Università di Pisa

Dipartimento di Informatica

Technical Report: TR-02-11

A multi-exchange heuristic for the single source capacitated facility location problem

R.K. Ahuja

Department of Industrial & Systems Engineering, University of Florida, Gainsville, FL, USA

ahuja@ufl.edu

J.B. Orlin

Sloan School of Management, MIT, Cambridge, MA, USA

jorlin@mit .edu

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

Dipartimento di Informatica, Università di Pisa, Pisa, Italy

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

12 Luglio 2002

ADDRESS: Corso Italia 40, 56125 Pisa, Italy.

TEL: +39 050 2212700

FAX: +39 050 2212726

A multi-exchange heuristic for the single source capacitated facility location problem

R.K. Ahuja

Department of Industrial & Systems Engineering, University of Florida, Gainsville, FL, USA

ahuja@ufl.edu

J.B. Orlin

Sloan School of Management, MIT, Cambridge, MA, USA

jorlin@mit .edu

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

Dipartimento di Informatica, Università di Pisa, Pisa, Italy

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

12 Luglio 2002

Abstract

This paper presents a very large scale neighborhood (VLSN) search algorithm for the capacitated facility location problem with single-source constraints. The neighborhood structures are induced by customer multi-exchanges and by facility moves. We consider both single-customer multi-exchanges, detected on a suitably defined *customer improvement graph*, and multi-customer multi-exchanges, detected on a *facility improvement graph* dynamically built through the use of a greedy scheme. Computational results for some benchmark instances are reported, which demonstrate the effectiveness of the approach for solving large-scale problems.

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

1 Introduction

The capacitated facility location problem with single source constraints, usually referred to as SSCFLP, is a well-known location problem which finds large applicability in many sectors, such as distribution systems planning or telecommunication networks design. The aim of the problem is to locate a number of facilities (e.g., plants, warehouses or concentrators), that have to serve at minimum cost a set of customers. The cost include fixed charges for opening the facilities and transportation costs for satisfying customer demands. Each customer j has an associated demand, w_j, that must be served by a single facility. Facilities can be located only at a finite number of prespecified sites, and there is a limit, s_i, to the total demand that a facility located at a site i can meet. The costs c_{ij} of supplying the demand of a customer j from a facility established at location i, as well as the fixed costs f_i for opening a facility at site i are known. Let

$$x_{ij} = \begin{cases} 1 & \text{if customer } j \text{ is assigned to a facility located at } i \\ 0 & \text{otherwise} \end{cases}$$

$$y_i = \begin{cases} 1 & \text{if a facility is located at candidate site } i \\ 0 & \text{otherwise} \end{cases}$$

Then, the problem can be stated mathematically as follows.

$$\begin{aligned} \min \quad & \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} + \sum_{i \in I} f_i y_i && (1) \\ \text{s.t.} \quad & \sum_{i \in I} x_{ij} = 1 \qquad \forall\ j \in J && (2) \\ & \sum_{j \in J} w_j x_{ij} \leq s_i y_i \qquad \forall\ i \in I && (3) \\ & x_{ij} \in \{0,1\} \qquad \forall\ i \in I, j \in J && (4) \\ & y_i \in \{0,1\} \qquad \forall\ i \in I, && (5) \end{aligned}$$

where $I = \{1, \ldots, n\}$ is the set of potential locations, and $J = \{1, \ldots, m\}$ is the set of customers.

The expression (1) establishes that the objective of the problem is the minimization of the sum of the fixed costs for opening facilities plus the shipment costs of meeting the customer demands. Assignment 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 maximum capacity; finally, the last two sets of constraints represent the integrality requirements.

SSCFLP belongs to the class of NP-hard problems. One of the first attempts to solve it to optimality is due to Neebe and Rao [13], who propose a branch and bound scheme based on the formulation of SSCFLP as a partitioning problem. Their method combines a column generation procedure with the use of a linear relaxation at each node of the enumeration tree to obtain bounds to the optimal solution. However, the use of exact methods for solving SSCFLP is restricted to small and medium size problems.

The most successful and extensively studied approaches to solve large instances of SSCFLP have been developed in the Lagrangean heuristic framework.

Barcelo and Casanova [3] propose a Lagrangean heuristic where they dualize the assignment constraints (2) and employ a two phase heuristic for finding feasible solutions:

the first phase selects a set of open plants; the second phase solves a generalized assignment problem to assign customers to this set through the use of a regret heuristic.

Klincewicz and Luss [12] devise a Lagrangean heuristic based upon relaxing the capacity constraints (3), thereby obtaining as Lagrangean subproblem an uncapacitated facility location problem. These subproblems are solved suboptimally using the dual ascent heuristic introduced by Erlenkotter [8]. An initial solution is provided by an add heuristic, while a final adjustment heuristic, based on cost differentials, is used to decrease the cost of the solution obtained from the Lagrangean phase.

Sridharan [17], as Barcelo and Casanova [3], dualizes the assignment constraints (2). He then separates the Lagrangean subproblem into n knapsack problems, one for each facility, and from their combined solutions gets a lower bound to the problem with a set of open facilities. This set of open facilities is then used to setup a single source transportation problem, whose optimal solution, obtained through branch and bound, provides a feasible solution and an upper bound to the original problem.

A similar approach based on the relaxation of the assignment constraints and the solution to a number of knapsack problems is devised by Pirkul [14]. According to a comprehensive study made by Beasley [5], Pirkul's heuristic is the one providing the best feasible solutions among the approaches mentioned so far.

In the same work [5], Beasley also proposes a different Lagrangean heuristic based upon relaxing both the demand constraints and the capacity constraints. The merit of his approach lies in its robustness, since it provides good quality solutions across a range of different location problems such as p-median, uncapacitated, capacitated and single source facility location problems.

Agar and Salhi [1] introduce efficient modifications to the Lagrangean framework proposed by Beasley. They also dualize both the assignment constraints (2) and the capacity constraints (3), and solve the Lagrangean problem by inspection to obtain a set of open facilities. A primal feasible solution is then built by solving a generalized assignment problem in two phases: they first solve an uncapacitated problem to assign customers to the open plants; then restore feasibility by a re-assignment procedure based on the definition of penalties and the solution of a knapsack problem.

Hindi and Pieńkosz [10] also employ Lagrangean relaxation dualising constraints (2). The calculation of lower bounds is along the same lines employed by Sridharan [17]. However, feasible solutions and upper bounds are computed by a different procedure that combines a greedy constructive heuristic, based on maximum regret, with a restricted neighborhood search. The authors report marginal improvements in the solution of some test problems previously solved by several search algorithms introduced by Delamaire et al. [7], and a branch and bound scheme proposed by Barcelo et al. [4].

A Lagrangean heuristic approach based on a repeated matching algorithm is described in the work by Rönnqvist et al. [16]. The authors dualize the assignment constraints (2) and use subgradient optimization. To obtain upper bounds from the Lagrangean solutions, they introduce a strong primal heuristic that solves a sequence of matching problems by deriving the matching costs from the solution of a number of knapsack problems. The results reported by the authors on three randomly generated sets of problems show some improvements compared to the results reported by Pirkul [14].

Holmberg et al. [11] incorporate the repeated matching Lagrangean heuristic into a branch and bound framework, and compare the resulting approach to the commercial code CPLEX.

The objective of this work is to propose a very large scale neighborhood (VLSN) technique for SSCFLP. The method consists of the alternating use of customer exchanges and facility moves which allow the replacement of the current solution with an improved solution in its neighborhood. Customer exchanges mainly affect the customer assignment among facilities already located, and are detected by searching for special paths or cycles on suitably defined improvement graphs. Facility moves aim at finding improving configurations of the open facilities. The neighborhood structures thus obtained are embedded in a local improvement algorithm which starts with a feasible solution and alternatively performs improving customer and facility moves until it gets a locally optimal solution. Restart mechanisms are also considered.

The rest of the paper is organized as follows. In Section 2, we set up some useful notations. Section 3 introduces some VLSN structures for SSCFLP. In Section 4, we describe the neighborhood induced by single-customer multi-exchanges and state the equivalence of the problem of finding improving moves in this neighborhood and the problem of detecting negative subset-disjoint cycles on the customer improvement graph. In Section 5, we address the multi-customer multi-exchange case and describe how to construct a dynamic graph through the use of a greedy scheme to detect cost-decreasing solutions in such a neighborhood. Section 6 is dedicated to the description of the facility neighborhood structure induced by facility opening, closing and transferring moves. Section 7 addresses the main issues concerning the implementation of local search procedures, while Section 8 reports the empirical results for a large suite of test problems. Finally, Section 9 presents conclusions and suggestions for further refinements and developments of the multi-exchange technique for SSCFLP.

2 Notation

Let S be a feasible solution to the problem (1)-(5). The set S can be represented as a partition of the customer set J into n subsets, i.e., $S = \{S_1, S_2, \ldots, S_n\}$, where each subset S_i, $i = 1, \ldots, n$, corresponds to a potential facility site and contains the set of customers assigned to that facility in the solution S. Eventually, S_i can be empty if no facility is established at location i.

The cost of each subset S_i, $i = 1, \ldots, n$, is given by

$$C(S_i) = \begin{cases} \sum_{j \in S_i} c_{ij} + f_i, & \text{if } S_i \neq \emptyset, \\ 0, & \text{otherwise,} \end{cases} \tag{6}$$

and the cost of the whole partition S is

$$C(S) = \sum_{i=1}^{n} C(S_i). \tag{7}$$

A solution S is feasible if the following inequality holds for $i = 1, \ldots, n$:

$$W(S_i) = \sum_{j \in S_i} w_j \leq s_i. \tag{8}$$

The *residual capacity*, r_i, of each subset S_i, i.e., the capacity still available at a facility located at i, is defined as:

$$r_i = s_i - W(S_i) \geq 0. \tag{9}$$

In the following, we will denote by $I(S)$ the set of open facilities in the current solution S, and by $F(j)$, $j \in J$, the facility serving customer j.

3 Very large scale neighborhood structures

The first VLSN structure we will consider for solving SSFLP is the customer neighborhood. Given a solution S to SSCFLP, the customer neighborhood of S is defined as the set of new solutions obtainable from S by exchanging customers among facilities in a cyclic manner. Such a neighborhood is constructed by moving among the facilities either single customers or subsets of customers conveniently chosen.

In particular, the single-customer neighborhood is defined in terms of single-customer cyclic or path exchanges as follows.

A *single-customer cyclic exchange* with respect to a solution S is defined as a customer sequence $R = (j_1, j_2, \ldots, j_q)$, 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 $r \neq s$, $r, s = 1, \ldots, q$. The cyclic exchange represents the following exchange of customers: customer j_1 is relinquished from facility $F(j_1)$ and assigned to facility $F(j_2)$, customer j_2 is relinquished from facility $F(j_2)$ and assigned to facility $F(j_3)$, and so on until customer j_q is relinquished from facility $F(j_q)$ and assigned to facility $F(j_1)$.

A cyclic exchange R is *feasible* if the capacity of each facility involved in the cycle is still satisfied after the exchange, i.e., if $w_{j_{r-1}} \leq r_{F(j_r)} + w_{j_r}$ for $r = 2, \ldots, q$, and

$w_{j_q} \leq r_{F(j_1)} + w_{j_1}$. A cyclic exchange R is *profitable* if the new solution S' obtained from S through R is such that $C(S') < C(S)$.

A *single-customer path exchange* $P = (j_1, j_2, \ldots, j_{q-1}, F_q)$ is similar to a single-customer cyclic exchange, but in this case the last element of the sequence, F_q, represents a facility location, and $F(j_r) \neq F(j_s) \neq F_q$ for $r \neq s$, with $r, s = 1, \ldots, q-1$. Such an exchange excludes movements of customers from F_q to $F(j_1)$. The definitions of feasibility and profitability for path exchanges are analogous to the ones for cyclic exchanges.

Each feasible single-customer cyclic or path exchange generates a new feasible solution S' from the current solution S. The set of all the new solutions obtainable from S through a feasible exchange R or P defines the single-customer multi-exchange neighborhood of S.

In order to efficiently detect improving single-customer moves, our approach does not enumerate and evaluate the large number of neighbors generated by single-customer exchanges. Instead, the neighborhood is searched only partially using a heuristic approach which detects special negative cost cycles in a suitably defined graph, called the *customer improvement graph*. This will be described in the following section.

Similarly, we define the multi-customer neighborhood in terms of multi-customer cyclic or path exchanges. Let us denote by $Q = \{i_1, i_2, \ldots, i_q\}$ a sequence of indexes of q subsets of the partition $S = \{S_1, S_2, \ldots, S_n\}$.

A *cyclic multi-customer exchange* is a sequence $U = \{U_{i_1}, U_{i_2}, \ldots, U_{i_{q-1}}, U_{i_q}\}$ of q sets of customers such that:

i) $U_{i_r} \subseteq S_{i_r}$ for $r = 1, \ldots, q$.

ii) Each subset U_{i_r} of the sequence belongs to a different subset S_i.

Any cyclic multi-customer exchange U uniquely defines a new solution $S' = \{S'_1, \ldots, S'_n\}$ in the neighborhood of S, obtained by shifting customer subsets along the sequence. Formally:

$$S'_{i_r} = \begin{cases} S_{i_r}, & \text{if } i_r \notin Q; \\ S_{i_r} \cup U_{i_{r-1}} \setminus U_{i_r}, & \text{if } i_r \in Q \text{ and } r = 2, \ldots, q; \\ S_{i_r} \cup U_{i_q} \setminus U_{i_r}, & \text{if } i_r \in Q \text{ and } r = 1. \end{cases} \tag{10}$$

The cyclic multi-customer exchange U is *feasible* if the capacity constraints are satisfied for each partition set S'_{i_r} of the new solution S'. A cyclic multi-exchange U is *profitable* if the new solution S' obtained from S through U is such that $C(S') < C(S)$.

A *path multi-customer exchange* is similar to a cyclic multi-exchange, but there is no subset of elements moving from S_{i_q} to S_{i_1}. A path multi-exchange is *feasible* if each modified partition subset is still feasible after the exchange, and *profitable* if the total cost of the subsets strictly decreases.

It is evident that the size of the multi-customer neighborhood can be excessively large if we consider all the possibilities of moving clusters of customers among facilities. In order to overcome this difficulty, first we try to reduce the number of possible clusters by limiting their size to a maximum of K elements, where K is an algorithm parameter. Second, we suggest a method to include in the neighborhood generation only those clusters which are more likely to produce improving neighbor solutions. This task is accomplished in the dynamic generation of the *facility improvement graph.*

The second VLSN structure we consider is the neighborhood induced by facility moves, which is defined as the set of solutions obtainable by changing the state of one facility from close to open or vice versa, or by swapping used and unused sites. A facility move can be performed by leaving the customer assignment almost unchanged or, occasionally, may require a complete reallocation of all customers. In practice, we only search a restricted facility neighborhood in the sense that only moves involving promising facilities are attempted.

4 Single-customer multi-exchanges

We start our discussion by describing how to build and explore the single-customer multi-exchange neighborhood for SSCFLP.

4.1 The Customer Improvement Graph

Given a solution S of SSCFLP, the *customer improvement graph* relative to S is a directed graph $G^c(S) = (N^c(S), A^c(S))$ defined as follows.

The set of nodes $N^c(S)$ contains a *regular node*, j, for each customer $j \in J$, a *pseudonode*, p_i, for each facility location i, and an *origin* node, o. The pseudonodes are added to $N^c(S)$ to model exchanges along paths rather than cycles.

The set of arcs $A^c(S)$ contains an arc (j, k) for each pair of regular nodes j and k in $N^c(S)$, provided that customers j and k are assigned to two different facilities in S, i.e., $F(j) \neq F(k)$, and that after the insertion of j in $S_{F(k)}$ and the removal of k from it, the capacity of facility $F(k)$ is not exceeded, i.e., $w_j - w_k \leq r_{F(k)}$. An arc connecting two nodes j and k, in fact, signifies that customer j leaves facility $F(j)$ and is assigned to facility $F(k)$, which in turn relinquishes customer k.

The cost α_{jk} of the arc (j, k) reflects the cost variation of facility $F(k)$ and it is therefore defined as

$$\alpha_{jk} = c_{F(k)j} - c_{F(k)k}. \tag{11}$$

$A^c(S)$ also contains an arc (j, p_i) from each regular node j to each pseudonode p_i, provided that customer j is not served by facility i in S, i.e., $F(j) \neq i$, and that j's demand does not exceed the residual capacity of facility i, i.e., $w_j \leq r_i$. An arc (j, p_i) signifies that customer j is moved to facility i but no customer is relinquished from i, and its cost is simply

$$\alpha_{jp_i} = c_{ij}. \tag{12}$$

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

$$\alpha_{oj} = \begin{cases} -c_{F(j)j}, & \text{if } |S_{F(j)}| > 1; \\ -c_{F(j)j} - f_{F(j)}, & \text{if } |S_{F(j)}| = 1. \end{cases} \tag{13}$$

Each arc (p_i, o) has cost:

$$\alpha_{p_io} = \begin{cases} f_i, & \text{if } S_i = \emptyset; \\ 0, & \text{otherwise.} \end{cases} \tag{14}$$

4.2 Negative subset-disjoint cycles

Having provided a full description of the customer improvement graph, we can now define negative subset-disjoint cycles in it [18].

Definition 4.1 *A directed cycle $(n_1, \ldots, n_q)$, with $n_r \in N^c(S)$, $r = 1, \ldots, q$, in the improvement graph $G^c(S)$ associated with solution S, is a negative subset-disjoint cycle if each one of its nodes, except the origin (if present) is associated with a different facility location and if the sum of the costs of its arcs is negative.*

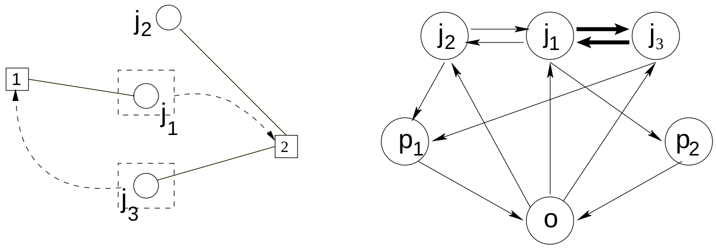
We can now state the equivalence between the problem of detecting a negative subset-disjoint cycle in $G^c(S)$ and the problem of finding a feasible profitable multi-exchange for S.

Property 4.1 *There is a one-to-one correspondence between the set of feasible cyclic [path] exchanges relative to the current solution S and the set of subset-disjoint cycles in $G^c(S)$. Furthermore, each cyclic [path] exchange is a profitable exchange if and only if the corresponding subset-disjoint cycle is negative.*

In particular, the fact that the customer improvement graph only includes feasible arcs with respect to the capacity constraints ensures that each exchange corresponding to a cycle in $G^c(S)$ is a feasible exchange. Furthermore, the way we define the costs of the arcs in $A^c(S)$ guarantees that the cost of each cycle in $G^c(S)$ accurately reflects the cost variation of the solution due to the corresponding customer exchanges. Hence, if the cycle cost is negative, the associated exchange is profitable, and vice-versa.

The node sequence $(n_1, \ldots, n_q)$, with $n_r \in N^c(S)$, $r = 1, \ldots, q$, corresponds to a path exchange if the subset-disjoint cycle contains the origin and a pseudonode; otherwise, it corresponds to a cyclic exchange.

Figure 1 shows the customer improvement graph associated with the solution S of a simple location problem with 2 facilities and 3 customers. A cyclic single-customer exchange involving customers j_1 and j_3 (a) and its corresponding cycle in $G^c(S)$ (b) are also illustrated.



(a) Solution S (b) $G^c(S)$

Figure 1: Customer Improvement Graph

4.3 Search for negative subset-disjoint cycles

As a consequence of Property 4.1, the problem of finding an improving move in the single customer neighborhood can be reformulated as the problem of identifying negative-cost subset-disjoint cycles in $G^c(S)$. This problem is known to be NP-complete [18]. However, the heuristic method proposed by Ahuja et al. [2] for the Capacitated Minimum Spanning Tree problem is able to detect such cycles very effectively, missing some of them only rarely. Hence, we used the same heuristic in our solution approach. The algorithm is based on a modification of the well known label-correcting algorithm for the shortest path problem. The main changes are introduced to check for path subset-disjointness. The modified label-correcting algorithm is run several times with different nodes as origin, since its success in finding negative subset-disjoint cycles is strictly related to the node from which the search is started. The reader is referred to [2] for further details concerning this algorithm.

5 Multi-customer multi-exchanges

Previous scientific works dealing with multi-exchange techniques (see for example [2, 9, 19, 18]), usually adopt a standard methodology for building the improvement graph which is basically the one we have just described for the single-customer move. Namely, each node of the graph corresponds to an element of the partition that can be moved, and each arc represents the transfer of the "tail" element toward the partition subset currently containing the "head" element.

This scheme becomes impractical when modeling multi-customer multi-exchanges, due to the exponential number of clusters that should be considered for movement. We therefore resort to a different variant to limit the graph growth, which consists of constructing the improvement graph dynamically and uses a greedy scheme to select only promising customer subsets to be moved between pairs of nodes. As it will become more explicit in the following sections, the exact meaning of each arc is established and becomes available only during the search phase, since only at that stage the moving customer subset is chosen, on the basis of the information gathered along the path up to that arc. Also, it is important to emphasize the fact that an arc can represent different customer subsets in the same execution of the search algorithm.

5.1 The Facility Improvement Graph

The *dynamic facility improvement graph* $G^f(S) = (N^f(S), A^f(S))$ used to model multi-customer multi-exchanges has a *regular node* i for each open facility, a *pseudonode*, p_h, for each facility location $h \in I$, and an *origin*, o.

The set of arcs $A^f(S)$ can contain an arc (i, h) between each pair of regular nodes. Such an arc indicates that the facility established at location i relinquishes a set of up to K customers which are then assigned to the facility located at h. It also implies the subsequent removal of a subset of at most K customers from S_h. The selection approach for choosing a convenient subset of up to K customers to be transfered from S_i to S_h, denoted as U_{ih}, will be described in the next section. Such a subset may not exist, in which case (i, h) is not included in the arc set. If it is included, its costs β_{ih} is defined as

$$\beta_{ih} = \sum_{j \in U_{ih}} c_{hj} - \sum_{j \in U_{ih}} c_{ij}. \tag{15}$$

The arc set $A^f(S)$ also contains an arc (i, p_h) from each regular node i to each pseudonode p_h, with $i \neq h$, with cost:

$$\beta_{ip_h} = \sum_{j \in U_{ih}} c_{hj} - \sum_{j \in U_{ih}} c_{ij}. \tag{16}$$

An arc (i, p_h) implies that no subset of customers is relinquished from facility h.

Additionally, $G^f(S)$ includes a set of directed arcs (o, i) which connect the origin to every regular node, plus a set of directed arcs (p_h, o) from each pseudonode p_h back to the origin. Each arc (o, i) has null cost, unless the subsequent removal of a customer subset from S_i leaves facility i empty, in which case its cost is $-f_i$. Such a condition can be checked only at running time, when the customer subset leaving facility i to another facility becomes available. Finally, the cost of each arc (p_h, o) is null if facility h has already customers assigned to it; otherwise, it is equal to the fixed cost, f_h, for opening it.

The inclusion of the origin o and of the pseudonodes p_h in the dynamic facility improvement graph has the usual rationale of modeling path exchanges. The use of path exchanges is particularly relevant in the multi-customer neighborhood since it frequently leads to changes in the facility locations. This possibility would be excluded if we restricted the search to the cyclic multi-customer exchange neighborhood only.

Figure 2 represents the dynamic facility improvement graph associated with the solution S of a problem with 9 customers and 4 facility locations. It also shows the path exchange corresponding to the cycle $\{o, 3, 1, 2, p_4\}$.

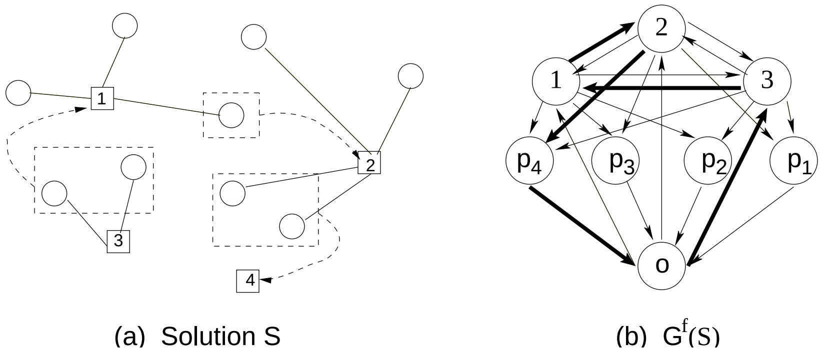


Figure 2: Dynamic Facility Improvement Graph

5.2 Negative subset-disjoint cycles

Definition 5.1 *A sequence of nodes $(n_1, \ldots, n_q)$ in the dynamic facility improvement graph $G^f(S)$, associated with solution S, is a negative subset-disjoint cycle if each one of its nodes, except the origin (if present) is associated with a different facility location, and if its overall cost is negative. The cost of a cycle is simply defined as the sum of the costs of its arcs.*

The problem of finding some feasible profitable multi-customer multi-exchanges for S can then be translated into the problem of detecting negative subset-disjoint cycles in $G^f(S)$. The correspondence between cycles in $G^f(S)$ and path or cyclic exchanges is based on the property that the neighborhood search algorithm not only finds subset-disjoint cycles, but also produces for each cycle a sequence of customer subsets which move among the facilities involved in it. Further, the method used for determining such customer subsets guarantees that their transfer along the sequence produces only feasible exchanges. These observations lead to the following property.

Property 5.1 *Each sequence of customer subsets uniquely identified by a subset disjoint cycle in $G^f(S)$ corresponds to a feasible cyclic or path exchange with respect to S. Furthermore, if the subset-disjoint cycle is negative, the corresponding cyclic or path exchange is a profitable exchange.*

The customer subset sequence corresponds to a path exchange if the subset-disjoint cycle contains the origin and a pseudonode; otherwise, it corresponds to a cyclic exchange.

It is to be noted that in this case the correspondence between cycles in $G^f(S)$ and path or cyclic exchanges for S is not a one-to-one correspondence. In fact, each feasible subset-disjoint cycle in $G^f(S)$ corresponds to a feasible exchange, but not all the possible feasible exchanges are mapped onto cycles in the dynamic facility improvement graph. The lack of this correspondence in both directions depends on our restriction to consider only a limited number of moving customer subsets.

5.3 Selection of moving customers

Before we proceed to the description of the customer selection policy, we introduce the definition of *candidate leaving subsets*.

Definition 5.2 *Given a customer subset S_h and a cluster of customers $U_e \notin S_h$, a subset U_l of S_h is a candidate leaving subset w.r.t. S_h and U_e if it satisfies the following condition: $W(U_l) \geq W(U_e) - r_h$.*

In other words, a subset of customers U_l is a candidate leaving subset w.r.t. S_h and an entering subset U_e if the capacity limit of facility h is not violated after the insertion of U_e in S_h and the removal of U_l from it.

Assume that, during the search for a subset-disjoint cycle, node h is reached from node i, and that we know the subset of customers U_{ih} entering S_h from S_i, as depicted in Figure 3.

Our customer selection policy consists in determining, for each node l reachable from node h, a subset of customer, $U_{hl} \in S_h$, to be reassigned from facility h to facility l in such a way to minimize the total cost for the reassignment. U_{hl} must be a candidate leaving subset w.r.t. S_h and U_{ih}, and can contain at most K customers.

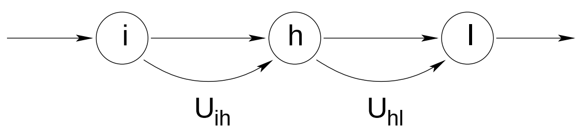


Figure 3: Subset Selection

Such a selection problem, SP, can be mathematically stated as a combinatorial optimization problem as follows.

$$\min \quad \sum_{j \in S_h} (c_{lj} - c_{hj}) x_j \tag{17}$$

$$\text{s.t.} \quad \sum_{j \in S_h} x_j \leq K \tag{18}$$

$$\sum_{j \in S_h} w_j x_j \geq W(U_{ih}) - r_h \tag{19}$$

$$x_j \in \{0, 1\} \qquad \forall\ j \in S_h. \tag{20}$$

The decision variable x_j takes value 1 if customer j is selected for the transfer from S_h to S_l, and 0 otherwise. Constraint (18) ensures that at most K customers are selected; constraint (19) guarantees that the selected customer subset is a candidate leaving subset with respect to the entering subset U_{ih} and S_h. The objective is to minimize the reassignment costs. A solution to the problem SP uniquely determines a subset U_{hl} by simply setting $U_{hl} = \{j \in S_h | x_j = 1\}$.

There are two situations in which the selection of the leaving subset U_{hl} requires the solution of a modified version of the selection problem SP. The first occurs when $i = o$, since in this case there is no subset moving from the origin into node h. This implies that the capacity constraint (19) is always satisfied, and SP can be easily solved by sorting the elements of S_h by non decreasing costs $(c_{lj} - c_{hj})$, and selecting the first in the order which have negative costs, up to a maximum number K. The second situation occurs when node l already belongs to the path from the origin to node h, implying that a subset-disjoint cycle has been detected. This cycle identifies a cyclic exchange along the sequence of facilities $l = i_1, i_2, \ldots, i_q = h$. When selecting the last customer subset $U_{i_q i_1}$ of the exchange, we must guarantee not only that its removal from $S_h = S_{i_q}$ restores the capacity constraint for facility h, according to (19), but also that its insertion in $S_l = S_{i_1}$ does not violate the capacity limit of facility l. Hence we must add to problem SP the additional constraint:

$$\sum_{j \in S_{i_q}} w_j x_j \leq r_{i_1} + W(U_{i_1 i_2}). \tag{21}$$

Problem SP has the structure of a knapsack problem in which an upper bound is imposed on the number of items that can be selected. Interesting approximation algorithms for knapsack problems with cardinality constraints have been proposed by Caprara et al. in [6]. In our experimental investigation, however, we simply implemented a greedy algorithm.

5.4 Search for negative subset-disjoint cycles

The algorithm we use to explore the facility improvement graph is built along the general lines of a label-correcting algorithm. However, it has to be noticed that the peculiar dynamic nature of $G^f(S)$ invalidates the shortest path optimality conditions on which label-correcting algorithms are based. Namely, we cannot state that if P is an optimal path from a node s to a node t in $G^f(S)$, and i is one of its intermediate nodes, then the subpath of P from s to i is an optimal path from s to i. Due to the lack of this property, the search for a shortest path or cycle in such a graph should maintain multiple labels for each node, each one corresponding to a different subpath through which the given node can be reached. Our heuristic approach aims at finding a negative subset-disjoint cycle, not necessarily the best. Hence, we choose to maintain, for each node i, only one cost label and, in order to decide when to update it, we check the Bellman conditions as in standard shortest path algorithms. Still, there are some cases in which we do not update a node label even though the Bellman conditions are violated [2]. One such case occurs when arcs emanating from the current node have already been examined during the search, and a change in the subpath leading to it would cause a cascade of subsequent changes in the cost of arcs which have already been considered. This issue will be further clarified in the detailed description of the algorithm given below.

The multi-customer neighborhood search algorithm maintains a *cost label* $d(i)$ and a *predecessor* $pred(i)$ for each node i in the graph, as in standard shortest path frameworks, plus a label $U(i)$ to store the subset of customers which moves into i from its predecessor along the path, and a label $min\_w(i)$ which indicates the minimum demand of the leaving customer subsets associated with the arcs (i,h) emanating from i. Formally, $min\_w(i) = \min_{(i,h)\in FS(i)} W(U_{ih})$, where $FS(i)$ denotes the forward star of node i. Labels $d(i)$, $pred(i)$ and $min\_w(i)$ are initially set to ∞, while $U(i)$ is the empty set.

The search starts at the origin o. At each subsequent iteration a node i is extracted from the set Q of candidate nodes still to be examined, and the following operations are executed until either Q is empty or we find a negative subset-disjoint cycle.

The path from the origin to i is checked for disjointness. If it is not disjoint, node i is skipped and another node is extracted from Q. Otherwise, all the arcs emanating from i are generated and examined. For each arc $(i,h) \in FS(i)$, two cases are possible.

1. *The path from o to h is a subset-disjoint path.* Then,
   (a) the subset of customers U_{ih} is found by solving problem (17)-(20);
   (b) the cost β_{ih} of arc (i,h) is computed according to the arc cost definition provided in Section 5.1, and using the subset U_{ih} determined at step (1a);

(c) if the Bellman conditions for (i, h) are satisfied, i.e., $d(h) \leq d(i) + \beta_{ih}$, the arc is skipped. Otherwise, one of the following three cases can occur:

   i. *h is a regular node and its forward star has not been examined yet.* In this case, h is inserted in Q and its labels are updated, i.e., $pred(h) := i$, $d(h) := d(i) + \beta_{ih}$, $U(h) := U_{ih}$. Also, if $W(U_{ih}) < min\_w(i)$, then $min\_w(i) := W(U_{ih})$.

   ii. *h is a regular node and its forward star has already been examined.* This implies that the customer subsets U_{hl} associated with the arcs (h, l) emanating from h have already been determined by solving instances of the selection problem SP. If the labels of h are changed, those subsets might no longer be candidate leaving subsets with respect to S_h and the new entering subset U_{ih}. This would lead to the need for recomputing some subsets U_{hl}, which in turn might cause some other capacity constraints to be violated. To avoid this problem, the labels of h are updated as in the previous step only if all the subsets U_{hl} are still candidate leaving subset after the update, i.e., if $W(U_{ih}) \leq r_h + min\_w(h)$.

   iii. *h is a pseudonode.* By adding arc (h, o) to the path up to h, a subset-disjoint cycle is obtained which includes a pseudonode and the origin. If the cost of the cycle is negative, a profitable path exchange has been detected and the algorithm terminates.

2. *The path from o to h is not a subset-disjoint path.* There are two possible cases.

   (a) *h is a regular node.* Then a cycle has been found. We execute steps (1a)-(1b) taking into account that the computation of U_{ih} at step (1a) requires the solution of problem SP with the additional constraint (21). If $d(i)+\gamma_{ih}-d(h) < 0$, a profitable cyclic exchange has been found and the algorithm terminates.

   (b) *h is a pseudonode.* This means that $h = p_l$ for some l, and node l has already been encountered along the path from o to d_l. In this case, arc (i, h) is disregarded since it would produce the same cycle which is generated when considering the arc from i to the regular node l.

6 The Facility Neighborhood Structure

The facility neighborhood is defined by three types of facility moves aiming, respectively, at opening a new facility, closing an existing facility, and transferring a facility to a different location. A facility move is legal only if it maintains the feasibility of the solution and is profitable if it results in a decrease of the total cost. The evaluation of a move profitability requires reassigning customers to the new set of open facilities, which is itself NP-hard. In order to limit the computational effort needed to find profitable neighbors, we only consider a restricted facility neighborhood induced by estimates of the cost savings. The effective cost of the solutions in the restricted neighborhood are then computed heuristically by attempting first a partial reassignment. If this fails, then a complete reassignment is considered.

6.1 The opening moves

The *facility opening move* attempts to establish a new facility at an unused location. The restricted neighborhood search scheme for this move can be described as follows.

For each unused location i, we identify the set $U_i = \{j \in J | c_{ij} - c_{F(j)j} < 0\}$. Let $z_i = \sum_{j \in U_i}(c_{ij} - c_{F(j)j}) + f_i$. We then sort the unused facility locations by non decreasing order of the saving estimate z_i. Locations with positive saving estimates are excluded from further consideration. The remaining locations are examined in turn as indicated by the ordering, and the profit of the corresponding neighbors is evaluated. As soon as a profitable move is detected, the exploration of the restricted neighborhood is terminated and the current solution is replaced by the improving neighbor.

The cost of a neighbor solution associated with the opening of a facility at location $i \in I \setminus I(S)$ is evaluated by first considering a partial customer reassignment, which only involves the customers in U_i, and reassigning all or part of them exclusively to the candidate facility i. In particular,

1. if $W(U_i) \leq s_i$, z_i is the effective cost saving for opening a facility at site i and allocating all the customers in U_i to it. A feasible and profitable move has then be detected;

2. if $W(U_i) > s_i$, we try to find a subset of U_i so that its assignment to i maintains feasibility while decreasing the overall cost of the solution. The problem of finding such a subset can be formulated as a knapsack problem (KP) in the following way. Let x_j be a binary variable which takes value 1 if customer j is selected, 0 otherwise.

$$\min \quad \sum_{j \in U_i} (c_{ij} - c_{F(j)j}) x_j + f_i \tag{22}$$

$$\text{s.t.} \quad \sum_{j \in U_i} w_j x_j \leq s_i \tag{23}$$

$$x_j \in \{0, 1\} \qquad \forall\ j \in U_i. \tag{24}$$

An approximation to the optimal solution of KP can be easily obtained by first redirecting to i the customers in U_i which produce the largest saving per unit demand volume.

Let $U_i^* = \{j \in U_i | x_j = 1\}$ be such an approximate solution to problem KP, and z_i^* its objective function value. If $z_i^* < 0$, the opening of a facility at i and the allocation of the customers in U_i^* to it results in a feasible and profitable move.

If the partial reassignment operation fails to detect an improving solution, we try a complete reassignment of the customers to the new set of open facilities $\bar{I} = I(S) \bigcup \{i\}$ by solving the following Generalized Assignment Problem (GAP). Let x_{ij} be a binary variable which takes value 1 if customer j is assigned to facility $i \in \bar{I}$, 0 otherwise.

$$\begin{aligned}
\min \quad & \sum_{i \in \bar{I}} \sum_{j \in J} c_{ij} x_{ij} && (25)\\
\text{s.t.} \quad & \sum_{i \in \bar{I}} x_{ij} = 1 \qquad \forall\ j \in J && (26)\\
& \sum_{j \in J} w_j x_{ij} \leq s_i \qquad \forall\ i \in \bar{I} && (27)\\
& x_{ij} \in \{0,1\} \qquad \forall\ i \in \bar{I}, j \in J. && (28)
\end{aligned}$$

In our local search algorithm, an approximate solution to GAP is obtained by the greedy algorithm proposed by Romeijn and Romero Morales in [15]. The approach is based on the definition of a weight function which is used to measure the desirability of assigning each customer to each facility. The greedy algorithm then schedules the customers according to a decreasing order of desirability. Let S' be the solution obtained by rearranging the customers among the facilities according to the indication provided by the solution to GAP, and let $C(S')$ be its cost. The set of constrains (27) guarantees that S' is a feasible solution. Hence, if $C(S') < C(S)$, a feasible profitable move has been detected.

6.2 The closing moves

The *facility closing move* attempts to close an existing facility and reallocate its customers among the remaining open facilities.

As for the opening move, we only explore a restricted neighborhood generated by closing moves. We associate with each location i in use, a saving estimate $z_i = -f_i + \sum_{j \in S_i}(c_{\hat{\imath}j} - c_{ij})$, where $\hat{\imath}$ denotes the closest facility to customer j among the facilities in $\bar{I} = I(S) \setminus \{i\}$. Used facility locations with positive z_i or such that $W(S_i) > \sum_{h \in I(S) \setminus \{i\}} r(S_h)$ are not considered for generating neighbor solutions. The moves associated with the remaining locations are evaluated in the non decreasing order of the saving estimates z_i until a profitable move is detected. When evaluating the cost for closing a facility at i, a first attempt at detecting a profitable move is made by reassigning only the customers currently assigned to i. This requires solving a Reduced Generalized Assignment Problem (RGAP) where the capacities s_i are replaced by the residual capacities r_i, for each $i \in \bar{I}$, and involving only the customers in S_i. RGAP is solved through the same greedy algorithm used for GAP [15]. If RGAP does not yield an improving solution, the reassignment of the whole set of customers to the new set of open facilities $\bar{I}$ is tried by solving GAP as for the opening move.

6.3 The transferring moves

In some cases, closing and opening moves might not generate cost-decreasing solutions if considered separately. However, improving solutions might derive from their combined utilization, i.e., from the *transferring move*. A transferring move displaces a facility from its current location and reopens it at a different site.

A first attempt to detect an improving solution through a transferring move is simply made by moving an existing facility to a different location, and reassigning the whole cluster of customers associated with the moving facility to the new location. Such a facility transfer from a site i to a site h is feasible if $W(S_i) \leq s_h$ and profitable if $\sum_{j \in S_i} c_{hj} + f_h < C(S_i)$. The checking of these two conditions is pretty inexpensive and hence the move can be attempted for each pair (i, h) of used-unused sites.

A more complex transferring move is also considered which completely redistributes customers to the new set of open facilities. The main issue concerning this move is how to conveniently select promising pairs of facility locations for which to attempt the move, in order to avoid trying all the possible combinations. The restricted neighborhood search in this case operates as follows: we first select the facility to be closed on the basis of the estimates of the customer redistribution costs, as described for the closing move in Section 6.2. Once the closing facility i has been chosen, we choose for the facility opening the location h which could capture the greatest amount of customers currently assigned to i, provided that $\sum_{i \in \bar{I}} s_i \geq \sum_{j \in J} w_j$, where $\bar{I} = I(S) \setminus \{i\} \bigcup \{h\}$. A customer $j \in S_i$ is considered captured by site h if $c_{hj} < c_{ij}$.

Once again the problem of allocating customers to a set of already known open facilities $\bar{I}$ to detect improving moves requires solving GAP.

7 Neighborhood search algorithm

The facility and customer neighborhood structures introduced in the previous sections are made operational within a neighborhood search algorithm. The general scheme of the algorithm is very simple and, in this context, we did not make any specific effort to extend and refine it through the use of more sophisticated guidance processes, such as tabu search. The basic algorithm simply proceeds from an initial feasible solution by a sequence of local changes which are obtained through facility moves or customer moves. Each move produces an improved solution, and the process is iterated until a local optimum is found.

This general local improvement framework leaves large scope of operations, thanks to the amplitude of possible ways of alternating facility and customer moves. A reasonable choice, whose efficiency has been corroborated by the computational results, works as follows. The simple transferring move, which is quite inexpensive and moves the facilities rather quickly to better locations is executed first. The customer assignment improvement is then committed to the search for profitable multi-customer exchanges on the facility improvement graphs. When no more profitable multi-customer exchanges can be detected, the algorithm tries the facility moves in the following order: opening, closing, and transferring. As soon as a facility move produces a cost-decreasing solution, the search for profitable multi-customer exchanges is restarted to improve the assignment of customers to the new set of open facilities. Only when all the facility moves fail, the finer single-customer exchange is executed to perfect the solution.

In order to circumscribe the tendency of the local improvement algorithm to get trapped at local optima, we have implemented a *multistart* mechanism by obtaining multiple initial feasible solutions through a Lagrangean relaxation/subgradient optimization procedure. Such a procedure is analogous to the one described in [11], Section 2. Namely, we apply Lagrangean relaxation to the assignment constraints (2), and obtain a problem which separates into n knapsack problems, one for each facility i. The solutions of the knapsack problems provide reduced costs associated with each facility location, values for the variables y_i and x_{ij}, and a lower bound to the problem. The dual problem is solved with subgradient optimization, and primal feasible solutions are obtained by appropriately reassigning customers which are either not assigned or assigned to multiple facilities. The reader is referred to [11] for further details on this procedure.

In our computational investigation, we use the Lagrangean heuristic to generate 500 feasible solutions, and select 30 of them as restart solutions for the local search algorithm. The selected solutions are the ones, with the lowest costs, which guarantee diversity with respect to the set of open facilities.

7.1 Some Implementation Issues

There are some issues related to the actual implementation of the algorithmic paradigm 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.

Some issues are related to the algorithm which explores the facility improvement graph, as described in Section 5.4. First, the rule used for choosing the next node to be extracted from Q typically conditions the kind of cycles or paths that are explored by the algorithm. In our experimental investigation, we consider three different implementations of Q, corresponding to a *breadth-first search (bfs)*, a *depth-first search (dfs)*, and a *best search (bs)* of the graph.

Second, the cycle detection heuristic is applied by selecting a regular node as the origin from which to start the search. However, many profitable multi-customer exchanges might be missed by considering only one node as the origin node. We thus apply the cycle search algorithm multiple times with different nodes as origin. More precisely, we maintain a list of the regular nodes which have not been still used as origins, and select one node at random from that list: if a subset-disjoint cycle is detected starting from that origin, then the current solution is updated and the list is re-initialized; otherwise a new node is selected from the list. The algorithm stops when the list becomes empty.

The last remark concerns some parameter options of the Lagrangean heuristic which generates the initial solutions. In the experimental tests, we initially set to 1 the parameter λ used in the computation of the step size of the subgradient method (see [11]), and half its value after p consecutive iterations with no improvement of the lower bound. Preliminary tests showed that variations of the parameter p affect the quality of the obtained solutions. Hence, we carried out the subsequent testing with two different settings for this parameter, namely $p = 5$ and $p = 10$.

8 Computational Study

In this section, we present some preliminary computational results for 4 sets of benchmark problems, which were randomly generated by Holmberg et al. [11]. The test problems in each subset differ for the distributions of the inputs and for the size, which ranges from 50 to 200 customers, and from 10 to 30 candidate facility locations. The Lagrangean heuristic with repeated matching proposed in [11] to solve these problems is one of the most efficient heuristics for SSCFLP. In particular, in a previous work [16], the authors compared its performance with Pirkul heuristic and showed that the solutions obtained through the repeated matching approach were closer to the optimal than those from the Pirkul method. The Lagrangean heuristic by Holmberg et al., denoted in the following as *LH*, was therefore chosen as a term of comparison for evaluating the performance of our VLSN algorithms. We also validate the quality of the obtained solutions through direct comparison with the optimal solutions found through the commercial code CPLEX (version 7.0).

Both the CPLEX application and the VLSN heuristics were coded in C++, compiled with the GNU g++ compiler and run on a PC with a Athlon/1200Mhz processor and 512 Mb RAM, under the RedHat Linux 7.1 operating system.

An extended experimentation performed to study the impact of the parameter K, i.e., the maximum number of customers that can be transferred simultaneously in the multi-exchange phase, showed that the best results are obtained when K ranges from 4% to 10% of the total number of customers. For the sake of brevity, we report the results only for the value of K in this interval which produced the best results for each problem set. Namely, K is fixed to 3 for set 1, 6 for set 2, 7 for set 3, and 12 for set 4. Preliminary tests also indicate that the queue implementation of Q, corresponding to the *bfs* strategy, is almost always dominated by the stack implementation (*dfs*) and by the priority queue implementation (*bs*). From the initial results, we also observed that a value of 10 for the Lagrangean parameter p is generally preferable to the value 5. In the light of these preliminary observations, we restrict the presentation of the results to the best algorithmic options (i.e., *dfs* and *bs*, and $p = 10$). We then point out some exceptions to these general trends.

Tables 1-4 show the computational results obtained by CPLEX, *LH* and the *dfs* and *bs* versions of our VLSN algorithm for the 4 problem sets respectively. Each table is structured as follows. The first column gives the problem name. The second column shows the objective value of the optimal solution found by CPLEX. The next three columns report the objective function values returned by *LH*, *dfs* and *bs*, followed by the % deviations of the objective function values for the three heuristics from the best known solutions. The percentage deviation is calculated as:

$$\frac{z_e - z^*}{z^*} \times 100\%. \tag{29}$$

In (29), z^* and z_e denote respectively the optimal objective function value and the objective function value of the solutions obtained by the heuristic under examination. The last two columns give the CPU-times in seconds required by the VLSN heuristics to perform the 30 restarts. Finally, the last row of each table shows the average percentage deviation and the average time for the whole problem set.

Both *dfs* and *bs* solve to optimality all the instances of the first set of problems, as shown in Table 1. The VLSN heuristics seem to perform worse than the Lagrangean heuristic, in terms of average percentage deviation, on the second set of problems (Table 2). This is due only to the problems from p28 to p32, which seem to be characterized by an intrinsec difficulty; in fact, CPLEX solved them in a significant amount of time (ranging from 130 to 620 seconds). On the other hand, all the other problems listed in Table 2 are solved to optimality by *dfs*. Furthermore, the VLSN heuristics outperform the Lagrangean heuristic on the third and fourth sets (Tables 3 and 4). In particular, the VLSN heuristics compute a better solution for problems p51, p56, p57, p58, p66, p67 and p70. Moreover, problems p19, p20, p55, p59 and p71 are solved to optimality by the proposed heuristics, but not by the LH heuristic.

As indicated before, for each problem set we report the results only for one value of K and for $p = 10$, since these choices generally produced the best results. There are some exceptions concerning the fourth set, which are commented in the following. Moreover it has to be outlined that the VLSN heuristics, when implemented with $p = 5$, solved to

Table 1: Computational results for problems p1-p24.

Pr.	Opt.	Objective Value			% Deviation			Time (sec.)	
No.	Sol.	*LH*	*dfs*	*bs*	*LH*	*dfs*	*bs*	*dfs*	*bs*
p1	8848	8848	8848	8848	0.000	0.000	0.000	0.20	0.21
p2	7913	7913	7913	7913	0.000	0.000	0.000	0.09	0.10
p3	9314	9314	9314	9314	0.000	0.000	0.000	0.25	0.24
p4	10714	10714	10714	10714	0.000	0.000	0.000	0.38	0.42
p5	8838	8838	8838	8838	0.000	0.000	0.000	0.12	0.13
p6	7777	7777	7777	7777	0.000	0.000	0.000	0.09	0.08
p7	9488	9488	9488	9488	0.000	0.000	0.000	0.19	0.20
p8	11088	11088	11088	11088	0.000	0.000	0.000	0.23	0.23
p9	8462	8462	8462	8462	0.000	0.000	0.000	0.31	0.31
p10	7617	7617	7617	7617	0.000	0.000	0.000	0.12	0.12
p11	8932	8932	8932	8932	0.000	0.000	0.000	0.27	0.27
p12	10132	10132	10132	10132	0.000	0.000	0.000	0.63	0.60
p13	8252	8252	8252	8252	0.000	0.000	0.000	1.11	1.13
p14	7137	7137	7137	7137	0.000	0.000	0.000	0.46	0.46
p15	8808	8808	8808	8808	0.000	0.000	0.000	0.59	0.53
p16	10408	10408	10408	10408	0.000	0.000	0.000	1.84	1.89
p17	8227	8227	8227	8227	0.000	0.000	0.000	0.77	0.75
p18	7125	7125	7125	7125	0.000	0.000	0.000	0.44	0.44
p19	8886	8887	8886	8886	0.011	0.000	0.000	0.64	0.59
p20	10486	10487	10486	10486	0.010	0.000	0.000	1.85	1.82
p21	8068	8068	8068	8068	0.000	0.000	0.000	0.46	0.47
p22	7092	7092	7092	7092	0.000	0.000	0.000	0.02	0.02
p23	8746	8746	8746	8746	0.000	0.000	0.000	0.35	0.35
p24	10273	10273	10273	10273	0.000	0.000	0.000	1.45	1.45
Avg.					0.001	0.000	0.000	0.54	0.53

Problems size: n = 10-20, m = 50

Table 2: Computational results for problems p25-p40.

Pr.	Opt.	Objective Value			% Deviation			Time (sec.)	
No.	Sol.	*LH*	*dfs*	*bs*	*LH*	*dfs*	*bs*	*dfs*	*bs*
p25	11630	11630	11630	11630	0.000	0.000	0.000	9.63	9.45
p26	10771	10771	10771	10771	0.000	0.000	0.000	5.08	4.89
p27	12322	12322	12322	12327	0.000	0.000	0.041	12.75	12.69
p28	13722	13722	13727	13727	0.000	0.036	0.036	13.60	14.11
p29	12371	12375	12379	12379	0.032	0.065	0.065	30.09	30.93
p30	11331	11346	11392	11379	0.132	0.538	0.424	23.54	23.27
p31	13331	13341	13436	13365	0.075	0.788	0.255	30.01	28.57
p32	15331	15361	15436	15385	0.196	0.685	0.352	34.08	32.99
p33	11629	11629	11629	11629	0.000	0.000	0.000	11.70	11.17
p34	10632	10632	10632	10632	0.000	0.000	0.000	6.30	6.28
p35	12232	12232	12232	12232	0.000	0.000	0.000	10.31	10.43
p36	13832	13832	13832	13832	0.000	0.000	0.000	10.50	10.71
p37	11258	11258	11258	11258	0.000	0.000	0.000	1.52	1.52
p38	10551	10551	10551	10551	0.000	0.000	0.000	1.54	1.54
p39	11824	11824	11824	11824	0.000	0.000	0.000	0.20	0.20
p40	13024	13024	13024	13024	0.000	0.000	0.000	1.79	1.79
Avg.					0.027	0.132	0.073	12.66	12.53

Problems size: n = 30, m = 150

optimality also problem p52 and problem p28 (in the latter case with $K = 8$).

Table 3: Computational results for problems p41-p55.

Pr.	Opt.	Objective Value			% Deviation			Time (sec.)	
No.	Sol.	*LH*	*dfs*	*bs*	*LH*	*dfs*	*bs*	*dfs*	*bs*
p41	6589	6589	6589	6589	0.000	0.000	0.000	2.38	2.41
p42	5663	5663	5663	5663	0.000	0.000	0.000	2.31	2.30
p43	5214	5214	5214	5214	0.000	0.000	0.000	0.05	0.05
p44	7028	7028	7028	7028	0.000	0.000	0.000	0.36	0.35
p45	6251	6251	6251	6251	0.000	0.000	0.000	1.93	1.75
p46	5651	5651	5655	5651	0.000	0.071	0.000	2.54	2.61
p47	6228	6228	6228	6228	0.000	0.000	0.000	0.12	0.13
p48	5596	5596	5596	5596	0.000	0.000	0.000	1.48	1.38
p49	5302	5302	5302	5302	0.000	0.000	0.000	1.09	1.04
p50	8741	8741	8757	8741	0.000	0.183	0.000	1.79	1.85
p51	7414	7469	7425	7425	0.742	0.148	0.148	5.47	5.52
p52	9178	9178	9180	9180	0.000	0.022	0.022	0.40	0.41
p53	8531	8531	8531	8531	0.000	0.000	0.000	1.90	1.80
p54	8777	8777	8777	8777	0.000	0.000	0.000	0.76	0.77
p55	7654	7672	7654	7654	0.235	0.000	0.000	1.76	1.80
Avg.					0.065	0.028	0.011	1.62	1.61

Problems size: n = 10-30, m = 70-100

Table 4: Computational results for problems p56-p71.

Pr.	Opt.	Objective Value			% Deviation			Time (sec.)	
No.	Sol.	*LH*	*dfs*	*bs*	*LH*	*dfs*	*bs*	*dfs*	*bs*
p56	21103	21163	21120	21118	0.284	0.081	0.071	15.77	15.94
p57	26039	26167	26075	26085	0.492	0.138	0.177	26.32	27.09
p58	37239	37792	37240	37284	1.485	0.003	0.121	46.60	45.97
p59	27282	27300	27282	27282	0.066	0.000	0.000	29.97	29.90
p60	20534	20534	20534	20534	0.000	0.000	0.000	3.27	3.22
p61	24454	24454	24454	24454	0.000	0.000	0.000	6.27	6.27
p62	32643	32643	32648	32651	0.000	0.015	0.025	28.68	27.15
p63	25105	25105	25108	25108	0.000	0.012	0.012	15.22	14.78
p64	20530	20530	20530	20530	0.000	0.000	0.000	0.18	0.18
p65	24445	24445	24445	24445	0.000	0.000	0.000	0.18	0.18
p66	31415	31504	31429	31420	0.283	0.045	0.016	8.14	8.16
p67	24848	24876	24849	24849	0.113	0.004	0.004	10.34	10.15
p68	20538	20538	20538	20538	0.000	0.000	0.000	3.32	3.32
p69	24532	24532	24532	24532	0.000	0.000	0.000	6.15	6.50
p70	32321	32393	32323	32323	0.223	0.006	0.006	27.04	25.95
p71	25540	25562	25540	25540	0.086	0.000	0.000	28.10	28.78
Avg.					0.189	0.019	0.027	15.97	15.85

Problems size: n = 30, m = 200

Now let us analyse the computational results in more detail. Generally, it is not possible to state the superiority of either of the two search strategies *dfs* and *bs*. This observation was confirmed by experiments with different values of the parameters K and p.

Moreover, the general tendency of *dfs* and *bs* to outperform *bfs*, and of the option $p = 10$ to be more attractive than $p = 5$, is not as sound on the forth set of problems, as shown in Table 5. The table reports the results obtained when p is set to 5 for all the three search strategies. It can be noticed that this choice yields the best average percentage deviations for every implementation of Q, finding some additional optimal solutions which were missed by the algorithms with $p = 10$. In particular, the *bfs* version used in combination with $p = 5$ finds the optimal solutions for two of the most difficult problems, namely p57 and p58; *dfs* for p70, and *bs* for p63 and p66. These last combinations of the algorithmic implementation options are the ones yielding the best average percentage deviation, only 0.01%, on the most difficult set of problems, compared to the 0.19% average percentage deviation of LH. By summarizing, the total number of problems solved to optimality by the proposed heuristics is therefore 63 over 71 tested instances, whereas the heuristic LH finds the optimum solution for 55 problems.

The different behavior of the heuristics on the 4 problem sets due to variations of the parameter p can be explained by noticing that for some medium size instances, as the ones in set 3, the number of structurally heterogeneous initial solutions generated by the Lagrangean procedure with $p = 5$ is quite small. As a consequence, the lack of sufficient diversification in the initial solutions thwarts the benefits of the restart mechanism. On the contrary, for big problems like p56-p71, a smaller value of p is sufficient to produce 30 well-diversified initial solutions, and the overall methodology seems to profit from this choice. On the small problems of set 1, the effect of p is negligible, since the optimal solutions are always obtained by every implementation in the very first restarts.

Table 5: Computational results for problems p56-p71. Option: p = 5

Pr.	Opt.	Objective Value			% Deviation			Time (sec.)		
No.	Sol.	*bfs*	*dfs*	*bs*	*bfs*	*dfs*	*bs*	*bfs*	*dfs*	*bs*
p56	21103	21126	21124	21120	0.109	0.100	0.081	16.05	16.52	15.80
p57	26039	26039	26040	26040	0.000	0.004	0.004	31.48	29.37	29.22
p58	37239	37239	37240	37240	0.000	0.003	0.003	48.57	47.55	47.00
p59	27282	27282	27282	27282	0.000	0.000	0.000	31.28	30.49	28.66
p60	20534	20534	20534	20534	0.000	0.000	0.000	0.10	0.10	0.10
p61	24454	24454	24454	24454	0.000	0.000	0.000	6.50	6.53	7.08
p62	32643	32651	32651	32651	0.025	0.025	0.025	29.78	29.87	28.30
p63	25105	25108	25108	25105	0.012	0.012	0.000	12.18	11.81	12.18
p64	20530	20530	20530	20530	0.000	0.000	0.000	0.12	0.12	0.12
p65	24445	24445	24445	24445	0.000	0.000	0.000	0.10	0.10	0.10
p66	31415	31420	31429	31415	0.016	0.045	0.000	7.72	8.14	8.12
p67	24848	24849	24849	24864	0.004	0.004	0.064	11.06	10.86	10.70
p68	20538	20538	20538	20538	0.000	0.000	0.000	3.84	3.86	3.83
p69	24532	24532	24532	24532	0.000	0.000	0.000	6.24	6.08	6.98
p70	32321	32332	32321	32332	0.034	0.000	0.034	26.65	29.39	29.30
p71	25540	25540	25540	25540	0.000	0.000	0.000	28.11	24.80	25.62
Avg.					0.012	0.012	0.013	16.23	15.87	15.82

As far as the CPU-time is concerned, direct comparisons between LH and the VLSN heuristics are not possible since our code and Holmberg et al. code were run on two different machines. However, the running time does not seem to be the critical issue for our approach. The most difficult problem, p58, for which neither LH nor the exact method proposed in [11] could find the optimal solution, required less than one minute to be solved to optimality by the *bfs* version of the VLSN algorithm. The other implementations found very good approximations to the optimal solution in about the same amount of time (see Tables 4 and 5). A well fine-tuned CPLEX application, which uses the network optimizer in the initial node and the dual simplex after branching, and which gives branching priority to the location variables y_i to improve performance, needed around two and a half hours to solve the same problem. On the average, the instances of the most difficult set were solved in less than 20 seconds by different implementations of the VLSN technique. Further, the running times reported in the tables refer to the 30 restarts, but, as shown in Table 6, much less restarts than the maximum allowed were needed for most of the instances. In many cases, especially for the instances of the first and second problem set, the best solutions were found in the first restart. Only 5 problems required more restarts (between 20 and 30) to be solved by *dfs* and *bs*, and around 50 out of the 71 problems were solved within the first 5 restarts. The first row in Table 6 reports the number of instances in each subset which were already solved to optimality by the Lagrangean heuristic used to generate initial solutions. Globally, 5 problems did not require the use of the VLSN heuristic.

Table 6: Number of problems solved in each restart interval.

No. of	p1-p24		p25-p40		p41-p55		p56-p71		All	
Restarts	*dfs*	*bs*	*dfs*	*bs*	*dfs*	*bs*	*dfs*	*bs*	*dfs*	*bs*
–	1	1	1	1	1	1	2	2	5	5
[1,5]	20	20	11	10	11	9	10	11	52	50
(5,10]	2	0	2	1	1	0	1	0	6	1
(10,20]	1	3	1	3	0	2	1	2	3	10
(20,30]	0	0	1	1	2	3	2	1	5	5

The analysis of the computational results obtained by the VLSN algorithms throws some light on multiple issues. Firstly, it testifies the efficiency of the proposed methodology for solving large scale problems. This is evident especially for the fourth set of problems, whereas on smaller problems an advanced and structurally complex solution approach such as our VLSN technique can be sometimes less efficient.

Secondly, it shows that the use of restart mechanisms can be sometimes inadequate to escape local optimality. When solving instances like p51, the VLSN heuristics may get trapped at local optima at the very first iterations and are never able to escape from it in subsequent restarts. The study of alternative metaheuristic frameworks, such as tabu search, could aid in overcoming this difficulty.

9 Conclusions

We proposed some VLSN algorithms for the capacitated facility location problem with single-source constraints. Computational results for some benchmark instances demonstrated the effectiveness of the approach for solving large-scale problems. In particular, the definition of multi-customer multi-exchanges, detected on a dynamic improvement graph, showed to be very effective in solving this challenging problem.

References

[1] M.C. Agar and S. Salhi. Lagrangean heuristics applied to a variety of large capacitated palnt location problems. *Journal of the Operational Research Society*, 49:1072–1084, 1998.

[2] R.K. Ahuja, J.B. Orlin, and D. Sharma. Multi-exchange neighborhood search algorithms for the capacitated minimum spanning tree problem. *Mathematical Programming*, 91:71–97, 2001.

[3] J. Barcelo and J. Casanova. A heuristic Lagrangean algorithm for the capacitated plant location problem. *European Journal of Operational Research*, 15:212–226, 1984.

[4] J. Barcelo, E. Fernandez, and K. Jornsten. Computational results from a new Lagrangean relaxation algorithm for the capacitated plant location problem. *European Journal of Operational Research*, 53:38–45, 1991.

[5] J. E. Beasley. Lagrangian heuristics for location problems. *European Journal of Operational Research*, 65:383–399, 1993.

[6] A. Caprara, H. Kellerer, U. Pferschy, and D. Pisinger. Approximation algorithms for knapsack problems with cardinality constraints. *European Journal of Operational Research*, 123:333–345, 2000.

[7] D. Delmaire, J.A. Diaz, E. Fernandez, and M. Ortega. Comparing new heuristics for the pure integer capacitated plant location problem. Dr97/10, dpt. e10, Univ. Politecnica de Catalunya, Spain, 1997.

[8] D. Erlenkotter. A dual-based procedure for uncapacitated facility location. *Operations Research*, 26:992–1009, 1978.

[9] A. Frangioni, E. Necciari, and M.G. Scutellà. A multi-exchange neighborhood for minimum makespan machine scheduling problems. Tr 17/00, University of Pisa, 2000.

[10] K.S. Hindi and Pieńkosz. Efficient solution of large scale, single-source, capacitated plant location problems. *Journal of the Operational Research Society*, 50:268–274, 1999.

[11] K. Holmberg, M. Ronnqvist, and D. Yuan. An exact algorithm for the capacitated facility location problems with single sourcing. *European Journal of Operational Research*, 113:544–559, 1999.

[12] J.G Klincewicz and H. Luss. A Lagrangean relaxation heuristic for capacitated facility location with single-source constraints. *Journal of the Operational Research Society*, 37:495–500, 1986.

[13] A.W. Neebe and M.R. Rao. An algorithm for the fixed-charge assigning users to sources problem. *Journal of the Operational Research Society*, 34:1107–1113, 1983.

[14] H. Pirkul. Efficient algorithm for the capacitated concentrator location problem. *Computers & Operations Research*, 14:197–208, 1987.

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

[16] M. Ronnqvist, S. Tragantalerngsak, and J. Holt. A repeated matching heuristic for the single-source capacitated facility location problem. *European Journal of Operational Research*, 116:51–68, 1999.

[17] R. Sridharan. A lagrangian heuristic for the capacitated plant location problem with single source constraints. *European Journal of Operational Research*, 66:305–312, 1991.

[18] P.M. Thompson and J.B. Orlin. Theory of cyclic transfers. Working paper, Operations Research Center, MIT, 1989.

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