Computers and Operations Research 86 (2017) 94–109

Contents lists available at ScienceDirect

Computers and Operations Research

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

Heuristics for the capacitated modular hub location problem

Arild Hoff[a,*], Juanjo Peiró[b,c], Ángel Corberán[b], Rafael Martí[b]

[a] Molde University College, Norway
[b] Departament d'Estadística i Investigació Operativa. Universitat de València, Spain
[c] Departamento de Economía Aplicada (Área de Métodos Cuantitativos para la Economía y la Empresa). Universidad de Burgos, Spain

ARTICLE INFO

Article history:
Received 30 January 2017
Revised 2 May 2017
Accepted 3 May 2017
Available online 4 May 2017

Keywords:
Hub location
Modular links
Heuristic algorithms
Memory structures

ABSTRACT

In this paper we study the hub location problem, where the goal is to identify an optimal subset of facilities (hubs) to minimize the transportation cost while satisfying certain capacity constraints. In particular, we target the single assignment version, in which each node in the transportation network is assigned to only one hub to route its traffic. We consider here a realistic variant introduced previously, in which the capacity of edges between hubs is increased in a modular way. This reflects the practical situation in air traffic where the number of flights between two locations implies a capacity in terms of number of passengers. Then, the capacity can be increased in a modular way, as a factor of the number of flights. We propose heuristic methods to obtain high-quality solutions in short computing times. Specifically, we implement memory structures to create advanced search methods and compare them with previous heuristics on a set of benchmark instances. Memory structures have been widely implemented in the context of the tabu search methodology, usually embedded in local search algorithms. In this paper we explore an alternative design in which the constructive method is enhanced with frequency information and the local search is coupled with a path relinking post-processing. Statistical tests confirm the superiority of our proposal with respect to previous developments.

1. Introduction

The capacitated single assignment hub location problem with modular link capacities (CSHLPMLC) was first formulated by Yaman and Carello (2005) as a variant of hub location problems used in telecommunication. Alumur and Kara (2008) define hubs as special facilities that serve as switching, transhipment and sorting points in many-to-many distribution systems where the problem is to find the best facilities to serve as hubs among a number of potential locations. According to Campbell and O'Kelly (2012), the key distinguishing features in hub location are:

1. Demand is associated with flows between origin-destination (OD) pairs.
2. Flows are allowed to go through hub facilities.
3. Hubs are facilities to be located.
4. There is a benefit of routing flows via hubs.
5. There is an objective that depends on the location of hub facilities and the routing of flows.

For the basic hub location problems, two additional features are included:

6. Paths between OD pairs visit at most two hubs.
7. Directs OD flows are not allowed.

The survey paper from Farahani et al. (2013) gives an extensive overview of models classification, solution techniques, and applications for various types of hub location problems.

The CSHLPMLC is defined in a backbone/tributary network (Klincewicz, 1998) where the different nodes are defined either as hubs or as terminals. The hubs are connected through a backbone network while the terminals are connected to hubs through tributary networks. The CSHLPMLC considers a fully connected backbone network where each terminal node is assigned to exactly one hub. Thus, direct transportation between nodes is not possible and any traffic from one terminal to another is routed first to the hub to which the terminal is connected. Then, it is routed to the receiving terminal if this terminal is connected to the same hub; otherwise, it is routed to the receiving terminal through the hub that terminal is connected to. The links in the backbone network have a given capacity, and the total traffic on an arc determines how many links on the same arc are needed. The decision when designing such a network is twofold: first, to decide which nodes will serve as hubs and, then, to assign the remaining terminals to

* Corresponding author.
E-mail addresses: Arild.Hoff@hiMolde.no (A. Hoff), Juanjo.Peiro@uv.es (J. Peiró), Angel.Corberan@uv.es (Á. Corberán), Rafael.Marti@uv.es (R. Martí).

http://dx.doi.org/10.1016/j.cor.2017.05.004
0305-0548/

the hubs. This practical problem is especially evident in air transportation, where it is impractical to schedule a direct flight between every pair of cities due to its high operational cost. Air flight companies thus resort to route their passengers through intermediate airports, usually called hubs. Other typical examples where the hub network idea is used are the telecommunication industry and postal services. Farahani et al. (2013) also point out that maritime industry, freight transportation companies, public transit, and message delivery networks are areas that can take advantage of this idea.

The CSHLPMLC variant was introduced and modeled in Yaman and Carello (2005). In that paper, authors proposed a branch-and-cut algorithm for its solution, and a tabu search metaheuristic to provide good upper bounds for the exact method. Later, Corberán et al. (2016) developed a heuristic method based on strategic oscillation to solve larger instances of the problem and presented computational results on 170 test instances taken from the US airline industry and the Australian postal service.

In this article we propose a new heuristic to solve the CSHLPMLC. We introduce memory structures (frequency information recorded during the search) to enhance the performance of our methods. Memory-based strategies were introduced in the context of the well-known tabu search methodology (Glover and Laguna, 1997), by which alternative forms of memory are appropriately combined with effective strategies for exploiting them. We propose simple and effective implementations of these structures, based on frequencies, in the construction method and on a path relinking post-processing. Additionally, we explore some extensions of the CSHLPMLC model and adapt our heuristics to target them. In particular, we consider the non-stop services (a model where direct shipping between some non-hubs is permitted, as in Aykin, 1995a), and different capacities in the nodes. As far as we know, these two extensions have not been considered before in the context of the capacitated modular hub location problem.

The structure of this paper is as follows. After the Introduction, a literature review follows in Section 2, while the detailed problem formulation is shown in Section 3. Then, Section 4 describes our construction and improvement methods, as well as the path relinking procedure. Section 5 presents the computational results, while Section 6 discusses how to consider non-stop services in our algorithm for the CSHLPMLC. Finally, Section 7 contains the conclusions of the research.

2. Literature review

Hub location problems have been the subject of research for more than 25 years, as pointed out by Campbell and O'Kelly (2012) in their excellent review "Twenty-five years of hub location research". According to, Bryan and O'Kelly (1999), the starting point of the analytical research on hub location problems was a seminal paper by O'Kelly (1987), where a mathematical programming formulation and two heuristics were first proposed for a hub location problem. Considering that even simple hub location models have proven difficult to solve, Bryan and O'Kelly (1999) pointed out that the research in this area soon focused on the development of heuristics. As with other challenging optimization problems, hub location has been an important test for most of the heuristic methodologies, such as tabu search, GRASP, Lagrangean heuristics, variable neighborhood search, or simulated annealing, to mention a few.

As explained in Alumur and Kara (2008), papers in this field often assume three things: the hub network (also known as backbone network) is complete, the economies of scale incorporate a discount factor (the so-called α), and that direct services among non-hub nodes are not allowed. These authors reviewed the two classic families of hub location problems (single and multiple assignment) and summarized the most relevant papers published until 2008. In *single assignment* (*allocation*) problems each node is assigned to a single hub, while in *multiple assignment* problems each node is assigned to more than one hub. This study examined more than 100 papers and showed that the number of papers on single assignment models published until 2008 was more than twice the number of papers devoted to multiple assignment models. In their conclusions, Alumur and Kara noted that until year 2000 hub location research was mainly focused on defining and formulating models, while after that year solution methodologies attracted the attention of most researchers.

Two other important classifications of hub location problems are related with the opening cost and the capacity of the hubs, respectively. Some models consider a fixed number of hubs (usually denoted by p), while others consider a fixed cost of opening a hub, making the number of hubs to open a decision variable. When the number of hubs is not fixed, a more realistic variant is to consider that hubs have a limited capacity. In line with this, Bryan and O'Kelly (1999) noted that it is a challenge in this field to incorporate key characteristics of real networks without making the model impracticable.

Research on these type of problems has traditionally considered a fully interconnected backbone network with no direct links between terminals (Aykin, 1995b; Ernst and Krishnamoorthy, 1996). An interesting extension is considered in Aykin (1995a), where hub locations and service types for the routes are determined together. In particular, this model is based on the problems arising in air passenger transportation, where direct (non-stop) flights between some non-hub cities are offered by the airlines in order to improve the service to their customers. As a matter of fact, low-cost flight companies mainly based their business in offering direct flights, instead of the classic hub-based flights. This is changing the current picture in airlines transportation, where both models are simultaneously considered. In line with this, Campbell and O'Kelly (2012) stated that "as the field has matured, the literature has moved to incorporate more realistic aspects of logistics and flow". In Aykin (1995a), the set of routes for which direct shipping is permitted is assumed to be predetermined (an input to the problem). Total cost is minimized considering that the cost per unit flow through hubs is lower than for non-stop services due to scale factors, but the distance traveled in direct flights is usually lower.

Sasaki and Fukushima (2003) introduced the problem with capacity constraints on both hubs and arcs. This problem was a multiple allocation problem with only one stop for the traffic between terminals and not involving modular links. Later, Correia et al. (2010) presented an extension of the single-assignment hub location problem in which hubs have a limited capacity, but each potential hub has a set of potential capacities available, and the size of the hub is part of the decision process. Other recent studies, such as Contreras et al. (2012), consider different levels of the capacities, which also provides more realistic models.

Classical hub location models usually have a linear objective function minimizing the overall operational cost. This total cost is computed as the product of a unitary cost by the number of passengers or items transported. Carello et al. (2004) considered the cost of installing on each edge the capacity needed to route the traffic on that edge. These modular link capacities (Yaman and Carello, 2005) result in a stepwise objective function. This characteristic models for example the real situation of the number of seats in a plane operating in a route. The total capacity of the route can be increased by adding more flights. We found this variant of the classical model very interesting since we believe that it addresses in a natural way the real situation of air transportation. We therefore consider this model and view our work as another extension of the classical hub location problems.

The concept of modular links was also addressed in Yaman (2008) but in the context of a star-star network. In such a network, all hubs are connected to one central hub instead to each other through a backbone network. Hence, no direct links are permitted between the terminals and the structure of the network is very different from the CSHLPMLS. A very recent paper by Tanash et al. (2017) also considers a hub location problem where flow-dependent transportation costs are modeled using modular link costs. In particular, these authors study a modular hub location problem with single assignments but where the full interconnection between hub nodes is not assumed. For this problem, the authors present two mixed integer programming formulations and propose a Lagrangean relaxation for one of them that allows to decompose the Lagrangean function into two independent subproblems, which can be solved efficiently. Based on this relaxation, and on a heuristic algorithm, the authors develop a branch and bound producing good computational results on benchmark instances with up to 75 nodes.

Another important extension of the classical models addresses the limitations from requiring full interconnectivity. Campbell et al. (2005) relax that restriction by locating discounted hub arcs instead. Although this approach represents a more realistic model than the traditional full connected one, Campbell and O'Kelly (2012) pointed out that most of the research still considers the traditional model. Other researchers like Yoon and Current (2008) and Calik et al. (2009) have also relaxed the constraint on full connectivity between the hubs, and Alumur et al. (2009) presents methods for designing incomplete hub networks for different variants of single allocation hub location problems. However, these variants do not include capacity restrictions on the hubs or the links. In our opinion, given the modular structure of the CSHLPMLS, it does not make much sense to consider an incomplete hub network, unless the traffic between two hubs is zero and no link is needed on that connection.

Hub location models are basically cost oriented or service oriented models. While in the hub median problem the objective is to minimize the total cost, in the hub center problem the objective is to minimize the maximum cost between origin-destination pairs. Alumur and Kara (2008) discussed on both models, and Peiró et al. (2016) proposed a bi-objective optimization model that accounts for both cost-efficiency and service. In the last paper, it is argued that the airline industry is such that if the routes connecting two terminal nodes (i.e., an origin and a destination) are far from ideal (e.g., a direct flight) it could result in a loss of customers to the competition. In order to keep operational costs in perspective, authors proposed a GRASP method to approximate the Pareto front of the bi-objective problem. Other authors, such as Elhedhli and Wu (2010) have interpreted the service in terms of congestion instead of the travel time addressed in previous approaches. They considered an objective function with three terms: transportation, congestion, and fixed cost, and proposed a Lagrangean heuristic to solve the model.

Yaman (2011) proposed a variant in which each node can be connected to at most r of the p hubs, called the Uncapacitated r-Allocation p-Hub Median Problem. According to this author, single allocation versions are too limited for real-world situations, while multiple allocation results in high fixed costs and complicated networks. Yaman therefore proposed an "in-between" model, generalizing both versions of the p-hub median problem. Peiró et al. (2014) proposed a metaheuristic for this problem. Extensive experimentation showed that their method is able to obtain high-quality solutions in short computing times.

Most of the hub location applications take place in the field of transportation or telecommunications. Note that in the latter setting, hubs usually refer to hardware, such as routers, concentrators, or switches to provide communications (electronic data) among a set of data centers. Although the underlying model is similar in both fields, the cost associated with installing or changing the location of the hubs is significantly lower in telecommunications. In this context, dynamic configurations have emerged as more realistic models than the traditional (static) ones. Contreras et al. (2011) proposed a model in which the dynamic nature of the problem is addressed by including three types of costs: opening, operating, and closing hubs. Campbell (2010) also studied dynamic models in the context of variations in the demand because of geographic expansion.

Contreras (2015) shows an overview of distinguishing features, assumptions and properties commonly considered in hub location problems. He presents a section on capacitated models, describing two types of capacity restrictions; one on the hub facilities and another on the arcs. In the case of capacitated arcs, splitting commodities could be an alternative in the multi-assignment problem, while considering additional links on the same arc is necessary in modular problems like the CSHLPMLS. Contreras (2015) also presents different topologies in hub networks, which are substantially more difficult to solve than the basic problem. Still, for the problems with capacity restrictions, it is usually common to assume a full backbone network between the hubs. In our paper, we will assume a complete hub subnetwork, although we will also discuss a way of considering non-stop services between terminals as an extension to the problem.

3. Problem formulation

In this paper we consider the *single assignment hub location problem*, where a terminal can only be assigned to a single hub. Let $G=(V, E)$ be a network, being V the set of nodes ($|V|=n$), and E the set of edges. For any pair of nodes $i, j \in V$, t_{ij} denotes the traffic to be transported from i to j that must be routed through at least one hub and at most two hubs, that is, paths between origins and destinations will never use more than two hubs and, as a result, hubs will be assumed to be fully interconnected. Assigning a terminal i to a hub k has a cost C_{ik}. For the sake of simplicity, we consider that each hub is assigned to itself. Each hub k has a maximum transit capacity Q_k^h, which indirectly limits the number of terminals that can be assigned to it, in terms of their total traffic. In this variant of the problem, the number of hubs is not specified beforehand, and opening a hub at node i has a fixed installation cost C_{ii}.

There are two types of edges between nodes: access edges, used to connect terminals with hubs, and backbone edges, used to connect hubs with other hubs. Each backbone edge has a maximum traffic capacity of Q^b (in each direction), which has to be understood as a capacity factor. In graph terms we will say that we add several links on the backbone edge. Let R_{kl} be the cost of each link on the edge connecting hubs k and l (i.e., the cost of a flight between k and l in air transportation). Therefore, if we represent by w_{kl} the number of edges (flights) between k and l, the maximum number of passengers that we can transport in this route is $Q^b w_{kl}$, and its associated cost $w_{kl}R_{kl}$. In short, the hub location problem considered in this paper consists of selecting a subset of nodes to be hubs, and assigning the rest of the nodes to them, in such a way that the transportation cost is minimized while satisfying the capacity constraints.

Yaman and Carello (2005) formulated the CSHLPMLC as a mixed integer non-linear program as follows:

Consider the variables:

- the assignment variable x_{ik} is equal to 1 if terminal i is assigned to hub k, and 0 otherwise. If node i receives a hub, then x_{ii} takes value 1,
- z_{kl} is the traffic on an arc $(k, l) \in A$,

- w_{kl} is the number of links on the edge $\{k, l\} \in E$.

The formulation is, then:

$$\min \sum_{i\in V}\sum_{k\in V} C_{ik}x_{ik} + \sum_{\{k,l\}\in E} R_{kl}w_{kl} \quad (1)$$

Subject to:

$$\sum_{k\in V} x_{ik} = 1, \ \forall i \in V \quad (2)$$

$$x_{ik} \le x_{kk}, \ \forall i \in V, \ \forall k \in V \setminus \{i\} \quad (3)$$

$$\sum_{i\in V}\sum_{j\in V}\left(t_{ij}+t_{ji}\right)x_{ik} - \sum_{i\in V}\sum_{j\in V} t_{ij}x_{ik}x_{jk} \le Q_k^h x_{kk}, \ \forall k \in V \quad (4)$$

$$z_{kl} \ge \sum_{i\in V}\sum_{j\in V} t_{ij}x_{ik}x_{jl}, \ \forall (k,l) \in A \quad (5)$$

$$Q^b w_{kl} \ge z_{kl}, \ \forall \{k,l\} \in E \quad (6)$$

$$Q^b w_{kl} \ge z_{lk}, \ \forall \{k,l\} \in E \quad (7)$$

$$x_{ik} \in \{0, 1\}, \ \forall i,k \in V \quad (8)$$

$$w_{kl} \ge 0 \text{ and integer}, \ \forall \{k,l\} \in E \quad (9)$$

$$z_{kl} \ge 0, \ \forall (k,l) \in A. \quad (10)$$

Constraints (2) imply that each node needs to be assigned to only one hub. Constraints (3) force node k to be a hub if a node i is assigned to it. Constraints (4) specify that the capacity of a given hub k cannot be less than the amount of traffic that transits through it, thus prohibiting allocations to k beyond its capacity Q_k^h. The subtraction of the second part on the left hand side of the constraint is to avoid double-counting of the traffic between pair of nodes both assigned to the same hub. Constraints (5) add up the traffics through a given arc (k, l). Finally, constraints (6) and (7) fix the number of links needed on each backbone edge.

Note that, as pointed out in Tanash et al. (2017), the above model can be seen as one in which flow-dependent costs are considered, since the total transportation cost is estimated not in terms of the per unit flow cost but in terms of the number of links used on each backbone edge. Although in the model proposed by Yaman and Carello (2005) all the hub capacities are equal, we have considered here the more general case in which they can be different. These authors also proposed a branch-and-cut algorithm and a heuristic method to solve this problem. The heuristic method, based on the tabu search methodology, feeds the branch-and-cut with a good initial upper bound. In particular, the solution provided by the metaheuristic is used to limit the number of variables considered by the exact method, by identifying a subset of nodes that represents the best potential locations for the hubs. The hubs selected in the best solution belong to this subset, as well as the two other hubs, which appear most often in the best solutions found by the metaheuristic.

Corberán et al. (2016) proposed a heuristic method, based on a strategic oscillation over the search space, to solve the CSHLPMLC. In particular, their method iteratively constructs and partially destructs a solution. In this way, hubs are selected and deselected in search of the optimal set of hubs. This procedure is coupled with two local searches, one based on swapping the assignment of terminals to hubs, and another based on exchanges of terminals to a different hub. Their computational experimentation showed that this method outperforms the tabu search heuristic in Yaman and Carello (2005), and that it is able to match the optimal solutions in the small size instances that CPLEX is able to solve.

4. Solution method

4.1. Construction method

A solution S for our problem consists of a set of hubs H and an assignment of each terminal to a hub. Note that with this assignment, the routing of the traffics between any pair of nodes is univocally determined through their respective hubs.

Corberán et al. (2016) proposed a constructive method to obtain an initial solution that selects nodes to be hubs in a greedy fashion. Specifically, the authors considered an evaluation function to discriminate among candidate nodes based on the costs. Their method iteratively selects the hub nodes until there are enough hubs (in terms of capacity) to assign all the nodes in the network. An important characteristic of that method is that, each time a node is selected as a hub, it performs the associated assignment of terminals to this hub in a greedy fashion ignoring future hub selections. In this paper, we propose an alternative construction method that performs the assignment step after the hub selection, thus taking into account the complete set of hubs.

Our construction method starts by estimating the number of hubs p that provides enough capacity to assign all the terminals. We basically consider the total traffic between all pairs i, j of nodes, and divide it by the average hub capacity Q^h, $Q^h = \frac{1}{|V|}\sum_{k\in V} Q_k^h$. In mathematical terms:

$$p = \left\lceil \frac{\sum_{i,j\in V} t_{ij}}{Q^h} \right\rceil$$

Note that traffics t_{ii} can take a positive value in some applications, as in the case of postal deliveries sent to a central hub for sorting.

Since nodes with large amount of traffics may not be assigned to the same hub, the value of p above can underestimate the required number of hubs to route all the traffic in the network. In particular, we compute the number of nodes for which their traffic exceeds half of the hub capacity:

$$\sum_{j\in V\setminus\{i\}} t_{ij} + \sum_{j\in V\setminus\{i\}} t_{ji} + t_{ii} > \frac{Q_i^h}{2}.$$

It is likely that two nodes verifying the expression above do not share the same hub since the sum of their traffics may be larger than Q^h. Therefore, if the number of nodes verifying this expression is larger than our estimation of p, we change p to be this number of nodes.

The method we propose here is a multi-start algorithm. Many multi-start methods in combinatorial optimization resort to randomization to perform multiple constructions. Among them, GRASP methodology (Festa and Resende, 2011) is probably one of the most popular. However, we have developed our constructive algorithm under a different paradigm: adaptive memory. Instead of randomization, we apply frequency values, which record past hub appearances to discourage their selection in future constructions.

To decide which hubs to open, we use an evaluation function, described in what follows. Suppose that some nodes have already been selected as hubs. We denote them by H'. In order to select the next hub, the following four elements are considered:

- **Traffic value**. For each node i, we compute the traffic through it (incoming, outgoing, and internal traffic) with origin or destination not in H' as $t(i) = \sum_{j\in V\setminus\{i\}} t_{ij} + \sum_{j\in V\setminus\{i\}} t_{ji} + t_{ii} - \sum_{j\setminus H'} t_{ij} - \sum_{j\setminus H'} t_{ji}$. We then compute the relative value as $\frac{t(i)}{\max_{j\in V\setminus H'} t(j)}$.
- **Opening cost**. We compute the relative fixed installation cost for each node i as $\frac{C_{ii}}{\max_{j\in V\setminus H'} C_{jj}}$.

- **Assignment cost**. For each node i we compute the $m = n/p$ nodes (i.e. the average number of terminals assigned to a hub) with lowest assignment cost to i. The absolute assignment cost of node i, $a(i)$, is defined as the sum of these m costs. The relative assignment cost is computed as $\frac{a(i)}{\max_{j \in V \setminus H'} a(j)}$.
- **Frequency value**. For each node i, we record in freq(i) the number of times (previous solutions) in which i has been a hub. The relative frequency is $\frac{\text{freq}(i)}{\max_{j \in V \setminus H'} \text{freq}(j)}$.

These four elements are merged into a single expression reflecting the attractiveness of a node to be selected as a hub. Since we cannot establish a priori their relative importance, we introduce some factors that will be empirically set. For each node i, attractive(i) is defined as:

$$\text{attractive}(i) = +\delta \frac{t(i)}{\max_{j \in V \setminus H'} t(j)} - \alpha \frac{C_{ii}}{\max_{j \in V \setminus H'} C_{jj}} - \beta \frac{a(i)}{\max_{j \in V \setminus H'} a(j)} - \gamma \frac{\text{freq}(i)}{\max_{j \in V \setminus H'} \text{freq}(j)},$$

where $\alpha, \beta, \gamma, \delta \in [0, 1]$, and $\alpha + \beta + \gamma + \delta = 1$.

Typically, a node with large traffics is attractive to be selected as a hub, whereas a high opening cost or large assignment costs discourages it. Regarding the frequencies, since we want to diversify the search, those nodes that have been selected as hubs in previous solutions are penalized in posterior constructions. In our computational experiments section, we will test different values of these parameters, which permits to isolate the effect of each of the three elements considered (apart from the frequency) and evaluate their contribution.

At each iteration, our multi-start constructive method selects the best hubs according to the attractive values. Once the hubs are selected, we proceed to assign the terminals to these hubs. Let H be the set of selected hubs. For each terminal i, we compute the best assignment cost $\bar{c}_i$ to the selected hubs as

$$\bar{c}_i = min_{h \in H} C_{ih} .$$

Then, we order all the terminals according to their $\bar{c}_i$-values, where the terminal with the minimum value comes first. Following this order, we assign each terminal to its best hub (the one in H with the minimum cost) if it has enough capacity. If this hub does not have enough capacity to route the traffics of terminal i because of previous assignments, we consider the second best hub in H for i according to the assignment cost. We proceed in this way, trying to assign terminal i to the best hub in H that can manage its traffic. It may happen that none of the hubs in H can accommodate a given terminal. In this case, we add this terminal to a list of *orphan nodes*. When the assignment procedure has explored all terminals, the orphan node with largest traffics is selected as a new hub and we assign as many orphan nodes as possible to it. This procedure, which searches among orphan nodes to select a new hub, is repeated until all nodes are assigned to a hub.

The following example illustrates the constructive method described above, that we call CM1. Suppose that we have a network with 15 nodes where we have estimated $p = 2$. We have computed attractive(i) for all the nodes and concluded that $H = \{3, 10\}$. Column "Order" of Table 1 shows the order of the terminals in V for assigning terminals to hubs according to the $\bar{c}_i$-values. The assignment process starts by terminal 4 and assigns it to hub 3, since it is the preferred hub for this terminal. The procedure continues by assigning terminal 1 to hub 10; terminal 7 to hub 3; terminals 2, 15, and 14 to hub 10; and terminal 9 to hub 3. By the time terminal 13 needs to be assigned, hub 10 is full of traffics, so terminal 13 is assigned to hub 3 (second best hub for terminal 13). Then, terminals 8, 11, and 5 are assigned to hub 3. At this point, hubs 3 and 10 are completely full, hence they cannot accommodate any other terminal. Node 6 (next terminal in the order) is then declared an orphan node. The same happens with node 12. Since no more nodes remain to be assigned, the orphan node with largest traffics is selected as a new hub. Suppose that this is the case of node 12. Therefore $H \leftarrow H \cup \{12\}$ and node 12 is assigned to itself. As there is still enough capacity in the new hub 12, terminal 6 is assigned to it. The assignment process finishes here because all terminals have been assigned to a hub.

Table 1
Example of ordered nodes and possible hubs in CM1.

Order	Terminal	Ordered hubs	
1st	4	3	10
2nd	1	10	3
3rd	7	3	10
4th	2	10	3
5th	15	10	3
6th	14	10	3
7th	9	3	10
8th	13	10	3
9th	8	3	10
10th	11	3	10
11th	5	3	10
12th	6	10	3
13th	12	10	3

Notice that the process implemented in CM1 is designed under the assumption that the best assignment for a terminal is the hub for which its assignment cost is the lowest. However, considering that this is a greedy process and that hubs are limited in terms of their capacity, it is clear that in many cases we cannot assign all the terminals to their preferred hub. For this reason, we also propose alternative constructive methods to implement different search strategies.

Constructive method CM2 orders the terminals in non-increasing order of the $\bar{c}_i$-values (i.e., the terminal with the largest value comes first). The rationale behind this rule is to assign first the terminal with highest minimum assignment cost to the hubs. Constructive method CM3 computes the lowest and the second lowest assignment cost for each terminal i:

$$\bar{c}_i = min_{h \in H} C_{ih} , \quad \bar{\bar{c}}_i = min_{h \in H \setminus \{h^*\}} C_{ih} , \quad \text{where } h^* = \arg min_{h \in H} .$$

Then, CM3 calculates the difference of the two values above, $d_i = \bar{\bar{c}}_i - \bar{c}_i$, to evaluate how "urgent" it is to assign i to its best hub. It is clear that if d_i is large, we should try to assign i to its best hub because otherwise the assignment cost will be greater. On the contrary, if d_i is low, the assignment costs of i to its best and second best hubs are very similar. CM3 orders the terminals according to the d_i-values in non-increasing order and assigns them to their best available hub in this order.

Once a solution S is obtained with one of the three methods above, we evaluate it. To this end, different costs are involved:

- The fixed cost of opening/installing hubs.
- The fixed cost of assigning each terminal to its associated hub.
- The cost of installing the links on the backbone edges needed to send the traffics between hubs.

As it is customary in multi-start methods, once a solution is constructed, we proceed to improve it. The improvement methods used are described in the next section.

4.2. Improvement methods

Corberán et al. (2016) consider two neighborhoods, *Npairs* and *Nalone*, to improve a solution by changing the assignments of terminals to hubs. *Npairs* implements a classical exchange in which two terminals i and j, assigned to hubs k and l respectively

($k \neq l$), swap their corresponding hubs (i.e., the move assigns i to hub l, and j to hub k). The authors proposed a local search method, *LSpairs*, which implements explores this neighborhood with a first improvement strategy, i.e. by scanning the list of terminals and applying this exchange every time terminals are assigned to different hubs and the objective function is reduced (while the capacity limits are satisfied). Note that an efficient computation of the objective value after a move requires a detailed study. We refer the reader to Corberán et al. (2016) for such a study.

As the authors mentioned, the *Npairs* neighborhood turns out to be too restrictive due to the capacity constraints, so they complemented it with the *Nalone* neighborhood. This second neighborhood performs a simple insertion move in which the assignment of a terminal is changed from a hub to another hub. As in the previous neighborhood, authors proposed a local search method, *LSalone*, based on this move, which implements a first improvement strategy.

The experimental testing in Corberán et al. (2016) shows that the combination of the two neighborhoods is able to significantly improve the constructed solutions. Here, we want to go a step further by including the possibility of changing the hub selection of a given solution in the neighborhood exploration. As a matter of fact, we believe that further reductions in the objective function can be achieved by permitting the local search method to test different sets of hubs. Hence, we will consider the options of including and removing a hub in/from a solution, as well as changing the set of hubs.

In order to include or remove a hub from a given solution we propose two new neighborhoods, *Nadd* and *Nremove*, which explore the possibility of increasing the number of hubs in a solution and decreasing it, respectively. In particular, *Nadd* tries to increase the current set of hubs by adding to it as many new hubs as possible meanwhile these inclusions provide better solutions. It starts by selecting the non-hub node i with better value of attractive(i). Node i is added to the current set of hubs and a new assignment of terminals to hubs is done. Note that this process always produces a feasible solution that is evaluated and, if its cost is better than that of the current solution, becomes the new solution to continue the exploration. Otherwise, the solution is discarded and the following candidate is considered. To speed up the exploration, the number of candidates is limited to twice the number of hubs of the initial solution.

Neighborhood *Nremove* explores the option of removing hubs from a solution. First, a hub h is selected to be removed. Therefore, all nodes assigned to h need to be reassigned, if possible, to the remaining hubs. Each of these nodes is assigned to the hub with less residual capacity in which it fits. If all the nodes are finally reassigned without violating the capacity of the hubs, we have obtained a new solution that is evaluated and compared against the current best solution, and the procedure continues. Otherwise, the removal of another hub is considered.

Since the evaluation of adding or removing a hub to/from a solution can be very costly, especially the computation of the number of backbone edges needed after any move, we propose a new neighborhood, *Ncluster*, in which we consider that each hub together with its assigned terminals form a set (or *cluster*) in the network. This neighborhood, also used in Contreras and Díaz (2008) and Ilić et al. (2010), explores the change of hub within each cluster. In this way, the hub in a cluster changes its status to become a terminal and one of the terminals in this cluster is now the new hub of the cluster if its capacity allows to. The rest of the terminals in the cluster remain the same but assigned now to the new hub.

The proposed neighborhood *Ncluster* exhibits a good tradeoff between search power and computational cost. On one hand, it considers changing the hub in a solution, which is a major change that may lead to different types of solutions in the solutions' space. On the other hand, as it limits the exploration to changes within a cluster, there is no need to recalculate the number of links on the backbone edges to compute the value of the new solution.

Algorithm 1
Procedure *LScluster*.

Input: (s)
1 continue ← *TRUE*;
2 **while** continue is *TRUE* **do**
3 continue ← *FALSE*;
4 Define $\mathcal{U} = \{U_{h \in H}\}$
5 **foreach** $h \in H$: $|U_h| \geq 2$ **do**
6 Compute nodes' evaluation $\text{eval}(i) = C_{ii} + \sum_{j \in S \setminus \{i\}} C_{ji}$, $\forall i \in S$
7 Order $i \in S$ by non-decreasing value of eval(i)
8 **foreach** $i \in S$: $T(U_h) \leq Q_i^h$ **do**
9 $\bar{H} = H \setminus \{h\} \cup i$
10 $\bar{S} = S \setminus \{i\} \cup h$
11 $U_i = \{i\} \cup \bar{S}$
12 $\bar{\mathcal{U}} = \{U_{j \in \bar{H}}\}$
13 Compute cost solution value using $\bar{\mathcal{U}}$
14 **if** cost of solution using $\bar{\mathcal{U}}$ < cost of solution using $\mathcal{U}$ **then**
15 $H \leftarrow \bar{H}$
16 $i \leftrightarrow h$
17 continue ← *TRUE*
18 **end**
19 **end**
20 **end**
21 **end-while**
Output: s

The associated procedure, *LScluster*, works as follows. Let U_h be a cluster of nodes formed by a hub h and the set S of its assigned terminals, $U_h = \{h\} \cup S$. Let us denote by $T(U_h)$ the sum of the traffics associated with cluster U_h.

Once a cluster is selected in the main loop of Algorithm 1 (steps 5 to 20), we explore its terminals for a possible swapping with the current hub in the order given by their evaluation. Given an element i in $U_h = \{h\} \cup S$, its evaluation is given by:

$$\text{eval}(i) = C_{ii} + \sum_{j \in S \setminus \{i\}} C_{ji} \ .$$

We explore the nodes in the cluster and perform the first improvement move. After we have scanned all the clusters and eventually performed one or more moves, *LSalone* and *LSpairs* are applied. It is clear that if the hub of a cluster changes, some of the nodes in another cluster could be assigned to the new hub. As mentioned, *Ncluster* does not check this point. Therefore, we consider the *Nalone and Npairs* neighborhoods to check these reassignments and eventually perform further changes after a hub move. The application of *LSalone* and *LSpairs* may change the composition of some clusters, but the number of hubs remain unchanged. Hence, we then apply *LSremove* and *LSadd* in order to consider possible changes in the number of hubs. We perform further steps in which we first go over the clusters with the *Ncluster* neighborhood exploration, and then apply the *LSalone, LSpairs, LSremove*, and *LSadd*, as long as the solution improves. Our local search ends when no further improvement is possible. The whole procedure, *LSall,* is described in Algorithm 2.

4.3. Path relinking

Path relinking (PR) was suggested as an approach to integrate intensification and diversification strategies in the context of tabu search (Glover and Laguna, 1997). This approach generates new solutions by exploring trajectories that connect high-quality solutions by starting from one of these solutions, called *initiating solution*, and generating a path in the neighborhood space that leads toward the other solutions, called *guiding solutions*. This is accomplished

Algorithm 2
Procedure *LSall*.

```
Input: (s)
continue ← TRUE;
while continue is TRUE do
    continue ← FALSE;
    Apply LScluster
    Apply LSpairs
    if solution has improved after LSpairs then
        continue ← TRUE;
    end
    Apply LSalone
    if solution has improved after LSalone then
        continue ← TRUE;
    end
    Apply LSremove
    if solution has improved after LSremove then
        continue ← TRUE;
    end
    Apply LSadd
    if solution has improved after LSadd then
        continue ← TRUE;
    end
end-while
Output: s
```

by introducing in the initiating solutions attributes contained in the guiding ones. In this way, we generate a sequence of intermediate solutions that "connect" the initiating solution with the guiding one.

The term relinking reflects the fact that this method links again two or more solutions with a path in the search space. In the original tabu search design, two or more good solutions are recorded during the search. Since tabu search describes a trajectory, these solutions can be viewed as linked in the search space by the chain of moves which originated them. After the tabu search execution, we can consider to create a new trajectory, or chain of moves, to go from one of these high-quality solutions to another one. This new trajectory is directed considering the target solutions, instead of the objective function as in the initial application of tabu search. The method is therefore called path relinking because it creates two paths joining two solutions.

Laguna and Martí (1999) adapted PR in the context of GRASP as a form of intensification. The relinking, in the context of multi-start algorithms, consists of finding a path between two solutions generated with the constructive method and, eventually, improve the solution in the path with a local search. Therefore, the relinking concept has a different interpretation within GRASP, since the solutions are not originally linked by a sequence of moves. The authors, however, kept the original name of the methodology in spite of the fact that the two solutions are linked for the first time. Resende et al. (2010) explored different implementations to hybridize these two methodologies:

- *Greedy Path Relinking*. In this method the moves in the path from a solution to another are selected in a greedy fashion, according to the objective function value.
- *Greedy Randomized Path Relinking*. Here the method creates a candidate list with the good intermediate solutions and randomly selects among them.
- *Truncated Path Relinking*. In this application of PR the path between two solutions is not completed. It is applied, for example, in problems where good solutions are found close to the end points (original solutions) in the path.
- *Evolutionary Path Relinking*. This method iterates over the set of high-quality solutions, applying successively the relinking mechanism. It has many similarities with the scatter search methodology (Laguna and Martí, 2003).

In this paper we explore a kind of *Greedy Truncated Path Relinking* to our problem as a post-process method. Let X and Y be two solutions to the CSHLPMLC, and let H_X and H_Y be their associated sets of hubs. The path relinking procedure PR(X, Y) starts with X, and gradually transforms it into Y, by swapping out hubs in X with hubs in Y. The hubs in both solutions, H_{XY}, will remain as hubs in all the intermediate solutions generated in the path between them. Let H_{X-Y} be the hubs in X that are not hubs in Y. H_{Y-X} is defined equivalently. Let $\mathrm{PR}_0(X, Y)=X$ be the initiating solution in the path from X to Y. To obtain the first solution $\mathrm{PR}_1(X, Y)$ in this path, we remove a hub $i \in H_{X-Y}$ and replace it with a hub $j \in H_{Y-X}$, thus obtaining

$$H_{\mathrm{PR}_1(X,\ Y)} = H_{\mathrm{PR}_0(X,\ Y)} \setminus \{i\} \cup \{j\}.$$

In the PR variant implemented here, the selection of the nodes i, j is the one minimizing the objective function. In general, to obtain $\mathrm{PR}_{t+1}(X,\ Y)$ from $\mathrm{PR}_t(X, Y)$, we evaluate the different hubs $i \in H_{\mathrm{PR}_t(X,\ Y)-Y}$ to be removed and the hubs $j \in H_{\mathrm{Y}-\mathrm{PR}_t(X,\ Y)}$ to be selected. The move associated with the minimum cost option is performed.

At each intermediate solution in the path from X to Y, a restricted neighborhood is explored to generate the next solution in the path. The neighborhood is restricted because only moves removing hub $i \in H_{\mathrm{PR}_t(X,\ Y)-Y}$ and selecting hub $j \in H_{\mathrm{Y}-\mathrm{PR}_t(X,\ Y)}$ are allowed. As the procedure moves from one intermediate solution to the next, the cardinalities of sets $H_{\mathrm{PR}_t(X,\ Y)-Y}$ and $H_{\mathrm{Y}-\mathrm{PR}_t(X,\ Y)}$ decrease by one element. Consequently, as the procedure nears the guiding solution, there are fewer allowed moves to explore. This is why Resende et al. (2010) suggested that the search tends to be less effective in the final stages. In truncated path relinking, a new stopping criterion is used. Instead of continuing the search until the guiding solution is reached, only a limited number of steps, $\mathrm{PR}_{\mathrm{steps}}$, are allowed, abandoning the path before reaching the final solution. We consider $\mathrm{PR}_{\mathrm{steps}}$ as a search parameter, and explore the performance of the method with different values in the next section.

In our procedure, described in Algorithm 3, we first apply a constructive method (either *CM1, CM2* or *CM3*) and the local search procedure *LSall* to populate a set with high-quality solutions (elite set, ES). Our PR mechanism is designed to work as a post-process method. For each pair of solutions in ES, the cardinality of hubs and the objective values are compared in order to decide which of them will be the starting and the guiding solution. In the case where the cardinalities of their corresponding set of hubs are different, the solution with the highest number of hubs is chosen as starting solution X, and the path relinking is performed against towards the guiding solution Y. When all hubs in H_{Y-X} have been incorporated, there are one or more hubs that should be removed in order to finally reach Y. The procedure does not do this and stops at this step, so the whole path is not examined completely. This is the reason to consider our procedure a *Truncated* PR. If H_X and H_Y have the same cardinality, the solution with poorest objective value is chosen as the starting point and the path relinking is performed towards the better solution. The assignment of terminals to the hubs of any intermediate solution explored during the path relinking process is performed using the same strategy of the construction process.

5. Computational experiments

This section describes the computational experiments that we have performed to test the effectiveness and efficiency of the procedures discussed above. The algorithms have been implemented in C and run on an Intel Xeon E3-1270 at 3.40 GHz and 16GB of RAM computer running Windows 7–64 bits.

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 solution in the given set. Found by dividing the objective value of the actual solution to the objective value of the best solution.
- Best: Number of instances for which a procedure is able to find the best-known solution.
- CPU: Average computing time in seconds employed by the algorithm.

From previous publications, we have identified a benchmark with 170 instances from three well-known data sets:

- The CAB (Civil Aviation Board) data set, 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 network. From this original file, a total of 23 instances with 10, 15, 20 and 25 nodes have been used.
- The AP (Australian Post) data set, based on real data from the Australian postal service and 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 (Beasley, 1990). We have used 70 instances with n ranging from 10 to 200. 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, some flows from one node to itself are positive (i.e., $t_{ii} > 0$ for a given i).
- The USA423 data set was introduced in Peiró et al. (2014), 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 3 months period are considered. From the original benchmark, we have employed 77 instances in the experimentation below with n ranging from 20 to 250.

Each instance includes the three matrices (t_{ij}), (C_{ij}), (R_{kl}) and the capacity values Q^b for the backbone edges and Q^h for the hubs. To obtain a unique capacity for each hub, values for Q^h_k have been generated from the original values Q^h as random numbers in the range $[Q^h, 1.5Q^h]$. The entire set of instances is available at www.optsicom.es.

The experimental part is divided into two main blocks. The first block (scientific testing) is devoted to study the performance of the components of the algorithm, as well as to determine the best values for the key search parameters. They have been performed on the same training set of 36 instances used in Corberán et al. (2016). The second block of experiments (competitive testing) has the goal of comparing our procedure with the best published methods.

5.1. Scientific testing

A subset of 36 of the 170 instances derived from the CAB, AP and USA423 data sets were chosen for the preliminary experiments. The set consists of: three instances with $15 \le n \le 25$ from the CAB set, 21 instances with $10 \le n \le 195$ from the AP set, and 12 instances with $20 \le n \le 150$ from the USA423 set. These instances have been classified as small, medium, and large, with 12 instances in each group.

In our first experiment we have compared the combination of the different elements of the attractive(i) function for the hub selection in the constructive method CM3 (see Section 2). Specifically, we have considered the following alternative sets of parameter values:

- AllAlpha = $\{\alpha = 1;\ \beta = 0;\ \gamma = 0;\ \delta = 0\}$, only considers the opening costs
- AllBeta = $\{\alpha = 0;\ \beta = 1;\ \gamma = 0;\ \delta = 0\}$, only considers the assignment costs
- AllDelta = $\{\alpha = 0;\ \beta = 0;\ \gamma = 0;\ \delta = 1\}$, only considers the traffics
- AllGamma = $\{\alpha = 0;\ \beta = 0;\ \gamma = 1;\ \delta = 0\}$, only considers the frequency values
- Alt1 = $\{\alpha = 0.25;\ \beta = 0.25;\ \gamma = 0.25;\ \delta = 0.25\}$
- Alt2 = $\{\alpha = 0.70;\ \beta = 0.10;\ \gamma = 0.10;\ \delta = 0.10\}$
- Alt3 = $\{\alpha = 0.10;\ \beta = 0.70;\ \gamma = 0.10;\ \delta = 0.10\}$
- Alt4 = $\{\alpha = 0.10;\ \beta = 0.10;\ \gamma = 0.70;\ \delta = 0.10\}$
- Alt5 = $\{\alpha = 0.10;\ \beta = 0.10;\ \gamma = 0.10;\ \delta = 0.70\}$
- Alt6 = $\{\alpha = 0.35;\ \beta = 0.20;\ \gamma = 0.10;\ \delta = 0.35\}$
- Alt7 = $\{\alpha = 0.40;\ \beta = 0.10;\ \gamma = 0.10;\ \delta = 0.40\}$
- Alt8 = $\{\alpha = 0.40;\ \beta = 0.15;\ \gamma = 0.05;\ \delta = 0.40\}$

Note that the four first alternatives isolate the effect of each element of attractive(i), so they measure their contribution to the complete evaluation. Among the other alternatives, Alt1 is the one where the four parameters have the same weight. In Alt2, Alt3, Alt4, and Alt5, one of the parameters receives a larger weight value (0.70), while the other coefficients get a smaller weight (0.10). For Alt2 the highest weight is given to α, which is associated with the fixed cost of opening a hub. Similarly, Alt3 gives more weight to the cost of assigning terminals to hubs, while Alt4 penalizes the frequency of occurrence in previous iterations, and Alt5 gives more weight to the amount of traffic through the node. The Alt6, Alt7, and Alt8 alternatives try some other combinations of weights, where α and δ have higher values than the other parameters. This is based on the initial findings indicating that the opening costs and the traffics are the two most important elements when selecting hubs.

Table 2 shows the average percentage deviation from the best solution obtained with each alternative method when constructing 100 solutions for each instance. The CM3 method for assigning nodes to hubs is used for the data shown in the table. As expected, the worst alternatives are those based on only one element. AllDelta (the alternative with $\delta = 1$) is the best among these alternatives, which seems to indicate that the traffic through a node is a very important factor to choose it as a hub, and produce better values for Dev than AllBeta, which was the one used in Corberán et al. (2016), based on the assignment costs of terminals to hubs.

Results in Table 2 clearly show that alternatives Alt1 and Alt2 build solutions with least deviation. We have performed the Friedman test with all the alternatives and obtained a p-value < 0.0001, which confirms that there are significant differences between them.

To complement the analysis above, we represent in Fig. 1 the boxplots of the percentage deviations of each alternative. These diagrams represent the 36 deviation values obtained with each alternative on the instances of the training set. It can be seen that Alt1 and Alt2 have the highest concentration of lower relative deviations, which means that they produce the best solutions. A non-parametric Wilcoxon test performed to compare Alt1 and Alt2 gave a p-value of 0.936, which indicates that there is no significant difference between these two alternatives.

After analyzing the methods for selecting hubs, we have studied the procedures of assigning terminals to hubs (CM1, CM2, and CM3). Table 3 shows the results of this experiment on the training set. For each constructive method, the combinations of parameters that have been found to be best in the previous experiment have been tested, Alt1 and Alt2. Table 3 shows that the lowest average percentage deviation (6.7%) is obtained with CM2 and parameter

Table 2
Average percentage deviation from the best solution in set (Dev).

Size	AllAlpha	AllBeta	AllDelta	AllGamma	Alt1	Alt2	Alt3	Alt4	Alt5	Alt6	Alt7	Alt8
Small	51%	65%	38%	79%	9%	8%	28%	73%	12%	6%	7%	6%
Medium	46%	30%	24%	45%	8%	6%	7%	32%	11%	11%	12%	12%
Large	55%	40%	13%	42%	8%	11%	7%	31%	13%	14%	15%	17%
Summary	51%	45%	25%	55%	8%	8%	14%	45%	12%	10%	12%	12%

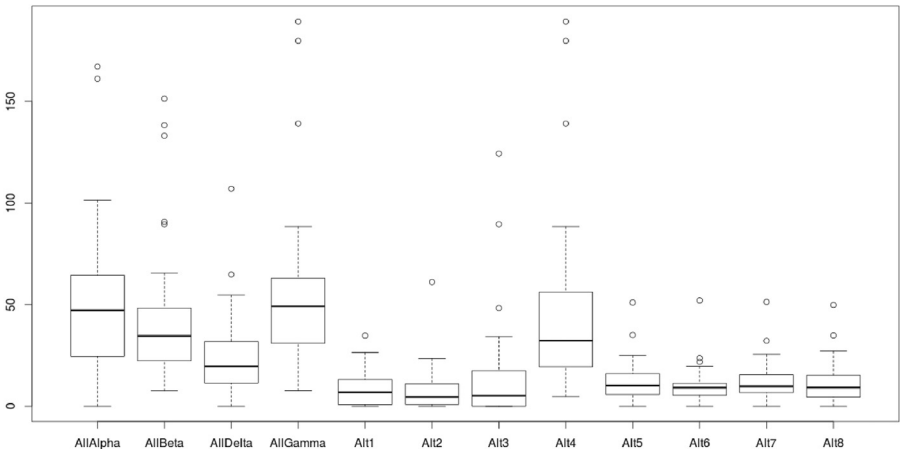

Fig. 1. Box plot of different alternatives.

Table 3
Average percentage deviation for constructive methods.

	CM1		CM2		CM3	
Size	Alt1	Alt2	Alt1	Alt2	Alt1	Alt2
Small	11.9%	8.7%	14.8%	8.2%	9.2%	8.4%
Medium	27.4%	22.4%	7.2%	3.6%	10.1%	7.4%
Large	51.2%	38.1%	5.2%	8.3%	8.6%	11.4%
Summary	30.2%	23.0%	9.1%	6.7%	9.3%	9.1%

Algorithm 3
Main search procedure.

Input: (G, maxIter, maxTime, α, β, γ, δ)
1 Obtain the value of p
2 ES$=\emptyset$
3 **while** iter $<$ maxIter **and** elapsedTime $<$ maxTime **do**
4 s $\leftarrow$ Constructive Method;
5 **if** s $\notin$ ES **then**
6 s' $\leftarrow$ *LSall(s)*
7 ES $\leftarrow$ ES $\cup$ {s'}
8 iter++
9 **end**
10 **end-while**
11 s* $\leftarrow$ PathRelinking(ES)
Output: s*

combination Alt2. On the other hand, CM2 with Alt1 or CM3 with Alt2 also present a quite good deviation, while the CM1 strategy gives much poorer results on both parameter combinations. A non-parametric Wilcoxon test performed to compare CM2 using Alt1 and Alt2 showed that there is no significant difference between these two configurations. We have chosen CM2 with Alt2 as the configuration for our constructive method and will use it in the following experiments.

In the third experiment, we have studied the contribution of the local search phase of the algorithm applied to the solutions obtained with the constructive method CM2 (and Alt2). In particular, we have considered the following local search variants, as described in Section 3:

- *LSalone+LSpairs* – The method combines these two local searches (Corberán et al., 2016).
- *LScluster* - The method only applies the local search *LScluster*.
- *LSreduce+LSadd* - The method combines these two local searches.
- *LSall* – All methods are applied as described in Algorithm 3.
- *LSAllOnce* - All methods are applied only once in the order given in Algorithm 3.

Table 4 reports the average percentage reduction from the constructed solution value obtained with these local search variants by running 50 iterations (construction+local search) on the instances in the training set. It shows that an important reduction is achieved if assignment-based local searches *LSalone+LSpairs* are combined. In particular, they exhibit an improvement of 9.4%. We can also observe from this table the good performance of the new local searches, *LScluster* and *LSreduce+LSadd*. Specifically, applying *LScluster* only, improves by 7.9% on average the constructed solutions, while applying *LSreduce+LSadd* improves by 12.3% on average. An advantage of *LScluster* is that it is very fast, since the amount of traffic between the clusters is constant and the number of edges used in the backbone network (W_{kl}) does not change. The LSAllOnce column shows the result when applying only once all the methods. This process improves the solutions obtained considerably, reaching a reduction of 18.5% on average. Finally, the *LSall* column shows the average results of the improvements obtained after applying the loop procedure described in Algorithm 2. Clearly, it is the best of the four alternatives.

Fig. 2 shows the search profile of the methods described above. Specifically, we represent the objective value of the best-known solution on a 130-nodes instance when applying different solution methods for 100 global iterations. We have added a sixth method, labeled "Construction", which represents the best value obtained when applying the constructive method without any local search. The other five lines represent the best solution value obtained with

Table 4
Local search percentage reduction from construction.

Size	LSalone+ LSpairs	LScluster	LSreduce+LSadd	LSallOnce	LSall
Small	−5.2%	−4.5%	−7.2%	−13.7%	−19.4%
Medium	−7.5%	−6.3%	−9.5%	−16.2%	−26.2%
Large	−15.6%	−12.8%	−20.4%	−25.5%	−33.1%
Summary	−9.4%	−7.9%	−12.3%	−18.5%	−26.2%

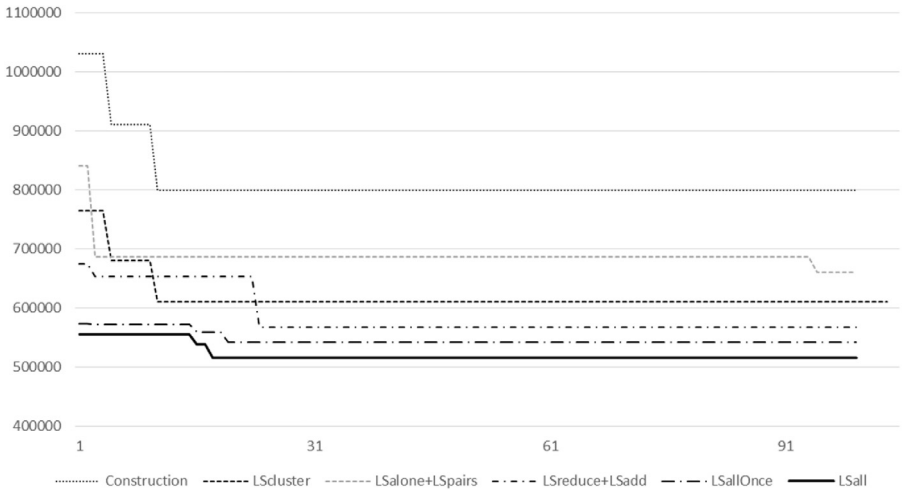

Fig. 2. Search profile for the different local search methods.

Table 5
Path relinking contribution.

Size	Deviation from best	CPU
Small	0.00%	+26.5%
Medium	−0.52%	+24.3%
Large	−1.24%	+22.6%
Summary	−0.59%	+24.5%

Table 6
Average percentage reduction in truncating path relinking.

PR_{steps}						
Size	1	2	3	4	5	6
Small	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Medium	0.23%	−0.48%	−0.52%	−0.52%	−0.52%	−0.52%
Large	−0.26%	−0.68%	−0.73%	−1.01%	−1.24%	−1.24%
Summary	−0.16%	−0.39%	−0.42%	−0.51%	−0.59%	−0.59%
CPU time	+3.5%	+5.46%	+6.4%	+6.8%	+7.1%	+8.8%

the application of the constructive method followed by each of the five local searches described above. The results in this diagram are in line with those reported in Table 4, where *LSall* is able to obtain the best results from the very beginning of the search process, and continues being the leader in the entire search horizon considered.

In the next experiment, we undertake to assess the performance of the PR post-process. To do this, all the instances in the training set have been executed for 50 iterations to create the solutions in the ES that will be used during PR. The path relinking procedure has been able to improve the previously obtained results in 10 out of the 36 instances in the training set, with an average improvement of 0.59%. Table 5 shows the results for the different groups of instances, together with the increase in the CPU time for including PR. It can be seen that the increase in the computing time decreases with the size of the instances and, in our opinion, the extra time consumed by PR is worth it, especially in the large size instances, where a 1.24% of improvement is obtained. In this experiment, we have also tested the strategy called two-sided path relinking (Festa and Resende, 2011), in which the best direction to create the path is determined. In particular, we have tested the reversal of the direction considered so far in the PR process, moving from the best towards the poorer solution, and it has not produced any significant difference in the results.

In our final experiment in the scientific testing, we explore the variant known as Truncated Path Relinking (Resende et al., 2010). As described in Section 4, this strategy considers the early termination of the PR without reaching the guiding solution, thus performing only a limited number of steps, PR_{steps}. Table 6 shows the average percentage improvement (reduction with respect to the initiating solution) achieved at each step of the path. As in previous tables, we divide the results according to the size of the instances and summarize them in an additional row. We include a final row with the average increase in the CPU time due to the application of PR.

In line with Table 5, Table 6 clearly shows the overall contribution of PR. Moreover, this table shows that we achieve in two steps the best possible result in the small instances and, therefore, it is a good strategy to truncate the path at an early stage. Medium and large instances require more steps to achieve the best results in the path. At some point (step 5) the improvement stagnates and further steps do not produce better results.

Table 7
Comparison with best previous method.

		Dev (%)		CPU		# Best	
Size	*# inst*	SO	AMP_50	SO	AMP_50	SO	AMP_50
Small	45	3.00	0.01	3.0	1.0	5	41
Medium	36	7.06	0.00	37.8	37.2	0	36
Large	27	13.66	0.00	310.4	359.6	0	27
Ex-large	37	15.55	0.00	1606.3	772.5	0	37
Huge	25	14.80	0.00	4916.3	2866.4	0	25
Summary	170	10.02	0.00	1130.7	654.9	5	166

Table 8
Comparison with CPLEX.

		CPLEX			AMP_50		
Instance	|V|	Value	CPU	Dev	Value	CPU	Dev
A1	10	89,766	13.2	0.00%	89,766	0.0	0.00%
A2	10	176,716	20.5	0.00%	176,716	0.0	0.00%
B	20	399,198	36,000	0.00%	410,563	0.7	2.85%
C	25	229,121	36,000	7.47%	213,202	0.9	0.00%
D	40	–	36,000	–	1,652,323	20.7	0.00%

5.2. Competitive testing

Once we have established the key-search parameters and explored the different variants of our algorithm, we compare it with the best previous method. Our procedure, denoted AMP_50, corresponds to a termination criteria of 50 executions of the constructive procedure producing the same number of solutions for the Elite Set. A full path relinking is then performed between the solutions in the Elite Set.

Table 7 reports the results of the comparison between the proposed procedure and the Strategic Oscillation method, SO, by Corberán et al. (2016). Since the results provided in this last paper were obtained on instances where the capacity of the hubs are equal, we have considered the full set of 170 instances with the original values Q^h and report the average percentage deviation (Dev), the running time in seconds (CPU), and the number of best solutions found with each method (#Best). These instances have been classified as small ($10 \leq n \leq 50$), medium ($55 \leq n \leq 100$), large ($110 \leq n \leq 150$), extra-large ($155 \leq n \leq 200$), and huge ($205 \leq n \leq 250$). Detailed results on each instance are shown in Tables 11–15 in the Appendix.

Table 7 clearly shows the superiority of the proposed procedure with respect to the previous SO method. Specifically, AMP_50 is able to obtain 166 best solutions out of the 170 instances considered, and shows an average computing time of 654.9 seconds. This compares favorably with the five best solutions obtained with SO, which uses 1130.7 seconds on average. We have performed a Wilcoxon test and the resulting p-value < 0.0001 confirms these conclusions.

As far as we know, it has not been published another method for solving the CSHLPMLC with different hub capacities Q_k^h. Thus, in order to compare our procedure for such instances, we attempted to obtain solutions by solving the formulation described in Section 3 by using CPLEX 12.6. Only two instances of small size could be solved to optimality with CPLEX within a time limit of 10 hours of CPU. The results obtained with CPLEX and with our algorithm on a small set of instances are shown in Table 8. In particular, the table reports the optimal values or the best upper bound found by CPLEX within the time limit and by AMP_50, the percentage deviation with respect to the optimal value or to the best upper bound found by any of the two procedures, and the computing time used by each method. Note that AMP_50 was able to find the optimal solution in the two instances CPLEX does, and that it obtained good solutions for the other three instances in very short computing times, while CPLEX was unable even to find a feasible solution for the last instance after 10 hours of CPU.

The results obtained with our algorithm on the whole set of instances are summarized in Table 9, while the detailed results on each instance can be found on the web site http://www.optsicom.es. The results obtained with two simpler versions of the algorithm (represented as “20” and “50”), consisting of executing 20 and 50 iterations of the main procedure but without applying the Path Relinking part, are also reported for comparison reasons. In particular, Table 9 shows the average percentage deviation with respect to the best solution found (Dev), the running time in seconds (CPU), and the number of best solutions found with each method (#Best) for each of the three methods.

Although, obviously, the best results are obtained with the full version of the procedure, the results reported in Table 9 clearly show the PR contribution to the good behavior of the whole algorithm. In particular, PR improves the results obtained after 50 iterations a 2.63%, 2.36%, and 4.02% on the large, extra-large and huge instances, respectively, at the expense of reasonable additional computing times.

6. An extension to the problem

As described in the literature review section, Aykin (1995a) considered a model based on the context of air passenger transportation where non-stop flights are offered between some non-hub cities. In this model, the set of routes for which direct shipping is permitted is predetermined. In other cases, the model itself determines the interest of establishing a non-stop connection between a pair of terminals. However, as far as we know, this option has never been studied in a problem with capacities for the hubs and a modular structure of the backbone network as the one in our problem, possibly because of its difficulty. Here, we approach this option in a simple way. First, the pairs of nodes for which direct transportation makes sense in terms of the problem data are identified. Then, we compare the solution obtained when solving the whole problem against the one obtained when these non-stop services have been established and the remaining traffics are transported through the hubs network.

Specifically, we first identify the pairs for which a non-stop service will be established as those with a high traffic between them and with a relatively low traffic with the rest of the nodes. To do this, we consider the external traffic $T(i)$ for each node i:

$$T(i) = \sum_{j \in V \setminus \{i\}} t_{ij} + \sum_{j \in V \setminus \{i\}} t_{ji} \ .$$

It is clear that if t_{ij} is low, a direct transportation between these two nodes makes no sense since direct transportation does not take advantage of the economies of scale and results in a more expensive service than the one provided by a hub-based

Table 9
Computational results.

		Dev			CPU			# Best		
Size	# inst	20	50	AMP_50	20	50	AMP_50	20	50	AMP_50
Small	45	1.14%	0.47%	0.00%	0.4	0.8	1.1	27	36	45
Medium	36	3.68%	0.68%	0.00%	11.1	27.2	35.6	7	27	36
Large	27	4.75%	2.63%	0.00%	114.1	288.8	396.4	2	10	27
Extra-large	37	4.62%	2.36%	0.00%	320.8	805.4	831.4	8	20	37
Huge	25	8.51%	4.02%	0.00%	1032.1	2573.1	2757.2	2	5	25
Summary	170	4.09%	1.79%	0.00%	242.2	605.5	657.2	46	98	170

Table 10
Comparison of solutions with and without direct services.

Instance	Z	Z'	Difference in #hubs	Ratio (%)	Direct services
60_500_60_69_60_1_50ap	547,029	505,472	0	4.90	33
70_500_60_69_60_1_50ap	628,311	622,608	0	1.03	33
80_800_69_50_80_1_60usa	607,666	514,502	−1	4.09	81
90_600_60_69_60_1_69usa	658,553	583,634	0	2.71	64
100_500_60_69_60_1_50usa	696,062	679,125	0	2.55	49

transportation. However, a relatively high traffic between two locations permits a non-expensive direct transportation. On the other hand, if these two nodes, i and j, exhibit a high traffic to the rest of the nodes ($T(i)$ and $T(j)$ respectively), they are good candidates to be hubs, and direct transportation will appear anyway.

We define a threshold t_p for the minimum pairwise traffic and another threshold t_t for the external traffic in such a way that if a pair of nodes (i, j) verifies that the traffic between them is larger than t_p and their external traffics are lower than t_t, then we consider that the t_{ij} traffic travels using a non-stop service. We call D to this set of pairs:

$$D = \left\{(i, j) : \ t_{ij} > t_p, \ t(i) \leq t_t, \ t(j) \leq t_t \ \right\}.$$

We empirically adjust the values of t_p and t_t to obtain a reasonable size for the set D. In particular, we estimate the number of hubs, and from it the average traffic through a hub. We use that number as a reference to compute t_p. From that value, we identify those nodes with a large or low traffic, and we adjust t_t from the traffic among the remaining nodes.

We propose a simple approach to evaluate the convenience of providing direct transportation for the pairs in D. Suppose that Q^d represents the traffic capacity of a direct link and R_d its cost. As with the backbone edges, if the traffic in that edge exceeds its capacity, a number of links $\frac{t_{ij}}{Q^d}$ to transport the whole traffic is considered. This has a cost of $R_d \frac{t_{ij}}{Q^d}$. We will assume that the traffic between node pairs in D is sent through direct links. Then, we can solve the problem of routing the remaining traffic through the hub subnetwork, but taking into account that none of the nodes in D can be hub. The solution with value Z'obtained with this method can be then compared against the solution of the standard problem, with value Z. The decision maker can analyze the values of Q^d and R_d for which $\sum_{(i,j)\in D} R_d \frac{t_{ij}}{Q^d} + Z' < Z$ (if any) in order to find out whether this extension can lead to improved outcomes.

Table 10 shows the results obtained for five medium size instances. The first number in the instance name denotes the number of nodes in the instance. Columns "Z" and "Z'" report the value of the best solution obtained with AMP_50 when no direct services are allowed and once the traffics sent through direct services have been removed from the problem, respectively. The comparison between $Z - Z'$ and the costs of the direct services associated with the pairs of terminals in D would help to decide if these direct services are interesting or not. The fourth column in Table 10 gives the difference in the number of hubs obtained for the two solutions. Finally, last two columns report the ratio between the traffics sent through direct services and the total traffic (in percentage) and the number of direct services selected, respectively.

7. Conclusions

Hub location problems are difficult combinatorial optimization problems that have recently received a lot of attention. In this paper, we have studied a capacitated modular hub location problem that has been modeled in the literature as a mixed integer nonlinear program. To solve the problem, we have proposed an algorithm using adaptive memory. We have tested the effects of a variety of search strategies, as those combining different constructive methods and neighborhood structures within a multi-start memory based framework. We have also explored a Path Relinking postprocess to obtain improved outcomes. Our experiments show that the adaptive memory features are capable of searching the solution space economically and effectively, outperforming existing approaches.

Acknowledgments

The authors want to thank three anonymous referees for their careful reading of the manuscript and for their many comments and suggestions that have contributed significantly to improve the article. This work was supported by the Spanish Ministerio de Economía y Competividad and Fondo Europeo de Desarrollo Regional (FEDER) (projects TIN-2015-65460-C02-01, MTM-2015-68097, ECO2013-47129-C4-3-R, ECO2016-76567-C4-2-R, and the PhD. grant BES-2013-064245), by the Generalitat Valenciana (project Prometeo 2013/049), and the Regional Government of Castilla y León and FEDER (projects BU329U14 and BU062U16). This support is gratefully acknowledged.

Appendix

Table 11
SO and AMP on small size instances.

	SO			AMP		
Instance	Value	Dev	CPU	Value	Dev	CPU
10_600_89_60_40_1_60_CAB	237,701	1.48%	0.22	234,243	0.00%	0.01
10_700_50_60_8_1_60_AP	273,801	9.82%	0.23	249,318	0.00%	0.00
10_700_50_60_8_1_60_CAB	253,255	3.86%	0.22	243,854	0.00%	0.01
10_700_69_40_8_1_50_CAB	257,691	4.49%	0.22	246,610	0.00%	0.01
10_800_60_60_6_1_69_AP	245,513	5.78%	0.23	232,104	0.00%	0.01
10_800_60_60_6_1_69_CAB	244,417	2.63%	0.58	238,144	0.00%	0.00
10_800_60_80_8_1_80_CAB	247,078	2.58%	0.22	240,872	0.00%	0.00
15_500_50_60_40_1_60_CAB	289,339	1.16%	0.36	286,027	0.00%	0.03
15_600_80_89_6_1_69_CAB	293,075	1.12%	0.44	289,839	0.00%	0.03
15_600_80_89_8_1_60_CAB	308,168	2.64%	0.41	300,251	0.00%	0.03
15_700_89_60_40_1_60_CAB	289,953	1.39%	0.39	285,965	0.00%	0.03
15_800_50_60_40_1_60_CAB	290,026	1.67%	0.39	285,274	0.00%	0.03
15_900_80_89_40_1_60_CAB	290,134	1.66%	0.41	285,390	0.00%	0.03
20_700_50_60_8_1_60_AP	406,266	4.28%	0.98	389,589	0.00%	0.08
20_700_50_60_8_1_60_CAB	176,142	1.20%	1.03	174,048	0.00%	0.09
20_700_50_60_8_1_60_USA	127,058	7.69%	0.92	117,986	0.00%	0.02
20_700_69_40_8_1_50_AP	408,538	0.06%	1.03	408,293	0.00%	0.10
20_700_69_40_8_1_50_CAB	187,305	5.07%	0.84	178,275	0.00%	0.09
20_800_60_60_6_1_69_CAB	164,351	0.00%	0.73	164,568	0.13%	0.10
20_800_60_80_8_1_80_AP	363,247	0.00%	1.19	363,247	0.00%	0.09
20_800_60_80_8_1_80_CAB	170,069	1.11%	0.75	168,204	0.00%	0.11
20_800_60_80_8_1_80_USA	121,071	4.94%	0.87	115,370	0.00%	0.02
20_900_80_89_40_1_80_CAB	151,342	0.01%	0.8	151,324	0.00%	0.10
25_600_80_60_6_1_40_CAB	210,578	1.02%	1.7	208,444	0.00%	0.26
25_600_80_89_6_1_69_CAB	192,213	0.00%	1.39	192,900	0.36%	0.27
25_600_80_89_8_1_60_CAB	214,944	4.15%	1.64	206,386	0.00%	0.25
25_650_69_69_6_1_50_CAB	210,278	4.10%	1.84	201,993	0.00%	0.24
25_650_69_69_6_1_70_CAB	162,862	0.91%	2.14	161,398	0.00%	0.32
25_800_89_60_40_1_80_CAB	179,893	0.00%	1.86	179,948	0.03%	0.28
25_900_80_89_40_1_80_CAB	181,207	0.34%	1.61	180,600	0.00%	0.30
30_600_80_89_8_1_60_AP	383,188	9.08%	3.46	351,284	0.00%	0.19
30_700_69_40_8_1_50_USA	211,640	0.12%	2.93	211,382	0.00%	0.42
35_600_80_89_8_1_60_AP	448,064	2.13%	3.98	438,722	0.00%	1.07
35_600_80_89_8_1_60_USA	206,472	1.98%	4.4	202,473	0.00%	0.60
35_700_80_50_8_1_69_AP	436,550	0.83%	4.23	432,959	0.00%	1.06
40_600_80_89_8_1_60_USA	288,503	4.34%	6.19	276,510	0.00%	2.92
40_700_80_50_8_1_69_AP	478,470	4.25%	6.43	458,982	0.00%	1.86
40_700_80_50_8_1_69_USA	282,629	4.44%	6.01	270,601	0.00%	2.83
45_600_80_89_8_1_60_AP	521,958	1.23%	8.44	515,630	0.00%	2.69
45_700_69_40_8_1_50_AP	589,832	0.61%	8.63	586,276	0.00%	2.70
45_700_69_40_8_1_50_USA	345,335	14.42%	6.83	301,809	0.00%	4.23
50_600_80_89_8_1_60_USA	347,675	9.33%	12.34	318,016	0.00%	6.24
50_700_69_40_8_1_50_AP	598,636	0.00%	11.5	599,205	0.10%	4.93
50_700_80_50_8_1_69_AP	562,786	3.24%	11.23	545,111	0.00%	4.61
50_700_80_50_8_1_69_USA	328,853	3.85%	10.87	316,654	0.00%	6.64

Table 12
SO and AMP on medium size instances.

	SO			AMP		
Instance	Value	Dev	CPU	Value	Dev	CPU
55_500_60_69_60_1_50_AP	551,996	4.46%	12.28	528,434	0.00%	7.32
55_500_60_69_60_1_50_USA	356,419	13.77%	11.51	313,285	0.00%	8.42
55_800_69_50_80_1_60_AP	627,685	0.65%	13.65	623,632	0.00%	6.71
55_800_69_50_80_1_60_USA	356,039	9.82%	10.31	324,208	0.00%	8.80
60_500_60_69_60_1_50_AP	597,551	7.41%	14.23	556,315	0.00%	9.56
60_600_60_69_60_1_69_USA	324,245	4.30%	14.34	310,881	0.00%	12.01
60_800_69_50_80_1_60_AP	664,374	3.18%	13.32	643,873	0.00%	10.31
60_800_69_50_80_1_60_USA	375,981	11.37%	14.96	337,606	0.00%	11.04
65_500_60_69_60_1_50_AP	594,968	0.55%	19.49	591,705	0.00%	13.08

(*continued on next page*)

Table 12 (*continued*)

	SO			AMP		
Instance	Value	Dev	CPU	Value	Dev	CPU
65_600_60_69_60_1_69_AP	596,768	4.29%	19.06	572,237	0.00%	12.69
65_600_60_69_60_1_69_USA	366,133	10.95%	21.09	330,006	0.00%	17.42
65_800_69_50_80_1_60_USA	405,756	10.67%	21.08	366,629	0.00%	16.35
70_500_60_69_60_1_50_AP	664,745	8.29%	24.56	613,848	0.00%	16.81
70_600_60_69_60_1_69_AP	615,656	2.60%	22.25	600,082	0.00%	18.08
70_600_60_69_60_1_69_USA	383,492	10.87%	29.03	345,887	0.00%	21.84
70_800_69_50_80_1_60_AP	769,108	3.93%	27.27	740,056	0.00%	18.55
75_500_60_69_60_1_50_USA	552,080	5.35%	29.14	524,031	0.00%	31.68
75_600_60_69_60_1_69_AP	665,111	2.24%	35.23	650,554	0.00%	22.03
75_600_60_69_60_1_69_USA	519,474	9.16%	31.44	475,886	0.00%	30.33
75_800_69_50_80_1_60_AP	828,245	5.22%	32.87	787,150	0.00%	21.92
80_500_60_69_60_1_50_AP	722,632	4.37%	40.25	692,408	0.00%	29.76
80_500_60_69_60_1_50_USA	632,672	7.78%	35.09	586,997	0.00%	45.36
80_800_69_50_80_1_60_AP	837,210	2.68%	39.53	815,360	0.00%	31.50
80_800_69_50_80_1_60_USA	673,561	9.46%	36.57	615,377	0.00%	46.32
85_500_60_69_60_1_50_AP	762,920	0.25%	62.71	760,980	0.00%	38.37
85_500_60_69_60_1_50_USA	777,825	12.60%	53.17	690,770	0.00%	74.37
85_800_69_50_80_1_60_AP	903,683	3.67%	60.42	871,710	0.00%	38.25
90_500_60_69_60_1_50_USA	804,494	15.66%	53.98	695,573	0.00%	80.35
90_600_60_69_60_1_69_AP	759,377	3.16%	70.56	736,086	0.00%	48.27
90_600_60_69_60_1_69_USA	708,086	10.21%	54.34	642,502	0.00%	72.99
90_800_69_50_80_1_60_AP	966,669	6.94%	70.72	903,969	0.00%	51.35
95_500_60_69_60_1_50_AP	905,808	11.02%	75.52	815,931	0.00%	56.49
95_500_60_69_60_1_50_USA	780,646	12.90%	64.88	691,424	0.00%	103.14
95_600_60_69_60_1_69_AP	826,451	3.49%	82.84	798,566	0.00%	56.01
95_600_60_69_60_1_69_USA	714,260	9.71%	65.63	651,071	0.00%	102.89
100_500_60_69_60_1_50_USA	792,405	11.03%	77.82	713,698	0.00%	150.43

Table 13
SO and AMP on large size instances.

	SO			AMP		
Instance	Value	Dev	CPU	Value	Dev	CPU
110_500_60_69_60_1_50_AP	996,975	4.46%	150.65	954,372	0.00%	108.51
110_600_60_69_60_1_69_AP	934,826	3.18%	142.15	906,009	0.00%	118.71
110_700_80_60_89_1_60_USA	1,122,255	22.57%	105.91	915,625	0.00%	265.08
110_800_69_50_80_1_60_USA	1,039,753	15.07%	117.52	903,544	0.00%	228.90
110_900_69_50_89_1_60_USA	1,105,052	17.71%	105.43	938,808	0.00%	258.70
120_500_60_69_60_1_50_AP	1,123,004	8.90%	206.13	1,031,226	0.00%	164.92
120_500_60_69_60_1_50_USA	1,117,652	17.78%	151.31	948,970	0.00%	328.15
120_700_80_60_89_1_60_USA	1,219,108	20.62%	163.57	1,010,735	0.00%	389.88
120_900_69_50_89_1_60_USA	1,175,745	10.15%	146.52	1,067,450	0.00%	415.56
125_500_60_69_60_1_50_AP	1,145,428	5.55%	192.06	1,085,187	0.00%	192.29
125_800_69_50_80_1_60_AP	1,371,476	10.16%	208.84	1,244,985	0.00%	207.28
130_600_60_69_60_1_69_AP	1,158,759	8.44%	291.91	1,068,564	0.00%	262.15
130_600_60_69_60_1_69_USA	1,040,651	9.62%	180.23	949,318	0.00%	446.82
130_800_69_50_80_1_60_USA	1,264,063	19.04%	227.08	1,061,879	0.00%	481.97
135_600_60_69_60_1_69_AP	1,229,722	9.88%	335.92	1,119,159	0.00%	316.02
135_800_69_50_80_1_60_USA	1,405,752	31.84%	251.49	1,066,218	0.00%	672.35
140_500_60_69_60_1_50_AP	1,412,597	12.76%	357.75	1,252,721	0.00%	370.81
140_700_80_60_89_1_60_USA	1,490,007	24.24%	286.76	1,199,339	0.00%	335.99
140_800_69_50_80_1_60_AP	1,570,932	12.14%	377.44	1,400,828	0.00%	395.23
140_900_69_50_89_1_60_USA	1,444,192	19.02%	275.01	1,213,396	0.00%	350.83
145_600_80_69_60_1_50_AP	1,462,902	8.92%	352.66	1,343,050	0.00%	452.93
145_600_80_69_60_1_50_USA	596,573	21.05%	710.75	492,827	0.00%	430.37
145_800_69_50_80_1_60_AP	1,603,039	8.27%	357.12	1,480,588	0.00%	456.14
145_800_69_50_80_1_60_USA	654,681	11.35%	647.73	587,954	0.00%	412.61
150_1000_69_60_80_1_69_USA	612,031	10.60%	837.07	553,365	0.00%	499.78
150_800_69_50_80_1_60_AP	1,734,477	10.90%	424.82	1,563,951	0.00%	646.73
150_900_69_60_80_1_89_USA	606,912	14.48%	775.96	530,125	0.00%	501.58

Table 14
SO and AMP on extra-large size instances.

	SO			AMP		
Instance	Value	Dev	CPU	Value	Dev	CPU
155_1000_69_60_80_1_69_USA	650,419	15.51%	919.46	563,085	0.00%	551.45
155_500_60_69_60_1_50_AP	1,427,231	18.26%	635.5	1,206,831	0.00%	557.88
155_800_69_50_80_1_60_AP	1,602,365	16.88%	604.55	1,371,004	0.00%	558.30
160_600_60_69_60_1_69_AP	685,604	15.20%	950.8	595,141	0.00%	373.73
160_700_80_60_89_1_60_USA	704,872	31.85%	1010.66	534,598	0.00%	758.45
160_800_69_50_80_1_60_AP	826,772	11.81%	1042.87	739,445	0.00%	383.80
160_900_80_50_60_1_69_AP	775,178	5.65%	1008.39	733,757	0.00%	398.09
160_900_89_50_60_1_69_USA	530,376	9.64%	1048.41	483,740	0.00%	663.18
165_1000_69_60_80_1_69_USA	665,409	12.88%	1302.26	589,474	0.00%	734.50
165_800_69_50_80_1_60_AP	876,244	20.72%	1109.94	725,874	0.00%	405.87
165_800_69_50_80_1_60_USA	704,389	9.39%	1302.59	643,904	0.00%	742.05
170_500_60_69_60_1_50_AP	714,822	29.82%	1152.12	550,621	0.00%	504.30
170_600_89_60_69_1_80_USA	551,600	11.36%	1304.46	495,349	0.00%	772.88
170_700_80_60_89_1_60_USA	609,581	8.55%	1361.95	561,585	0.00%	968.87
170_900_69_60_80_1_89_USA	613,150	5.99%	1403.26	578,471	0.00%	735.98
170_900_80_50_60_1_69_AP	754,262	5.00%	1251.03	718,370	0.00%	562.75
175_500_60_69_60_1_50_AP	649,148	8.51%	1427.86	598,241	0.00%	554.70
175_600_60_69_60_1_69_AP	648,587	5.99%	1664.5	611,906	0.00%	591.87
175_800_69_50_80_1_60_USA	716,778	9.73%	1625.83	653,231	0.00%	915.04
175_900_69_60_80_1_89_USA	602,257	7.49%	1434.86	560,301	0.00%	919.48
180_1000_69_60_80_1_69_USA	740,453	15.80%	1970.72	639,398	0.00%	938.22
180_600_60_69_60_1_69_AP	746,918	21.72%	1804.62	613,648	0.00%	673.60
180_600_89_60_69_1_80_USA	558,479	12.62%	1739.88	495,902	0.00%	1122.06
180_800_89_69_89_1_89_AP	797,621	7.82%	1921.67	739,750	0.00%	695.90
185_500_60_69_60_1_50_AP	858,506	39.68%	1872.1	614,628	0.00%	691.73
185_600_80_89_89_1_89_AP	700,282	7.51%	2029.67	651,382	0.00%	672.15
185_600_89_60_69_1_80_USA	530,783	8.74%	1880	488,142	0.00%	1202.84
185_800_69_50_80_1_60_AP	890,070	12.48%	1971.64	791,304	0.00%	688.01
185_900_69_60_80_1_89_USA	621,650	11.92%	2158.96	555,466	0.00%	1004.08
190_600_60_69_60_1_69_AP	773,840	22.08%	2200.58	633,895	0.00%	792.25
190_600_80_89_89_1_89_AP	800,719	22.29%	2288.98	654,744	0.00%	812.07
190_600_89_60_69_1_80_USA	522,508	7.69%	2190.87	485,190	0.00%	1515.13
190_700_89_69_89_1_89_USA	575,933	12.61%	2177.05	511,453	0.00%	1490.72
190_800_69_50_80_1_60_AP	1,031,614	29.15%	2298.88	798,755	0.00%	800.36
195_600_60_69_60_1_69_AP	774,165	19.89%	2487.11	645,712	0.00%	941.72
195_800_69_50_80_1_60_AP	1,107,739	34.67%	2431.53	822,538	0.00%	943.84
195_900_89_89_89_1_69_AP	1,093,874	28.39%	2449.22	852,008	0.00%	946.40

Table 15
SO and AMP on huge size instances.

	SO			AMP		
Instance	Value	Dev	CPU	Value	Dev	CPU
200_500_60_69_60_1_50_AP	661,633	4.18%	2686.72	635,108	0.00%	1045.68
200_700_80_60_89_1_60_USA	743,444	21.81%	2976.42	610,310	0.00%	1927.87
200_700_89_69_89_1_89_USA	608,272	12.78%	2985.89	539,340	0.00%	1852.44
200_800_69_50_80_1_60_AP	763,358	3.33%	2731.83	738,765	0.00%	1068.13
200_800_89_69_89_1_89_USA	645,449	16.29%	2995.71	555,040	0.00%	1895.10
205_800_69_50_80_1_60_USA	800,686	15.46%	3086.01	693,477	0.00%	1681.57
205_900_69_60_80_1_89_USA	670,343	10.23%	3570.73	608,148	0.00%	1705.52
210_800_69_50_80_1_60_USA	811,601	11.14%	3832.57	730,273	0.00%	1790.77
210_900_69_60_80_1_89_USA	699,431	17.84%	3743.74	593,529	0.00%	1743.47
215_800_69_50_80_1_60_USA	897,348	18.32%	3927.14	758,429	0.00%	2085.28
215_900_69_60_80_1_89_USA	736,413	18.89%	3653.79	619,427	0.00%	1928.71
220_800_69_50_80_1_60_USA	872,528	20.56%	4275.09	723,734	0.00%	2473.09
220_900_69_60_80_1_89_USA	725,843	12.03%	4485.24	647,915	0.00%	2328.85
225_800_69_50_80_1_60_USA	978,703	12.63%	4993.8	868,963	0.00%	2775.11
225_900_69_60_80_1_89_USA	812,905	10.95%	4668.05	732,658	0.00%	2681.13
230_800_69_50_80_1_60_USA	994,501	15.83%	6096.23	858,609	0.00%	3213.37
230_900_69_60_80_1_89_USA	859,790	18.35%	5934.23	726,502	0.00%	3065.26
235_800_69_50_80_1_60_USA	1,067,897	10.43%	6223.85	967,074	0.00%	3838.36
235_900_69_60_80_1_89_USA	967,918	21.74%	6017.28	795,102	0.00%	3698.01
240_800_69_50_80_1_60_USA	1,106,592	12.75%	6079.38	981,444	0.00%	4447.55
240_900_69_60_80_1_89_USA	914,468	14.51%	6852.66	798,589	0.00%	4358.11
245_800_69_50_80_1_60_USA	1,209,802	24.60%	7913.75	970,942	0.00%	5020.98
245_900_69_60_80_1_89_USA	913,041	10.91%	7197.28	823,255	0.00%	4956.09
250_800_69_50_80_1_60_USA	1,132,659	13.50%	7800.79	997,938	0.00%	4978.50
250_900_69_60_80_1_89_USA	991,167	20.94%	8180.05	819,535	0.00%	5100.97

References

Alumur, S., Kara, B.Y., 2008. Network hub location problems: the state of the art. Eur. J. Oper. Res. 190, 1–21.

Alumur, S., Kara, B.Y., Karasan, O.E., 2009. The design of single allocation incomplete hub networks. Transp. Res. Part B 43, 936–951.

Aykin, T., 1995a. The hub location and routing problem. Eur. J. Oper. Res. 83, 200–219.

Aykin, T., 1995b. Networking policies for hub-and-spoke systems with application to the air transportation system. Transp. Sci. 29, 201–221.

Beasley, J.E., 1990. OR-Library: distributing test problems by electronic mail. J. Oper. Res. Soc. 41, 1069–1072.

Bryan, D.L., O'Kelly, M.E., 1999. Hub-and-spoke networks in air transportation: an analytical review. J. Reg. Sci. 39, 275–295.

Calik, H., Alumur, S., Kara, B.Y., Karasan, O.E., 2009. A tabu-search based heuristic for the hub covering problem over incomplete hub networks. Comput. Oper. Res. 36, 3088–3096.

Campbell, J.F., 2010. Designing hub networks with connected and isolated hubs. In: Proc. 43rd Hawaii International Conference Systems Society HICSS-43. Koloa, Kauai.

Campbell, J.F., Ernst, A.T., Krishnamoorthy, M., 2005. Hub arc location problems: part I-Introduction and results. Manage. Sci. 51, 1540–1555.

Campbell, J.F., O'Kelly, M.E., 2012. Twenty-five years of hub location research. Transp. Sci. 46, 153–169.

Carello, G., Della Croce, F., Ghirardi, M., Tadel, R., 2004. Solving the hub location problem in telecommunications network design: a local search approach. Networks 44, 94–105.

Contreras, I., 2015. Hub location problems. In: Laporte, G., Nickel, S., Saldanha da Gama, F. (Eds.), Location Science. Springer.

Contreras, I., Cordeau, J.F., Laporte, G., 2011. The dynamic uncapacitated hub location problem. Transp. Sci. 45, 18–32.

Contreras, I., Cordeau, J.F., Laporte, G., 2012. Exact solution of large-scale hub location problems with multiple capacity levels. Transp. Sci. 46, 439–459.

Contreras, I., Díaz, J.A., 2008. Scatter search for the single source capacitated facility location problem. Ann. Oper. Res. 157, 73–89.

Corberán, Á., Peiró, J., Campos, V., Glover, F., Martí, R., 2016. Strategic oscillation for the capacitated hub location problem with modular links. J. Heuristics 22, 221–244.

Correia, I., Nickel, S., Saldanha da Gama, F., 2010. Single-assignment hub location problems with multiple capacity levels. Transp. Res. Part B 44, 1047–1066.

Elhedhli, S., Wu, H., 2010. A Lagrangean heuristic for hub-and-spoke systems design with capacity selection and congestion. INFORMS J. Comput. 22, 282–296.

Ernst, A.T., Krishnamoorthy, M., 1996. Efficient algorithms for the uncapacitated single allocation p-hub median problem. Location Sci. 4, 139–154.

Farahani, R.Z., Hekmatfar, M., Arabani, A.B., Nikbakhsh, E., 2013. Hub location problems: a review of models, classification, solution techniques and applications. Comput. Ind. Eng. 64, 1096–1109.

Festa, P., Resende, M.G.C., 2011. GRASP: basic components and enhancements. Telecommun. Syst. 46, 253–271.

Glover, F., Laguna, M. (1997). Tabu Search, Kluwer, Norwell, MA.

Ilić, A., Urošević, D., Brimberg, J., Mladenović, N., 2010. A general variable neighborhood search for solving the uncapacitated single allocation p-hub median problem. Eur. J. Oper. Res. 206, 289–300.

Klincewicz, J.G., 1998. Hub location in backbone/tributary network design: a review. Location Sci. 6, 307–335.

Laguna, M., Martí, R., 1999. GRASP and path relinking for 2-layer straight line crossing minimization. INFORMS J. Comput. 11, 44–52.

Laguna, M., Martí, R., 2003. Scatter Search: Methodology and Implementations in C. Kluwer Academic Publishers, New York.

O'Kelly, M.E., 1987. A quadratic integer program for the location of interacting hub facilities. Eur. J. Oper. Res. 32, 393–404.

Peiró, J., Corberán, Á., Laguna, M., Martí, R., 2016. Models and Solution Methods for the Uncapacitated r-Allocation p-Hub Equitable Center Problem. University of Valencia Technical Report.

Peiró, J., Corberán, Á., Martí, R., 2014. GRASP for the uncapacitated r-allocation p-hub median problem. Comput. Oper. Res. 43, 50–60.

Resende, M.G.C., Gallego, M., Duarte, A., Martí, R., 2010. GRASP and path relinking for the Max-Min diversity problem. Comput. Oper. Res. 37, 498–508.

Sasaki, M., Fukushima, M., 2003. On the hub-and-spoke model with arc capacity constraints. J. Oper. Res. Soc. Japan 46, 409–428.

Tanash, M., Contreras, I., Vidyarthi, N., 2017. An exact algorithm for the modular hub location problem with single assignments. Comput. Oper. Res. 85, 32–44.

Yaman, H., 2008. Star p-hub median problem with modular arc capacities. Comput. Oper. Res. 35, 3009–3319.

Yaman, H., 2011. Allocation strategies in hub networks. Eur. J. Oper. Res. 211, 442–451.

Yaman, H., Carello, G., 2005. Solving the hub location problem with modular link capacities. Comput. Oper. Res. 32, 3227–3245.

Yoon, M.-G., Current, J., 2008. The hub location and network design problem with fixed and variable arc costs: formulation and dual-based solution heuristic. J. Oper. Res. Soc. 59, 80–89.