	0.0	0.3	0.94	0.43	1.18	142632.42
AP65	5	2	3	0.75	2	0.0	0.0	0.0	1.09	0.46	1.51	130848.81
AP65	5	3	3	0.75	2	0.0	0.0	0.0	1.47	0.57	4.16	130127.46
AP65	5	4	3	0.75	2	0.0	0.0	0.0	1.88	0.73	3.03	130094.52
AP65	6	2	3	0.75	2	0.0	0.0	1.7	1.75	0.92	1.81	123292.08
AP65	6	3	3	0.75	2	0.0	1.5	0.0	2.14	1.02	4.09	122959.75
AP65	6	4	3	0.75	2	0.0	1.4	0.1	3.09	1.31	4.40	122871.81
AP65	6	5	3	0.75	2	0.0	1.4	0.1	3.79	1.64	6.62	122871.81
AP65	7	2	3	0.75	2	0.1	0.1	0.0	2.58	1.18	2.76	116951.08
AP65	7	3	3	0.75	2	0.2	0.2	0.0	3.17	1.55	6.52	116665.46
AP65	7	4	3	0.75	2	0.0	0.0	0.8	4.21	1.96	8.41	116601.77
AP65	7	5	3	0.75	2	0.0	0.0	0.9	5.53	2.45	9.19	116590.58
AP65	7	6	3	0.75	2	0.0	0.0	0.9	6.48	2.99	14.49	116590.58
AP65	8	2	3	0.75	2	0.0	0.0	0.0	2.83	1.51	3.85	111622.72
AP65	8	4	3	0.75	2	0.0	1.0	0.0	4.43	2.41	12.57	111239.47
AP65	8	6	3	0.75	2	0.0	0.0	0.1	7.59	4.08	13.18	111231.25
AP65	8	7	3	0.75	2	0.0	1.0	0.1	9.41	4.30	22.31	111231.25

Medium-sized hard instances												
						DEV (%)			**CPU**			
Inst	p	r	χ	α	δ	**SS**^{A}	**SS**^{B}	**GRASP**	**SS**^{A}	**SS**^{B}	**GRASP**	**Best known**
AP70	3	2	3	0.75	2	0.0	0.0	2.0	0.62	0.26	0.56	158038.30
AP70	4	2	3	0.75	2	0.1	0.1	0.0	0.99	0.43	1.95	142720.05
AP70	4	3	3	0.75	2	0.0	0.0	1.0	1.22	0.53	1.70	142626.15
AP70	5	2	3	0.75	2	0.2	0.2	0.0	1.58	0.65	2.51	132562.81
AP70	5	3	3	0.75	2	0.0	0.0	0.0	1.96	0.76	4.56	132100.10
AP70	5	4	3	0.75	2	0.0	0.0	0.0	2.58	0.97	6.15	132055.96
AP70	5	3	3	0.75	2	0.0	0.0	0.0	3.40	1.59	6.29	123645.87
AP70	5	4	3	0.75	2	0.0	0.0	0.0	3.93	2.03	5.49	123601.74
AP70	5	5	3	0.75	2	0.0	0.0	0.0	4.82	2.54	7.18	123601.74
AP70	7	2	3	0.75	2	0.2	0.2	0.0	2.42	1.35	6.07	117996.74
AP70	7	3	3	0.75	2	0.0	0.0	0.0	3.32	1.75	11.29	117525.65
AP70	7	4	3	0.75	2	0.0	0.0	0.0	4.54	2.06	11.29	117485.79
AP70	7	5	3	0.75	2	0.0	0.0	0.0	5.65	2.71	12.06	117485.26
AP70	7	6	3	0.75	2	0.0	0.0	0.0	6.36	3.25	15.83	117485.26
AP70	7	3	3	0.75	2	0.0	0.7	0.0	4.19	1.94	8.58	112134.88
AP70	7	4	3	0.75	2	0.0	0.6	0.6	5.78	2.63	16.02	112098.09
AP70	7	5	3	0.75	2	0.0	0.6	0.0	7.23	3.11	16.82	112082.12
AP70	7	6	3	0.75	2	0.0	0.6	0.0	8.96	4.23	24.68	112082.12
AP70	7	7	3	0.75	2	0.0	0.6	1.1	10.24	5.04	49.23	112082.12
AP75	3	2	3	0.75	2	0.1	0.1	0.0	0.76	0.31	0.58	158171.28
AP75	4	2	3	0.75	2	0.0	0.0	0.0	1.45	0.58	2.14	142854.97
AP75	4	3	3	0.75	2	0.1	1.1	0.0	1.75	0.66	2.14	142668.41
AP75	5	2	3	0.75	2	0.0	0.0	0.0	2.00	0.79	3.37	132822.87
AP75	5	3	3	0.75	2	0.0	0.0	0.0	2.39	0.92	4.66	132387.75
AP75	5	4	3	0.75	2	0.0	0.0	0.0	3.10	1.20	7.07	132365.64
AP75	6	2	3	0.75	2	0.1	0.1	0.0	2.43	1.30	3.95	125657.15
AP75	6	3	3	0.75	2	0.0	0.0	0.0	3.32	1.59	7.01	125224.59
AP75	6	4	3	0.75	2	0.0	0.0	0.5	4.66	2.04	7.63	125184.65
AP75	6	5	3	0.75	2	0.0	0.0	0.1	5.82	2.55	14.15	125184.65
AP75	7	2	3	0.75	2	0.0	0.8	0.0	3.47	1.67	7.39	119237.88
AP75	7	3	3	0.75	2	0.0	0.0	0.0	5.08	2.28	10.08	118808.16
AP75	7	4	3	0.75	2	0.0	0.0	0.0	6.25	2.91	15.12	118786.38
AP75	7	5	3	0.75	2	0.0	0.0	0.0	7.95	3.65	13.32	118786.38
AP75	7	6	3	0.75	2	0.0	0.0	0.0	9.32	4.47	15.73	118786.38
AP75	8	2	3	0.75	2	0.0	0.0	0.2	3.68	2.19	4.18	114690.98
AP75	8	4	3	0.75	2	0.0	0.6	0.0	6.60	3.28	22.15	113400.50
AP75	8	7	3	0.75	2	0.0	0.0	0.0	11.16	5.56	24.54	114086.67

Medium-sized hard instances												
						DEV (%)			**CPU**			
Inst	p	r	χ	α	δ	**SS**^{A}	**SS**^{B}	**GRASP**	**SS**^{A}	**SS**^{B}	**GRASP**	**Best known**
AP80	3	2	3	0.75	2	0.1	0.1	0.0	0.92	0.37	1.44	158202.73
AP80	3	3	3	0.75	2	0.0	0.0	0.0	1.95	0.70	2.12	143102.38
AP80	5	2	3	0.75	2	0.0	0.0	0.0	2.08	0.89	3.17	132915.50
AP80	5	3	3	0.75	2	0.0	0.0	0.0	2.82	1.14	4.85	132446.27
AP80	5	4	3	0.75	2	0.0	0.0	0.0	3.63	1.44	6.71	132424.08
AP80	6	2	3	0.75	2	0.1	0.1	0.0	3.25	1.60	5.83	125743.83
AP80	6	3	3	0.75	2	0.0	0.0	0.0	4.27	2.03	8.36	125284.10
AP80	6	4	3	0.75	2	0.0	0.0	0.1	5.66	2.64	11.12	125258.38
AP80	6	5	3	0.75	2	0.0	0.0	0.1	7.18	3.31	9.75	125258.38
AP80	7	2	3	0.75	2	0.0	0.6	0.6	4.02	2.06	4.35	119597.86
AP80	7	3	3	0.75	2	0.0	0.0	0.0	5.87	2.61	12.48	119132.73
AP80	7	4	3	0.75	2	0.0	0.0	0.8	7.61	3.52	17.05	119112.52
AP80	7	5	3	0.75	2	0.0	0.0	0.1	9.90	4.55	25.55	119105.55
AP80	7	6	3	0.75	2	0.0	0.0	0.1	11.76	5.43	21.25	119105.55
AP80	7	3	3	0.75	2	0.0	1.9	0.6	6.27	3.35	19.47	113787.42
AP80	7	5	3	0.75	2	0.0	1.2	0.1	10.58	5.48	26.02	114404.95
AP80	7	6	3	0.75	2	0.0	1.2	0.4	13.59	6.03	32.28	114404.95
AP80	7	7	3	0.75	2	0.0	1.2	0.0	15.65	7.34	55.93	114404.95
AP85	3	2	3	0.75	2	0.0	0.0	0.0	1.35	0.45	1.62	158274.33
AP85	4	2	3	0.75	2	0.0	0.1	0.0	1.76	0.73	3.11	142919.25
AP85	4	3	3	0.75	2	0.0	0.0	0.0	2.36	0.95	3.96	142822.33
AP85	5	2	3	0.75	2	0.2	0.2	0.0	2.77	1.34	6.08	133552.92
AP85	5	3	3	0.75	2	0.0	0.0	0.0	3.71	1.67	6.13	133110.33
AP85	5	4	3	0.75	2	0.0	0.0	0.0	4.63	2.16	10.08	133081.65
AP85	6	2	3	0.75	2	0.0	0.0	0.0	4.86	1.72	5.82	126462.13
AP85	6	3	3	0.75	2	0.0	0.0	0.4	5.18	2.33	7.05	125925.59
AP85	6	4	3	0.75	2	0.0	0.0	0.1	6.20	2.22	18.55	125849.01
AP85	6	5	3	0.75	2	0.0	0.0	0.1	8.59	3.74	18.91	125849.01
AP85	7	2	3	0.75	2	0.0	0.3	0.0	4.55	2.32	6.68	120735.00
AP85	7	3	3	0.75	2	0.1	0.1	0.0	6.02	2.80	15.98	119872.79
AP85	7	4	3	0.75	2	0.0	0.0	0.6	8.04	3.56	15.41	119852.39
AP85	7	5	3	0.75	2	0.0	0.0	0.0	10.17	4.47	31.81	119837.13
AP85	7	6	3	0.75	2	0.0	0.0	0.1	12.01	5.40	38.39	119837.13
AP85	8	2	3	0.75	2	0.2	0.7	0.0	5.77	3.02	12.15	114508.00
AP85	8	5	3	0.75	2	0.0	0.0	0.1	14.49	6.68	26.24	114966.55
AP85	8	6	3	0.75	2	0.0	0.0	0.1	17.65	8.30	53.54	114966.55

Medium-sized hard instances												
						DEV (%)			**CPU**			
Inst	p	r	χ	α	δ	**SS**^{A}	**SS**^{B}	**GRASP**	**SS**^{A}	**SS**^{B}	**GRASP**	**Best known**
AP90	3	2	3	0.75	2	0.0	0.0	0.0	1.28	0.44	1.94	157612.16
AP90	4	2	3	0.75	2	0.1	0.1	0.0	1.83	0.76	2.92	142465.46
AP90	4	3	3	0.75	2	0.0	0.0	0.0	2.65	1.07	7.77	142342.85
AP90	5	2	3	0.75	2	0.1	0.1	0.0	3.79	1.47	5.43	132771.23
AP90	5	3	3	0.75	2	0.0	0.0	0.0	4.35	1.79	8.05	132486.47
AP90	5	4	3	0.75	2	0.0	0.0	0.0	5.62	2.28	11.28	132422.07
AP90	6	2	3	0.75	2	0.1	0.1	0.0	3.86	1.74	9.31	125465.09
AP90	6	3	3	0.75	2	0.0	0.0	0.1	5.47	2.25	11.34	125176.85
AP90	6	4	3	0.75	2	0.0	0.0	0.1	6.51	2.65	18.00	125055.28
AP90	6	5	3	0.75	2	0.0	0.0	0.1	7.90	3.27	17.64	125055.28
AP90	7	2	3	0.75	2	0.1	0.1	0.0	5.72	2.72	10.30	119666.91
AP90	7	3	3	0.75	2	0.0	0.0	0.4	7.55	3.51	16.32	119379.94
AP90	7	4	3	0.75	2	0.0	0.0	0.2	10.04	4.48	41.87	119198.08
AP90	7	5	3	0.75	2	0.0	0.0	0.1	12.39	5.28	36.33	119190.81
AP90	7	6	3	0.75	2	0.0	0.0	0.1	14.94	6.84	58.77	119190.81
AP90	8	2	3	0.75	2	0.0	0.0	0.0	6.07	3.44	10.55	114099.35
AP90	8	3	3	0.75	2	0.0	0.0	1.1	8.03	4.17	27.21	113872.57
AP90	8	4	3	0.75	2	0.0	0.0	0.2	11.06	5.07	19.20	113776.78
AP90	8	5	3	0.75	2	0.0	0.0	0.4	13.60	6.36	38.81	113732.39
AP90	8	6	3	0.75	2	0.0	0.0	0.1	16.91	7.47	47.03	113732.39
AP90	8	7	3	0.75	2	0.0	0.8	0.1	18.50	9.25	81.84	113732.39
AP95	3	2	3	0.75	2	0.0	0.0	0.0	1.67	0.53	1.50	157684.46
AP95	4	2	3	0.75	2	0.1	0.1	0.0	2.17	0.86	3.60	142538.31
AP95	4	3	3	0.75	2	0.1	0.1	0.0	2.55	1.06	5.49	142457.05
AP95	5	2	3	0.75	2	0.1	0.1	0.0	3.51	1.38	6.49	133014.42
AP95	5	3	3	0.75	2	0.0	0.0	0.0	4.79	1.76	14.01	132763.09
AP95	5	4	3	0.75	2	0.0	0.0	0.0	6.19	2.27	12.57	132686.94
AP95	6	2	3	0.75	2	0.0	0.0	0.0	5.28	1.99	6.35	125700.19
AP95	6	3	3	0.75	2	0.0	0.0	0.1	6.59	2.58	11.73	125435.39
AP95	6	4	3	0.75	2	0.0	0.0	0.1	8.63	3.22	24.37	125311.86
AP95	6	5	3	0.75	2	0.0	0.0	0.1	10.54	3.98	17.80	125311.86
AP95	7	2	3	0.75	2	0.0	0.1	0.0	6.38	3.23	11.90	119897.57
AP95	7	3	3	0.75	2	0.0	0.0	0.1	8.06	3.81	20.17	119628.97
AP95	7	4	3	0.75	2	0.0	0.2	0.0	10.96	4.70	30.11	119422.38
AP95	7	5	3	0.75	2	0.0	0.1	0.1	13.88	5.88	37.85	119422.38
AP95	7	6	3	0.75	2	0.0	0.1	0.1	16.18	7.11	36.05	119422.38
AP95	8	2	3	0.75	2	0.0	0.0	0.4	8.32	4.22	15.76	114236.69
AP95	8	3	3	0.75	2	0.0	0.0	0.0	10.01	4.52	25.82	114351.51
AP95	8	4	3	0.75	2	0.0	0.0	0.4	14.33	5.87	37.18	113900.72
AP95	8	5	3	0.75	2	0.0	0.0	0.0	18.42	7.34	53.85	113890.35
AP95	8	6	3	0.75	2	0.0	0.0	1.0	21.56	8.83	85.85	113868.30
AP95	8	7	3	0.75	2	0.0	0.0	0.0	25.07	10.85	51.21	113868.30

Large-sized hard instances												
						DEV (%)			**CPU**			
Inst	p	r	χ	α	δ	**SS**^{A}	**SS**^{B}	**GRASP**	**SS**^{A}	**SS**^{B}	**GRASP**	**Best known**
AP100	3	2	3	0.75	2	0.0	0.0	0.0	1.99	0.60	0.45	158043.08
AP100	4	2	3	0.75	2	0.0	0.0	0.0	2.78	1.01	1.12	143208.40
AP100	4	3	3	0.75	2	0.0	0.0	0.0	4.00	1.41	1.21	143086.43
AP100	5	2	3	0.75	2	0.1	0.1	0.0	4.24	1.74	2.11	133815.35
AP100	5	3	3	0.75	2	0.0	0.0	0.0	5.73	2.21	1.53	133569.22
AP100	5	4	3	0.75	2	0.0	0.0	0.0	6.91	2.83	1.57	133483.00
AP100	6	2	3	0.75	2	0.1	0.1	0.0	6.30	2.45	3.36	126523.14
AP100	6	3	3	0.75	2	0.0	0.0	0.0	8.60	3.18	2.46	126228.60
AP100	6	4	3	0.75	2	0.0	0.0	0.1	11.44	4.11	5.26	126107.94
AP100	6	5	3	0.75	2	0.0	0.0	0.1	14.13	5.17	3.59	126107.94
AP100	7	2	3	0.75	2	0.1	0.1	0.0	7.60	3.40	4.14	120697.19
AP100	7	3	3	0.75	2	0.0	0.0	0.0	10.37	4.40	6.02	120471.47
AP100	7	4	3	0.75	2	0.0	0.2	0.2	13.67	5.55	6.62	120187.66
AP100	7	5	3	0.75	2	0.0	0.0	0.2	18.03	7.21	5.34	120164.59
AP100	7	6	3	0.75	2	0.0	0.1	0.1	19.49	9.52	8.97	120234.75
AP100	8	2	3	0.75	2	0.0	0.1	0.0	9.73	4.63	3.18	114709.50
AP100	8	3	3	0.75	2	0.1	0.1	0.0	11.67	5.23	6.27	114439.93
AP100	8	4	3	0.75	2	0.0	0.0	0.1	15.68	6.68	9.50	114315.28
AP100	8	5	3	0.75	2	0.0	0.0	1.2	20.19	8.42	7.40	114298.12
AP100	8	6	3	0.75	2	0.0	0.0	1.1	24.68	10.28	7.98	114296.13
AP100	8	7	3	0.75	2	0.0	0.0	0.0	27.87	12.15	12.93	114296.13
AP150	3	2	3	0.75	2	0.0	0.0	0.0	4.95	1.79	1.68	158742.05
AP150	4	2	3	0.75	2	0.1	0.1	0.0	8.66	3.33	3.57	143811.40
AP150	4	3	3	0.75	2	0.0	0.0	0.0	10.70	3.82	2.94	143696.46
AP150	5	2	3	0.75	2	0.0	0.0	1.6	13.06	5.02	6.93	134590.93
AP150	5	3	3	0.75	2	0.0	0.1	1.8	17.36	5.93	9.85	134053.76
AP150	5	4	3	0.75	2	0.0	0.0	0.0	20.77	6.89	9.59	134022.43
AP150	6	2	3	0.75	2	0.1	0.1	0.0	19.62	6.95	8.39	127219.49
AP150	6	3	3	0.75	2	0.0	0.0	0.0	25.01	8.73	14.03	126935.58
AP150	6	4	3	0.75	2	0.0	0.0	0.0	29.01	10.38	11.20	126871.13
AP150	6	5	3	0.75	2	0.0	0.0	0.0	39.97	13.43	16.45	126871.13
AP150	7	2	3	0.75	2	0.1	0.1	0.0	27.73	9.20	18.00	121297.48
AP150	7	3	3	0.75	2	0.0	0.0	0.2	33.58	11.11	26.64	121101.93
AP150	7	4	3	0.75	2	0.0	0.0	0.0	48.84	15.48	19.35	120922.63
AP150	7	5	3	0.75	2	0.0	0.0	0.3	58.01	17.32	21.78	120965.44
AP150	7	6	3	0.75	2	0.0	0.0	0.0	68.40	20.69	27.71	120965.44
AP150	8	2	3	0.75	2	0.0	0.0	1.0	33.95	13.09	20.70	115486.83
AP150	8	3	3	0.75	2	0.0	0.0	0.6	45.65	16.29	33.58	115609.81
AP150	8	5	3	0.75	2	0.0	0.0	0.0	83.73	28.37	43.13	115108.06
AP150	8	6	3	0.75	2	0.0	0.0	0.0	102.16	34.79	61.19	115105.52
AP150	8	7	3	0.75	2	0.0	0.0	0.7	118.18	41.23	55.18	115105.52
AP200	3	2	3	0.75	2	0.0	0.0	0.0	10.93	3.96	5.67	159987.41
AP200	4	2	3	0.75	2	0.0	0.0	0.0	24.46	7.55	11.99	144755.16
AP200	4	3	3	0.75	2	0.0	0.0	0.0	33.80	9.03	12.00	144611.12
AP200	5	2	3	0.75	2	0.0	0.0	0.0	36.78	10.49	21.50	137408.43
AP200	5	3	3	0.75	2	0.0	0.4	0.3	43.53	11.50	22.56	136914.54
AP200	5	4	3	0.75	2	0.0	0.2	0.2	64.65	16.60	33.38	136777.91
AP200	6	2	3	0.75	2	0.0	0.0	0.0	38.48	14.40	37.03	130235.76
AP200	6	3	3	0.75	2	0.0	1.2	0.0	50.76	16.11	44.05	129883.62
AP200	6	4	3	0.75	2	0.0	0.0	0.0	76.71	23.27	54.33	129817.47
AP200	6	5	3	0.75	2	0.0	0.0	1.2	93.25	28.60	30.80	129817.47
AP200	7	2	3	0.75	2	0.0	0.2	0.6	57.63	18.31	49.65	123989.21
AP200	7	3	3	0.75	2	0.1	0.2	0.0	78.94	23.21	56.82	123670.80
AP200	7	4	3	0.75	2	0.0	0.2	0.2	110.69	30.88	73.03	123661.35
AP200	7	5	3	0.75	2	0.0	0.2	0.2	139.14	38.47	89.25	123658.33
AP200	7	6	3	0.75	2	0.0	0.2	0.0	163.59	45.90	63.66	123658.33
AP200	8	2	3	0.75	2	0.0	0.1	0.5	89.36	33.12	61.56	118125.17
AP200	8	3	3	0.75	2	0.0	0.2	0.5	106.03	41.38	79.85	117828.62
AP200	8	4	3	0.75	2	0.0	0.0	0.0	104.03	36.42	136.11	117719.51
AP200	8	5	3	0.75	2	0.0	0.0	0.4	142.68	46.31	121.85	117709.98
AP200	8	6	3	0.75	2	0.0	0.0	0.6	162.28	56.19	118.79	117709.98
AP200	8	7	3	0.75	2	0.0	0.0	0.4	180.24	65.34	112.22	117709.98