GRASP for the Uncapacitated r-Allocation p-hub Median Problem

Juanjo Peiró, Ángel Corberán, Rafael Martí
Departament d'Estadística i Investigació Operativa
Universitat de València, Spain

Abstract.

In this paper we propose several heuristics for the Uncapacitated r-Allocation p-Hub Median Problem. In the classical p-hub problem, given a set of nodes with pairwise traffic demands, we must select p of them as hub locations and route all traffics through them at a minimum cost. We target here an extension, called the r-allocation p-hub median problem, recently proposed by Yaman (2011), in which every node is assigned to r of the p selected hubs ($r \leq p$) and we are restricted to route the traffic of the nodes through their associated r hubs.

As it is usual in this type of problems, our method has three phases: location, assignment and routing. Specifically, we propose a heuristic based on the GRASP methodology in which we consider three local search procedures. The combinatorial nature of this problem makes them time-consuming. We therefore propose a filtering mechanism to discard low-quality constructions and skip its improvement, saving its associated running time. We perform several experiments to first determine the values of the key-search parameters of our methods and then to compare with previous algorithms. Computational results on 465 instances show that while only small instances can be optimally solved with exact methods, the heuristics are able to find high-quality solutions on larger instances in short computing times. Moreover, when targeting the classical p-hub versions (with $r = 1$ or $r = p$), our heuristic is competitive with the state of the art methods.

Key-Words: p-hub, heuristics, GRASP, combinatorial optimization.

1. INTRODUCTION

The p-hub problem is a classical optimization problem (O'Kelly, 1987) in which, given a set of nodes with pairwise traffic demands, we have to choose p of them as hub locations and route all the traffic through these hubs at a minimum cost. For each pair of nodes i and j, there is a traffic t_{ij} that needs to be transported. It is generally assumed that direct transportation between non-hub nodes is not possible, and the t_{ij} traffic travels on a path $i \to h_i \to h_j \to j$, where h_i and h_j are hubs assigned to i and j, respectively.

There are two extensively studied versions of the p-hub problem regarding the allocation strategy: the single allocation and the multiple allocation versions. In the *single allocation* p-hub problem, each node is assigned to one of the p hubs , only allowing to send and receive traffic through this single hub. In the *multiple allocation* p-hub problem, each node can send and receive traffic through any of the p hubs.

Transporting the t_{ij} units flow through the path $i \to h_i \to h_j \to j$ has an associated cost $c_{ij}(h_i, h_j)$, usually computed as $c_{ij} = t_{ij}\left(\chi d_{ih_i} + \alpha d_{h_i h_j} + \delta d_{h_j j}\right)$, where d_{ih_i} is the distance between i and h_i (similarly for the $d_{h_i h_j}$ and $d_{h_j j}$), and χ, α and δ are unit rates for collection (origin-hub), transfer (hub-hub) and distribution (hub-destination), respectively. Generally, α is used as a discount factor to provide reduced unit costs on arcs between hubs, so typically $\alpha < \chi$ and $\alpha < \delta$. Therefore, a solution is determined by a set of hubs, the node-to-hubs assignments, and the travel paths for each pair of nodes. The sum of the $c_{ij}(h_i, h_j)$ values for all (i, j) pairs is the solution cost or value. The problem then consists of finding the hubs, assignments and paths that minimize the total cost of transportation.

During the last years p-hub problems have been widely studied due to the increase in the number of applications in telecommunications, transportation and logistics. The p-hub network, based on transshipment nodes, provides a better utilization of transporters. Different versions of the problem include, single and multiple hub allocations (as mentioned above), capacity constraints, fixed costs, and maximum travel time, to mention the most common ones. The p-hub median problem belongs to the class of $\mathcal{NP}$-hard problems. Even when the set of hubs is given, the sub-problem of optimal allocation of non-hub nodes to hubs is also $\mathcal{NP}$-hard (Love, Morris and Wesolowski, 1988). We refer the reader to Kratica et al. (2007), Milanović (2010), Ilić et al. (2010), Gelareh and Nickel (2011), García, Landete and Marín (2012), and Campbell and O'Kelly (2012) for some of the most interesting papers on the subject.

A new evolutionary approach was presented by Milanović (2010) for solving the multiple allocation version of the problem. Integer encoding of individuals was used to ensure their feasibility, whose quality was evaluated using a fitness function. By applying genetic operators of selection, crossover, and mutation, future generations were produced. Duplicated individuals were removed from the population in the next generation, being also limited to a certain percentage the individuals with the same objective value but different genetic code. Fine grained tournament selection (*FGTS*) was used, as well as standard one-point crossover operator and the idea of *frozen bits* to increase the diversity of the genetic material. The

results demonstrate the usefulness of the proposed approach with new best solutions for three standard instances.

A VNS approach was presented by Ilić et al. (2010) for the single version of the problem. Three neighborhoods were proposed for the VNS scheme, using the idea of hubs as clusters: *allocate* tries to change the allocations of every non-hub node, leaving all other elements unchanged; *alternate* preserves all clusters, changing the location of a hub from one node to other from the same cluster, assigning the remaining nodes of the cluster to this new hub; *new locate* tries to increase the diversity of solutions obtained selecting nodes from out of a cluster to be hub and then assigning the nodes at the cluster to other hubs. The authors also presented how to efficiently update data structures for calculating new total flow and cost in the network. Both sequential and nested strategies of the VNS were proposed, outperforming the best-known heuristic in terms of effort and quality solutions for the single version.

Recently, Yaman (2011) proposed a very interesting variant of this problem in which each node can be connected to at most r of the p hubs, called the Uncapacitated r-Allocation p-Hub Median Problem (UrApHMP). The motivation of this variant comes from the fact that the single allocation version, in which a node is connected (assigned) to a single hub is too restricted for real-world situations, while the multiple allocation variant, where each node can use any of the p hubs to route its traffic, results in high fixed costs and complicated networks. The r-allocation p-hub problem, being $r \leq p$, generalizes both versions of the p-hub median problem. When $r = 1$ we are at the single allocation version, whereas if $r = p$, we have the multiple allocation version. Yaman (2011) proposed a mixed integer programming formulation for this generalized version and performed a computational study to first compare the r-allocation version with the single and multiple variants, and then to optimally solve small and medium size instances. She observed in instances with 50 and 75 nodes (and 3, 4, and 5 hubs), that when a node is allowed to be allocated to two hubs, the solutions are significantly cheaper than the single allocation solutions (about 2.0% on average) and slightly more expensive than the multiple version (about 0.3% on average).

In this paper we propose a heuristic for the Uncapacitated r-Allocation p-Hub Median Problem. It is based on the GRASP methodology, and implements three local search procedures. Additionally, we propose a filtering mechanism to discard low-quality solutions and selectively apply the local searches to "promising" solutions to obtain high quality solutions in short computing times. As far as we know, these are the first heuristics proposed for this more general p-hub median model. Computational results on 465 instances show that while only small instances can be optimally solved with exact methods, the heuristics are able to find high-quality solutions on larger instances in short computing times. Moreover, when targeting the classical p-hub versions (with $r = 1$ or $r = p$), our heuristic is competitive with the state of the art methods.

2. MATHEMATICAL PROGRAMMING FORMULATION

The single allocation version of the p-hub median problem was formulated for the first time by O'Kelly (1987) as a quadratic integer program. This formulation resulted in a very difficult problem to be solved. Campbell (1994) formulated the p-hub median problem as an integer program, but this formulation contained many variables and constraints ($O(n^4)$). In the following paragraphs we briefly describe the formulation that we use in our computational study to evaluate the performance of the proposed heuristics.

Given a network with a set of nodes N and a set of arcs A, let t_{ij} be the amount of traffic to be routed from node i to node j, i.e., through the arc (i,j), and let d_{ij} be its associated unit routing cost. The r-allocation p-hub problem is then formulated (Yaman, 2011) in terms of the following variables: Given a node k, $z_{kk} = 1$ if the node is a hub (i.e., if a hub is set or located at this node), and $z_{kk} = 0$ otherwise. Given a non-hub node i and a hub k, $z_{ik} = 1$ if node i is assigned or allocated to node k, and 0 otherwise. Finally, f_{ijkl} is the proportion of the traffic t_{ij} from node i to node j that travels along the path $i \to k \to l \to j$, where k and l are hubs. With these variables, the problem can be formulated as follows:

$$min \sum_{i\in N}\sum_{j\in N}\sum_{k\in N}\sum_{l\in N} t_{ij}(\chi d_{ik} + \alpha d_{kl} + \delta d_{lj}) f_{ijkl} \quad (1)$$

$Subject\ to$:

$$\sum_{k\in N} z_{ik} \le r \quad \forall i \in N \quad (2)$$

$$z_{ik} \le z_{kk} \quad \forall i,k \in N \quad (3)$$

$$\sum_{k\in N} z_{kk} = p \quad (4)$$

$$\sum_{k\in N}\sum_{l\in N} f_{ijkl} = 1 \quad \forall i,j \in N \quad (5)$$

$$\sum_{l\in N} f_{ijkl} \le z_{ik} \quad \forall i,j,k \in N \quad (6)$$

$$\sum_{k\in N} f_{ijkl} \le z_{jl} \quad \forall i,j,l \in N \quad (7)$$

$$f_{ijkl} \ge 0 \quad \forall i,j,k,l \in N \quad (8)$$

$$z_{ik} \in \{0,\ 1\} \quad \forall i,k \in N \quad (9)$$

Constraints (2) ensure that each node is allocated to at most r hubs, where hubs are assigned according to (3). In addition, constraint (4) limits to p the number of hubs. Finally, constraints

(5) to (7) are associated with the routing of the traffic between each pair of nodes i, j through their corresponding hubs k, l.

In our computational experiments, we have tested this formulation and studied the effectiveness of our heuristics in terms of their ability to find the optimal solution on small size instances.

3. GRASP

The GRASP methodology was developed in the late 1980s by Feo and Resende (1989) and the acronym was coined in Feo and Resende (1995). Basically, each GRASP iteration consists in constructing a trial solution with some greedy randomized procedure and then applying local search to the constructed solution. The construction phase is iterative, randomized, greedy, and adaptive. This two-phase process is repeated until some stopping condition is satisfied. A best local optimum found over all local searches is returned as the solution of the heuristic. We refer the reader to Festa and Resende (2011) for a recent survey of this metaheuristic.

The algorithm in Figure 1 shows the pseudo-code for a generic GRASP for minimization. The greedy randomized construction seeks to produce a diverse set of good-quality solutions from which to apply the local search phase. Let x be the partial solution under construction in a given iteration and let C be the candidate set with all the remaining elements that can be added to x. The GRASP construction uses a greedy function $g(c)$ to measure the contribution of each candidate element $c \in C$ to the partial solution x. A restricted candidate list RCL is the subset of candidate elements from C with good evaluations according to g. In particular, if g_{min} and g_{max} are the minimum and maximum evaluations of g in C respectively, then

$$RCL = \{c \in C \mid g(c) \leq g_{min} + \alpha(g_{max} - g_{min})\},$$

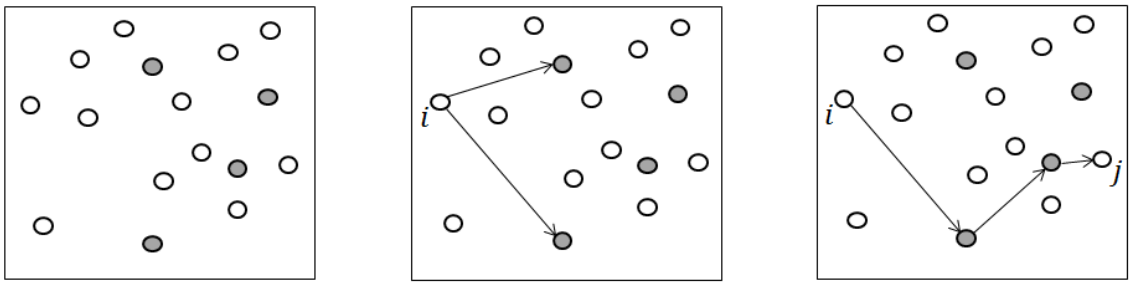
where α is a number in $[0, 1]$.

```
begin GRASP
1   f* ← ∞;
2   while stopping criterion not satisfied do
3       x ← ∅;
4       Compute C with the candidate elements that can be added to x;
5       while C ≠ ∅ do
6           For all c ∈ C compute g(c).  g_min = min_{c∈C} g(c)  and g_max = max_{c∈C} g(c);
7           Define RCL ← {c ∈ C | g(c) ≤ g_min + α(g_max − g_min)} with α ∈ [0, 1] ;
8           Select c* at random from RCL(C);
9           Add c* to partial solution: x ← x ∪ {c*};
10          Update C with the candidate elements that can be added to x;
11      end-while
12      if x is infeasible then
13          Apply a repair procedure to make x feasible;
14      end
15      x ← LocalSearch(x);
16      if f(x) < f(x*) then
17          x* ← x; f* ← f(x);
18      end
19  end
20  return x*;
```

Figure 1. GRASP algorithm for minimization of $f(x)$.

3.1 Construction

To obtain a solution for the UrApHMP we first select the p hubs (step 1) and then determine the assignments of each node to r of the p hubs (step 2). Finally (step 3), for each pair of nodes we identify the routing of their traffic through the appropriated hubs. Figure 2 illustrates these steps.



Step 1. Locating p hubs | Step 2. Assigning r hubs | Step 3. Routing traffics

Figure 2. Construction steps

Let h be a candidate location for a hub. For any node j that could be assigned to h in step 2 we need to consider that all the traffic from j to any other node i, could be routed through h, and in the objective function we would have this traffic t_{ji} multiplied by d_{jh}. We therefore consider the evaluation of this assignment, $e(j,h)$, as:

$$e(j,h) = d_{jh} \sum_{i=1}^{n} t_{ji}$$

Note that, since we want to evaluate the attractiveness of h to be a hub, we compute $e(j,h)$ for every node j in the graph. On the other hand, it is reasonable to assume that if h is a hub, only a fraction of the nodes will be assigned to it. This is the reason to consider, for the evaluation $g(h)$ of h, only the k nodes with lowest $e(j,h)$ value. Let us assume that they are $j_1, \dots, j_k$. In mathematical terms, $g(h) = \sum_{s=1}^{k} e(j_s, h)$. We now apply the standard method in GRASP to construct a restricted candidate list RCL with good hub locations according to this greedy evaluation g, computing g_{min} and g_{max} as described above. Note that, as it is customary in GRASP, g is an adaptive function. Once a hub h_1 is selected, in the following construction steps, when computing $e(j,h)$ for a new candidate h, we do not sum the term $t_{h_1 h}$ since hubs do not need to be assigned to other hubs to route their traffic. We finish step 1 when the p hubs have been selected. Let $H = \{h_1, h_2, \dots, h_p\}$ be the set of these hubs.

In the step 2 of our constructive method, r of the p hubs are allocated to each node in the graph. Specifically, for each node i we evaluate its allocation value $alloc(i,h)$ to any hub h previously selected. Note that for any node j to which we need to send traffic from i, this traffic has to be sent through some of their assigned hubs. In other words, to transport the t_{ij} units, a path $i \to h_i \to h_j \to j$ will be used. To simplify the combinatorial problem of determining simultaneously h_i and h_j, we compute $alloc(i,h)$ as

$$alloc(i,h) = d_{ih}\sum_{j=1}^{n} t_{ij} + \sum_{j=1}^{n} t_{ij}d_{hj} \quad \forall\, i \in N, \qquad \forall h \in H,$$

where the first term computes the cost associated with the arc from i to h, and the second one estimates the cost associated with the arcs from h to all destinations j. We then compute $alloc(i,h)\ \forall\, h = h_1, h_2, \dots, h_p$, and assign to i the r hubs with the best (minimum) allocation values. Let H_i be the set of r hubs assigned to node i ($H_i \subseteq H$).

Note that in step 1 we select hubs in a greedy randomized fashion and in step 2 we assign them to nodes in a greedy way without any random element. We have empirically found that the randomization in step 1 is enough to obtain a diversified set of solutions in our problem. Adding a randomized component in step 2 would result in lower quality solutions.

Finally, in step 3, we route all the traffics at their minimum cost. For each pair i and j, we have to determine the hubs $h_i \in H_i$ and $h_j \in H_j$ minimizing the routing cost. In mathematical terms, from the expression of the objective function, and given $h_i \in H_i$ and $h_j \in H_j$, we denote

$$c_{ij}(h_i, h_j) = t_{ij}\left(\chi d_{ih_i} + \alpha d_{h_ih_j} + \delta d_{h_jj}\right).$$

The routing cost from i to j, c_{ij}, is then obtained by searching the hubs $h_i \in H_i$ and $h_j \in H_j$ minimizing the expression above, i.e.

$$c_{ij} = \min_{h_i \in H_i,\, h_j \in H_j} c_{ij}(h_i, h_j)$$

Since there is a small number of hubs assigned to each node (r typically takes a value between 2 and 6), an exhaustive exploration in this final step can be performed. Specifically, for each pair (i,j) we consider the r^2 associated pairs of hubs to determine the minimum cost c_{ij}. Note that even when H_i and H_j have a common hub, it cannot be ensured that the best route will be through that hub, and the computation of all the possibilities mentioned above is needed.

3.2 Solution representation

As described in Subsection 3.1, three steps are applied to construct a solution: location, assignment and routing. Therefore, to represent a solution we need to specify these three aspects. In particular, a solution S is given by a set of hubs H, a matrix of assignments A, and two matrices of hubs $\mathcal{H}^1$ and $\mathcal{H}^2$ specifying the traffic routes. In particular, $S = (H, A, \mathcal{H}^1, \mathcal{H}^2)$, where:

$$H = \{h_1, h_2, \dots, h_p\} \subseteq N$$

$$A = [a_{ij}]_{i=1,\dots,n;\ j=1,\dots,r},\ a_{ij} \in H_i$$

$$\mathcal{H}^1 = [\mathcal{h}^1_{ij}]_{i=1,\dots,n;\ j=1,\dots,n},\ \mathcal{h}^1_{ij} \in H_i \quad , \quad \mathcal{H}^2 = [\mathcal{h}^2_{ij}]_{i=1,\dots,n;\ j=1,\dots,n},\ \mathcal{h}^2_{ij} \in H_j\,.$$

The set H specifies the p hubs in the solution. Each row i of matrix A contains the r hubs assigned to node i, H_i. Finally, for each pair of nodes i and j, we need to represent the path $i \to h_i \to h_j \to j$ used to route the traffic. Matrix $\mathcal{H}^1$ provides the first hub in the route and matrix $\mathcal{H}^2$ the second one, i.e. $\mathcal{H}^1(i,j) = h_i$ and $\mathcal{H}^2(i,j) = h_j$.

Note that the best hubs to route the traffic t_{ij} from i to j may be different than those to route the traffic t_{ji} from j to i. As described above, the best hubs to route t_{ij} are those minimizing the cost expression $c_{ij}(h_i, h_j) = t_{ij}\left(\chi d_{ih_i} + \alpha d_{h_i h_j} + \delta d_{h_j j}\right)$, which is not a symmetric expression in terms of i and j when the coefficients χ and δ take different values. Since we need to specify the hubs in the path from i to j and the hubs in the path from j to i, a solution is represented using two separated matrices. Given a pair (i,j), $h_{ia}, h_{ib} \in H_i = \{h_{i1}, \dots, h_{ir}\}$ denote the hubs assigned to i in the two associated paths. Similarly, $h_{jc}, h_{jd} \in H_j = \{h_{j1}, \dots, h_{jr}\}$ are the hubs assigned to j. In particular, $\mathcal{H}^1(i,j) = h_{ia}$, $\mathcal{H}^2(i,j) = h_{jc}$, $\mathcal{H}^1(j,i) = h_{jd}$, and $\mathcal{H}^2(j,i) = h_{ib}$. This is illustrated in Figure 3.

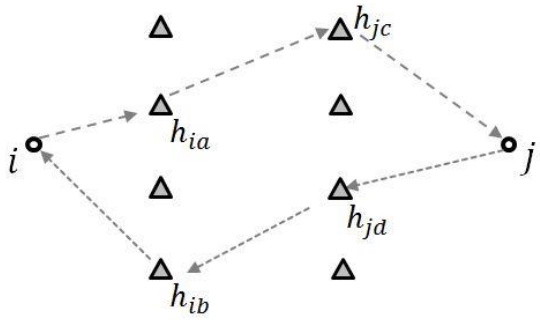


Figure 3. Paths between i and j.

3.3 Improvements

Since the proposed method consists of three steps, and the last one is solved optimally, we can apply two types of improvements to the results obtained at step 1 and step 2: changing the hubs selected, and changing the assignments of hubs to nodes.

Given a solution $S = (H, A, \mathcal{H}^1, \mathcal{H}^2)$, we consider two neighborhoods, N_H and N_A, to improve S. N_H implements a classical exchange in which a hub node h_i is removed from H, and a non-hub node $h_i' \in N \setminus H$ becomes a hub. In other words, we move the hub from node h_i to node h_i', thus obtaining $H' = \{h_1, h_2, \dots, h_i', \dots, h_p\}$. On the other hand, neighborhood N_A does not change the hub selection and it only considers the node assignments. In particular, for a node i with assigned hubs $H_i = \{h_{i1}, \dots, h_{ia}, \dots, h_{ir}\} \subseteq H$, this neighborhood exchanges an assigned hub with a non-assigned one. In mathematical terms, we replace $h_{ia} \in H_i$ with $h_{ia}' \in H \setminus H_i$ obtaining $H_i' = \{h_{i1}, \dots, h_{ia}', \dots, h_{ir}\}$.

Specifically, the first neighborhood (N_H) explores the possibility of exchanging a hub, replacing a node h_i in H by another node $h_i' \in N \setminus H$. The element h_i to be removed from the solution

is chosen by an evaluation cost that determines the most expensive hub in S in terms of its contribution to the objective function. For a pair of nodes (i,j), we route its traffic t_{ij} in S through the path $i \to h_i \to h_j \to j$ with cost $c_{ij}(h_i,h_j) = t_{ij}\left(\chi d_{ih_i} + \alpha d_{h_ih_j} + \delta d_{h_jj}\right)$. We split this cost into two parts:

$$c_{ij}^1(h_i) = t_{ij}\left(\chi d_{ih_i} + \alpha\frac{1}{2}d_{h_ih_j}\right) \text{ and } c_{ij}^2(h_j) = t_{ij}\left(\alpha\frac{1}{2}d_{h_ih_j} + \delta d_{h_jj}\right)$$

We consider that $c_{ij}^1(h_i)$ reflects the cost associated with the use of h_i in the path $i \to h_i \to h_j \to j$. Similarly, $c_{ij}^2(h_j)$ provides an evaluation of the cost of using h_j in this path. In order to measure the total cost associated to a given hub, we sum these values up for all pairs of nodes:

$$c_h^1 = \sum_{i,j} c_{ij}^1(h) \quad , \qquad c_h^2 = \sum_{i,j} c_{ij}^2(h)$$

Hence, we express the evaluation of the cost associated with each hub h as $cost_h = c_h^1 + c_h^2$. Now, h^* is selected as the hub such that $cost_{h^*} = max_{h\in H}\{cost_h\}$. In other words, h^* is, according to this estimation, the most expensive hub in the solution. Therefore, it can be considered as a good candidate to be replaced.

The local search LS_H performs moves in N_H as long as the objective function improves. At each iteration, it selects the hub with the largest contribution to the objective function (according to the estimation above) and searches for a non-hub node to reduce the solution value. We implement here the so-called *first strategy*, in which we perform the first improving move in the neighborhood (instead of scanning the entire neighborhood to determine the best one). Starting from a random element, we examine in increasing order the non-hub nodes searching for the first improving exchange. LS_H terminates when no improving move is found and the current solution is returned as the output of the procedure.

Note that the evaluation of a move in LS_H is quite time consuming. Given a solution $S = (H, A, \mathcal{H}^1, \mathcal{H}^2)$, any change in H affects the rest of components in S. Specifically, when node h_i is replaced by node h_i' in H, it is needed to re-evaluate the hub assignment of all the vertices assigned to h_i, since they cannot use this hub anymore and could use any other hub to route their traffic, not necessarily h_i'. In mathematical terms, for any vertex v such that $h_i \in H_v$ the best hub $h \in H \setminus \{h_i\} \cup \{h_i'\}$ replacing h_i in H_v has to be selected. Therefore, we have to scan all the vertices assigned to h_i to compute the value associated with this exchange and re-compute matrix A. Moreover, to evaluate these assignments we need to compute and evaluate the routes (i.e., to re-compute $\mathcal{H}^1$ and $\mathcal{H}^2$). Note that even those routes not using h_i have to be re-computed since the new hub in H_v could provide a better route than the current one. Since the update of A, $\mathcal{H}^1$ and $\mathcal{H}^2$ requires a significant computational effort, an alternative would be just to replace h_i with h_i' in H_v, remaining all the other elements in S the same. We explore this alternative in the next neighborhood, but we can anticipate that, as expected, it produces lower quality solutions in lower running times (as compared with the entire exploration). The complete evaluation of any trial move has been implemented in LS_H.

The second neighborhood N_A only considers moves on A without changing H. In other words, N_A explores the possibility of exchanging a hub h_{ia} assigned to node i with one of the hub

nodes $h'_{ia} \in H \setminus H_i$ not assigned to i. Based on this neighborhood, we propose two local search procedures, LS_{A1} and LS_{A2}. LS_{A1} implements a simple exploration consisting of replacing h_{ia} with h'_{ia} in all the routes from/to node i. As mentioned above, this could lead to sub-optimal solutions since we are not exploring all the assignments. Consider, for example, the update of the route $i \to h_{ia} \to h_j \to j$ when we replace h_{ia} with h'_{ia} in H_i. With LS_{A1} we would obtain the path $i \to h'_{ia} \to h_j \to j$. Specifically, given a solution $S = (H, A, \mathcal{H}^1, \mathcal{H}^2)$, when LS_{A1} performs a move changing h_{ia} by h'_{ia}, a new solution is obtained in which H remains unchanged, A only changes in one element (h_{ia} with h'_{ia} in row i), $\mathcal{H}^1$ changes h_{ia} with h'_{ia} in row i (in all its appearances), and similarly $\mathcal{H}^2$ in column i. Note that this move can be done very quickly but it does not consider whether any other hub in H_i can provide a better route from i to j. The local search procedure LS_{A1} performs moves in N_A as long as the objective function improves, exploring the assignments in increasing order, and performing the first improving move found. This local search terminates when no improving move is found, and the current solution is returned as the output of the method.

Method LS_{A2} also exchanges the hub assigned to a node i (as LS_{A1}) but also explores the other hubs in $H_i \setminus \{h_{ia}\}$ to determine the best one for each particular route starting and finishing at i. Given a solution $S = (H, A, \mathcal{H}^1, \mathcal{H}^2)$, when LS_{A2} performs a move and changes h_{ia} with h'_{ia}, a new solution is obtained in which H remains unchanged, the matrix A only changes one element (h_{ia} with h'_{ia} in row i), but now $\mathcal{H}^1$ and $\mathcal{H}^2$ are completely recomputed. Since we cannot assure that any hub in a route remains unchanged, routes are computed from scratch in Step 3 of the constructive method. It is clear that this move is computationally more expensive than the one implemented in LS_{A1}. However, as it will be shown in the comparison of both methods in Section 5, the extra running time is justified in those cases in which we want to match the optimal solution, since LS_{A1} is not able to reach it although it obtains very good results. In order to reduce the computational effort of the algorithm, a filtering mechanism to discard low-quality constructions is proposed. It is described in the next section.

4. FILTERING SOLUTIONS

After a number of iterations, it is possible to estimate the fractional improvement achieved by the application of the improvement phase and use this information to increase the efficiency of the search (Laguna and Martí, 1999). Let us define the fractional improvement in the iteration t as:

$$P(t) = \frac{c(S_t) - c(S_t^*)}{c(S_t)}$$

where S_t is the solution constructed at iteration t, $c(S_t)$ is its value, and S_t^* is the improved solution obtained applying an improvement method to S_t (and $c(S_t^*)$ is its value). This improvement method can be any of the three described in Section 3.3, LS_H, LS_{A1} or LS_{A2}, or a combination of them.

After $tmax$ iterations, the mean $\widehat{\mu_P}$ and standard deviation $\widehat{\sigma_P}$ of the improvement P can be estimated as:

$$\widehat{\mu_P} = \frac{\sum_{t=1}^{tmax} P(t)}{n}, \qquad \widehat{\sigma_P} = \sqrt{\frac{\sum_{t=1}^{tmax} (P(t) - \widehat{\mu_P})^2}{tmax - 1}}$$

Then, at a given iteration $q > tmax$, these estimates can be used to determine whether it is "likely" that the improvement phase will be able to improve enough the current construction to produce a better solution than the current best one, S_{best}. If this is not the case, we could discard the constructed solution and skip its improvement, saving its associated running time. In particular, we calculate the minimum fractional improvement $\Delta c(q)$ that is necessary for a construction S_q to be better than S_{best}, as:

$$\Delta c(q) = \frac{c(S_q) - c(S_{best})}{c(S_q)}$$

If $\Delta c(q)$ is close to $\widehat{\mu_P}$, applying the improvement method to the current solution S_q would probably produce a solution S_q^* better than S_{best}. Therefore, in order to save computing time, the improvement method is only applied to the promising solutions S_q, according to this estimation. We can formulate this filtering mechanism as:

if $\Delta c(q) < \widehat{\mu_P} + \lambda \widehat{\sigma_P}$, then apply the improvement method to S_q; otherwise, discard it.

λ is a search parameter representing a threshold on the number of standard deviations away from the estimated mean percentage improvement. Preliminary experiments to test the effect of different λ values have been performed and are reported in Section 5.

5. COMPUTATIONAL EXPERIMENTS

This section describes the computational experiments performed to test the efficiency of the GRASP heuristics. The procedures have been implemented in C and the integer linear programming formulation described in Section 2 have been solved using CPLEX 12.4, the most recent version of CPLEX when the experiments were carried out. The results reported in this section were obtained with an Intel i7 @ 2.7 GHz and 4GB of RAM computer running Windows 7. The metrics that we use to measure the performance of the algorithms are:

- **Value**: Average objective value of the best solutions obtained with the algorithm on the instances considered in the experiment.
- **Dev**: Average percentage deviation from the best-known solution (or from the optimal solution, if available).
- **Best**: Fraction of instances in a set for which a procedure is able to find the best-known solution.
- **CPU**: Average computing time in seconds employed by the algorithm.

5.1 Test Problems

We have tested our algorithms on three sets of instances:

(1) The **CAB** (Civil Aviation Board) data set. It is based on airline passenger flows between some important cities in the United States. It consists of a data file, presented by O'Kelly in 1987, with the distances and flows of a 25 nodes graph. From this original file, 75 instances with 25 nodes and $p = 1,\ldots, 5$, and $r = 1,\ldots, p$ have been generated by several authors. The following parameter values have been widely used: $\chi = 1, \delta = 1$, and $\alpha = 0.2, 0.4, 0.6, 0.8$, and 1.

(2) The **AP** (Australian Post) data set. It is based on real data from the Australian postal service and was presented by Ernst and Krishnamoorthy in 1996. 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 360 instances with $n = 40, 50, 70, 75, 80, 85, 90, 95, 100, 150$ and 200 nodes. For those instances with $40 \leq n \leq 50$, p ranges from 1 to 5. For those with $70 \leq n \leq 95$, p ranges from 1 to 8, and for those with $100 \leq n \leq 200$, p takes values between 1 and 20. In all these cases, $r \in \{1, \ldots, p\}$. According with previous articles, cost parameter values are $\chi = 3$, $\alpha = 0.75$ and $\delta = 2$. Regarding the flows between nodes, these instances do not have symmetric flows (i.e., for a given pair of nodes i and j, t_{ij} is not necessarily equal to t_{ji}). Moreover, flows from one node to itself can be positive (i.e., t_{ii} can be strictly positive for a given i).

(3) The **USA423** data set. This is a new family of instances that we introduce here 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 3 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.

5.2 Parameter calibration

From the set of 465 instances derived from the CAB, AP and USA423 data sets described before, we have used a subset of 45, with different sizes and values of p and r, to calibrate the parameters in our method. The entire set, as well as this training set, are available at http://www.optsicom.es.

In our first experiment we study the constructive method described in Section 3.1 in terms of solution quality and diversification power. Clearly, the performance of this solution generator depends on the value of its two parameters, α, defining the size of the Restricted Candidate List, and k, determining the number of elements in which the evaluation is based on. In order to determine effective values for these parameters, we have created a measure of diversity for

a set of solutions. To perform an effective exploration of the solution space, the constructive method has to be able to generate solutions of a different structure which, in our specific problem, can be interpreted in terms of using different hubs. We therefore compute, in a set of constructed solutions, the number of different hubs used. Specifically, given a set of solutions $P = \{S_1, S_2, \dots, S_q\}$, where $H_1, H_2, \dots, H_q$ are their corresponding sets of hubs, the diversity measure, $div(P)$, is the number of elements in the set obtained as the union of these sets of hubs. In mathematical terms,

$$div(P) = \left| \bigcup_{i=1}^{q} H_i \right|.$$

This diversity metric can be easily interpreted as the number of different hubs in the set of solutions. The larger this value is, the more diversity the algorithm is able to produce. In the first experiment we have generated $q = 100$ solutions with the constructive method, $k = 1.0$, and different values of α. Table 1 shows the metrics described above: Dev, Best and CPU, as well as the average diversity measure, div, on the 45 instances of the training set. This table indicates that the best solutions in terms of quality are obtained with $\alpha = 0.8$, since the algorithm is able to obtain an average percentage deviation of 3.3% and 13 best known solutions, which compares favorably with the other results. Moreover, with $\alpha = 0.8$, we also obtain the best results in terms of diversity since the div statistic exhibits a value of 64, which is larger than or equal to the rest of diversity values in this table. We therefore set $\alpha = 0.8$ in the rest of the experiments.

α	Dev	Best	div	CPU
0.1	11.0%	12	27	0.483
0.2	6.7%	12	41	0.486
0.3	6.7%	3	51	0.485
0.4	5.3%	6	56	0.484
0.5	5.4%	2	58	0.485
0.6	4.7%	3	61	0.489
0.7	4.6%	2	62	0.493
0.8	3.3%	13	64	0.485
0.9	3.3%	11	64	0.487
random	3.5%	10	62	0.468

Table 1. Constructive method with different α values.

Table 2 shows the results of the second experiment in which we run the constructive method with $\alpha = 0.8$ and different values of the k parameter on the training set instances. Results in this table show that $k = 1.0$ obtains the best values in terms of quality and diversity. We therefore set $\alpha = 0.8$ and $k = 1.0$ in the rest of the experiments.

k	Dev	Best	div	CPU
0.8	1.4%	30	63	0.480
0.9	1.6%	28	63	0.462
1.0	1.1%	33	64	0.462
1.1	1.2%	28	63	0.462
1.2	0.9%	31	64	0.462
1.3	1.8%	27	63	0.461
1.4	2.0%	25	63	0.462
1.5	1.7%	26	63	0.461
1.6	1.9%	26	63	0.462
1.7	1.8%	27	63	0.462
1.8	1.6%	29	63	0.462
1.9	1.4%	26	63	0.462
2.0	1.5%	25	63	0.463

Table 2. Constructive method with different k values.

With the goal of supporting our conclusions about the performance of the proposed procedures, we performed the non-parametric Friedman test for multiple correlated samples to the best solutions obtained by the proposed constructive method with each parameter value in Tables 1 and 2. This test computes, for each instance, the rank value of each method according to solution quality. Then, it calculates the average rank value for each method across all instances. If the averages differ greatly, the associated p-value or level of significance is small. The resulting p-values of 0.001 and 0.034 obtained with the individual best values in Tables 1 and 2, respectively, indicate that there are statistically significant differences among the variants tested. The ranks values produced by these tests confirm the selection of $\alpha = 0.8$ and $k = 1.0$.

5.3 Different GRASP designs

With the search parameters set as indicated above, we proceed to compare the relative merit of the GRASP variants. In particular, we explore the contribution of the three local search algorithms proposed in Section 3.3, LS_H, LS_{A1} and LS_{A2}, when applied separately or in combination. Table 3 reports the results of five different methods when solving the 45 instances in the training set by generating 100 solutions for each instance. The first one is simply the constructive method with no local search, C, and it is considered as a baseline in this experiment. The next two are GRASP algorithms formed with the constructive method plus either LS_H or LS_{A2}, denoted $C + LS_H$ and $C + LS_{A2}$, respectively. Finally, the last two GRASP methods combine in their local search phase LS_H with LS_{A1} or with LS_{A2}. In particular, in $C + LS_H + LS_{A1}$, each constructed solution is improved first with LS_H, and the resulting local optimum is then submitted to LS_{A1}, which provides the output of the entire method. Similarly, $C + LS_H + LS_{A2}$ applies LS_{A2} to the solutions obtained with LS_H. Table 3 shows the average results obtained according to the size of the instances, classified as small ($1 \leq n \leq 45$), medium ($50 \leq n \leq 95$) , and large ($100 \leq n \leq 200$).

Algorithm	Dev	CPU
C	8.11%	0.09
large	8.59%	0.21
medium	10.81%	0.09
small	6.32%	0.01
C + LS_H	0.25%	61.90
large	0.10%	179.16
medium	0.06%	34.87
small	0.46%	0.55
C + LS_{A2}	7.85%	55.03
large	8.38%	169.66
medium	10.59%	18.99
small	6.00%	0.33
C + LS_H + LS_{A1}	0.20%	62.52
large	0.09%	181.17
medium	0.06%	34.94
small	0.35%	0.56
C + LS_H + LS_{A2}	0.00%	88.80
large	0.00%	260.83
medium	0.00%	45.48
small	0.00%	0.81

Table 3. Comparison of GRASP variants.

Results in Table 3 clearly show that, as expected, $C + LS_H + LS_{A2}$ obtains the best solutions overall (0.00% deviation from best), although it requires the longest running times of the five methods (88.80 seconds on average). Comparing the 0.25% average deviation achieved by $C + LS_H$ on 61.90 seconds with the 7.85% achieved by $C + LS_{A2}$ on 55.03 seconds, we can confirm that LS_H performs a more efficient exploration of the search space than LS_{A2}. However, LS_H is not able to reach the best known solutions by itself, and it needs the post-processing of LS_{A2} to match the best values (as shown with $C + LS_H + LS_{A2}$).

We applied the Friedman test for paired samples to the data used to generate Table 3. The resulting *p*-value of 0.000 obtained in this experiment clearly indicates that there are statistically significant differences among the five methods tested. A typical post-test analysis consists of ranking the methods under comparison according to the average rank values computed with this test. According to this, we obtain that the $C + LS_H + LS_{A2}$ method is the best overall with an average rank of 1.36, followed by $C + LS_H + LS_{A1}$ with an average rank of 2.24 and $C + LS_H$ with 2.40. Finally, we obtain the two methods with larger rank values (as compared with the previous methods): $C + LS_{A2}$ (4.09) and C (4.91).

We compare now the results of $C + LS_H$ and $C + LS_{A2}$ shown in Table 3 with two well-known non-parametric tests for pairwise comparisons: the Wilcoxon test and the Sign test. The former one answers the question: Do the two samples (solutions obtained with both methods in our case) represent two different populations? The resulting *p*-value of 0.000 indicates that the values compared come from different methods. On the other hand, the Sign test computes the number of instances on which an algorithm supersedes another one. The resulting *p*-value of 0.000 indicates that $C + LS_H$ is the clear winner between both methods.

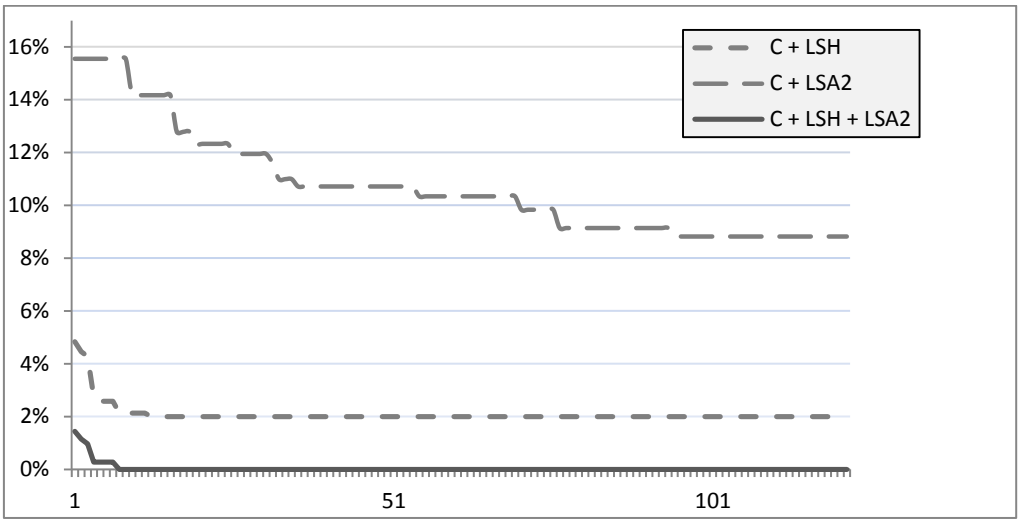


Figure 4. Search Profile.

To complement the results above, we study now the search profile of the most interesting combinations of local searches. Figure 4 shows the progression of the average deviation found by three methods for the training set of instances during 120 iterations of search time. The figure shows how most improvements on the best solution obtained with the C+LSH+LSA2 and C+LSH methods are achieved early in the search (i.e., within 10% of the number of iterations). After that point, both methods stagnate, and only exhibit a marginal improvement in the next iterations. On the other hand, C+LSA2 performs worse, with an average percentage deviation of several orders of magnitude larger than the other methods.

$tmax$	λ	Dev	CPU	# of skip
10	-1	0.152%	15.0	76.0
	0	0.135%	34.4	44.5
	1	0.000%	53.9	16.3
	2	0.000%	60.7	4.2
	3	0.000%	62.4	1.1
20	-1	0.151%	21.3	67.8
	0	0.151%	39.1	39.9
	1	0.072%	53.9	14.8
	2	0.000%	59.4	2.9
	3	0.000%	60.3	0.6

Table 4. Filtering GRASP constructions.

In our next experiment we test the efficiency of the filtering mechanism described in Section 4. Table 4 reports the results obtained with the $C + LS_H + LS_{A2}$ method when running with different values of the two filter parameters: $tmax$, the number of initial iterations for which we compute the mean $\widehat{\mu_P}$ and standard deviation $\widehat{\sigma_P}$ of the improvement achieved with the local search, and λ, the value to compute the filtering threshold $\widehat{\mu_P} + \lambda\widehat{\sigma_P}$. Specifically, it reports the average Dev and CPU, as in the previous experiments, and the average number of solutions discarded for improvement, # of skip, out of the 100 solutions constructed.

Table 4 shows that, as expected, the larger the λ value, the longer the CPU time. In other words, with low λ values the method discards more constructed solutions and therefore requires less CPU time than with larger values (given that it saves the computation of the local search associated with the discarded solutions). For example, with $\lambda = 0$ and $tmax = 20$, an average of 39.9 out of the 100 solutions constructed are discarded and the method only applies the local search to the remaining solutions. This has two consequences; the first one is that the CPU time is 39.1, which is lower than the 88.80 reported in Table 3 in which no filtering was applied and the 100 constructed solutions were submitted to the local search. The second one is that the average percentage deviation is 0.151% instead of the 0.00% of the unfiltered GRASP. On the other hand, this table shows that there are small differences when $tmax = 10$ or 20 with a slightly improvement in the former case. Therefore, in the following experiments we set $tmax = 10$ and denote the method as GRASP(λ) selecting the λ value in each experiment according to this trade-off between computing time and solution quality.

5.4 Comparison with Optimal Values

In Section 2 we have described the integer formulation proposed in Yaman (2011) for the UrApHMP. We have applied this formulation with CPLEX to solve 105 small instances with n ranging from 25 to 50, and $p, r = 1, 2, 3, 4,$ and 5. Table 5 reports the average percentage deviation with respect to the optimal solution of GRASP(λ) with $\lambda = 0$, which filters many constructions as shown above, and $\lambda = 3$, in which the filter is basically not applied. Note that in this experiment, instead of reporting the number of best solutions found, we report the number of instances in which the method is able to match the optimal solution (Opt).

			GRASP(0)			GRASP(3)		
n	**p**	**# instances**	**Opt**	**Dev**	**CPU**	**Opt**	**Dev**	**CPU**
	1	5	5	0.00%	0.04	5	0.00%	0.05
	2	10	10	0.00%	0.12	10	0.00%	0.21
25	3	15	15	0.00%	0.22	15	0.00%	0.49
	4	20	15	0.17%	0.42	15	0.17%	1.17
	5	25	24	0.03%	0.92	24	0.03%	2.14
	Total	75	69	0.06%	0.48	69	0.06%	1.15
	1	1	1	0.00%	0.16	1	0.00%	0.27
	2	2	2	0.00%	0.56	2	0.00%	1.04
40	3	3	3	0.00%	0.79	3	0.00%	2.18
	4	4	4	0.00%	2.48	4	0.00%	5.75
	5	5	4	0.04%	4.18	4	0.04%	9.10
	Total	15	14	0.01%	2.30	14	0.01%	5.16
	1	1	1	0.00%	0.27	1	0.00%	0.50
	2	2	2	0.00%	1.09	2	0.00%	1.85
50	3	3	3	0.00%	3.29	3	0.00%	5.22
	4	4	4	0.00%	7.16	4	0.00%	11.46
	5	5	5	0.00%	11.01	5	0.00%	19.77
	Total	15	15	0.00%	6.40	15	0.00%	10.97

Table 5. GRASP deviations from the optimal value.

Table 5 shows that the GRASP method is able to obtain the optimal solution in most cases. Even the filtered variant, GRASP(0), obtains 98 optimal values out of the 105 instances considered. It is worth mentioning that CPLEX requires up to 3 hours of computing time to obtain the optimal value on the instances with $n = 50$, and that larger instances cannot be solved due to memory requirements. Therefore we cannot provide the optimal values for larger instances.

5.5 Comparison with Optimal Assignments

As it has been described before, to obtain a solution for this problem we face a three step process:

1. Selecting the p hubs,
2. Determining the assignments of each node to r of the p hubs, and
3. Identifying, for each pair of nodes, the traffic through the appropriated hubs.

In the previous section, we have considered the linear integer formulation in Yaman (2011) to obtain the optimal solution for the small instances of this problem. In order to measure the quality of the results obtained with our method, and considering that we cannot solve larger instances using this formulation, we have adapted it to optimally solve the assignment and routing subproblems (steps 2 and 3 above). In particular, we first use our heuristic to select the set of p hubs H. Then, to assign the nodes to hubs and compute the traffics among nodes, we solve the following integer problem:

$$min \sum_{i\in N}\sum_{j\in N}\sum_{k\in H}\sum_{l\in H} t_{ij}\left(d_{ik} + \alpha d_{kl} + d_{lj}\right) f_{ijkl} \quad (1)$$

$Subject\ to$:

$$\sum_{k\in H} z_{ik} \leq r \quad \forall i \in N \quad (2)$$

$$\sum_{k\in H}\sum_{l\in H} f_{ijkl} = 1 \quad \forall i,j \in N \quad (3)$$

$$\sum_{l\in H} f_{ijkl} \leq z_{ik} \quad \forall i,j \in N,\ \forall k \in H \quad (4)$$

$$\sum_{k\in H} f_{ijkl} \leq z_{jl} \quad \forall i,j \in N,\ \forall l \in H \quad (5)$$

$$f_{ijkl} \geq 0 \quad \forall i,j \in N,\ \forall k,l \in H \quad (6)$$

$$z_{ik} \in \{0,\ 1\} \quad \forall i \in N,\ \forall k \in H \quad (7)$$

The formulation above, in which the set of hubs is fixed, is a special case of the one described in Section 2. We solve it with CPLEX 12.4 and obtain the optimal solution of the assignment and routing subproblems for this specific set of hubs. It is clear that since H has been obtained heuristically, the optimality of the resulting solution cannot be guaranteed. However, with this

experiment, we are able to test whether our algorithm is obtaining or not the optimal solution in steps 2 and 3 for a given solution of step 1.

Table 6 shows, for the 318 medium and large instances in the AP set, its number of nodes (n), the number of instances tested for each value of n (# instances), the number of instances in which the GRASP obtains the optimal solution in steps 2 and 3 (Opt), and the average percentage deviation with respect to the assignment and routing optimal values (Dev). The results in this table clearly show that our GRASP algorithm is able to match the optimal assignment and routing values in all the AP instances tested.

Size	n	# instances	Opt	Dev
Medium	70	35	35	0.00%
	75	35	35	0.00%
	80	35	35	0.00%
	85	35	35	0.00%
	90	35	35	0.00%
	95	35	35	0.00%
	Total	210	210	0.00%
Large	100	36	36	0.00%
	150	36	36	0.00%
	200	36	36	0.00%
	Total	108	108	0.00%

Table 6. GRASP deviations from the assignment and routing optimal values on AP instances.

In order to study the behavior of our algorithm on larger instances, we have repeated the above experiment on the set of 30 instances generated from the USA423 set. As it has been mentioned, all the 30 instances have 423 nodes, while p takes values in $\{3, 4, 5, 6, 7\}$ and $\mathrm{r} \in \{2, \ldots, \mathrm{p}-1\}$. Two different sets of values for the cost parameters χ, α, δ have been used for each instance: 0.1, 0.07, 0.09 (denoted by A), and 0.09, 0.075, 0.08 (denoted by B). The results are shown in Table 7, which are average values for all the instances with a given value of p and values of $r \in \{2, \ldots, p-1\}$. CPU GRASP column denotes the maximum time allowed to the GRASP, while CPU CPLEX shows the time needed by CPLEX to get the optimal allocation and routing values for the set of p hubs obtained with the GRASP. All times are shown in minutes.

Although in these instances the GRASP algorithm cannot match the optimal assignment and routing values, in our opinion the results shown in Table 7 are very good. They show a deviation from the optimal values that never exceeds a 0.85% on average, and only in two out of the 30 instances slightly exceeds 1%. Note that CPLEX needs more than 20 hours of computing time on average to solve these instances.

Cost parameters	p	# instances	Dev	CPU GRASP	CPU CPLEX
A	3	1	0.00%	30	21.5
A	4	2	0.01%	30	224.5
A	5	3	0.12%	30	197.7
A	6	4	0.21%	60	148.1
A	7	5	0.82%	60	1232.1
B	3	1	0.00%	30	0.5
B	4	2	0.01%	30	52.6
B	5	3	0.20%	30	38.7
B	6	4	0.19%	60	627.5
B	7	5	0.78%	60	280.4

Table 7. GRASP deviations from the assignment and routing optimal values on USA423 instances.

5.6 Comparison with Previous Heuristics

As far as we know, there is no previous heuristic for the $UrApHMP$, in which each node can be connected to at most r of the p hubs. However, two particular cases of this general problem, the single and the multiple allocation problems, have been extensively studied, and we can compare our method with previously proposed heuristics for these two cases. In particular, in the *Uncapacitated Multiple Allocation p-Hub Median Problem*, each node can send and receive traffic through any of the p hubs, while in the *Uncapacitated Single Allocation p-Hub Median Problem*, each node is assigned to one of the p hubs, only allowing sending and receiving traffic through this single hub. Although we have designed our GRASP algorithm to solve the general *Uncapacitated r-Allocation p-Hub Median Problem* and it does not take advantage of the specific characteristics of these two special cases, we can apply it to solve them and to ascertain if the GRASP is able to compete with recently published methods specially proposed for these multiple and single versions.

Table 8 shows the results of our GRASP when solving the multiple allocation problem. This table shows the results obtained with the GRASP for a single iteration (one construction plus the local search), denoted GRASP_1, and for 10 iterations, GRASP_10. The table also reports the results of the evolutionary approach recently proposed by Milanović (2010) for the multiple version. The three algorithms are applied on the 61 AP instances reported in that paper. Specifically, Table 8 shows the size and number of instances in each row, and, for each method, the number of instances in which it obtains the best known solution, the average percentage deviation with respect to this best, and the CPU in seconds. Results for the evolutionary method are directly taken from Milanović (2010), and therefore running times are only indicative and cannot be directly compared with GRASP computing times.

		GRASP_1			GRASP_10			Evolutionary		
n	**# instances**	**Best**	**Dev**	**CPU**	**Best**	**Dev**	**CPU**	**Best**	**Dev**	**CPU**
10	7	5	0.77%	0.00	7	0.00%	0.01	7	0.00%	0.05
20	7	6	0.27%	0.02	7	0.00%	0.10	7	0.00%	0.16
25	7	5	0.20%	0.04	7	0.00%	0.21	7	0.00%	0.24
40	9	7	0.28%	0.41	9	0.00%	1.94	9	0.00%	1.19
50	9	7	0.08%	0.83	8	0.00%	4.87	9	0.00%	7.36
100	11	6	0.22%	40.91	10	0.00%	321.00	10	0.01%	67.93
200	11	5	0.10%	780.42	9	0.00%	1879.25	9	0.08%	346.62
Total	61	41	0.25%	148.30	57	0.00%	373.12	58	0.01%	76.07

Table 8. GRASP vs. evolutionary method with $r = p$.

Results in Table 8 clearly show that the GRASP algorithm is able to obtain state-of-the-art results for the multiple allocation problem. Even GRASP_1, the version in which only one solution is generated, performs remarkably well, obtaining results of similar quality to those reported with the evolutionary method by Milanović (2010), specifically designed for this problem. As expected, the GRASP_10 variant is more time consuming than GRASP_1 and is able to match, and in some cases surpass, the evolutionary method. In particular, it obtains a new best known value of 92646.38 in the AP instance with n=200 and p=15 for which the previously best known value was 92669.64.

Table 9 shows the results for the single allocation problem. As in the previous experiment, we report the results with GRASP_1 and GRASP_10. We also report in this table the results of the Genetic Algorithm (GA) described in Kratica et al. (2007). This table shows the Dev and CPU values for each method on the 14 AP instances reported in that paper.

		GRASP_1		GRASP_10		GA	
n	**p**	**Dev**	**CPU**	**Dev**	**CPU**	**Dev**	**CPU**
	2	0.04%	0.17	0.04%	0.98	0.00%	13.29
	3	0.00%	0.41	0.00%	4.38	0.00%	16.64
	4	0.03%	0.41	0.03%	4.40	0.00%	17.76
100	5	0.04%	0.91	0.04%	5.29	0.00%	22.11
	10	0.00%	3.47	0.00%	34.80	0.00%	40.73
	15	1.43%	8.42	0.26%	111.51	0.00%	57.54
	20	0.04%	16.08	0.04%	67.89	0.00%	79.20
	2	0.00%	1.82	0.00%	15.88	0.00%	100.72
	3	0.07%	3.61	0.07%	43.48	0.00%	111.77
	4	0.03%	5.78	0.03%	30.48	0.00%	131.16
200	5	0.45%	10.08	0.26%	54.43	0.08%	169.73
	10	0.39%	39.63	0.39%	454.83	0.00%	259.14
	15	1.07%	142.34	0.56%	1213.85	0.04%	313.02
	20	1.88%	331.82	0.11%	3432.02	0.20%	374.75
Average		0.39%	40.35	0.13%	391.02	0.02%	121.97

Table 9. GRASP vs. GA with $r = 1$.

To complement the information above, Table 10 reports the results of our two GRASP variants and the VNS approach presented by Ilić et al. (2010) on the 8 AP instances reported in that paper. Note that these are the hardest AP instances for this problem.

		GRASP_1		GRASP_10		VNS	
n	**p**	**Dev**	**CPU**	**Dev**	**CPU**	**Dev**	**CPU**
100	5	0.04%	0.91	0.04%	5.29	0.00%	0.08
	10	0.00%	3.47	0.00%	34.80	0.00%	0.67
	15	1.43%	8.42	0.26%	111.51	0.00%	3.22
	20	0.04%	16.08	0.04%	67.89	0.00%	3.57
200	5	0.45%	10.08	0.26%	54.43	0.00%	5.16
	10	0.39%	39.63	0.39%	454.83	0.00%	5.60
	15	1.07%	142.34	0.56%	1213.85	0.00%	17.66
	20	1.88%	331.82	0.11%	3432.02	0.00%	12.98
Average		0.66%	69.09	0.21%	671.83	0.00%	6.12

Table 10. GRASP vs. VNS with $r = 1$.

Tables 9 and 10 show that the GRASP variants are able to obtain good results on the single allocation version, although they behave slightly worse than the specialized algorithms for this problem. In particular, the GA by Kratica et al. (2007), see Table 9, shows an average percentage deviation of 0.02% obtained in 121.97 seconds, while GRASP_1 and GRASP_10 obtain an average percentage deviation of 0.39% and 0.13% in 69.06 and 671.83 seconds, respectively. The VNS by Ilić et al. (2010) performs remarkably well since it is able to achieve an average percentage deviation of 0.00% in 6.12 seconds. Note, however, that the GRASP algorithm is not designed to exploit the particular characteristics of the single version of this problem, as it does the VNS, and the objective of this comparison is to show that it performs relatively well across different types of p-hub problems.

5.7 Run time distribution

Aiex, Resende and Ribeiro (2007) observed that the variable *time-to-target-value* usually has in GRASP an exponential distribution. Time-to-target (TTT) plots display on the ordinate axis the probability that an algorithm will find a solution at least as good as a given target value within a given running time, shown on the abscissa axis. TTT plots are used to characterize the running times of stochastic algorithms for combinatorial optimization. Specifically, for each instance/target pair, the running times are sorted in increasing order. We associate with the i-th sorted running time t_i a probability $p_i = (i - {}^1/_2)/n$ and plot the points (t_i, p_i). The resulting diagram shows the cumulative probability distribution plot and permits to check whether a given algorithm has or not an exponential distribution.

We ran 100 times our GRASP for the UrApHMP on a representative instance, stopping when a solution is found with objective value equal to the best known for this instance. For each run we recorded the running time. Each run is independent of the other by using a different initial seed for the random number generator. With these 100 running times, we plot the time-to-target plot (run time distributions) shown in Figure 4, in which we add the theoretical

exponential distribution. This experiment confirms the expected exponential runtime distribution for our GRASP. Therefore, linear speed is expected if the algorithm is implemented in parallel.

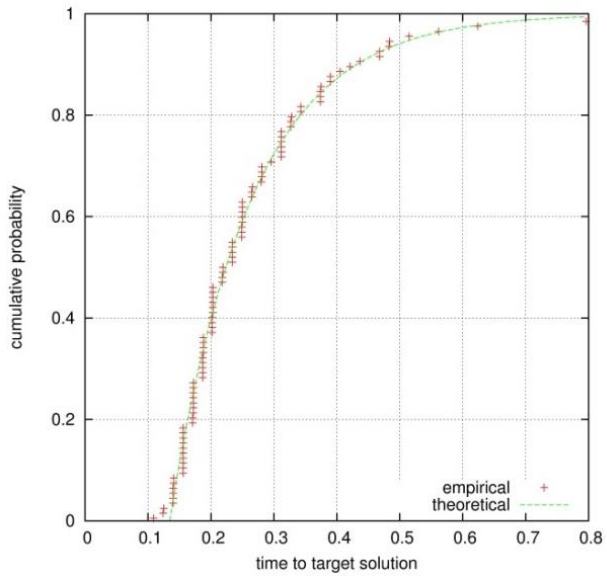


Figure 4. Time to target plot.

6. CONCLUSIONS

We have developed a heuristic procedure based on the GRASP methodology that provides high quality solutions for the Uncapacitated r-Allocation p-Hub Median Problem. We have explored the critical issue of which solution-generation-method proves effective to obtain a good set of solutions in terms of quality and diversity. We have defined three neighbourhoods in the local search and a filtering mechanism to selectively apply it.

Overall experiments with 465 instances were performed to assess the merit of the procedures developed here. Our implementation was shown to be competitive in a set of instances previously reported in the literature. Moreover, the procedure has been shown to be robust in terms of solution quality within a reasonable computational effort. The proposed method was compared with the linear integer formulation implemented in CPLEX and with previous heuristics for different p-hubs variants (single and multiple allocation problems). The experimentation shows that the GRASP method is able to obtain high quality solutions across different p-hub problems.

Finally, a new set of instances, USA423, has been proposed. It is based on real airline data and consists of a data file concerning 423 cities in the United States, where real distances and passenger flows for an accumulated 3 months period are considered.

ACKNOWLEDGEMENTS

This work was supported by the research projects TIN2009-07516, MTM2009-14039-C06-02, TIN2012-35632-C02, and MTM2012-36163-C06-02.

Authors want to thank the company Data In, Information Out (DIIO) for providing us with new real data for the test problems. DIIO (www.diio.net) is a world leader firm in aviation business intelligence tools.

REFERENCES

Aiex, R. M., M. G. C. Resende, and C. C. Ribeiro. 2007. TTT plots: A perl program to create time-to-target plots. Optimization Letters 1, no. 4: 355-366.

Campbell, J. F. 1994. Integer programming formulations of discrete hub location problems. European Journal of Operational Research 72, no. 2: 387-405.

Campbell, J. F. and M. E. O'Kelly. 2012. Twenty-five years of hub location research. Transportation Science 46, no. 2: 153-169.

Ernst, A. and M. Krishnamoorthy. 1996. Efficient algorithms for the uncapacitated single allocation p-hub median problem. Location Science 4, no. 3: 139–154.

Feo, T. A. and M. G. C. Resende. 1989. A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters 8, no. 2: 67-71.

Feo, T. A. and M. G. C. Resende. 1995. Greedy randomized adaptive search procedures. Journal of Global Optimization 6, no. 2: 109-133.

Festa, P. and M. G. C. Resende. 2011. GRASP: Basic components and enhancements. Telecommunication Systems 46, no. 3: 253-271.

García, S., M. Landete, and A. Marín. 2012. New formulation and a branch-and-cut algorithm for the multiple allocation p-hub median problem. European Journal of Operational Research 220, no. 1: 48-57.

Gelareh, S. and S. Nickel. 2011. Hub location problems in transportation networks. Transportation Research Part E: Logistics and Transportation Review 47, no. 6: 1092-1111.

Ilić, A., D. Urošević, J. Brimberg, and N. Mladenović. 2010. A general variable neighborhood search for solving the uncapacitated single allocation p-hub median problem. European Journal of Operational Research 206, no. 2: 289-300.

Kratica, J., Z. Stanimirović, D. Tošić, and V. Filipović. 2007. Two genetic algorithms for solving the uncapacitated single allocation p-hub median problem. European Journal of Operational Research 182, no. 1: 15-28.

Laguna, M. and R. Martí. 1999. GRASP and path relinking for 2-layer straight line crossing minimization. INFORMS Journal on Computing 11, no. 1: 44-52.

Love, R.F., Morris, J.G., and Wesolowski, G.O. 1988. Facilities location: Models and methods. Elsevier Science Publishing Co., New York.

Milanović, M. 2010. A new evolutionary based approach for solving the uncapacitated multiple allocation p-hub median problem. In Soft Computing in Industrial Applications, AISC 75 (X. Z. Gao et al., eds.). Springer-Verlag, Berlin, pp: 81-88.

O'Kelly, M. E. 1987. A quadratic integer program for the location of interacting hub facilities. European Journal of Operational Research 32, no. 3: 393-404.

Yaman, H. 2011. Allocation strategies in hub networks. European Journal of Operational Research 211, no. 3: 442-451.