Large scale local search heuristics for the Capacitated Vertex p-Center Problem

M.P. Scaparra S. Pallottino M.G. Scutellà

Dipartimento di Informatica, Università di Pisa, Pisa, Italy

[scaparra,pallo,scut]@di.unipi.it

Abstract

This paper investigates the application of very-large neighborhood search techniques for solving the capacitated vertex p-center problem. We characterize a local search neighborhood in terms of path and cyclic exchanges of customers among facilities, and exploit principles borrowed from network optimization theory to efficiently detect cost decreasing solutions in such a neighborhood. We complement the multi-exchange methodology with a relocation mechanism specifically designed to perform facility location adjustments. The validity of the proposed approach is supported by empirical investigation and performance comparisons with the commercial code CPLEX.

Keywords: location problems, large scale optimization, multi-exchange

1 Introduction

The vertex p-center problem consists of locating p facilities and assigning customers to them so as to minimize the longest distance between a customer and the facility to which it is assigned. The term *vertex* refers to the fact that the set of candidate facility sites, coinciding with the set of demand points, can be represented as the vertex set, V, of a complete graph. The distance between any two sites i and j can then be translated into a positive length, d_{ij}, associated with the arc connecting nodes i and j in the graph. In this paper we address the capacitated version of the vertex p-center problem, denoted by CVPCP, which is characterized by the presence of upper limits to the service that each facility can supply. Each node $j \in V$ has a weight, w_j, associated with it, representing the customer demand for service at that node. The model under study requires that the total customer demand assigned to each facility i does not exceed its capacity, s_i, while enforcing each customer demand to be entirely allocated to a single facility.

The capacitated vertex p-center problem can be stated mathematically as follows:

$$
\begin{aligned}
\text{(CVPCP)} \qquad & \min && W && && (1)\\
& \text{s.t.} && \sum_{i\in V} x_{ij} && = 1 \quad \forall\ j \in V && (2)\\
& && \sum_{j\in V} w_j x_{ij} - s_i y_i && \le 0 \quad \forall\ i \in V && (3)\\
& && \sum_{i\in V} d_{ij} x_{ij} - W && \le 0 \quad \forall\ j \in V && (4)\\
& && \sum_{i\in V} y_i && = p && (5)\\
& && x_{ij} \in \{0,1\} && \forall\ i \in V, j \in V && (6)\\
& && y_i \in \{0,1\} && \forall\ i \in V. && (7)
\end{aligned}
$$

Constraints (2) ensure that all customers are allocated to exactly one center; constraints (3) enforce the total demand of customers assigned to a facility not to exceed its capacity; constraints (4) simply force the variable W to be at least the longest customer-facility distance; the total number of open facilities is set to p by (5); the last two sets of constraints represent the variables integrality requirements; finally, formula (1), together with constraints (4), states that the objective to be minimized is the maximum distance between a demand node and the facility to which it is assigned.

The minimax criterion characterizing the p-center objective is traditionally used in location applications to minimize the adverse effects of worst case scenarios in providing service to the customers. Common application settings for such an objective, such as public school district planning or medical facility system design, naturally impose the presence of limits to the total demand that each facility can supply. Also, in some emergency facility location problems the inclusion of the capacity constraints can be used as a tool for hedging the risk of server unavailability and, consequently, for tackling congestion and delay issues. Unfortunately, the capacity requirement greatly increases the difficulty of the problem, which even in the uncapacitated case is well-known to be NP-hard [9]. A capacitated version of the p-center problem was first posed by Bar-Ilan et al. [4] to address the problem of locating centers in distributed communication networks. The authors refer to the problem as the Balanced p-Center Problem, since it aimes at achieving workload balancing among centers. In order to avoid centers overloading and bottleneck problems, Bar-Ilan et al. bound the maximum number of sites assigned to any particular center, and propose a constant ratio approximation algorithm with a factor of 10 to solve the balanced problem. The approach combines the elegant approximation techniques of Hochbaum and Shmoys [8] for the unbalanced p-center problem to choose centers, and network flow techniques to assign customers. Jaeger and Goldberg [11] address the capacitated p-center problem on tree networks. They show that, in this particular case, the problem can be solved in polynomial time when the facility capacities are identical. Khuller and Sussmann [10] consider the p-center problem where each facility can be assigned at most L demand nodes. They give a polynomial approximation algorithm with a factor of 5 for the capacitated multi-p-center problem, i.e., for the problem simplification which allows more centers to be colocated at the same site. For the more difficult case enforcing the selection of p distinct sites, they obtain an approximation factor of 6.

In this paper, we address a generalization of the aforementioned capacitated p-center problems, where facilities can have different capacities and the total customer demand assigned to each facility, rather than the number of customers, is limited by the individual site capacity. We propose a very large scale neighborhood (VLNS) technique to solve the problem, which relies on the definition of a multi-exchange neighborhood structure and exploits flow based improvement algorithms to efficiently detect improving solutions in such a neighborhood. The methodology is complemented by a relocation mechanism specifically designed to perform facility location adjustments.

The first attempt to apply multi-exchange algorithms to problems with minimax objectives is due to Frangioni et al. [6], who devised new local search algorithms for the minimum makespan machine scheduling problem. In the location arena, multi-exchange techniques were used for the first time by Ahuja et al. [2] for the single-source capacitated facility location problem.

The plan of the paper is as follows. In Section 2 we set the notation and some underlying definitions that will be used throughout the discussion. In Section 3 we cover the main issues characterizing the multi-exchange neighborhood structure. We start by defining path and cyclic multi-exchanges for CVPCP; we then show how to identify them in terms of subset-disjoint cycles in a suitably defined improvement graph; we propose several cycle detection heuristics and show how a local reoptimization phase can be embedded in the cycle search. In Section 4 we describe the neighborhood search algorithm, while in Section 5 we report and compare the results obtained under different cycle detection policies. To conclude the paper, we suggest in which direction to convey future research efforts.

2 Notation

Let S be a feasible solution to the problem (1)-(7). S can be represented as a partition of the set of vertices, V, into p subsets, each one corresponding to an open facility and containing all the demand nodes assigned to it in S. Namely, $S = \{S_1, \ldots, S_p\}$ such that $\bigcup_{i=1}^{p} S_i = V$ and $S_i \bigcap S_h = \emptyset$ for $i, h = 1, \ldots, p$ $(i \neq h)$.

We denote by $l(i)$ the *active* location for subset S_i, i.e., the node which hosts the facility serving the customers in S_i. The cost of each subset S_i, $i = 1, \ldots, p$, is then

$$C(S_i) = \max\{d_{l(i)j} | j \in S_i\}, \tag{8}$$

and the cost of the whole partition S is

$$C(S) = \max\{C(S_i) | i = 1, \ldots, p\}. \tag{9}$$

$C(S)$ is also called the *bottleneck cost* for the solution S. Every subset S_i such that $C(S) = C(S_i)$ is a *bottleneck subset*, and each active location $l(i)$ corresponding to a bottleneck subset is a *bottleneck facility*. We denote by $B(S)$ the set of bottleneck subsets in S, i.e., $B(S) = \{i \in \{1, \ldots, p\} : C(S_i) = C(S)\}$. Each bottleneck subset S_i, $i \in B(S)$, has a set of *bottleneck customers*, $J(i)$, associated with it, which contains the demand nodes with maximum distance from the active location in that subset, i.e., $J(i) = \{j \in S_i : d_{l(i)j} = C(S)\}$.

A solution S to CVPCP is feasible if the following inequalities hold:

$$W(S_i) = \sum_{j \in S_i} w_j \leq s_{l(i)}, \qquad i = 1, \ldots, p; \tag{10}$$

that is, S is feasible if, for each subset, the capacity of the facility at the active location is not exceeded by the total demand of the customers assigned to it. The *residual capacity* of each subset S_i is defined as:

$$r_{l(i)} = s_{l(i)} - W(S_i). \tag{11}$$

Finally, we denote by $F(j)$, $j \in V$, the facility serving customer j in the solution S; that is, $F(j) = l(i)$ if $j \in S_i$.

3 The multi-exchange neighborhood

The multi-exchange neighborhood structure for CVPCP is similar to the one provided in [2] for the single source capacitated facility location problem (SSCFLP), the main difference residing in the definition of exchange profitability. As in [2], two kind of multi-exchanges are defined, i.e., cyclic and path exchanges.

A *cyclic exchange* with respect to a solution S is defined as a customer sequence $R = (j_1, j_2, \ldots, j_q)$, $q \leq p$, such that each customer j_r in the sequence is assigned to a different open facility in S, i.e., $F(j_r) \neq F(j_s)$ for $r \neq s$, $r, s = 1, \ldots, q$. A cyclic exchange R represents the movement of customer j_r from $F(j_r)$ to $F(j_{r+1})$, for each $r = 1, \ldots, q-1$, and of customer j_q from $F(j_q)$ to $F(j_1)$. R is *feasible* if the capacity of each facility involved in the cycle is still satisfied after the exchange, i.e., if $w_{j_r} \leq r_{F(j_{r+1})} + w_{j_{r+1}}$ for $r = 1, \ldots, q-1$ and $w_{j_q} \leq r_{F(j_1)} + w_{j_1}$. R is *profitable* if the new solution S' obtained from S through R is such that:

$$C(S') < C(S), \tag{12}$$

or

$$C(S') = C(S),\ B(S') \subseteq B(S),\ J'(i) \subseteq J(i), \forall\ i \in B(S'),\quad \bigcup_{i \in B(S')} J'(i) \subset \bigcup_{i \in B(S)} J(i). \quad (13)$$

In (13), $J'(i)$ denotes the new set of bottleneck customers related to the bottleneck subset $S'_i \in S'$. This definition of profitability states that an exchange is profitable either if it decreases the objective function value or if it reduces the number of bottleneck customers while not worsening the bottleneck cost, without creating new bottleneck subsets and new bottleneck customers.

We also consider an alternative and narrower definition of profitability in which (13) is substituted by:

$$C(S') = C(S), \quad B(S') \subseteq B(S), \quad J'(i) \subset J(i),\ \forall\ i \in B(S'). \quad (14)$$

We will refer to the multi-exchanges which satisfy condition (12) or condition (14) as *strictly profitable* multi-exchanges. According to the new definition, a multi-exchange is strictly profitable either if it decreases the objective function value or if the number of bottleneck customers is reduced in each bottleneck subset, and neither bottleneck subsets nor bottleneck customers are generated.

A *path exchange* $P = (j_1, j_2, \ldots, j_{q-1}, F)$, where F is a facility other than $F(j_r)$, $r = 1, \ldots, q-1$, is similar to a cyclic exchange, but in this case no customer belonging to F moves to $F(j_1)$. The definition of feasibility, profitability and strict profitability of a path exchange are identical to the corresponding definitions stated for the cyclic exchanges.

Each feasible cyclic or path exchange generates a new feasible solution S' from the current solution S. The totality of these new solutions defines the multi-exchange neighborhood of S. Such a neighborhood can be modelled in a compact way through a *customer improvement graph*, and searched by means of network flow optimization techniques. This will be the subject of the following sections.

3.1 The customer improvement graph

The *customer improvement graph* related to a solution S of CVPCP is the directed graph $G(S) = (N(S), A(S))$ defined as follows. The set of nodes $N(S)$ contains a *regular* node for each demand node j, a *pseudonode*, p_i, for each active location $l(i)$, $i = 1, \ldots, p$, and an *origin* node o. The arc set $A(S)$ contains an arc (j, k) for each pair of regular nodes j and k in $N(S)$, provided that the three following conditions are satisfied:

(c.1) $F(j) \neq F(k)$;

(c.2) $w_j - w_k \leq r_{F(k)}$;

(c.3) $d_{F(k)j} < C(S)$.

Such an arc (j,k) signifies that customer j is removed from facility $F(j)$ and assigned to facility $F(k)$. It also implies the removal of customer k from $F(k)$. Condition (c.1) states that such an arc exists only if customers j and k are not assigned to the same facility in the current solution S. Condition (c.2) guarantees that the customer movement implied by the arc does not violate the capacity constraints of facility $F(k)$. Finally, condition (c.3) asserts that an arc (j,k) exists only if the distance between customer j and facility $F(k)$ is strictly less than the bottleneck cost of the current solution. This last condition deserves particular attention since it guarantees that no move associated with an arc of the customer improvement graph generates new bottleneck customers.

Note that the inclusion in $N(S)$ of regular nodes which correspond to active locations amounts to assuming that, even if a node is active, the demand of the associated customer can be assigned to a facility sited at a different node.

The arc set $A(S)$ also contains an arc (j, p_i) from a regular node j to a pseudonode p_i, provided that customer j is not served by facility $l(i)$ in the current solution S, that its demand does not exceed the residual capacity of facility $l(i)$, and that the distance between j and $l(i)$ is strictly smaller than the bottleneck cost, that is:

(c.4) $F(j) \neq l(i)$;

(c.5) $w_j \leq r_{l(i)}$;

(c.6) $d_{l(i)j} < C(S)$.

An arc (j, p_i) implies the transferring of customer j from its current facility to the facility located at $l(i)$, without moving any customers from such a facility.

Finally, we have an arc (p_i, o) from each pseudonode back to the origin, and an arc (o, j) from the origin to each regular node.

These arcs will be used in order to model the feasible path exchanges in terms of suitable cycles of the customer improvement graph, which include the origin node.

3.2 Subset-disjoint cycles

Profitable multi-exchanges of customers can be modelled as special disjoint cycles on the customer improvement graph, as stated by the following definition.

Definition 3.1 *Consider the customer improvement graph $G(S)$ associated with a solution S. A directed cycle $(n_1, \ldots, n_q)$, with $n_r \in N(S)$, $r = 1, \ldots, q$, is a subset-disjoint cycle if each one of its nodes is associated with a different facility location (with the exception of the origin node, if present).*

From the above definition the following property can be easily proved.

Property 3.1 *Each subset-disjoint cycle in $G(S)$ corresponds to a feasible cyclic or path exchange with respect to S.*

In particular, the fact that the customer improvement graph only includes feasible arcs with respect to the capacity constraints, as expressed by conditions (c.2) and (c.5), ensures that each exchange corresponding to a subset-disjoint cycle in $G(S)$ is a feasible exchange. The node sequence $(n_1, \ldots, n_q)$, with $n_r \in N(S)$, $r = 1, \ldots, q$, corresponds to a path exchange if it contains the origin and a pseudonode; otherwise, it corresponds to a cyclic exchange.

Note that the correspondence stated by Property 3.1 does not hold in both directions. In fact, there are some cyclic or path exchanges which are not mapped into the customer improvement graph, due to the definition of the arc set $A(S)$. These exchanges are the ones which worsen the current bottleneck cost, defined in (9); they are excluded from the mapping by conditions (c.3) and (c.6) on the arc existence. This simplification is made possible by the particular nature of the CVPCP objective function, which ensures that local moves which worsen the current bottleneck cost can be safely excluded from the neighborhood, since they can never yield improving solutions. This is not necessarily true for problems with a minsum objective function, where often moves that locally worsen the objective function cannot be ruled out.

The minmax objective characterizing CVPCP also allows to characterize the subset-disjoint cycles, which correspond to profitable multi-exchanges, independently of an arc cost definition. We recall that, for problems with minsum objectives, profitable multi-exchanges usually correspond to negative subset-disjoint cycles, and the notion of negativity is connected to costs associated with the arcs in the related improvement graph. Similar observations were already made in [6] for the minimum makespan parallel machine scheduling problem as opposed to some vehicle routing problems [14] and to the capacitated minimum spanning tree problem [3].

The above observation is formalized in the following property.

Property 3.2 *Given a customer improvement graph $G(S)$, there is a one-to-one correspondence between the set of the subset-disjoint cycles which contains at least one bottleneck customer and the profitable cyclic or path exchanges for the corresponding solution S.*

Proof: Consider a subset-disjoint cycle which contains a bottleneck customer $\hat{\jmath}$ and does not include the origin and a pseudonode. Two cases are possible. (i) $\hat{\jmath}$ is the only bottleneck customer in the solution S. Condition (c.3) guarantees that the cost of assigning $\hat{\jmath}$ to the new facility, as indicated by the cycle information, is strictly less than the bottleneck cost. Also, the same condition ensures that the new assignments of the customers involved in the cycle do not equal or exceed the bottleneck cost. Hence, the cost of the new solution will be strictly decreased and the corresponding exchange is a profitable exchange according to (12). (ii) Other customers besides $\hat{\jmath}$ have the same bottleneck distance from the facilities they are assigned to. If all these customers belongs to different subsets and they are all included in the cycle, the same reasoning of case (i) proves that the cost of the new solution is strictly decreased and, hence, the cycle corresponds to a profitable exchange. Otherwise, if any bottleneck customer belongs to the same subset of $\hat{\jmath}$ or belongs to a different subset but it is not included in the cycle, then the bottleneck cost of the solution cannot decrease. However, the bottleneck distance is decreased at least for customer $\hat{\jmath}$, and condition (c.3) guarantees that all the other customer moves, corresponding to the arcs in the cycle, do not create new bottleneck customers. Hence, the move does not worsen the objective function, does not increase the number of the bottleneck subsets, and reduces the number of bottleneck customers for at least one bottleneck facility. The corresponding exchange is then a profitable cyclic exchange according to (13). The case of cycles containing the origin and a pseudonode can be proved in a similar way, by using condition (c.6) instead of (c.3) when needed.

The opposite correspondence can be proved in a similar way.

Property 3.2 is modified as follows when the definition of strict profitability is considered.

Property 3.3 *There is a one-to-one correspondence between the set of strictly profitable cyclic and path exchanges for the current solution S, and the set of subset-disjoint cycles in $G(S)$ which contain exactly one bottleneck customer for each bottleneck subset.*

Property 3.3 can be easily proved following the same lines as for Property 3.2.

Hereafter we will refer to the cycles characterized by Property 3.2 as *1-disjoint cycles*, and to the cycles characterized by Property 3.3 as *t-disjoint cycles*, where t denotes the number of bottleneck subsets in the current solution.

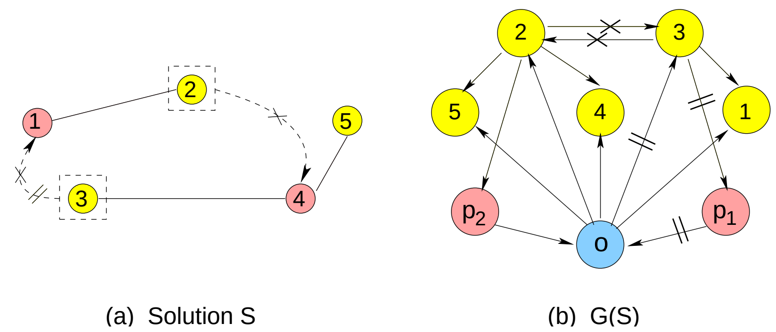


Figure 1: Customer improvement graph

Figure 1(b) shows the customer improvement graph associated with the solution S of a simple location problem with 5 demand nodes and $p = 2$, depicted in Figure 1(a). The active facility locations for the two partition subsets $S_1 = \{1, 2\}$ and $S_2 = \{3, 4, 5\}$ are node 1 and node 4, respectively. S_2 is the only bottleneck subset, with bottleneck facility 4 and bottleneck customer 3. In this example, the two subset-disjoint cycles containing the bottleneck customer 3 (the ones marked in the picture) correspond to profitable exchanges, with no need to define the arc costs. These are the cycle $\{(2,3),(3,2)\}$ and the cycle $\{(o,3),(3,p_1),(p_1,o)\}$. The second one corresponds to the path exchange which only reassigns customer 3. Note that, since 3 is the only bottleneck customer, the two exchanges are also strictly profitable. Moreover, note the exclusion from the customer improvement graph of arcs such as $(5,2)$, since the Euclidean distance between customer 5 and the facility location $F(2)=1$ is greater than the current bottleneck distance (from customer 3 to the facility location 4).

The problem of looking for a subset-disjoint cycle in $G(S)$ which includes at least one bottleneck customer is in general NP-complete. Even more so, the problem of looking for subset-disjoint cycles which include one bottleneck customer for each bottleneck subset. A more detailed treatment of this point can be found in [6], where special structural properties of the improvement graphs associated with bottleneck-type problems have been investigated. The difficulty of the problem suggests to devise heuristic approaches to search the graph in a reasonable amount of computational time.

3.3 Search for subset-disjoint cycles

As we pointed out in Section 3.2, the characterization of profitable multi-exchanges in terms of subset-disjoint cycles leaves out of consideration the definition of costs for the arcs in $G(S)$.

Nevertheless, we introduce here the notion of cost as a tool to drive the selection of the nodes during the subset-disjoint cycle computation. In particular, similarly to what has been proposed in [6], we provide two different definitions of the arc costs, each one suggesting a different way of processing the nodes during the search. According to the first definition, the cost of each arc measures the cost reduction of the bottleneck subset associated with that arc, if any; according to the second definition, the cost of each arc measures the gap between the cost of the subset associated with the head node of the arc after the move, and the current bottleneck cost. In both definitions, all the costs associated with the arcs are either negative or equal to zero.

The first cost definition suggests to perform the search of subset-disjoint cycles through a label correcting algorithm for shortest path problems suitably modified. We will refer to the cycles detected by this technique as *1SP-cycles* and *tSP-cycles*, according to whether they look for 1-disjoint cycles or t-disjoint cycles, respectively. The second definition encourages the use of an approach which is a modification of the algorithm for computing bottleneck paths in a directed graph. The cycles returned by such an approach will be referred to as *1BP-cycles* and *tBP-cycles*. Below, we provide the formal definitions of the arc costs for each case, starting from the one which produces SP-cycles.

3.3.1 The arc cost definitions

Hereafter we will refer to the second maximum distance from the active location of a subset S_i, $l(i)$, and the customers in S_i as $d_2(i)$, for each subset S_i. For simplicity of notation, we will refer to the first maximum distance $C(S_i)$, defined by (8), also as $d_1(i)$. Observe that, occasionally, it can be $d_1(i) = d_2(i)$ if the first and the second furthest customers in S_i have the same distance from $l(i)$.

The cost α_{jk} of each arc (j,k) connecting two regular nodes $j \in S_i$ and $k \in S_h$ in the customer improvement graph $G(S)$ is defined as:

$$\alpha_{jk} = \begin{cases} \max\{d_{l(h)j}, d_2(h)\} - C(S), & \text{if } d_{l(h)k} = C(S); \\ 0, & \text{otherwise.} \end{cases} \tag{15}$$

Equation (15) states that the cost of a generic arc (j,k) is equal to the cost variation of the subset S_h which contains k, provided that S_h is a bottleneck subset. If customer k is a bottleneck customer, the cost is given by the difference between the new maximum distance in S_h after the removal of k and the insertion of j ($\max\{d_{l(h)j}, d_2(h)\}$), and the bottleneck cost. Otherwise, the cost is 0.

Similarly, the cost of an arc (o,k), from the origin to a regular node $k \in S_h$, is:

$$\alpha_{ok} = \begin{cases} d_2(h) - C(S), & \text{if } d_{l(h)k} = C(S); \\ 0, & \text{otherwise.} \end{cases} \tag{16}$$

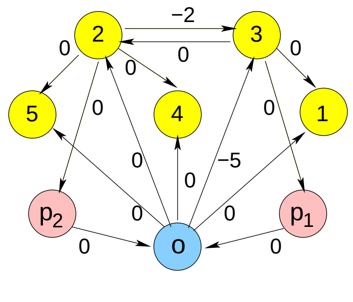
The arcs (j, p_i) from regular nodes to pseudonodes and the arcs (p_i, o) from pseudonodes back to the origin have null cost. From the above definitions it is easy to observe that an arc (j,k) or an arc (o,k) can have a negative cost only if k is the only bottleneck customer in the facility $F(k)$.

$$\begin{array}{c|ccccc|} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 4 & 2 & 8 & 10 \\ 2 & 4 & 0 & 5 & 5 & 4 \\ 3 & 2 & 5 & 0 & 7 & 8 \\ 4 & 8 & 5 & 7 & 0 & 2 \\ 5 & 10 & 4 & 8 & 2 & 0 \\ \hline \end{array} = [d_{ij}]$$

$S_1 = \{1,2\}$ $\quad l(1) = 1$

$S_2 = \{3,4,5\}$ $\quad l(2) = 4$

(a) Solution Data



(b) G(S)

Figure 2: Cost definition for SP-cycles

Figure 2 shows the arc costs α for the customer improvement graph associated with the problem instance of Figure 1. The current solution information and the distance matrix are now made explicit. Subset S_2 is the bottleneck subset in the solution, with bottleneck cost $d_{43} = 7$. The only arcs with non zero costs in $G(S)$ are the ones entering node 3, which represents the only bottleneck customer. In particular, arc $(o, 3)$ has cost -5 because the removal of 3 from S_2 reduces the cost of S_2 from 7 to $d_{45} = 2$. Arc $(2, 3)$ implies the insertion of customer 2 into S_2, besides the removal of 3. Such a move reduces the cost of S_2 by 2 units since the new maximum distance after the move is $d_{42} = 5$. Hence, the cost of the arc is -2.

According to the second definition, the cost β_{jk} of each arc (j, k) connecting two regular nodes $j \in S_i$ and $k \in S_h$ in the customer improvement graph $G(S)$ is:

$$\beta_{jk} = \begin{cases} \max\{d_{l(h)j}, d_2(h)\} - C(S), & \text{if } d_{l(h)k} = d_1(h); \\ \max\{d_{l(h)j}, d_1(h)\} - C(S), & \text{otherwise.} \end{cases} \tag{17}$$

Equation (17) states that the cost of a generic arc (j, k) measures how far from the bottleneck value is the cost of subset S_h after the insertion of customer j and the removal of customer k from it. Two cases are possible depending on whether k is the customer having the maximum distance from $l(h)$ in S_h or not.

The cost of an arc (o, k), from the origin to a regular node $k \in S_h$, is:

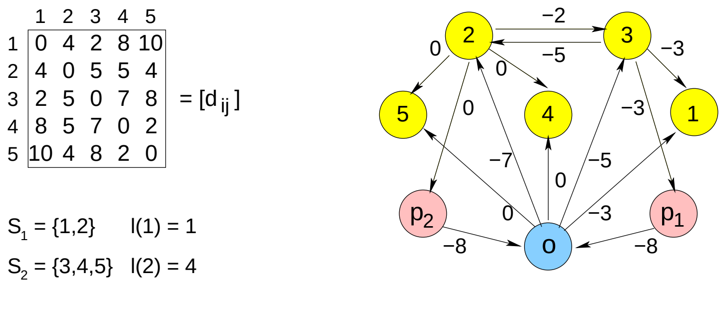
$$\beta_{ok} = \begin{cases} d_2(h) - C(S), & \text{if } d_{l(h)k} = d_1(h); \\ d_1(h) - C(S), & \text{otherwise.} \end{cases} \tag{18}$$

Each arc (j, p_h) from a regular node $j \in S_i$ to a pseudonode p_h has cost:

$$\beta_{jp_h} = \max\{d_{l(h)j}, d_1(h)\} - C(S). \tag{19}$$

Finally, the arcs (p_i, o) from pseudonodes back to the origin have cost $-M$, where M is a big positive number (for example, $M = C(S) + 1$). Note that, if the second cost definition is used, the number of arcs with negative cost is larger than in the case of α cost; one reason is that a cost β_{jk}, or a cost β_{ok}, can be zero only when $F(k)$ is a bottleneck facility, which remains bottleneck after the removal of customer k.

Figure 3 shows the arc costs β for the customer improvement graph associated with the problem instance of Figure 1. As an example, arc $(3, 2)$ in $G(S)$ has cost -5 because, after the insertion of customer 3 and the removal of customer 2, the new cost of subset S_1 is 5 units below the current bottleneck value. In this particular example, the maximum arc cost of each cycle in the graph reflects the effective reduction of the solution cost deriving from the move associated with the cycle. For instance, the maximum arc cost of cycle $(2, 3, 2)$ is -2 and, in fact, the cyclic exchange which swaps customer 2 and 3 reduces the



(a) Solution Data (b) G(S)

Figure 3: Cost definition for BP-cycles

cost of the new solution from 7 to 5. This is not always true when the current solution contains more than one bottleneck customer.

The above observations make more clear why the second definition of the arc costs suggests to detect subset-disjoint cycles in $G(S)$ through algorithms for computing bottleneck path trees in directed graphs, i.e., algorithms which look for paths that minimize the maximum arc cost.

3.3.2 Heuristics for SP-cycle computation

The heuristics developed for the detection of SP-cycles, denoted in the following as SP heuristics, are modifications of the well-known label-correcting algorithm for the shortest path problem (for more details, see [1]). From this approach, they inherit the use of a cost label, $d(j)$, and a predecessor index, $pred(j)$, associated with each node of the graph, and the fact that they operate by successively updating the cost labels, until they satisfy the optimality Bellman conditions, i.e., until:

$$d(k) \leq d(j) + c_{jk}, \qquad \forall \ (j,k) \in A(S), \tag{20}$$

where c_{jk} denotes the cost associated with the arc (j,k) in the arc set $A(S)$ of the customer improvement graph.

The main variations from this general framework concern the check for subset disjointness and the enforcement on the number of bottleneck elements present in the cycle. We propose two SP heuristics, $1S$ and tS, which look respectively for $1SP$-cycles and tSP-cycles through the use of the cost definitions (15)-(16). We first consider the functioning of the heuristic $1S$, and then indicate the appropriate changes needed for implementing tS. In the following we denote by nb_j the number of bottleneck customers included in the path from the origin of the graph to node j, and by Q the set of candidate nodes still to be examined.

The search of $1SP$-cycles is started at the origin o and, at each iteration, a node j is extracted from Q. The following steps are then executed until either Q is empty or a $1SP$-cycle is found. The path from the origin o to j is checked for disjointness. If the path is not disjoint, node j is discarded and another node is selected from Q. Otherwise, each arc (j, k) emanating from j is examined. If $d(k) > d(j) + c_{jk}$, then there are three cases to consider:

1. *Node k is a pseudonode.* If the current path from o to k is subset disjoint, and $nb_k \geq 1$ (this condition can be checked together with the disjointness of the path), then the addition of arc (k, o) to the path determines a $1SP$-cycle. The cycle corresponds to a profitable path exchange, and $1S$ stops.

2. *Node k is a regular node and the path from o to k is subset-disjoint.* In this case, the labels of node k are updated, i.e., $d(k) := d(j) + c_{jk}$ and $pred(k) := j$, and node k is inserted in Q, if not already present.

3. *Node k is a regular node and the path from o to k is not subset-disjoint.* If k belongs to the disjoint path from o to j, and at least one of the nodes in the portion of the path from k to j corresponds to a bottleneck customer (i.e., $nb_j - nb_{pred(k)} \geq 1$), a $1SP$-cycle has been found. The cycle corresponds to a profitable cyclic exchange, and $1S$ stops. Otherwise, arc (j, k) is discarded.

The heuristic tS is similar to $1S$. The main difference concerns the criterion driving the updating of the labels. Contrary to what we did in the previous case and to what has been suggested in [6], the heuristic looking for tSP-cycles favors the number of bottleneck customers rather than the costs when deciding whether to process an arc and, eventually, update the head node labels. This means that an arc (j, k), such that k is a regular node and the path from o to k is subset-disjoint, is examined and the labels of node k are updated only if:

- k is a bottleneck customer and $nb_j + 1 > nb_k$;
- k is not a bottleneck customer and $nb_j > nb_k$;
- k is a bottleneck customer, $nb_j + 1 = nb_k$ and $d_k > d_j + c_{jk}$;
- k is not a bottleneck customer, $nb_j = nb_k$ and $d_k > d_j + c_{jk}$.

The above conditions state that the primary criterion driving the label update of a node k is the number of bottleneck customers included in the paths to reach k. If both the path through j and the current path used to reach k have the same number of bottleneck elements, the decision is delegated to the usual optimality conditions associated with the cost labels. Obviously, case (1) and case (3) of the procedure for detecting $1SP$-cycles need also to be modified in tS, in the sense that the algorithm stops only if exactly t bottleneck customers belong to the detected subset-disjoint path or cycle ($nb_k = t$ in case (1), and $nb_j - nb_{pred(k)} = t$ in case (3)).

It is immediate to observe that, compared with the standard label correcting approach, the SP heuristics require an extra $O(p)$ time for each node selection from Q, in order to check the disjointness of the path (a disjoint path can involve at most p nodes). The checking of the number of bottleneck customers can be performed simultaneously.

Note that the search of SP-cycles can start also from a regular node. In such a case, the above presented search algorithm can be easily adapted in order to discover profitable path exchanges.

3.3.3 Heuristics for BP-cycle computation

The heuristics developed for the detection of BP-cycles, denoted in the following as BP heuristics, are based on the cost definitions (17)-(19), and they are a modification of the algorithm which looks for a bottleneck path tree in a directed graph. The bottleneck path tree paradigm can be derived from the shortest path label-correcting framework by simply substituting condition (20) with

$$d(k) \leq \max\{d(j), c_{jk}\}, \qquad \forall\ (j,k) \in A(S), \tag{21}$$

and by changing the label updating operation for those arcs violating the optimality conditions with $d(k) := \max\{d(j), c_{jk}\}$.

Therefore, the general scheme described for the heuristic $1S$ remains unchanged for $1B$, except for the different check of the optimality condition and the different updating of the labels of the nodes. The heuristic tB is analogous to tS, and also in this case it is the number of bottleneck customers rather than the costs to guide the arc processing and the label updating.

3.3.4 Some implementation issues

There are some issues related to the actual implementation of the algorithmic paradigms outlined so far which deserve particular notice, since the way they are handled can have a substantial impact on the effectiveness of the overall methodology.

One of these issues concerns the implementation of the set of candidate nodes Q, used by the cycle detection algorithms. In our empirical investigation we tested three possibilities of implementing Q, corresponding to the first-in first-out policy (*queue*), to the last-in first-out policy (*stack*) and to the selection of the node with the minimum cost label (*priority queue*).

Another issue concerns the dynamic construction of the customer improvement graph $G(S)$. Even if we characterized the subset-disjoint cycle detection procedures assuming that $G(S)$ had already been built, in practice we implicitly construct the graph during the search phase. That is, the existence of an arc (j, k) is checked during the cycle computation, when node j is extracted from Q and its forward star is built and examined.

The last remark is related to the visit of the paths. In our implementation, we applied the cycle detection heuristics with some regular node as the starting node of the paths. The choice of the starting node conditions the kind of cycles that are explored by the algorithm. Further, if we consider only one node as the starting node, we might miss many profitable exchanges. We thus searched the graph using every regular node as the starting node, stopping only when all these nodes have been tried unsuccessfully.

3.4 Recentering operations

Sometimes the detection of cycles corresponding to profitable multi-exchanges fails due to the restrictions on the arc existence imposed by conditions (c.3) and (c.6). An untimely stopping of the algorithm, in fact, can occur when the movement of all the bottleneck customers is barred by the excessive costs of assigning them to different subsets, if these costs, as stated by the aforementioned conditions, are only based on the current configuration of the receiving subsets. For instance, an arc (j, k), with $k \in S_h$, is left out of consideration by the algorithm if the cost of assigning j to $l(h)$ exceeds the current bottleneck cost. However, the move associated with (j, k) might lead to an improving solution if, after the insertion of j and the removal of k, we would change the active facility location for S_h.

We call *recentering operation* the operation which computes the optimal active location for a subset S_h, considering as candidate sites only the nodes, currently in S_h, which are not active locations for some subsets. The last condition is introduced in order to avoid the possibility of assigning the customers in S_h to a facility which already serves a different subset. The optimal active location of a subset is also referred to as the *local center* of the subset. The local center of a subset S_h can be computed by performing the following steps (Step 2 guarantees the feasibility of the local center with respect to the capacity constraints, whereas Step 3 excludes all the facilities which are currently active locations):

1. let $\bar{D}$ be the submatrix of the distance matrix which only contains the rows and the columns associated with the nodes in S_h;
2. discard all the rows j for which $s_j < W(S_h)$;
3. discard all the rows j for which $j = l(i)$ for some $i = 1, \dots, p$;
4. compute the maximum entry of each remaining row of $\bar{D}$;
5. the node which corresponds to the row with the minimum maximum entry is the local center for subset S_h.

The multi-exchange neighborhood can be enlarged by adding some arcs to the customer improvement graph with the scope of modelling recentering operations and, hence, allowing the relocation of some facilities. In order to limit the additional computational time needed to execute the local reoptimization, we restrict the use of the recentering operation only to those arcs whose tail node corresponds to a bottleneck customer, since those arcs are more likely to reduce the bottleneck costs.

Consider an arc (j, k) such that j is a bottleneck customer and $k \in S_h$. If (j, k) violates condition (c.3), but it satisfies conditions (c.1) and (c.2), we calculate the local center for the subset $S'_h = S_h \cup \{j\} \setminus \{k\}$, and we check if the following condition is satisfied:

(c.7) $d'_1(h) < C(S)$,

where $d'_1(h)$ is the cost of subset S'_h when the facility is located at its local center.

If (c.7) holds, we include (j, k) in the arc set $A(S)$ and assign a cost to it which depends on the cycle search heuristic in use.

Namely, if a heuristic of the SP family is used, the cost of the arc is:

$$\alpha_{jk} = \begin{cases} d'_1(h) - C(S), & \text{if } d_{l(h)k} = C(S); \\ 0, & \text{otherwise.} \end{cases} \tag{22}$$

If a BP heuristic is used, the cost is:

$$\beta_{jk} = d'_1(h) - C(S). \tag{23}$$

Similarly, for the arcs (j, p_h), if condition (c.6) fails, but conditions (c.4) and (c.5) are satisfied and j is a bottleneck customer, we recenter the facility of the new subset $S'_h = S_h \cup \{j\}$, we check condition (c.7) and, if this condition holds, we assign to the arc the cost:

$$\alpha_{jp_h} = \begin{cases} d'_1(h) - C(S), & \text{if } d_{l(h)k} = C(S); \\ 0, & \text{otherwise,} \end{cases} \tag{24}$$

in case of SP heuristics, and

$$\beta_{jp_h} = d'_1(h) - C(S), \tag{25}$$

for BP heuristics.

We will refer to the two families of heuristics which also include the recentering operation during the search phase as *SP-ER* and *BP-ER*, respectively. Clearly, if a cycle detected through *SP-ER* and *BP-ER* contains one or more arcs whose existence depends on the recentering operation, the new solution must be updated accordingly, by also changing the set of the active facility locations.

The execution of the recentering operation for each arc outgoing a bottleneck node might still be quite time-consuming. In order to mitigate the impact of such an operation on the overall time needed to detect a profitable exchange, we also propose an alternative scheme to *SP-ER* and *BP-ER*. The new scheme performs an *approximate recentering operation*, meaning that the local center used by the operation to check the existence of an arc is not necessarily the local center of the new subset resulting from the move associated with that arc. In fact, for each bottleneck customer, the local center is computed only once for each partition subset, and not once for every arc outgoing the bottleneck node. We indicate the two resulting approaches by *SP-AR* and *BP-AR*.

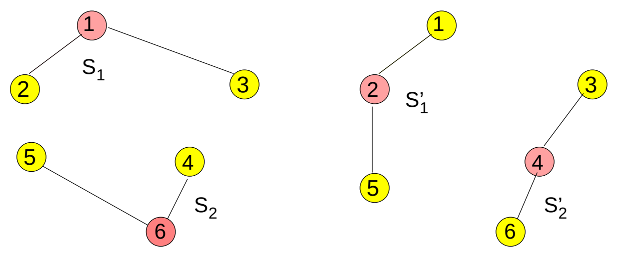
More in detail, *SP-AR* and *BP-AR* operate as follows (we recall that the arc existence is checked during the search phase): each time a bottleneck node $j \in S_i$ is extracted from Q, its forward star is scanned in clusters, where each cluster corresponds to a subset S_h with $h \neq i$. If a cluster S_h satisfies condition (c.3), meaning that $d_{l(h)j} < C(S)$, all his customers are processed in the usual way. Otherwise, if S_h cannot receive customer j because condition (c.3) is violated, we perform the recentering operation with respect to the subset $S'_h = S_h \cup \{j\}$, i.e., without taking into account the leaving customer. We then check condition (c.7). If the condition is violated, all the customers in the cluster are skipped. Otherwise, each arc (j, k) with k in S_h is treated in the usual way, by verifying if the disjointness and the feasibility conditions are satisfied, and processing it accordingly. If j is not a bottleneck customer, *SP-AR* and *BP-AR* operate in the standard way.

The costs used in this scheme to detect subset-disjoint cycles are still the ones defined by (22)-(24) or (23)-(25). Obviously, the local centers computed by *SP-AR* and *BP-AR* are approximations of the effective local centers of the subsets involved in the moves, since the removal of customers, which in this case is neglected by the operation, can affect the optimal position of the facilities. As a consequence, these two approaches might miss some profitable exchanges, which instead can be identified by *SP-ER* and *BP-ER*.

Figure 4 illustrates a case where the exact version of the recentering operation succeeds in detecting a profitable exchange, whereas the approximate version fails.

	1	2	3	4	5	6
1	0	3	6	7	8	10
2	3	0	9	7	4	9
3	6	9	0	3	10	6
4	7	7	3	0	6	2
5	8	4	10	6	0	5
6	10	9	6	2	5	0

(a) Distance Matrix



(b) Solution S (c) Solution S'

Figure 4: Exact Recentering Operation

The current solution S of a simple problem with 6 demand nodes is shown in Fig. 4 (b). During the cycle search, an arc such as $(5, 3)$ is generated only if we recenter the facility serving the customers in S_1. The recentering operation with respect to the subset $S_1' = S_1 \cup \{5\} \setminus \{3\} = \{1, 2, 5\}$ relocates the facility from node 1 to node 2. Since the cost of S_1', $d'(1) = d_{25} = 4$, is strictly less than the current bottleneck cost, $d_{45} = d_{13} = 6$, arc $(5, 3)$ is generated. Similarly, the recentering operation with respect to S_2' relocates the facility from node 6 to node 4.

The inclusion of such an arc in a cycle can produce an improving solution such the one shown in Fig. 4 (c). Had we considered the recentering operation with respect to the subset $S_1' = S_1 \cup \{5\} = \{1, 2, 3, 5\}$, we would have not moved the facility from node 1. Consequently, arc $(5, 3)$ (and also $(5, 2)$ and $(5, 1)$) would have not been generated, and improving solutions such as S' would have been incautiously disregarded.

Both the approximate and the exact versions of the recentering operation will be tested in our computational experimentation, to show the trade-off between the computational effort required by the two approaches and the quality of the resulting solutions.

4 The neighborhood search algorithms

In order to test the effectiveness of our neighborhood structure, we implemented fairly simple local improvement algorithms which use profitable multi-exchanges to produce improving solutions. In the context of this work, we did not make any specific effort to fine tune it, neither we investigated the possibility of incorporating the multi-exchange neighborhood structure in a more general metaheuristic framework, such as tabu search or simulated annealing.

Our neighborhood search algorithms perform rather elementary steps: they start with a feasible solution S, and subsequently apply one of the disjoint-cycle detection heuristics to find a profitable multi-exchange on the customer improvement graph $G(S)$. Every time a profitable multi-exchange is found, the current solution is updated accordingly. A single run would normally end when no more subset-disjoint cycles can be found. However, we add to this general and quite standard local search framework an additional step denoted as *relocation*, which, at the end of each run, tries to move the facilities to a better location relatively to the subset of customers they are currently serving. A simple way of performing the relocation is by executing the recentering procedure described in Section 3.4 for each subset S_h of the current solution, i.e., by relocating each facility at the local center of the customer subset assigned to it. We also consider an extension of this relocation strategy, in which the choice of the new center for a customer subset is not restricted to the nodes currently in that subset, but involves all of the non-active locations. In the following we will refer to the first approach as *restricted relocation*, while the term *relocation* will be used for the more general approach. If at least one facility is relocated during the relocation step, the search for profitable multi-exchange can be resumed from the new solution.

In accordance to the usual practice in local search schemes, we apply the algorithm several times, starting with different initial solutions. To generate multiple feasible solutions, we adopt two restart mechanisms.

1) Random Restart. In each restart, we randomly select p nodes from the set of candidate sites. If the capacity of the p nodes is not sufficient to cover the total demand of the customers, we generate another set of p locations. When a set of p nodes which can supply all the total demand is found, we open the p facilities at those nodes and assign the customers to them by solving a *Generalized Assignment Problem* (GAP).

GAP is solved through the greedy heuristic proposed in [13]. The solution thus obtained is then improved through a local reoptimization step.

2) Random Perturbation. The first initial solution is generated with the same procedure described for the Random Restart mechanism. In subsequent restarts, a feasible solution is computed by slightly perturbing the set of the active facility locations resulting from the last iteration, and assigning the customers to the new set of facilities by solving GAP. In particular, the perturbation is made by moving δ randomly selected facilities to δ locations, also randomly chosen, which do not already have a facility established at them and such that the total capacity of the new set of active locations can satisfy the total demand.

The employment of these restart mechanisms might seem quite questionable, especially for the use of GAP to determine the customer assignment. The objective function of GAP, in fact, is of a minsum nature, so that its solution does not give any guarantee of quality for CVPCP. In some preliminary experiments, we replaced GAP with a greedy algorithm specifically designed for CVPCP. However, the use of this procedure did not show any visible gain and, hence, the successive experimentation was carried out maintaining the restart version which employs the GAP solution. We are highly confident that the study of more elaborate and specialized restart mechanisms would add effectiveness to our solution methodology.

5 The computational study

We now present some computational experience. We initially considered all the implementation options: simple and strict profitability of the exchanges, SP-cycles and BP-cycles, random restart and random perturbation mechanisms, queue, stack and priority queue implementations of the set of the candidate nodes Q. From the preliminary results obtained, we then made an attempt at deducing general trends which make some particular choices functionally more advantageous than others. We dropped the alternatives which seem to be dominated with respect to the quality of the obtained solutions, so that we can restrict the final analysis of the results only to the most promising versions of the multi-exchange technique.

5.1 The test problems

To the best of the authors knowledge, no benchmark test problems for the capacitated vertex p-center problem exist in the literature. Therefore, we tested and compared our multi-exchange heuristics on three different data sets originally generated for other location problems.

An initial set of instances was drawn from the OR-Library [5]. The set contains 20 instances originally designed for the capacitated p-median problem. Each input data file contains information concerning the number of customers, the number of facilities to be sited, the demand and capacity at each node, and the coordinates of each customer location, i.e., the same information needed to set up a CVPCP instance. From the location coordinates, we computed the distance among the demand nodes by using Euclidean distances and rounding them down to the nearest integer. The OR-Library contains 2 groups of 10 instances, with 50 demand nodes and 5 facilities to be located, and 100 demand nodes and 10 facilities to be located, respectively. In all of the problems the capacity is supposed to be the same for every facility location.

The second set of instances was chosen to test the efficiency of our heuristics on large-scale problems. The set was made available by Lorena and Senne [12] for the capacitated p-median problem and comprises real data referring to the central area of São José dos Campos city. The set includes 6 large instances whose size ranges from 100 to 402 customers, and from 10 to 40 centers. Also in this case, all of the facility sites have equal capacity.

The third set of instances was included to show the behavior of our heuristics on problems with non-euclidean distances. The set includes 2 networks which were randomly generated by Galvão and ReVelle [7] for the maximal covering location problem. For both networks, the lengths of the arcs were sampled from a uniform distribution. Then, an all-pair shortest path algorithm was used to compute the distances among the demand nodes. Since the original data files, with 100 and 150 nodes respectively, did not contain information about the location capacities, we generated the capacities for each problem in two different ways. The first way assigns equal capacity to each site. The second way assigns to each location one of 3 possible capacity values (low, medium and high) with probability 0.4, 0.4 and 0.2 respectively. In both cases, the capacity values were chosen by taking into account the number of centers p, so that in the resulting problems the capacities were neither too low (unfeasible problems) or too high (trivial problems). For each problem, we generated the capacities for two different values of p (namely, $p = 5$ and p=10 for the first problem, and $p = 10$ and $p = 15$ for the second problem), thus obtaining a total number of 8 instances for this set. These instances are available at the address http://www.di.unipi.it/di/groups/optimize/research.

In the absence of existent terms of comparison, we validated the quality of the solutions found by the multi-exchange heuristics on the basis of the solutions obtained with the commercial CPLEX 7.0 mixed-integer optimizer. We supplied the CPLEX application with several problem-specific directives to improve its performance. For instance, we forced CPLEX to use the network optimizer at the initial node, so as to exploit the underlying network structure of CVPCP, and the dual simplex method after branchings. Further, we controlled the path that CPLEX traverses in the tree by assigning a branching priority to the variables y_i associated with the facility locations, since their value is likely to have a bigger impact on the objective function. We also induced CPLEX to emphasize feasibility rather than optimality as it solves the problems. Several experiments with different parameter settings proved that this combination yields consistent gains in efficiency and time consumption. In spite of this parameter calibration, the CPLEX application could not solve the large problems in the second set. The quality of the solutions for this set was hence estimated with respect to the best solution found by our heuristic implementations.

Both the CPLEX application and the multi-exchange heuristics were coded in C++, compiled with the GNU g++ compiler and run on a PC with a AthlonMP/2400Mhz processor and 1 Gb RAM, under the RedHat Linux 7.1 operating system.

For each multi-exchange variant, we executed 200 independent runs for the problems of the first and third sets, and 100 runs for the larger problems of the second set. The running times reported in the tables refer to the total number of runs.

5.2 The test results

We first present the results of some preliminary tests performed on the first problem set to test the impact of the restart mechanisms and the recentering operations.

Tables 5.1 and 5.2 report the results for the random restart and the perturbation restart mechanism, respectively. The results were produced by the *1S* heuristic, with Q implemented as a queue and the restricted relocation strategy. Each table collates the three recentering policies: no recentering (NR), approximate recentering (AR), and exact recentering (ER). The first column gives the problem name. The second column shows the objective value of the optimal solution found by CPLEX, z_{bk}. The next three columns give the objective function value returned by the multi-exchange heuristics, z_e, followed by three columns which report the computational times in seconds. Finally, the % deviations of the multi-exchange heuristics results from the best known solutions are given in the last three columns. The percentage deviation is calculated as:

$$\frac{z_e - z_{bk}}{z_{bk}} \times 100\%. \tag{26}$$

Table 5.1
Comparison based on the recentering operation for $1S$
(Options: queue and random restart)

Prob.	Opt.	Objective Value			Time			% Deviation		
No.		NR	AR	ER	NR	AR	ER	NR	AR	ER
p1	29	29	29	29	2.64	1.79	4.70	0.00	0.00	0.00
p2	33	33	33	33	3.60	2.63	6.42	0.00	0.00	0.00
p3	26	28	26	28	3.69	2.53	6.02	7.69	0.00	7.69
p4	32	32	32	32	3.62	2.73	6.66	0.00	0.00	0.00
p5	29	29	30	29	3.24	2.25	5.45	0.00	3.45	0.00
p6	31	32	32	32	3.26	2.48	5.63	3.23	3.23	3.23
p7	30	33	30	32	3.51	2.82	5.91	10.00	0.00	6.67
p8	31	31	32	32	3.15	2.25	5.54	0.00	3.23	3.23
p9	28	29	29	29	3.20	2.20	5.13	3.57	3.57	3.57
p10	32	36	36	36	3.59	2.43	5.59	12.50	12.50	12.50
p11	19	21	23	23	34.50	17.56	70.90	10.53	21.05	21.05
p12	20	24	22	23	34.44	19.20	68.92	20.00	10.00	15.00
p13	20	21	23	21	32.49	16.58	66.88	5.00	15.00	5.00
p14	20	23	22	21	35.87	19.10	63.43	15.00	10.00	5.00
p15	21	23	23	22	35.54	17.26	67.20	9.52	9.52	4.76
p16	20	22	22	21	38.37	20.11	70.59	10.00	10.00	5.00
p17	22	24	24	24	34.03	18.36	66.83	9.09	9.09	9.09
p18	21	23	23	23	31.70	15.80	65.17	9.52	9.52	9.52
p19	21	22	23	22	35.85	20.31	67.30	4.76	9.52	4.76
p20	21	24	23	22	34.21	17.26	61.88	14.29	9.52	4.76
Avg.					19.13	10.28	36.41	7.24	6.96	6.04

Table 5.2
Comparison based on the recentering operation for $1S$
(Options: queue and perturbation restart)

Prob.	Opt.	Objective Value			Time			% Deviation		
No.		NR	AR	ER	NR	AR	ER	NR	AR	ER
p1	29	29	29	29	2.68	1.81	4.41	0.00	0.00	0.00
p2	33	33	33	33	3.77	2.65	6.48	0.00	0.00	0.00
p3	26	28	28	26	3.89	2.52	5.71	7.69	7.69	0.00
p4	32	32	32	32	3.66	2.60	6.35	0.00	0.00	0.00
p5	29	30	29	29	3.48	2.17	5.95	3.45	0.00	0.00
p6	31	32	32	32	3.44	2.15	4.98	3.23	3.23	3.23
p7	30	32	32	33	3.71	2.63	6.36	6.67	6.67	10.00
p8	31	32	31	31	3.29	2.22	5.39	3.23	0.00	0.00
p9	28	29	29	29	3.38	2.14	5.35	3.57	3.57	3.57
p10	32	36	36	36	4.00	2.34	5.16	12.50	12.50	12.50
p11	19	23	20	22	31.93	13.94	60.55	21.05	5.26	15.79
p12	20	24	21	22	34.06	14.05	59.85	20.00	5.00	10.00
p13	20	23	22	21	31.93	12.88	54.70	15.00	10.00	5.00
p14	20	22	21	21	33.29	15.26	56.98	10.00	5.00	5.00
p15	21	22	22	23	35.36	13.11	60.42	4.76	4.76	9.52
p16	20	23	21	21	37.46	15.57	63.23	15.00	5.00	5.00
p17	22	23	24	23	33.54	15.15	58.01	4.55	9.09	4.55
p18	21	22	23	22	28.03	12.08	49.65	4.76	9.52	4.76
p19	21	23	22	23	35.25	14.26	61.10	9.52	4.76	9.52
p20	21	23	23	23	33.67	14.35	57.22	9.52	9.52	9.52
Avg.					18.59	8.19	31.99	7.72	5.08	5.40

From the analysis of these first two tables, we notice that in both cases, the two multi-exchange heuristics which implement the recentering operation outperform the version without recentering in terms of average percentage deviation. As expected, the version ER is slower than the other versions, but often provides the best results. The AR option proves to be a good trade-off between the solution quality and the computational time: it outperforms both the NR and ER versions when it comes to the CPU-time, and in some cases it even returns better solutions than the version with exact recentering. As for the restarting policies, when the relevant heuristics (AR and ER) are considered, the perturbation restart mechanism (Table 5.2) performs better than the random restart mechanism (Table 5.1), yielding better percentage deviation and running slightly faster.

The same tendencies (inferiority of the random restart and no recentering versions) were corroborated by the results obtained with the other cycle detection strategies, the stack and priority queue implementations of the set Q, and with the relocation mechanism. For the sake of brevity, we do not report the results for all the implementation options and refer the interested reader to the website http://www.di.unipi.it/di/groups/optimize/research, where exhaustive result tables are provided.

In the light of these preliminary observations, we restricted the subsequent analysis of the results to the algorithmic implementations which use the random perturbation restart mechanism and the recentering operations. In Tables 5.3 and 5.4, we compare the performance of the heuristics $1S$, tS, $1B$ and tB. Since the 4 heuristics showed a similar behavior for different implementations of Q and relocation strategies, we only report the results relative to the stack implementation and the restricted relocation. The two tables differ with respect to the recentering operation: the approximate recentering is used in Table 5.3, while Table 5.4 displays the results for the exact recentering.

Table 5.3
Comparison based on the cycle type
(Options: stack, perturbation restart and approximate recentering)

Pr.	Objective Value					Time				% Deviation			
No.	Opt.	$1S$	tS	$1B$	tB	$1S$	tS	$1B$	tB	$1S$	tS	$1B$	tB
p1	29	29	29	29	29	1.58	1.85	2.02	2.25	0.00	0.00	0.00	0.00
p2	33	33	33	33	33	2.66	2.84	3.47	3.96	0.00	0.00	0.00	0.00
p3	26	28	26	28	26	2.31	2.67	3.70	3.28	7.69	0.00	7.69	0.00
p4	32	32	32	32	32	2.92	2.76	3.81	3.73	0.00	0.00	0.00	0.00
p5	29	29	30	30	29	2.15	2.28	2.91	2.99	0.00	3.45	3.45	0.00
p6	31	32	32	32	32	2.08	2.33	2.92	2.97	3.23	3.23	3.23	3.23
p7	30	31	32	32	33	2.49	2.84	3.31	3.54	3.33	6.67	6.67	10.00
p8	31	32	34	31	31	2.16	2.57	3.05	2.95	3.23	9.68	0.00	0.00
p9	28	29	29	29	29	2.06	2.41	2.81	2.97	3.57	3.57	3.57	3.57
p10	32	35	35	35	32	2.41	2.60	3.49	3.37	9.38	9.38	9.38	0.00
p11	19	21	22	22	21	16.59	19.84	26.30	31.10	10.53	15.79	15.79	10.53
p12	20	23	24	25	23	15.28	21.13	30.33	30.36	15.00	20.00	25.00	15.00
p13	20	22	22	20	22	13.19	18.96	24.06	26.62	10.00	10.00	0.00	10.00
p14	20	20	21	21	22	14.67	21.26	27.75	33.60	0.00	5.00	5.00	10.00
p15	21	23	24	24	23	14.96	20.33	26.25	28.08	9.52	14.29	14.29	9.52
p16	20	21	23	21	21	16.42	22.38	33.41	38.14	5.00	15.00	5.00	5.00
p17	22	23	23	24	23	16.67	21.03	29.66	35.11	4.55	4.55	9.09	4.55
p18	21	23	23	21	23	12.88	20.93	19.59	23.69	9.52	9.52	0.00	9.52
p19	21	22	22	22	23	16.03	21.81	31.39	35.39	4.76	4.76	4.76	9.52
p20	21	23	21	23	23	16.14	25.91	25.12	35.16	9.52	0.00	9.52	9.52
Avg.						8.88	11.94	15.37	17.36	5.44	6.74	6.12	5.50

The two tables show that, apart from a few exceptions, the search strategies which look

Table 5.4
Comparison based on the cycle type
(Options: stack, perturbation restart and exact recentering)

Pr.	Objective Value					Time				% Deviation			
No.	Opt.	$1S$	tS	$1B$	tB	$1S$	tS	$1B$	tB	$1S$	tS	$1B$	tB
p1	29	29	29	29	29	4.65	5.44	5.36	5.91	0.00	0.00	0.00	0.00
p2	33	33	33	33	33	6.28	6.79	8.60	9.12	0.00	0.00	0.00	0.00
p3	26	28	26	26	26	5.50	6.33	7.57	7.82	7.69	0.00	0.00	0.00
p4	32	32	32	32	32	6.63	7.49	7.89	8.03	0.00	0.00	0.00	0.00
p5	29	29	30	29	30	5.67	6.13	7.15	7.70	0.00	3.45	0.00	3.45
p6	31	32	32	32	32	5.41	5.95	6.83	7.40	3.23	3.23	3.23	3.23
p7	30	30	33	30	30	6.06	7.22	7.75	8.36	0.00	10.00	0.00	0.00
p8	31	32	31	32	31	5.37	6.43	7.02	7.55	3.23	0.00	3.23	0.00
p9	28	29	29	29	29	5.27	5.86	6.45	6.71	3.57	3.57	3.57	3.57
p10	32	35	32	35	36	5.41	5.96	6.85	7.37	9.38	0.00	9.38	12.50
p11	19	22	22	21	22	63.01	93.03	98.02	118.77	15.79	15.79	10.53	15.79
p12	20	21	22	22	23	64.60	84.20	91.45	129.35	5.00	10.00	10.00	15.00
p13	20	20	22	20	22	61.74	82.94	95.41	105.60	0.00	10.00	0.00	10.00
p14	20	20	22	20	23	59.36	96.55	87.28	122.46	0.00	10.00	0.00	15.00
p15	21	22	23	21	22	62.84	94.90	96.00	120.09	4.76	9.52	0.00	4.76
p16	20	21	22	20	21	64.70	95.28	102.38	126.14	5.00	10.00	0.00	5.00
p17	22	23	24	23	24	63.40	99.66	92.40	129.22	4.55	9.09	4.55	9.09
p18	21	22	22	23	22	52.19	87.54	77.66	99.19	4.76	4.76	9.52	4.76
p19	21	22	22	22	23	66.68	103.10	101.14	132.76	4.76	4.76	4.76	9.52
p20	21	23	24	22	23	56.54	104.06	85.19	126.78	9.52	14.29	4.76	9.52
Avg.						33.67	50.24	50.02	64.22	4.06	5.92	3.18	6.06

for 1-cycles are generally preferable to the ones which look for t-cycles. In particular, the heuristic $1B$ in conjunction with the exact recentering policy (Table 5.4) is the one yielding the best results: it solves half of the problems to optimality, with an average deviation of only 3.18%. Further, even if it runs slightly slower than other implementations, it always takes less than about 2 minutes to solve the most difficult instances p11-p20, whereas CPLEX needs up to 15 hours and never less than 1.5 hours to solve the same problem instances.

Additional evidences of the heuristics behavior are provided in Table 5.5, which summarizes the results for the relevant implementation options (i.e., perturbation restart and recentering operations). The table, which displays the average percent deviation on the whole set, shows that no strict dominance can be established neither among different implementations of the set of candidate nodes Q, nor between the two relocation mechanisms. Also, it confirms that the bottleneck heuristic $1B$ outperforms the other cycle search strategies in most cases. Finally, we can deduce that the exact recentering is a more sound approach than the approximate recentering.

The tests performed on the second set of large problems reinforced most of the considerations already drawn for the first set. The dominance of the perturbation restart over the random restart mechanism and of the recentering operations over the no recentering version were even more pronounced on this set. We hence do no report the results for those dominated options (see the website for a complete account of the results) and focus on the most efficient implementations. Table 5.6 displays the average percent deviation

Table 5.5
Average percent deviation for OR Library (Beasley 1990) instances

Q	Recent.	Restricted Relocation				Relocation			
	Op.	$1S$	tS	$1B$	tB	$1S$	tS	$1B$	tB
queue	appr.	5.08	5.29	5.07	6.78	6.36	6.40	4.86	6.36
queue	exact	5.40	6.36	4.92	4.87	5.07	6.24	3.54	6.84
stack	appr.	5.44	6.74	6.12	5.50	6.99	7.16	4.49	6.98
stack	exact	4.06	6.02	3.18	6.06	4.65	6.88	4.04	6.52
pr. queue	appr.	5.56	6.63	4.82	6.29	6.23	6.19	5.65	6.23
pr. queue	exact	4.51	6.67	4.79	5.78	3.89	6.11	4.19	5.42

Table 5.6
Average percent deviation for Lorena and Senne (2003) instances

Q	Recent.	Restricted Relocation				Relocation			
	Op.	$1S$	tS	$1B$	tB	$1S$	tS	$1B$	tB
queue	appr.	12.82	13.91	12.80	12.52	10.03	10.63	10.31	11.42
queue	exact	6.59	6.99	13.64	11.43	8.99	9.75	9.55	11.15
stack	appr.	9.10	9.19	13.65	11.73	11.32	11.65	11.84	13.30
stack	exact	5.72	7.31	12.23	13.11	9.84	10.04	12.60	13.32
pr. queue	appr.	11.34	10.92	12.56	15.39	11.62	11.96	12.93	13.31
pr. queue	exact	8.36	9.01	6.43	11.07	6.89	8.26	12.56	12.10

with respect to the best known solutions found by our heuristics on the whole set. Even in this case, the exact recentering seems more robust than the approximate recentering, while no dominance is noticeable among the different implementations of Q. Again, none of the relocation mechanisms shows a consistent superiority over the other. It can be observed that on this set, the heuristics $1S$ and tS outperform their bottleneck counterparts (5 of the 6 best solutions were in fact found by the SP versions under different combinations of the other implementation options).

Finally, we report in Table 5.7 the results obtained on the set of problems characterized by non-euclidian distances. The percent deviation for this set was computed with respect to the best solutions found by CPLEX. For 4 of the 8 problems, these solutions were not proved to be optimal within the maximum execution time of 48 hours we imposed on the CPLEX application.

Table 5.7
Average percent deviation for Galvão and ReVelle (1996) instances

Recent.	Restart	Restricted Relocation				Relocation			
Op.		$1S$	tS	$1B$	tB	$1S$	tS	$1B$	tB
queue	appr.	6.71	6.97	6.31	6.86	5.86	7.24	5.92	6.43
queue	exact	5.46	6.40	5.04	6.43	4.93	5.33	5.74	5.05
stack	appr.	6.26	5.89	6.68	6.41	5.84	6.54	6.26	6.28
stack	exact	4.76	5.72	5.46	6.29	5.48	6.00	5.42	5.72
pr. queue	appr.	7.39	7.25	6.80	6.98	6.15	6.86	6.40	6.17
pr. queue	exact	5.28	5.45	6.14	5.59	4.93	5.18	5.60	5.17

The tests performed on this set confirm most of the algorithmic trends already noticed for the two previous sets. Note that in this case, as for the first set, the heuristics based on the search of 1-cycles perform slightly better. In particular, the version $1S$ with exact recentering and restricted relocation is the one providing the best average percent deviation (4.76%) on this set. Also, the heuristic $1S$ is steadily faster than the other heuristics, under each implementation option combination and for all of the problem sets (for the details related to the running times the reader is referred to the website).

It has to be noted that some of the best results for this last problem set were obtained when the relocation was used in combination with the random restart mechanism (see the website), while the restricted relocation continues to perform better with the perturbation restart scheme.

The results presented in this work refer to instances characterized by p/n ratios smaller than or equal to 10%. Empirical results from other tests showed that the performance of the proposed heuristics deteriorates for larger values of this ratio. We believe that this difficulty could be appreciably overcome by complementing the multi-exchange technique with more sophisticated restart mechanisms and more specific facility moves.

The execution times of the multi-exchange heuristics were in general quite modest, considering the size of the problems solved. The most computationally expensive implementations were the ones making use of the exact recentering operation. However, even the slowest implementations never took more than around 15 minutes to solve the largest problems (the ones with 400 nodes in the second set). For the other problems the time was remarkably lower. The computational times for each variant are listed in the tables available at the website.

6 Conclusions and future research

In this paper, we have presented some classes of multi-exchange algorithms for solving the capacitated vertex p-center problem, which is well-known to be one of the most difficult models in discrete location theory. We enhanced the effectiveness of the multi-exchange paradigm by more problem-specific tools, such as the recentering operation. The results are really encouraging. For small and medium size problems, even the less efficient implementations were able to find good approximations of the optimal or best known solutions, and their running times were very modest if compared to the times required by CPLEX. Furthermore, we were able to solve large-scale problems which can not be solved by general purpose optimization software packages.

Several areas of future work can still be considered to add efficiency to the multi-exchange methodology. First, new restart mechanisms, tailored to the characteristics of the problem, should be devised in order to produce better initial solutions. Second, the multi-exchange neighborhood structure potentials should be exploited inside more sophisticated metaheuristic frameworks, such as tabu search. Finally, the multi-exchange neighborhood could be extended to include the simultaneous movement of clusters of customers among facilities. We are confident that this extension may be particularly effective for CVPCP: the transferring of more customers at once, in fact, might yield significant changes to the configuration of each subset of customers, thus facilitating the relocation of the facilities.

Acknowledgements

Comments and suggestions of an anonymous referee were very appreciated. In particular, we wish to thank him for suggesting the extension of the relocation mechanism. The research has been partially supported by MIUR-SORSA and INDAM-GNAMPA.

References

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, Network Flows: theory, algorithms and applications, Prentice Hall, New Jersey, 1993.

[2] R.K. Ahuja, J.B. Orlin, S. Pallottino, M.P. Scaparra, and M.G. Scutellà, A multi-exchange heuristic for the single source capacitated facility location problem, TR 02-11, Dipartimento di Informatica, University of Pisa, 2002 (submitted to *Management Science*).

[3] R.K. Ahuja, J.B. Orlin, and D. Sharma, New neighborhood search structures for the capacitated minimum spanning tree problem, *Mathematical Programming* 91 (2001), 71-97.

[4] J. Bar-Ilan, G. Kortsarz, and D. Peleg, How to allocate network centers, *Journal of Algorithms* 15 (1993), 385-415.

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

[6] A. Frangioni, E. Necciari, and M.G. Scutellà, A multi-exchange neighborhood for minimum makespan machine scheduling problems, TR 00-17, Dipartimento di Informatica, University of Pisa, 2000 (to appear in *Journal on Combinatorial Optimization*).

[7] R.D. Galvão, and C. ReVelle, A Lagrangean heuristic for the maximum covering location problem, *European Journal of Operational Research* 88 (1996), 114-123.

[8] D.S. Hochbaum, and D. Shmoys, A unified approach to approximation algorithms for bottleneck problems, *Journal of the A.C.M.* 33 (1986), 533-550.

[9] O. Kariv, and S.L. Hakimi, An algorithmic approach to network location problems. Part 1: The p-centers, *SIAM Journal on Applied Mathematics* 37 (1979), 513-538.

[10] S. Khuller, and Y.J. Sussmann, The capacitated k-center problem, *SIAM Journal on Discrete Mathematics* 13(3) (2000), 403-418.

[11] M. Jaeger, and J. Goldberg, A polynomial algorithm for the equal capacity p-center problem on trees, *Transportation Science* 28(2) (1994), 167-175.

[12] L.A.N. Lorena, and E.L.F. Senne, A column generation approach to capacitated p-median problems, *Computers and Operations Research* 2003, to appear.

[13] H.E. Romeijn, and D. Romero Morales, A class of greedy algorithms for the generalized assignment problem, *Discrete Applied Mathematics* 103 (2000), 209-235.

[14] P.M. Thompson, and H.N. Psaraftis, Cyclic transfer algorithms for multi-vehicle routing and scheduling problems, *Operations Research* 41 (1993), 935-946.