Strategic Oscillation for the capacitated hub location problem with modular links

Ángel Corberán[(a)], Juanjo Peiró[(a)], Fred Glover[(b)], and Rafael Martí[(a)]

[(a)]Departament d'Estadística i Investigació Operativa, Universitat de València, Spain
[(b)]OptTek Systems, Boulder (CO), USA

March 21, 2014

Abstract

The capacitated single assignment hub location problem with modular link capacities is a variant of the classical hub location problem in which the cost of using edges is not linear but stepwise, and the hubs are restricted in terms of transit capacity rather than in the incoming traffic. This problem was introduced by Yaman and Carello in [18] and treated by a branch-and-cut and a tabu search metaheuristic. We propose a metaheuristic algorithm based on Strategic Oscillation, which was originally introduced in the context of tabu search. Our method incorporates several designs for constructive and destructive algorithms, together with associated local search procedures, to balance diversification and intensification for an efficient search. Computational results on 118 instances show that, in contrast to exact methods that can only solve small instances optimally, our metaheuristic is able to find high-quality solutions on larger instances in short computing times. In addition, our new method which joins tabu search strategies with strategic oscillation outperforms the previous tabu search implementation.

Keywords and phrases: hub location problem, modular link costs, tabu search, strategic oscillation, iterated greedy.

1 Introduction

Discrete facility location problems related to the design of transportation networks are one of the most extensively studied problems in combinatorial

optimization due to the variety and importance of their fields of application. There are several variants of discrete facility location problems. Some of the most important are: the p-median problem, the p-center problem, the maximal covering location problem, and the hub location problem, to name just a few. In all of them, one is given a network $G = (V, E)$ with a set of demand nodes V, and a set of edges E. For each pair of nodes i and $j \in V$, there is a traffic t_{ij} (of goods, people, etc.) that needs to be transported. Depending on each variant of the problem, additional specific characteristics can be found, such as a fixed cost of opening a facility at a potential location, or limitation on its capacity. We refer the reader to the excellent surveys [3] and [12] for a detailed description of the most prominent facility location problems.

In this paper we study the Hub Location Problem (HLP), in which a set of facility locations is selected from a given set of potential locations V. The goal is to identify an optimal subset of facilities in order to minimize a transportation cost function while satisfying a set of constraints. For the sake of simplicity, we call these facilities *distribution centers* or simply *hub nodes*. The rest of the nodes in the network are called *terminal nodes*. It is assumed that direct transportation between terminals is not possible and, therefore, the traffic t_{ij} travels along a path $i \to h_i \to h_j \to j$, where i and j are assigned to hubs h_i and h_j, respectively. The reader can find in [1] and [6] a complete description of HLP variants.

Among the family of Hub Location Problems, we focus on a specific variant known as the *Capacitated Single assignment Hub Location Problem with Modular Link Capacities* (CSHLPMLC). This problem was formulated as a quadratic mixed integer programming problem by Yaman and Carello [18]. These authors also proposed a branch-and-cut algorithm to solve optimally the problem together with a metaheuristic to obtain good initial solutions. As proved in [17], the CSHLPMLC is $\mathcal{NP}$-hard. In what follows we summarize the characteristics of this problem:

- G is a connected network. V is the set of demand nodes. For each pair of nodes i and j, there is a traffic t_{ij} to be transported through the edges E.
- All the nodes in the network are demand points (i.e. terminal nodes), as well as potential hub locations (i.e. hub nodes).
- Each node i is assigned to only one hub h_i.
- Hubs can be located at any node i of the network, with an associated installation cost C_{ii}.
- The number of hubs used is not fixed a priori. A solution can have any number of hubs (from 1 to $|V|$).
- All hubs have the same capacity Q^h, which limits the total traffic transiting through them.

- Edges between hubs have capacity Q^b. If the traffic between two hubs exceeds this amount, additional edges with Q^b capacity are added.

The CSHLPMLC consists of selecting a subset of nodes to be hubs and assigning the rest of the nodes to them in such a way the transportation cost is minimized while satisfying the capacity constraints.

Many heuristics and metaheuristics have been proposed to solve different variants of hub location problems, including GRASP [14], VNS [9], Tabu Search [18], and several complex hybrid techniques. In this paper we present a simple, easily adaptable and powerful algorithm, based on the Iterated Greedy–Strategic Oscillation (SO) methodology. The purpose of this paper is to investigate the SO proposal, which alternates between constructive and destructive phases as a basis for creating a competitive method for this hub location problem.

The structure of the remainder of this paper is as follows. We begin by summarizing the previous work by Yaman and Carello [18], which as far as we know is the only published paper devoting attention to this specific problem. In particular, the problem definition, the notation used, as well as the formulation proposed in [18] are described in Section 2, while the heuristic method in [18] is described in Section 3. We then describe in Section 4 the elements of our SO method, including the memory structures employed in our implementation. Finally, Section 5 presents a comparison between methods, optimal results, as well as several experiments to determine the values of the key-search parameters. Computational outcomes on 150 instances show that, while only small instances can be optimally solved with exact methods, our metaheuristic is able to find high-quality solutions on larger instances in short computing times, and outperforms the previous tabu search implementation.

2 A non-linear programming formulation

Let $G = (V, E)$ be a network with node set $V = \{1, \ldots, n\}$ and edge set E. For any pair of nodes $i, j \in V$, t_{ij} denotes the traffic to be transported from i to j, where $t_{ii} = 0$ for any node i.

Each node i is either a terminal node or a hub node (*terminal* and *hub* for short). A terminal can only be assigned to a single hub. A hub is assigned to itself. The hubs and the edges among them define a complete subgraph. Let $H \subseteq V$ be the subset of hubs. Opening a hub at node i has a fixed installation cost C_{ii}. Each hub i has a capacity Q^h limiting the total amount of traffic transiting through i.

There are two types of edges between nodes: edges of the first type are used to connect terminals with hubs, and we call them *access edges* in reference to the access to the network they provide. Let m_i be the number of access edges needed to route the incoming and outgoing traffic at node

i, and let Q^a be the maximum capacity an access edge allows to transfer through it. So,

$$m_i = max\left\{\left\lceil\frac{\sum_{j\in V} t_{ij}}{Q^a}\right\rceil, \left\lceil\frac{\sum_{j\in V} t_{ji}}{Q^a}\right\rceil\right\}.$$

The cost of installing m_i edges between terminal i and hub k is denoted by C_{ik}. Edges of the second type are used to transfer traffics between hubs, and we call them *backbone edges*. E_B denotes the set of backbone edges, defined as $E_B = \{\{k,l\} : k,l \in H, k < l\}$. Each backbone edge has a maximum traffic capacity of Q^b (in each direction). We define A as the set of arcs associated with the edges in E_B, $A = \{(k,l) : k,l \in H, k \neq l\}$.

If nodes k and l are hubs, the amount of traffic on arc (k,l), denoted as z_{kl}, is the sum of the traffics that have to be transported from nodes assigned to k to nodes assigned to l. The capacity Q^b of a given edge $\{k,l\}$ cannot be less than the maximum of traffic on its corresponding arcs (k,l) and (l,k), and the cost of installing the edge is denoted by R_{kl}. This edge capacity Q^b can, for example, be understood as the capacity of an airplane. If $2 \times Q^b \geq z_{kl} > Q^b$, two copies of the edge (two airplanes) are needed, even if the second one transports less than Q^b passengers, and a fixed cost R_{kl} for each airplane has to be paid. This reflects the non-linear nature of the costs R_{kl}. This modular link characteristic makes this model much more realistic that the linear cost version.

If nodes k and l are hubs, the amount of traffic on arc (k,l), denoted as z_{kl}, is the sum of the traffics that have to be transported from nodes assigned to k to nodes assigned to l. The capacity Q^b of a given edge $\{k,l\}$ cannot be less than the maximum of traffic on its corresponding arcs (k,l) and (l,k), and the cost of installing the edge is denoted by R_{kl}. This edge capacity Q^b can be understood as the capacity of an airplane, for instance. If $z_{kl} > Q^b$, several copies of the edge (like considering several airplanes) are needed. This reflects the non-linear nature of the costs R_{kl}. We pay a fixed cost R_{kl} for an airplane which transports a maximum of Q^b passengers between hubs k and l. If two airplanes are needed, $2 \times R_{kl}$ would be the cost to pay, even if one of the airplanes transports less than Q^b passengers. This modular link characteristic makes this model much more realistic that the linear cost version.

Three different costs have to be considered in this problem: The opening costs of the hubs (C_{kk}), the assignment costs of terminals to hubs (C_{ik}), and the traffic costs between hubs. While cost C_{ik} corresponds to that of transporting all the traffic involving i through hub k, R_{kl} represents the cost of using only one edge $\{k,l\}$. This last cost has to be multiplied by the number of copies needed of the edge $\{k,l\}$. So, we face to two types of decisions, the *binary decision* of assigning a terminal to a hub and the *integer decision* associated with how many copies of the edges among hubs will be used. Figure 1 shows a diagram which represents the hubs as shaded

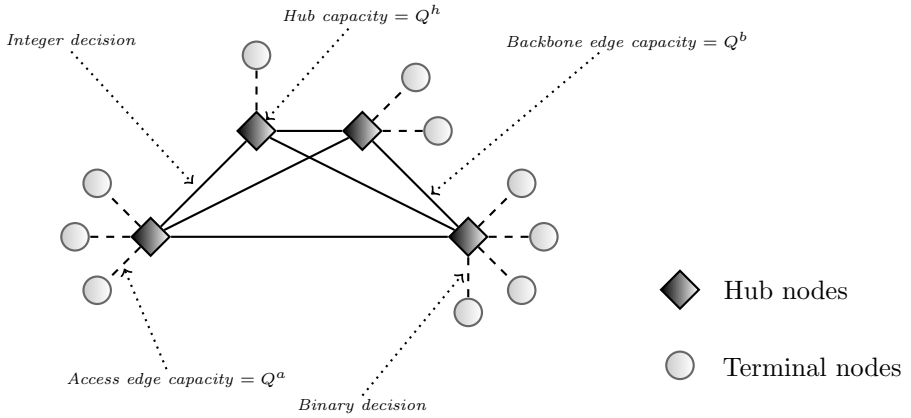

Figure 1: Different costs in the CSHLPMLC

squares, the terminals as circles, the assignments of terminals to hubs in dashed lines, and the different capacities involved.

The following variables are defined in [18] in order to provide the following mathematical programming model:

- 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 a backbone arc $(k,l) \in A$ and w_{kl} is the number of copies of the edge $\{k,l\} \in E_B$.

Then, the capacitated single assignment hub location problem with modular link capacities can be formulated as follows ([18]):

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

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

$$x_{ik} \leq x_{kk} \quad \forall i \in V, \quad \forall k \in V \setminus \{i\} \tag{3}$$

$$\sum_{i\in V}\sum_{j\in V}(t_{ij}+t_{ji})x_{ik} - \sum_{i\in V}\sum_{j\in V} t_{ij}x_{ik}x_{jk} \leq Q^h x_{kk} \quad \forall k \in V \tag{4}$$

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

$$Q^b w_{kl} \geq z_{kl} \quad \forall \{k,l\} \in E \tag{6}$$

$$Q^b w_{kl} \geq z_{lk} \quad \forall \{k,l\} \in E \tag{7}$$

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

$$w_{kl} \in \mathbb{Z}_+ \quad \forall \{k,l\} \in E \tag{9}$$

Constraints (2) imply that each node has to be assigned to only one hub. Constraints (3) force node k to be a hub if node i is assigned to it. Constraints (4) say 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 maximum capacity Q^h. Constraints (5) add up the traffics through a given backbone arc (k,l). Finally, constraints (6) and (7) fix the number of copies needed of each backbone edge.

We have tested this formulation on a small set of instances to check if our metaheuristic would be able to catch the optimal solution obtained using exact methods on this formulation. Results of this comparison can be found in Subsection 5.4.

3 Previous methods

A metaheuristic and a branch-and-bound algorithm are proposed for the CSHLPMLC in [18] based on the idea of finding a set of hub nodes that represents, in a sense, the best hubs that can be selected heuristically. This set is called the *concentration set*. The resulting reduced problem, where hubs can be chosen only among the nodes of the concentration set, is called the *concentrated problem*, and is solved by a branch-and-cut method. The metaheuristic is based on two methodologies: a tabu search (TS) for the hub location subproblem and a local search for assigning terminals to hubs.

As described in [8], tabu search is a metaheuristic that guides a local search procedure to explore the solution space beyond local optimality.

The main steps of the proposed metaheuristic algorithm [18] are:

1. To construct an initial feasible solution by means of a greedy algorithm that starts with an empty set of hubs and adds hubs one by one until a feasible solution is reached. The hubs are added trying to keep the assignment cost as low as possible.

2. To apply a tabu search to the location subproblem. The neighborhood is generated by applying three different moves: a new hub is opened (adding move), a hub is removed from the set of hubs (removing move), and a hub is removed and a new one is opened in another terminal (swapping move).

3. To apply a local search to the 20 best solutions found in the previous step for the assignment subproblem. The neighborhood is generated

by applying two different moves: a terminal is moved from a hub to another, and two terminals swap their hubs.

A comparison between the above tabu search, PrevTS, and the one we propose in Section 4 is presented in Section 5.

4 Strategic Oscillation

The structure of a neighborhood in tabu search goes beyond that used in local search by embracing the types of moves used in constructive and destructive processes (where the foundations for such moves are accordingly called constructive neighborhoods and destructive neighborhoods). Following basic tabu search principles, memory structures can be implemented within a constructive process to favor (or avoid) the inclusion of certain elements in the solution previously identified as attractive (or unattractive). Such expanded uses of the neighborhood concept reinforce a fundamental perspective of TS, which is to define neighborhoods in dynamic ways that can include serial or simultaneous consideration of multiple types of moves.

This dynamic neighborhood approach applies not only to the types of neighborhoods used in "solution improvement methods" (sometimes called "local search methods") but also applies to constructive neighborhoods used in building solutions from scratch - as opposed to transitioning from one solution to another. Although it is commonplace in the metaheuristic literature to restrict the word "neighborhood" to refer solely to transitions between solutions as embodied in improvement methods, constructive neighborhoods have been proposed as an important ingredient of search processes from the very beginning of the TS methodology, as documented by Glover and Laguna [8]. Nevertheless, tabu search methods for exploiting constructive neighborhoods have rarely been applied in computational studies.

Our tabu search approach for the CSHLPMLC is additionally based on the strategic oscillation methodology [7, 8]. Strategic oscillation (SO) is closely linked to the origins of tabu search, and operates by orienting moves in relation to a critical level, as identified by a stage of construction. In particular, we consider a constructive/destructive type of strategic oscillation, where constructive steps "add" elements and destructive steps "drop" elements.

More recently, constructive and destructive neighborhoods have been applied within a simplified and effective method known as Iterated Greedy (IG) [10], which generates a sequence of solutions by iterating over a greedy constructive heuristic which, like strategic oscillation, uses two main phases: *destruction* and *construction*. IG is a memory-less metaheuristic easy to implement that has exhibited state-of-the-art performance in some settings (see for example [5, 11, 15]). We sketch the form of this method because its

simplicity gives a convenient foundation for embedding it in a more complete strategic oscillation approach.

Briefly described, the *destruction* phase of IG removes selected solution components from a previously constructed complete candidate solution. The *construction* procedure then applies a greedy constructive heuristic to reconstruct a complete candidate solution. Once a candidate solution has been completed, an acceptance criterion decides whether the newly constructed solution will replace the incumbent solution. The algorithm iterates over these steps until a stopping criterion is met. Optionally, a local search can be applied after construction phases for improved outcomes. Algorithm 1 shows an outline of the IG method.

As shown in Algorithm 1, the IG method starts from a complete initial solution S (Initialise()) and then iterates through a main loop which first generates a partial candidate solution S_p by removing a fixed number of elements (n_h hubs in our case) from the complete candidate solution S (Destruction-phase(S, n_h)) and next reconstructs a complete solution S_c starting with S_p (Construction-phase(S_p)). In the local search phase (Local-Search-phase(S_c)), an improvement procedure is performed in order to find better solutions near the reconstructed solution. Before continuing with the next loop, an acceptance criterion (AcceptanceCriterion(S, S_i)) decides whether the solution returned by the local search procedure, S_i, becomes the new incumbent solution. The process iterates through these phases until a computation limit t_{max} is reached. The best solution, S_{best}, generated during the iterative process is kept to provide the final result.

```
   Input: G, t_max, n_h
   Output: S_best
 1 S ← Initialise();
 2 S_best ← S;
 3 while t_max is not reached do
 4 |   S_p ← Destruction-phase(S, n_h);
 5 |   S_c ← Construction-phase(S_p);
 6 |   S_i ← Local-Search-phase(S_c);
 7 |   if S_i is better than S_best then
 8 |   |   S_best ← S_i;
 9 |   end
10 |   if AcceptanceCriterion(S, S_i) then
11 |   |   S ← S_i;
12 |   end
13 end
```

Algorithm 1: Iterated Greedy pseudocode

We have considered two different acceptance criteria in the scheme shown

in Algorithm 1:

- *'Replace if better' acceptance criterion.* The new solution is accepted only if it provides a better objective function value [19].
- *'Random walk' acceptance criterion.* An IG algorithm using the above criterion may lead to stagnation situations of the search due to insufficient diversification [16]. At the opposite extreme is the random walk acceptance criterion, which always applies the destruction phase to the most recently visited solution, irrespective of its objective function value. This criterion clearly favors diversification over intensification, because it promotes a stochastic search in the space of local optima.

4.1 Strategic Oscillation for the CSHLPMLC

The metaheuristic we propose for solving the capacitated single assignment hub location problem with modular link capacities embodies the iterated greedy approach within the strategic oscillation method by including simple recency and frequency memory strategies derived from tabu search, as proposed in the original SO methodology.

Finding an initial feasible solution

We define a feasible solution S to be an assignment of the terminal nodes to hubs in such a way traffics from every origin to every destination can be transferred using these hubs.

Let h be a candidate location node for a hub. For any node j that can be assigned to h, with cost C_{jh}, we have to consider that all the traffic from and to j has to be routed through h. In order to evaluate the attractiveness of h as a hub, $g(h)$, we consider first the nodes with the lowest C_{jh} value, adding as many nodes as the capacity permits. Let us assume, without loss of generality, that they are $j_1, \ldots, j_{u(h)}$. In mathematical terms,

$$g(h) = C_{hh} + \frac{\sum_{s=1}^{u(h)} C_{j_s h}}{u(h)},$$

where the first term in the expression corresponds to the installation cost.

The hub h_1 with lowest evaluation $g(h)$ is selected as a hub and the terminals used in the computation of $g(h_1)$ are assigned to it. Then, the attractiveness function is computed again for the remaining nodes without considering the terminals already assigned to h_1. This iterative procedure is applied until we have selected enough hubs to assign all the nodes in the network.

At this stage, a feasible solution $S = (H, \mathbf{A})$ is available, where H is the set of hubs, and $\mathbf{A}$ the set of assignments. We represent by (i, h) the

assignment of terminal i to hub h. The set $\mathbf{A}$ contains the n pairs reflecting these assignments. Since hub h is assigned to itself, $\mathbf{A}$ contains the pair (h, h). Moreover, $\mathbf{A}^h$ denotes the set of nodes assigned to h, i.e. $\mathbf{A}^h = \{i \in V : (i, h) \in \mathbf{A}\}$.

Evaluation of a feasible solution

Different nature costs are involved when evaluating S:

- The fixed cost of opening/installing hubs.
- The fixed cost of assigning each terminal to its associated hub.
- The cost of installing the backbone edges needed to transfer the traffic between hubs. This cost is computed as follows. Given two hubs k and l, the total amount of traffic on the arc $(k, l) \in A$ is obtained as

$$z_{kl} = \sum_{i \in A^k} \sum_{j \in A^l} t_{ij}.$$

  Then, the maximum amount of traffic that will travel through the edge $\{k, l\}$ is $T_{kl} = max\{z_{kl}, z_{lk}\}$. Since each edge $\{k, l\}$ has a maximum capacity Q^b, it will be necessary to replicate this edge $w_{kl} = \left\lceil \frac{T_{kl}}{Q^b} \right\rceil$ times. Given that R_{kl} is the cost of installing a copy of edge $\{k, l\}$, the total cost is $\sum_{\{k,l\} \in E_B} R_{kl} w_{kl}$.

Destruct and construct to improve the hubs selection

Since we apply this constructive procedure several times, we have modified $g(h)$ to incorporate information about previous constructions. Specifically, we define the frequency $freq(h)$ as the number of times (solutions) in which node h has been selected as a hub. Then, to discourage the selection of those nodes already selected as hubs, we consider, instead of $g(h)$, a new evaluation function $g'(h)$:

$$g'(h) = \frac{g(h)}{max_g} + \gamma \frac{freq(h)}{max_{freq}} \ ,$$

where γ is a search parameter that weights the second term in $g'(h)$.

Following the strategy described in the Iterated Greedy approach, once a solution S is constructed (using $g'(h)$ to guide the process), we partially deconstruct it by removing some of its elements, obtaining S_p. In our context, it means that we deselect some of its hub nodes. Note that the terminals that were assigned to these unselected hubs are now unassigned. We have called these unassigned nodes, including the deselected hubs, *orphan* nodes, and denoted as $\mathbf{O}$ the set of all of them, i.e. $\mathbf{O} = \bigcup_{h \in H_S \setminus H_{S_p}} \mathbf{A}^h$.

The greediness of the construction process results, in some cases, in hubs with more capacity than the used by their assigned nodes. In fact, we have empirically found that some of the orphan nodes could eventually be assigned to some of the remaining hubs in S_p. Hence, the first step in our reconstruction process is to check if the remaining capacity of the hubs belonging to S_p permits to assign any orphan node to them (thus removing it from set $\mathbf{O}$). Afterwards, we select as a new hub the node $h^* \in \mathbf{O}$ with the lowest $g'(h)$ value. We remove h^* and its terminal nodes from $\mathbf{O}$, update $freq(h^*)$ by adding one unit, and iteratively perform more construction steps until all the nodes in $\mathbf{O}$ have been assigned or selected as hubs, obtaining a new feasible solution S_c. This destructive-constructive process is repeated until a stopping criterion is met, which in our algorithm is simply a maximum number of iterations defined as η.

In our SO method we remove hubs from the solution at random. A second search parameter δ indicates the percentage of removed hubs. This randomization implies a strong diversification component in the search which balances the intensification of our greedy constructive algorithm. Given the second term in the g' value with the frequency record, the removed hubs are unlikely to be selected in the next iterations. In Section 5 we study the performance of the proposed algorithm, denoted as SO1, for different values of γ, η, and δ.

In SO1 the evaluation function $g^{'}(h)$ penalizes the choice as hubs of those nodes that were already selected in previous iterations. A second SO algorithm, SO2, where we use the original evaluation function $g(h)$ and a classic tabu list (the customary type of tabu search recency memory embodied in a tabu list) has also been considered. These two designs are usual ways of implementing a recency memory structure in a constructive method. We believe that their comparison is therefore of interest to disclose their relative merit (see Section 5 for our conclusions on this point).

In SO2 we construct an initial solution according to the $g(h)$ evaluation. Then, we randomly remove a percentage δ of the hubs. The removed hubs are added to a tabu list and become tabu for a given number of iterations (constructions) denoted by τ (tabu tenure). The same construction method, guided here by $g(h)$, is applied to reconstruct the solution by selecting new non-tabu hubs. Thus, SO1 applies a strategic oscillation based on frequencies, while SO2 is based on a short term tabu list.

In order to speed up the process and search for new solutions, we have included a slight modification in the assignment of orphan nodes in the SO2 procedure. In particular, when we check if an orphan node can be assigned to a non removed hub, we follow the order given by the tabu list. We first try to assign the tabu nodes since they cannot be hubs in this iteration. Moreover, within the tabu nodes, we try first those that recently gain the tabu status (those that we strongly forbid to be hubs). As is usually done in tabu search implementations, we include an aspiration criterion to override

the tabu status by permitting, in this case, a tabu node to be a hub if the capacity constraints would compel this in the assignment process.

Improvements on the assignments

When a new feasible solution $S_c = (H, \mathbf{A})$ is obtained, an improvement procedure on the assignment of terminals to hubs is applied. Two neighborhoods, N_{pairs} and N_{alone}, are proposed to improve S_c:

N_{pairs} implements a classical exchange in which two terminals i and j, assigned to hubs k and l, swap their corresponding hubs. This exchange can be done when nodes i and j are not hubs, they are assigned to different hubs, and when the new assignments do not violate the capacity constraints. To compute the cost of this exchange:

- Only the assignment costs of i and j are updated: $C_{ik} + C_{jl}$ is subtracted from the total assignment cost and the new assignment costs $C_{il} + C_{jk}$ are added.
- The cost of the backbone edges also need to be recomputed. Given a backbone edge $\{p, q\}$, the new traffic T'_{pq} traversing $\{p, q\}$ is $T'_{pq} = max\{z'_{pq}, z'_{qp}\}$, where the values of z'_{pq} and z'_{qp} are computed as follows:
  - If $p \neq k, l$ and $q \neq k, l$, the traffics through backbone edge $\{p, q\}$ do not change, $z'_{pq} = z_{pq}$ and $z'_{qp} = z_{qp}$.
  - If exactly one end node of $\{p, q\}$ is k or l, for instance $p = k$ $(q \neq l)$,

$$z'_{pq} = z_{pq} - \sum_{s \in A^q} t_{is} + \sum_{s \in A^q} t_{js} \quad \text{and} \quad z'_{qp} = z_{qp} - \sum_{s \in A^q} t_{si} + \sum_{s \in A^q} t_{sj}.$$

  - If $\{p, q\} = \{k, l\}$,

$$z'_{pq} = z_{pq} - \sum_{s \in A^l} t_{is} - \sum_{s \in A^k \setminus \{i\}} t_{sj} + \sum_{s \in A^l \setminus \{j\} \cup \{i\}} t_{js} + \sum_{s \in A^k \setminus \{i\}} t_{si}$$

    and

$$z'_{qp} = z_{qp} - \sum_{s \in A^k} t_{js} - \sum_{s \in A^l \setminus \{j\}} t_{si} + \sum_{s \in A^k \setminus \{i\} \cup \{j\}} t_{is} + \sum_{s \in A^l \setminus \{j\}} t_{sj}.$$

  From the new traffics T'_{pq} traversing the backbone edges $\{p, q\} \in E_B$, we compute the number of copies that are needed of each edge and its cost, $\left\lceil \frac{T'_{pq}}{Q^b} \right\rceil \times R_{pq}$. Only if the total cost of the new assignment is lower than the cost of the current solution (first improvement strategy), the exchange is done.

Sometimes N_{pairs} proves to be a poor neighborhood as it is quite restrictive. For this reason we also propose N_{alone}, which implements another classical movement. In N_{alone}, a terminal i, previously assigned to hub k, is now assigned to another hub l. To compute the cost of the new assignment:

- Only the assignment cost of i needs to be updated from the total assignment cost: C_{ik} is subtracted from the total cost and C_{il} is added.
- The cost of the backbone edges also need to be recomputed. Let $i \in \mathbf{A}^k$. We try to assign i to another hub l in order to get a cost reduction. As in N_{pairs}, given a backbone edge $\{p, q\}$, the values of z'_{pq} and z'_{qp} to obtain T'_{pq} are computed as follows:
  - If $p \neq k, l$ and $q \neq k, l$, the traffics through backbone edge $\{p, q\}$ do not change, $z'_{pq} = z_{pq}$ and $z'_{qp} = z_{qp}$.
  - If $p = k$ or $q = k$, since hub k looses its assigned node i (suppose $p = k$),
$$z'_{pq} = z_{pq} - \sum_{s \in A^q} t_{is} \quad \text{and} \quad z'_{qp} = z_{qp} - \sum_{s \in A^q} t_{si}.$$
  - If $p = l$ or $q = l$, since node i is assigned now to hub l (suppose $p = l$),
$$z'_{pq} = z_{pq} + \sum_{s \in A^q} t_{is} \quad \text{and} \quad z'_{qp} = z_{qp} + \sum_{s \in A^q} t_{si}.$$
  - If $p = k$ and $q = l$,
$$z'_{pq} = z_{pq} - \sum_{s \in A^l} t_{is} + \sum_{s \in A^k \setminus \{i\}} t_{si} \quad \text{and} \quad z'_{qp} = z_{qp} - \sum_{s \in A^l} t_{si} + \sum_{s \in A^k \setminus \{i\}} t_{is}.$$

  As in N_{pairs}, from the new traffics T'_{pq} we compute the cost of the copies needed of each backbone edge. Again, the exchange is done only if the new cost is lower than the cost of the current solution.

Special solutions

Throughout this study we have been able to identify, from time to time, two kinds of special solutions regarding the number of hubs installed: solutions where there is **only one hub** and all nodes are assigned to it, and solutions where **all nodes are hubs** and each node is assigned to itself. We call them special solutions because in both cases the assignment of terminals to hubs cannot be modified. This makes useless the previously described local search procedures.

Once the whole process of destruction-construction ends, we check if such special cases correspond to feasible solutions with which compare the best solution found during the strategic oscillation process.

5 Computational experiments

In this section we describe the computational experiments performed to test the efficiency of the proposed strategic oscillation metaheuristic. The algorithm has been implemented in C using Mingw 4.6 compiler. The mathematical programming formulation described in Section 2 has been solved using Cplex 12.5, the most recent version of Cplex when the experiments were carried out. The results reported in this section were obtained with an Intel i7-3770 at 3.40GHz 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-known solution (or from the optimal solution, if available).
- 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.

5.1 Test instances

To test the performance of the proposed metaheuristic, we have generated a new set of 118 instances from two well-known instances in the ORLIB[1] [2] and from another instance recently proposed in [14]. Unfortunately, it has not been possible to obtain the original instances used by Yaman and Carello [18]. A detailed description of our instances follows:

1. 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 [13], with the distances and flows of a 25 nodes network. From this original file, a total of 15 instances with 10, 15, 20 and 25 nodes have been generated.
2. The **AP** (Australian Post) data set, based on real data from the Australian postal service and presented by Ernst and Krishnamoorthy in 1996 [4]. The size of the original data file is 200 nodes. Smaller instances can be obtained using a code from ORLIB. As with CAB, we have generated different instances from the original file. We have extended this set of instances by generating 60 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,

[1] ORLIB website http://people.brunel.ac.uk/∼mastjjb/jeb/info.html

t_{ij} is not necessarily equal to t_{ji}). Moreover, in the original instance some flows from one node to itself are positive (i.e., $t_{ii} > 0$ for a given i). We have modified them in such a way $t_{ii} = 0, \forall i$, in order to adapt the data to the model definition.

3. The **USA423** data set. This family of instances was introduced by Peiró, Corberán and Martí in [14], 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 data, 43 instances have been generated with n ranging from 10 to 185.

Each original instance includes the traffic and the traveling cost per unit matrices. From these two, we have generated the matrices t_{ij}, C_{ij}, R_{kl}, and the capacity values Q^a, Q^b, Q^h. While the t_{ij} traffic matrix is the original one, matrices C_{ij} and R_{kl} have been created to incorporate the assignment, installation, and inter-hub transportation costs.

The experiments are divided into two main blocks. The first block (Sections 5.2 and 5.3) is devoted to study the behavior of the components of the solution procedure, as well as to determine the best values for the search parameters. The second block of experiments (Sections 5.4 and 5.5) has the goal of comparing our procedure with the best published methods. To be able to test the effectiveness of our strategies, the first set of experiments is performed on a subset of instances to test how well our choices generalize to the entire set of problems.

From the 118 instances derived from the CAB, AP and USA423 data sets, the tuning experiments are performed on the following subset of 36 instances: 3 instances from the CAB set with $15 \leq n \leq 25$, 21 instances with $10 \leq n \leq 195$ from the AP set, and 12 instances with $20 \leq n \leq 150$ from USA423. We refer to these 36 instances as the *training set* and to the remaining 82 instances as the *testing set*. On the other hand, the instances have been classified as small ($10 \leq n \leq 50$), medium ($55 \leq n \leq 100$), and large ($105 \leq n \leq 200$). The entire set of instances is available at `www.optsicom.es`.

5.2 Parameter calibration

We initially perform several experiments to study the constructive-destructive method described in Section 4 in terms of solution quality and diversification power. In all the preliminary experiments we executed the strategic oscillation method for 100 global iterations ($\eta = 100$).

We first compare in SO1 the Random Walk and the Replace if Better acceptance criteria for different values of δ. The results are shown in Table 1, in which the average percentage deviation of the solution values with respect to the best value obtained in this experiment are given. As can be

seen, the best results are obtained with the values $\delta = 40\%$ and $\delta = 50\%$ for the first and second criteria, respectively. In order to compare both sets of results we have performed two well-known non-parametric tests for pairwise comparisons: the Wilcoxon test and the Sign test. On the one hand, the Wilcoxon test answers the question: Do the two samples (the solutions obtained with both methods in our case) represent two different populations? The resulting probability value of 0.001 indicates that the compared values come from different methods. On the other hand, the Sign test computes the number of instances on which an algorithm beats the other one. The resulting probability value of 0.004 indicates that the Replace if Better criterion is significantly better than Random Walk.

Average deviations for different δ values								
		10%	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**
Random Walk	small	10.6%	11.0%	7.2%	8.1%	6.6%	9.4%	9.2%	9.8%
	medium	9.8%	12.6%	12.2%	12.9%	13.2%	13.2%	14.9%	14.1%
	large	14.2%	12.1%	13.7%	11.3%	12.6%	13.3%	11.9%	13.3%
	average	11.5%	11.9%	11.0%	10.7%	10.8%	12.0%	12.0%	12.4%
Replace if Better	small	17.5%	14.3%	12.4%	8.0%	7.3%	7.4%	9.2%	13.5%
	medium	13.2%	5.5%	2.4%	4.3%	6.7%	9.7%	10.3%	10.3%
	large	14.4%	13.4%	11.7%	10.8%	6.5%	9.0%	10.5%	9.6%
	average	15.0%	11.1%	8.8%	7.7%	6.8%	8.7%	10.0%	11.1%

Table 1: Comparison of the two acceptance criteria for different values of δ

In a second experiment, the effect of parameter γ on the SO1 method is studied. According to the conclusion of the previous experiment, the Replace if Better strategy is the one selected. Table 2 shows the average percentage deviations obtained for all the γ values tested. Additionally, we tested a random-based variant, labeled as Rand, in which an integer random number is uniformly selected between 1 and 9 at each iteration. The results in this table clearly show that this last variant outperforms the others. Therefore, from now on, this variant (SO1 with Replace if Better strategy, $\delta = 50\%$ and random selection of γ), denoted SO1 for short, is the one selected for the rest of experiments.

Average deviations for different γ values						
	1	**3**	**5**	**7**	**9**	**Rand**
small	4.4%	4.4%	4.4%	4.4%	4.4%	0.4%
medium	3.1%	3.1%	3.1%	3.1%	3.1%	2.2%
large	2.4%	2.4%	2.4%	2.4%	2.4%	1.2%
average	3.3%	3.3%	3.3%	3.3%	3.3%	1.3%

Table 2: Results obtained for different values of γ

The effectiveness of generating multiple solutions in our strategic oscil-

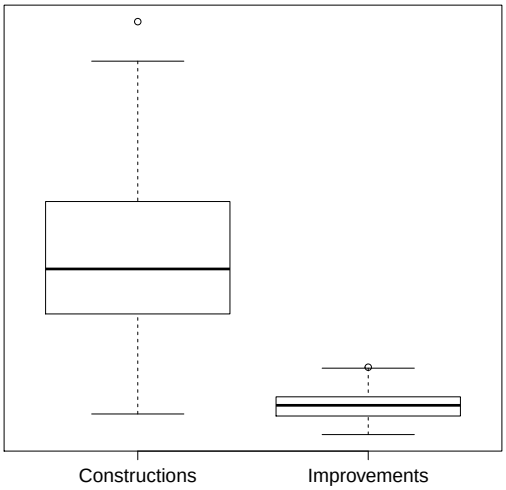

Figure 2: Boxplot of 100 iterations for instance 150-1000-69-60-80-1-69-USA

lation method has also been tested, since this algorithm relies on obtaining good and diverse solutions to serve as the starting point for the local search procedures. Figure 2 shows the boxplot of the SO1 method with and without the local search on a representative instance (150-1000-69-60-80-1-69-USA). The left boxplot shows the values of the 100 solutions found without applying the local search procedures, while the right one shows the results obtained after applying them. This plot clearly shows that different solutions are obtained in most of the runs. As expected, the variant with the local search obtains better solutions, as compared with the one without improvements, but with lower dispersion.

Another experiment was carried out to calibrate the value of the τ parameter in SO2. Since we did not observe any significant differences among the tested values, we do not report here the obtained results. A default value for parameter τ of 4 has been chosen.

5.3 Algorithm designs

In this section we compare the two strategic oscillation variants according to the memory structure used. SO1 applies $g^{'}(h)$ for hub selection, while SO2 uses $g(h)$ and implements a tabu list. The results obtained in this experiment are summarized in Table 3, where we report the number of best solutions found, out of 36, by each variant, as well as the average computing time used. This table shows that SO1 obtains better solutions than SO2. In particular, SO1 is able to match all the best known solutions, while SO2

only obtains 11 out of 36 instances, which represents an average percentage deviation of 9.2%. As a result of this experiment, from now on we select SO1 and simply denote it by SO.

	SO1			**SO2**		
	Dev	**# Best**	**CPU**	**Dev**	**# Best**	**CPU**
small	0.0%	12	0.32	9.2%	4	0.24
medium	0.0%	12	6.57	13.2%	1	3.41
large	0.0%	12	181.65	5.3%	6	151.56
summary	0.0%	36	62.85	9.2%	11	51.74

Table 3: Comparison between SO1 and SO2

5.4 Comparison with optimal values

In Section 2 we have described the formulation proposed in [18] for the CSHLPMLC, which contains quadratic constraints. We used Cplex to solve 30 instances with n ranging from 10 to 30, and only 11 instances could be solved to optimality. As far as we know, solving such an instance depends on the properties of its constraint matrix. The results obtained with Cplex for the optimally solved instances are reported in Table 4, as well as those obtained with the SO metaheuristic. In particular, it shows the average percentage deviation with respect to the optimal solution obtained with Cplex and the computing time used by each method.

Table 4 shows that the SO method is able to obtain the optimal solution in most cases (9 out of 11). In fact, for the only two instances in which SO was not able to find the optima, we allowed the algorithm to run for 200 iterations, but the results did not improve. As expected, Cplex required much more computing time than SO to obtain the optimal value. It is worth mentioning that Cplex used the total number of cores of the CPU (8 cores in our case) compared to only a few used by the SO algorithm.

5.5 Comparison with a tabu search algorithm

Since it was not possible to compare the SO procedure with Cplex on larger instances, in order to test its behavior we have compared our algorithm with the tabu search algorithm (PrevTS) described in [18]. In this section we use the instances in the training and testing sets. Table 5 shows the average percentage deviation with respect to the best solution known (Dev), the number of best solutions found (# Best), and the computing time (CPU) of both methods on the 36 training set instances.

Table 5 clearly shows that our SO method (which incorporates tabu search memory structures) outperforms the previously proposed tabu search

	Cplex		SO	
	Value	**CPU**	**Dev**	**CPU**
A1H	72710	24.02	4.0%	0.13
A2H	105477	254.30	0.0%	0.02
A3H	77516	23.69	20.7%	0.13
A4H	188200	139.00	0.0%	0.02
B1H	45636	75.80	0.0%	0.13
B2H	23818	4.66	0.0%	0.12
B3H	51387	31.08	0.0%	1.54
B4H	25410	4.71	0.0%	0.03
C1H	43526	4297.98	0.0%	0.29
C2H	43505	3304.48	0.0%	0.28
C3H	57905	33891.33	0.0%	0.30

Table 4: Comparison between Cplex and SO on small-size instances

	PrevTS			SO		
	Dev	**# Best**	**CPU**	**Dev**	**# Best**	**CPU**
small	4.87%	5	1.44	1.28%	7	0.33
medium	4.68%	1	18.57	0.52%	11	6.61
large	16.79%	2	240.50	1.33%	10	154.31
summary	8.78%	8	86.84	1.04%	28	53.75

Table 5: Comparison between PrevTS and SO on the training set instances

approach. In particular, SO matches 28 out of 36 instances, while PrevTs is only able to match 8 instances. On the other hand, SO exhibits a 1.04% average percent deviation and PrevTS 8.78% achieved in 53.75 and 86.84 seconds, respectively. It must be noted that, as mentioned in [18], the objective of the authors when developing PrevTS was to obtain relatively good initial solutions for their exact method, while our SO has been designed to obtain high quality solutions in short running times.

We compare now the performance of SO and PrevTS on the testing set, to measure the ability of our algorithm (SO) to target instances not included in the training set. In particular, we consider 82 instances classified according to their size into small, medium and large. Note that the large set includes instances with $n = 200$. Table 6 shows the results of this experiment in terms of the average deviation with respect to the best known value (Dev) and number of instances in which each method is able to match this best value (# Best). This table shows that SO obtains better results than PrevTS in significantly lower running times. Specifically, PrevTS has an average deviation of 8.18% obtained in 379.33 seconds, while SO has an average deviation of 1.23% obtained in 78.82 seconds. Additionally, we have performed the Sign and Wilcoxon test with the results of this experiment. The probability values of 0.00 obtained in both tests confirm the superiority of SO. Tables 7, 8 and 9 in the Appendix show the individual results of this experiment, to provide the reader with a detailed information for further experimentation.

In our last experiment, we compare SO and PrevTs over the time. Figure 3 shows the evolution throughout the search of the best value obtained with each method on a representative instance. The search profile depicted in this figure shows that SO (dashed line) obtains better solutions than PrevTS from the very beginning of the search. On the other hand, the PrevTS needs some time to reach relatively good solutions, and after 200 seconds, it seems to stagnate, while SO still exhibits a marginal improvement.

	PrevTS			**SO**		
	Dev	**# Best**	**CPU**	**Dev**	**# Best**	**CPU**
small	8.26%	7	1.27	0.25%	20	0.33
medium	5.95%	5	88.15	1.23%	20	12.33
large	9.86%	7	902.18	1.99%	29	192.09
summary	8.18%	19	379.33	1.23%	69	78.82

Table 6: Comparison between PrevTS and SO on the testing set instances

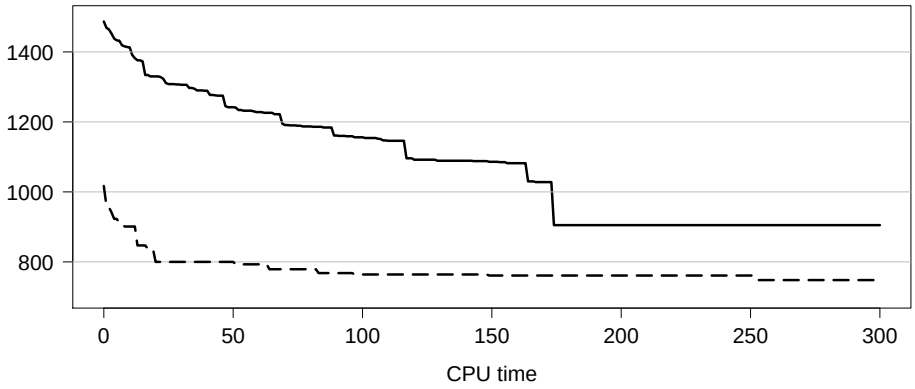

Figure 3: Search Profile for SO (dashed line) and PrevTs (plain line)

6 Conclusions and future research

We have proposed a new metaheuristic based on Strategic Oscillation for the Capacitated Single assignment Hub Location Problem with Modular Link Capacities. This problem was introduced by Yaman and Carello [18] as an interesting variant of the classical hub location problem in which the cost of using edges is not linear but stepwise, and the hubs are restricted in terms of transit capacity rather than in the incoming traffic. Our proposed method incorporates several designs for constructive and destructive algorithms making use of tabu search memory structures together with associated local search procedures. The computational experiments show that our algorithm is able to find high-quality solutions in short computing times, and outperforms a previously published tabu search procedure.

We envision that future enhancements of our method may be possible by employing additional strategies derived from the strategic oscillation and tabu search methodology, such as replacing the recourse to randomization with more strategic elements (as by removing hubs strategically using tabu search memory in the destructive phases), and joining recency memory and frequency memory instead of treating them independently.

Acknowledgements

This work was supported by the Spanish Ministerio de Economía y Competitividad (projects TIN-2012-35632-C02, MTM-2012-36163-C06-02, and grant BES-2013-064245) and by the Generalitat Valenciana (project Prometeo 2013/049). Authors want to thank Hande Yaman for providing us some data files for the test problems in Table 4.

References

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

[2] J. E. Beasley. OR-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

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

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

[5] L. Fanjul-Peyroa and R. Ruiz. Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, 207(1):55–69, 2010.

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

[7] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166, 1977.

[8] F. Glover and M. Laguna. *Tabu search*. Kluwer, Norwell, MA, 1997.

[9] A. Ilić, D. Urošević, J. Brimberg, and N. Mladenović. A general variable neighborhood search for solving the uncapacitated single allocation p-hub median problem. *European Journal of Operational Research*, 206(2):289–300, 2010.

[10] L.W. Jacobs and M.J. Brusco. A local-search heuristic for large set-covering problems. *Naval Research Logistics*, 42(7):1129–1140, 1995.

[11] M. Lozano, D. Molina, and C. García-Martínez. Iterated greedy for the maximum diversity problem. *European Journal of Operational Research*, 214(1):31–38, 2011.

[12] M. T. Melo, S. Nickel, and F. Saldanha da Gama. Facility location and supply chain management - a review. *European Journal of Operational Research*, 196(2):401–412, 2009.

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

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

[15] R. Ruiz and T. Stützle. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, 187(3):1143–1159, 2008.

[16] R. Ruiz and T. Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049, 2008.

[17] H. Yaman. *Concentrator Location in Telecommunication Networks*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium, Dec 2002.

[18] H. Yaman and G. Carello. Solving the hub location problem with modular link capacities. *Computers & Operations Research*, 32(12):3227–3245, 2005.

[19] K.C. Ying and H.M. Cheng. Dynamic parallel machine scheduling with sequence-dependent setup times using an iterated greedy heuristic. *Expert Systems with Applications*, 37(4):2848–2852, 2010.

Appendix

Type	Instance	PrevTS			SO		
		Value	**Dev**	**CPU**	**Value**	**Dev**	**CPU**
	10_700_50_60_8_1_60CAB	243854	0.00%	0.02	246501	1.09%	0.00
	10_700_69_40_8_1_50CAB	246610	0.00%	0.02	246610	0.00%	0.00
	10_800_60_60_6_1_69AP	283138	19.50%	0.03	236944	0.00%	0.00
	10_800_60_60_6_1_69CAB	238144	0.00%	0.02	238144	0.00%	0.01
	10_800_60_80_8_1_80CAB	240872	0.00%	0.02	242976	0.87%	0.01
	15_600_80_89_6_1_69CAB	291191	0.00%	0.09	291695	0.17%	0.02
	20_700_50_60_8_1_60AP	405786	1.15%	0.24	401162	0.00%	0.06
	20_700_50_60_8_1_60CAB	205817	16.93%	0.21	176011	0.00%	0.05
	20_700_50_60_8_1_60USA	137854	7.31%	0.13	128464	0.00%	0.05
	20_700_69_40_8_1_50AP	432081	1.34%	0.32	426347	0.00%	0.05
	20_700_69_40_8_1_50CAB	214323	17.11%	0.18	183015	0.00%	0.04
	20_800_60_60_6_1_69CAB	191608	15.06%	0.21	166528	0.00%	0.04
Small	20_800_60_80_8_1_80AP	385461	4.50%	0.56	368864	0.00%	0.04
	20_800_60_80_8_1_80CAB	197141	14.23%	0.23	172585	0.00%	0.03
	25_600_80_60_6_1_40CAB	245429	16.07%	0.46	211448	0.00%	0.12
	25_600_80_89_8_1_60CAB	242246	15.53%	0.49	209684	0.00%	0.12
	25_650_69_69_6_1_70CAB	221333	19.05%	0.83	185921	0.00%	0.15
	30_700_69_40_8_1_50USA	246833	29.63%	0.77	190419	0.00%	0.12
	35_600_80_89_8_1_60AP	478595	0.00%	0.27	491376	2.67%	0.45
	35_600_80_89_8_1_60USA	232360	11.32%	1.30	208733	0.00%	0.48
	40_700_80_50_8_1_69USA	330583	6.91%	3.13	309213	0.00%	0.85
	45_700_69_40_8_1_50AP	611473	0.00%	2.45	621057	1.57%	1.21
	45_700_69_40_8_1_50USA	339338	0.35%	3.29	338168	0.00%	0.97
	50_700_69_40_8_1_50AP	669803	3.97%	9.06	644246	0.00%	1.51
	50_700_80_50_8_1_69USA	374127	6.45%	7.43	351456	0.00%	1.86

Table 7: SO and PrevTS on small size instances of the testing set

Type	Instance	PrevTS			SO		
		Value	**Dev**	**CPU**	**Value**	**Dev**	**CPU**
	55_500_60_69_60_1_50AP	525642	0.00%	8.65	581649	10.65%	2.80
	55_500_60_69_60_1_50USA	378822	12.96%	9.63	335367	0.00%	2.54
	55_800_69_50_80_1_60AP	609771	0.00%	7.06	657315	7.80%	2.87
	60_500_60_69_60_1_50AP	615449	7.57%	5.90	572141	0.00%	3.86
	60_600_60_69_60_1_69USA	391313	20.28%	18.07	325348	0.00%	3.53
	60_800_69_50_80_1_60AP	702917	3.50%	4.94	679162	0.00%	3.55
	60_800_69_50_80_1_60USA	424505	14.63%	18.86	370335	0.00%	3.59
	65_500_60_69_60_1_50AP	665600	7.39%	12.27	619823	0.00%	6.18
	65_800_69_50_80_1_60USA	402210	0.00%	18.98	419754	4.36%	5.37
	70_600_60_69_60_1_69AP	658758	2.85%	22.66	640480	0.00%	8.38
	70_600_60_69_60_1_69USA	428069	18.73%	38.43	360551	0.00%	6.79
	70_800_69_50_80_1_60AP	728949	0.00%	27.69	782537	7.35%	8.13
Medium	75_600_60_69_60_1_69USA	568456	14.30%	77.59	497343	0.00%	9.64
	75_800_69_50_80_1_60AP	868528	2.23%	36.96	849573	0.00%	9.85
	80_500_60_69_60_1_50AP	780236	0.88%	58.75	773396	0.00%	14.69
	80_500_60_69_60_1_50USA	724155	11.89%	92.88	647214	0.00%	13.57
	80_800_69_50_80_1_60USA	690769	1.06%	94.44	683508	0.00%	13.35
	85_500_60_69_60_1_50AP	886289	5.40%	152.40	840859	0.00%	17.97
	85_800_69_50_80_1_60AP	1003536	2.50%	96.61	979071	0.00%	19.00
	90_500_60_69_60_1_50USA	904709	13.10%	277.96	799944	0.00%	19.81
	90_600_60_69_60_1_69USA	742016	2.43%	166.94	724440	0.00%	18.76
	95_500_60_69_60_1_50AP	972840	5.25%	68.41	924290	0.00%	29.08
	95_600_60_69_60_1_69AP	859425	1.79%	86.81	844308	0.00%	31.30
	95_600_60_69_60_1_69USA	723454	0.00%	284.44	727768	0.60%	23.02
	100_500_60_69_60_1_50USA	877730	0.08%	516.37	877069	0.00%	30.66

Table 8: SO and PrevTS on medium size instances of the testing set

Type	Instance	PrevTS			SO		
		Value	**Dev**	**CPU**	**Value**	**Dev**	**CPU**
Large	110_600_60_69_60_1_69AP	1174251	17.70%	154.39	997699	0.00%	25.32
	120_500_60_69_60_1_50AP	1349557	15.87%	113.15	1164724	0.00%	38.85
	125_500_60_69_60_1_50AP	1498128	21.36%	208.45	1234498	0.00%	38.01
	130_600_60_69_60_1_69AP	1426980	12.67%	145.97	1266488	0.00%	39.19
	130_800_69_50_80_1_60USA	1663758	18.14%	537.61	1408324	0.00%	44.06
	135_600_60_69_60_1_69AP	1610997	30.12%	263.48	1238055	0.00%	61.29
	135_800_69_50_80_1_60USA	1584036	10.60%	4960.59	1432262	0.00%	53.01
	140_500_60_69_60_1_50AP	1691066	12.60%	579.65	1501836	0.00%	60.31
	140_800_69_50_80_1_60AP	2130072	24.93%	293.33	1705035	0.00%	66.37
	145_800_69_50_80_1_60AP	2114457	25.49%	381.02	1684972	0.00%	73.90
	145_800_69_50_80_1_60USA	715853	0.00%	227.14	715853	0.00%	109.78
	150_800_69_50_80_1_60AP	2200307	21.93%	698.27	1804633	0.00%	92.54
	150_900_69_60_80_1_89USA	584419	0.00%	272.34	584419	0.00%	134.87
	155_1000_69_60_80_1_69USA	727403	1.04%	430.02	719931	0.00%	85.78
	155_800_69_50_80_1_60AP	2049217	22.06%	1408.05	1678796	0.00%	112.21
	160_800_69_50_80_1_60AP	749451	0.00%	221.53	961809	28.34%	160.21
	165_1000_69_60_80_1_69USA	712118	4.70%	783.95	680178	0.00%	174.06
	165_800_69_50_80_1_60USA	767748	4.45%	838.54	735012	0.00%	199.45
	170_500_60_69_60_1_50AP	588784	0.00%	207.78	690343	17.25%	177.26
	170_900_69_60_80_1_89USA	623614	0.00%	872.50	623614	0.00%	205.92
	175_500_60_69_60_1_50AP	889474	3.78%	615.16	857100	0.00%	202.84
	175_600_60_69_60_1_69AP	710911	5.20%	695.56	675774	0.00%	203.61
	175_800_69_50_80_1_60USA	790880	0.67%	1358.05	785590	0.00%	255.05
	175_900_69_60_80_1_89USA	652884	0.95%	1466.26	646769	0.00%	225.42
	180_1000_69_60_80_1_69USA	754360	2.26%	1536.09	737662	0.00%	234.88
	180_600_60_69_60_1_69AP	840658	6.74%	764.75	787587	0.00%	281.14
	185_800_69_50_80_1_60AP	796454	0.00%	1595.31	941838	18.25%	377.93
	185_900_69_60_80_1_89USA	615474	0.00%	576.85	615474	0.00%	288.66
	190_800_69_50_80_1_60AP	1212471	15.76%	631.85	1047414	0.00%	436.89
	195_600_60_69_60_1_69AP	759008	4.64%	1971.14	725377	0.00%	459.45
	200_500_60_69_60_1_50AP	868511	25.27%	1641.45	693294	0.00%	619.96
	200_800_69_50_80_1_60AP	815087	6.76%	2419.51	763466	0.00%	608.59

Table 9: SO and PrevTS on large size instances of the testing set