A Multi-Exchange Heuristic for the Single-Source Capacitated Facility Location Problem

Article *in* Management Science · February 2003

DOI: 10.2139/ssrn.337621 · Source: RePEc

5 authors, including:

James B. Orlin
Massachusetts Institute of Technology
253 PUBLICATIONS **13,798** CITATIONS
SEE PROFILE

Maria Paola Scaparra
University of Kent
35 PUBLICATIONS **1,110** CITATIONS
SEE PROFILE

Maria Grazia Scutellà
Università di Pisa
62 PUBLICATIONS **1,188** CITATIONS
SEE PROFILE

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

Maria Paola Scaparra

Abstract— **Capacitated facility location problems with single source constraints (SSCFLP) represent a major class of location problems which are characterized by the placement of upper limits on facility supplies and the requirement that each customer be served by a single facility. Those two features greatly increase the difficulty of the problem so that exact methods can solve only small instances. This justifies the need for devising new efficient heuristic methodologies for handling hard, large scale problems.**

In this paper we investigate the application to SSCFLP of a new class of local search algorithms, based on the concept of multi-exchange neighborhood search. The neighborhood structure we propose relies on the exchange of sets of customers among facilities along special paths or cycles detected in a suitably defined composite improvement graph. Starting with an initial feasible solution, each multi-exchange allows the replacement of the current solution with an improved solution in its neighborhood until some termination criterion is satisfied. Due to the large scale of the neighborhood structure, an improved solution is computed heuristically by exploring the neighborhood space through efficient *network flow based improvement algorithms*.

This paper is a preliminary version of a work still in progress [3].

Keywords— **location problems, local search, multi-exchange**

I. Introduction

THE capacitated facility location problem with single source constraints, usually referred to as SSCFLP, deals with the location of a number of plants which have to serve at minimum cost a set of customers distributed in a discrete space. Each customer has an associated demand which must be served by a single plant. The plants can be located only at a finite number of prespecified sites and there are constraints on the total demand that each plant can meet. The total costs to be minimized include fixed charges for opening the facilities and transportation costs for satisfying customer demand. The aim of the problem is to optimally locate plants and assign customers demand to them.

The problem can be stated mathematically as follows:

$$\min \quad \sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij} + \sum_{i\in I} f_i y_i \tag{1}$$

$$\text{s.t.} \quad \sum_{i\in I} x_{ij} = 1 \qquad \forall\ j \in J \tag{2}$$

$$\sum_{j\in J} w_j x_{ij} \le s_i y_i \qquad \forall\ i \in I \tag{3}$$

$$x_{ij} \in \{0,1\} \qquad \forall\ i \in I, j \in J \tag{4}$$

$$y_i \in \{0,1\} \qquad \forall\ i \in I \tag{5}$$

where:
$I = \{1,\dots,n\}$ is the set of potential locations.
$J = \{1,\dots,m\}$ is the set of customers.
$f_i =$ fixed cost for establishing a facility at location i.
$c_{ij} =$ cost of supplying all the demand of customer j from a plant established at location i.
$w_j =$ demand of customer j.
$s_i =$ capacity of facility i, i.e. maximum amount of demand that can be allocated to a plant located at i.
$y_i =$ decision variable which takes either value 1 or 0 according to whether facility location i is established or not.
$x_{ij} =$ decision variable which takes value 1 if customer j is assigned to facility i and 0 otherwise.

Formula (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 demand of customers. Demand 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 is a special case of the capacitated facility location problem, but its additional feature requiring that each customer can only be supplied from one facility makes it generally more difficult. Consequently, exact algorithms can solve only small sized instances. An example of an exact procedure to solve SSCFLP is the branch and bound algorithm proposed by Neebe and Rao [14]. Their solution approach, which is based on the formulation of SSCFLP as a partitioning problem, 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.

As problem instances grow in size, the use of heuristic approaches becomes more appealing so that approximations to the optimal solution can be obtained in a reasonable amount of computational time.

The most successful approaches thus far to solve SSCFLP are Lagrangean heuristics, i.e. heuristics based on the Lagrangean relaxation of some set of constraints and the use of subgradient optimization methods to solve the Lagrangean dual. Lagrangean heuristics have the advantage of providing both a lower and an upper bound to the

Department of Computer Science, University of Pisa, Corso Italia 40, 56125 Pisa, Italy. E-mail: scaparra@di.unipi.it .

optimal solution, thus producing an upper bound on the duality gap at each iteration of the procedure.

Klincewicz and Luss [12] propose a Lagrangean relaxation heuristic where they dualize the capacity constraints (3), thereby obtaining as Lagrangean subproblem an uncapacitated facility location problem. These subproblems are solved using the dual ascent algorithm introduced by Erlenkotter [7].

Barcelo and Casanova [4], Sridharan [17] and Pirkul [15] device solution procedures based upon relaxing the customer demand constraints (2). The heuristic proposed by Barcelo and Casanova [4] consists of two phases: the plant selection phase, which terminates when the total capacity of the open plants just exceeds the total demand, and the assignment phase, which makes use of a regret heuristic to assign customers to plants after improving the list of open plants through an interchange procedure. Sridharan [17] separates the Lagrangean subproblem into n knapsack problems, one for each facility, and from their combined solutions he gets a lower bound to the problem with a set of open facilities. This set of open facilities is then used to setup a transportation problem, whose optimal solution provides a feasible solution and an upper bound to the original problem. A similar approach based on the relaxation of the demand constraints and the solution to a number of knapsack problems is devised by Pirkul [15]. According to comparisons made by Beasley [5], Pirkul's heuristic provides the best feasible solutions to SSCFLP.

In the same work [5], Beasley also proposes a different Lagrangean heuristic based upon relaxing both the demand constraints and the capacity constraints. The procedure makes use of subgradient optimization to maximize the lower bound obtained from the relaxation. The information contained in the solution so found is then used in an attempt to construct a feasible solution to the original problem. Even if the heuristic proposed by Pirkul gives better solutions for SSCFLP than the procedures proposed by Beasley, the merit of the last approach lies in its robustness since it provides good quality solutions across a range of different location problems (test results are provided for p-median, uncapacitated, capacitated and single source facility location problems).

Geoffrion and Graves [10] propose a different approach for SSCFLP based on Benders' decomposition and apply the method to a generalized version of the problem which considers multiple commodities and multiple stages in distribution.

Finally, a new heuristic approach based on a repeated matching algorithm is described in the work by Rönnqvist et al. [16]. In each iteration of their heuristic, the authors solve a matching problem to optimality, by deriving the matching costs from the solution of a number of knapsack problems. The results obtained are encouraging and the repeated matching approach seems to be quite attractive, especially with regard to potential extensions to applications which include additional complicating constraints.

In this paper we investigate the application to SSCFLP of a new class of local search algorithms based on the concept of multi-exchange neighborhood search. This method is a generalization of the two-exchange neighborhood search, a very effective and largely used approach for solving difficult classes of combinatorial optimization problems. The neighborhood structure we propose relies on the exchange of sets of customers among facilities along special paths or cycles detected in a suitably defined composite improvement graph. Starting with an initial feasible solution, each multi-exchange allows the replacement of the current solution with an improved solution in its neighborhood until some termination criterion is satisfied. Due to the large scale of the neighborhood structure, an improved neighbor solution is computed heuristically without enumerating and evaluating all feasible solutions in the multi-exchange neighborhood.

The rest of the paper is organized as follows. In section II we represent a generic solution of SSCFLP as a partition of the set of customers and we introduce some useful notations. In section III we define the neighborhood structure by describing cyclic and path exchanges, the composite improvement graph and the criteria driving the selection of moving subsets of customers. Section IV details how to efficiently explore the composite improvement graph to find improving exchanges, while section V addresses the main issues concerning the implementation of the local search procedure. Finally, in section VI we make some conclusive remarks.

II. Notation

Let S be a feasible solution to the problem (1)-(5). Then S can be represented as a partition of the 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. Of course, 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 $i = 1, \ldots, n$:

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

We then define the *residual capacity* of each subset S_i as:

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

Further, let U_{ih} be a subset of customers currently in S_i which can be moved to S_h (the need for doubly subindexing the subset of moving customers U_{ih} derives from the fact that the composition of such a subset depends on both the

relinquishing subset S_i and the destination subset S_h, as will become clearer in the following sections). The *capacity gap value for leaving subsets* with respect to the subset U_{ih} which enters S_h is given by:

$$gl(U_{ih}, S_h) = W(U_{ih}) - r(S_h), \tag{10}$$

where $W(U_{ih})$ is the total demand of the customers in U_{ih}. $gl(U_{ih}, S_h)$ represents the demand excess supplied by facility h after the assignment to it of the new set of customers U_{ih}. A non positive value of this quantity denotes that facility h's capacity is not exceeded.

A subset $U_{hl} \in S_h$ is called a *candidate leaving subset* with respect to U_{ih} and S_h if it satisfies the following condition:

$$W(U_{hl}) \geq gl(U_{ih}, S_h) \tag{11}$$

In the context of the neighborhood structure which is described next, this condition means that if U_{ih} is the subset of customers to be moved to location h from the current facility i, then U_{ih} can substitute only those subsets of S_h which restore solution feasibility with respect to the capacity constraints.

Similarly, we can define the *capacity gap value for entering subsets* with respect to a subset U_{hl} which leaves the partition set S_h as:

$$ge(U_{hl}, S_h) = r(S_h) + W(U_{hl}). \tag{12}$$

This quantity represents the capacity availability of facility h after the removal of subset U_{hl}.

A subset U_{ih} is called a *candidate entering subset* with respect to U_{hl} and S_h if the following condition is satisfied:

$$W(U_{ih}) \leq ge(U_{hl}, S_h) \tag{13}$$

This condition means that if U_{hl} is the subset of customers to be moved from facility h to another facility l then U_{hl} can be substituted in S_h only by those subsets which do not exceed the capacity availability of h.

In the following sections we will propose a neighborhood structure and use the notions of *candidate leaving subsets* and *candidate entering subsets* to establish feasibility conditions under two different exploration strategies of such a neighborhood.

III. Neighborhood Structure

The neighborhood structure we propose is based on the concept of *ejection chains* introduced by Glover [11] which is characterized by sequences of moves leading from one solution to another by linked steps in which changes in selected elements cause other elements to be *ejected* from their current state, position or value assignment.

Our approach embodies the basic idea underlying ejection chains procedures since at each iteration some elements of a subset S_i are ejected from it and moved to another subset, thus forcing other elements in that subset to move to yet another subset and so on. The chain can have a variable length and can be cyclic or not (if the chain is cyclic, we will refer to it as a cyclic exchange; if not, we will call it a path exchange). Each chain uniquely identifies a new solution. The search for improving solutions obtained through exchanges of customers along a chain relies on *network flow based improvement algorithms* [1] since each improving move is characterized in terms of graph structures, and computed by algorithms on graphs. Cyclic exchanges and path exchanges for SSCFLP can be defined as follows.

A. Cyclic and Path Exchanges

Let S be a feasible solution to SSCFLP represented by a partition of the set of customers J into n subsets as described above. Let us denote by $Q = \{i_1, i_2, \ldots, i_q\}$ a sequence of indexes of q subsets of the partition.

A *cyclic multi-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 all $r = 1, \ldots, q$.

ii) For each pair r, p with $r \neq p$ $(r, p \in \{1, \ldots, q\})$, we have that $i_r \neq i_p$ (in other words each subset U_{i_r} of the sequence belongs to a different subset S_i).

Any cyclic multi-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:

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

The cyclic multi-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 considered an *improving* cycle if the new solution S' obtained from S through U is such that $C(S') < C(S)$.

A *path multi-exchange* is similar to a cyclic multi-exchange but there is no subset of elements moving from S_{i_q} to S_{i_1}. This implies that S_{i_1} will surely end up with less elements in the new solution, while S_{i_q} will gain new elements. The definition of a feasible path multi-exchange is derived from the definition of feasible cyclic multi-exchange with little modifications.

Each feasible multi-exchange (cyclic or path) generates a new feasible solution S' from the current solution S. The set of all the new solutions obtainable from S through a feasible multi-exchange defines the multi-exchange neighborhood of S. The size of this neighborhood can be very large and the search for an improving solution (possibly a local optimum) might require a heavy computational effort. In order to find a good cyclic or path exchange in the neighborhood while keeping the computational time reasonable, we search this neighborhood only partially using a heuristic approach.

The adopted scheme features a network flow based approach which searches the multi-exchange neighborhood by finding a disjoint negative cycle in an improvement graph. The strategy of searching the neighborhood using network

flow based algorithms was already embodied in the solution approach of several hard combinatorial optimization problems: Thompson and Psaraftis [19] and Gendreau et al. [9] use this strategy to solve vehicle routing problems; a similar approach is provided by Ahuja et al. [2] for the capacitated minimum spanning tree problem; Frangioni et al. [8] apply the multi-exchange techniques to the minimum makespan machine scheduling problem; Talluri [18] devices a strategy based on the search of better solutions using improvement graphs for the daily airline fleet assignment problem. Analogous approaches have been used in many other contexts. For a more complete reference see Ahuja et al. [1].

Here we will describe how the basic idea underlying all these schemes can be adapted for solving SSCFLP. We start from a description of a special improvement graph, called a *composite improvement graph*, and we show how the problem of finding an improving multi-exchange U for the current solution S can be reformulated as the search for a directed disjoint cycle with negative cost on this composite graph.

B. Improvement Graph

The composite improvement graph used to compute a neighborhood solution of the current solution S to SSCFLP, is built by merging a number K of simple exchange graphs $G^k(S) = (N^k, A^k)$, $k = 1, \ldots, K$. Each simple exchange graph $G^k(S)$ can be described as follows. Theoretically, $G^k(S)$ contains as many nodes as the number of subsets of each partition set S_i with exactly k customers. In practice, however, the inclusion of any possible subset of cardinality k would cause the size of $G^k(S)$ to grow excessively. Hence, we only consider a subgraph of it in which, for each pair of facilities i and h, there is only one node corresponding to a single subset of cardinality k that can be moved from S_i to S_h. Such a subgraph will then have $O(n(n-1))$ nodes. In order to further decrease the size of each $G^k(S)$, we choose a compact representation of it, where we maintain only a single node, denoted by i^k, for each subset S_i provided that $|S_i| \geq k$. The exact meaning of i^k is established and becomes available only at running time and depends on the search algorithm used to reach it during the neighborhood exploration phase. Note that i^k can represent different customer subsets in the same running of the search algorithm.

A generic arc (i^k, h^k) in the simple exchange graph G^k has the following meaning: the facility established at location i relinquishes a set of k customers which are then assigned to the facility located at h. The selection approach for choosing a convenient subset of k customers to be transfered from S_i to S_h will be covered in section III-C. Such a subset may not exist, in which case (i^k, h^k) is not included in the arc set A^k of G^k.

In order to consider movements of up to K customers between each facility pair, we merge the K simple graphs into a *composite improvement graph* $H^c(S) = (N^c, A^c)$.

The set of nodes N^c is given by the union of all the nodes of the G^k's, $k = 1, \ldots, K$, plus an extra node, called the *source node*, and n *dummy nodes*, d_i, one for each facility $i \in I$.

The set of arcs A^c is given by all the arcs of the K simple graphs, plus the arcs connecting the nodes of different simple graphs which are not associated with the same subset S_i. Namely, a node i^k in the simple graph G^k, $k = 1, \ldots, K$, is connected to every node of each graph G^w, with $w \neq k$, except node i^w. As for arcs belonging to the same simple graph, a directed arc (i^k, h^w) corresponds to the transfer of a subset of k customers from S_i to S_h (denoted in the following as U_{ih}). Also, such an arc implies the subsequent removal from S_h of a subset of w customers. If a subset U_{ih} of k elements cannot be found according to the subset selection strategies which will be described in III-C, A^c does not contain the arc (i^k, h^w).

Additionally, the composite improvement graph includes a set of directed arcs which connect the source node to every node i^k in N^c, plus a set of directed arcs from each dummy node d_i, $i = 1, \ldots, n$, to the source node and a set of arcs from each node i^k to each dummy node d_h with $i \neq h$.

The composite improvement graph $H^c(S)$ associated with a solution S can be seen as a multi-layered directed graph where each layer of vertices corresponds to a facility location and contains up to K nodes, plus a *dummy* node d_i which allows the transfer of elements between subsets along a path rather than a cycle. A sketch of the composite improvement graph is given in figure 1. The sets of nodes in the rectangular boxes represent two different layers of the graph.

The inclusion of the source node s and of the dummy nodes d_i in the composite improvement graph derives from the need for possibly removing sets of customers from facilities without adding new customers to them. In fact, the presence of an arc from the source node to any other node i^k implies that the subset S_i loses a subset of k customers but no other subset is added to it. An arc connecting a node i^k to a dummy node d_h belonging to two different layers i and h means that a subset of k users U_{ih} moves from S_i into S_h but no subset moves out of S_h. From each dummy node, the only possibility is to go back to the source node. Thus, each cycle in the graph which contains the source node and, hence, a dummy node corresponds to a path exchange.

The use of path exchange is crucial for SSCFLP since we might eventually eject all the elements of a subset S_i without replacing them. This move would cause the closing of the facility currently opened at site i thereby saving the fixed cost associated to it. Also, a new facility can be opened at a site i if we select a cycle passing through dummy node d_i, given that S_i is initially empty. The possibility of changing the state of a facility from open to close and vice versa would not be allowed if we restricted the search to the cyclic exchange neighborhood only.

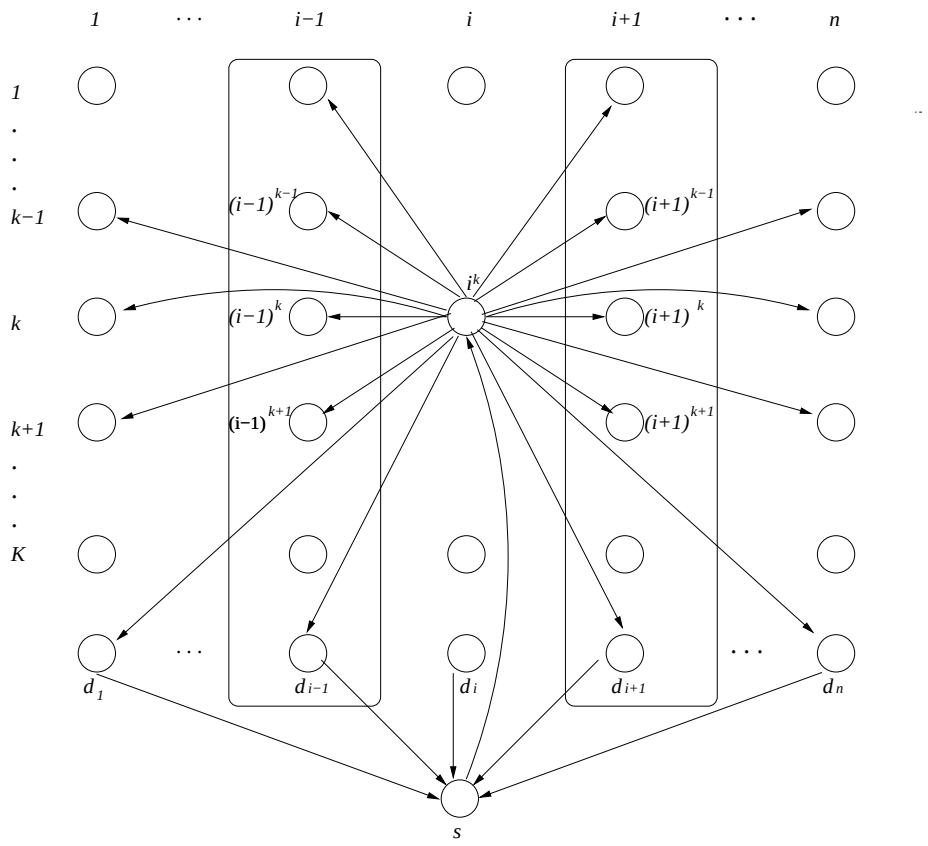

Fig. 1. Composite Improvement Graph

B.1 Arc Costs

Before we define the costs for each type of arc in A^c, it is worthwhile to underline an important property of the composite improvement graph, deriving from our definition of the neighborhood structure, i.e.: H^c is an implicitly defined dynamic graph since the costs of its arcs cannot be determined a priori, but only during the neighborhood exploration phase. It is only at that stage, in fact, that we can know the exact meaning of each node i^k and, consequently, the subset of customer which is to be moved.

In the following definition of the arc costs we will assume that the customer subset U_{ih} to be moved from facility i to facility h has already been computed. The idea is that the cost of each arc (i, h) must reflect the cost decrease of the relinquishing subset S_i and the cost increase of the receiving subset S_h. Clearly, the costs must take into account the possible savings from closing a facility with no customers assigned to it as well as the fixed costs due to the assignment of customers to a facility previously closed. We assign the savings for facility closing to the arcs from the source node to regular nodes and the costs for facility opening to the arcs from dummy nodes to the source node (we remind that the change of a facility state from open to close and vice versa is possible only along cycles which include the source node and a dummy node).

The cost of each type of arc in the composite improvement graph can be defined as follows.

The cost $\gamma_{i^k h^w}$ of each arc (i^k, h^w) connecting two regular nodes i^k and h^w is:

$$\gamma_{i^k h^w} = \sum_{j \in U_{ih}} c_{hj} - \sum_{j \in U_{ih}} c_{ij} \tag{14}$$

Equation (14) states that the cost of the generic arc (i^k, h^w) is equal to the sum of the cost increase of S_h due to insertion of U_{ih} and the cost decrease of S_i due to removal of U_{ih}.

The above definition also holds for each arc (i^k, d_h) from a regular node to a dummy node, i.e.:

$$\gamma_{i^k d_h} = \sum_{j \in U_{ih}} c_{hj} - \sum_{j \in U_{ih}} c_{ij} \tag{15}$$

Each arc from the source node to a regular node i^k has null cost unless the cardinality of S_i is exactly k, in which case the cost of the arc is equal to the cost saving for closing facility i, i.e.:

$$\gamma_{s i^k} = \begin{cases} 0 & \text{if } |S_i| \neq k \\ -f_i & \text{if } |S_i| = k \end{cases} \tag{16}$$

Finally, the cost $\gamma_{d_h s}$ of each arc from a dummy node back to the origin is null if facility h has already customers

assigned to it; it is equal to the fixed cost for opening facility h if subset S_h is initially empty, i.e:

$$\gamma_{d_h s} = \begin{cases} 0 & \text{if } S_h \neq \emptyset \\ f_h & \text{if } S_h = \emptyset \end{cases} \quad (17)$$

B.2 Negative subset-disjoint cycles

Having provided a full description of the composite improvement graph, we can now define negative subset-disjoint cycles.

Definition III.1: We say that a directed cycle $\{i_1^k, \ldots, i_q^w\}$ with $k, w \in \{1, \ldots, K\}$, in the composite improvement graph $H^c(S)$ associated with solution S is a *negative subset-disjoint cycle* if each one of its nodes corresponds to a different subset S_i 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 equivalence between the problem of detecting a negative subset-disjoint cycle in $H^c(S)$ and the problem of finding a feasible profitable multi-exchange for S is based on the property that the neighborhood search algorithm, which will be described in section IV, 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 methods used for determining such customer subsets guarantee that the move keeps the solution feasible, as described in section III-C.

From the above definitions and observations the following property can be easily proved:

Property III.1: Each sequence of customer subsets uniquely identified by a subset disjoint cycle in $H^c(S)$ corresponds to a feasible cyclic [path] exchange with respect to S. Furthermore, if the subset-disjoint cycle is negative, the corresponding cyclic [path] exchange is a profitable exchange.

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

As an example, if $\{i_1^k, i_2^w \ldots, i_q^z\}$ is the cycle returned by the search algorithm, the customer subset sequence $\{U_{i_1 i_2}, U_{i_2 i_3} \ldots, U_{i_q i_1}\}$, with $|U_{i_1 i_2}| = k, |U_{i_2 i_3}| = w, \ldots, |U_{i_q i_1}| = z$, corresponds to a cyclic exchange. If the cycle has the form $\{s, i_1^k, \ldots, i_{q-1}^z, d_{i_q}\}$, the customer subset sequence $\{U_{i_1 i_2}, \ldots, U_{i_{q-1} i_q}\}$, with $|U_{i_1 i_2}| = k, \ldots, |U_{i_{q-1} i_q}| = z$, corresponds to a path exchange.

As a result of property III.1, we have transformed the problem of finding profitable exchanges in the multi-exchange neighborhood into identifying negative subset-disjoint cycles into the composite improvement graphs H_S^c.

In order to detect negative subset-disjoint cycles in the composite improvement graph, we adopt two different algorithms, subsequently referred to as the *forward search algorithm* and *backward (or reverse) search algorithm*. The two algorithms, which will be detailed in section IV, differ in the way they process the arcs during the search. Namely, the first scans the outgoing arcs of the node under examination, while the latter scans its incoming arcs. As a consequence, if we execute a forward search of the graph, the customer subset entering a node is selected before the customer subset which leaves it; if we use the backward search, the entering customers are selected after the leaving customers.

We now describe how to perform the selection of moving subsets in each of the two search approaches.

C. Selection of Moving Subsets

We will consider three different ways to select the subsets to be moved from or into a subset S_h during the search of forward or backward exchanges, respectively. This corresponds to selecting and working on different subgraphs of the composite improvement graph.

C.1 Subset Selection in Forward Neighborhood Search

Assume that during the forward search for a subset-disjoint cycle node h is reached from node i, so that we know that U_{ih} is the subset of customers moving from S_i into S_h. The situation is depicted in figure 2. Now, for each node l reachable from node h, we must determine the "best" subset of customers to be transfered from facility h to facility l. (For simplicity of notation, we omit the index k associated with each node. In the following, we just imply that i, h and l are generic nodes of H^c and that we are looking for a subset of exactly k elements to be transfered).

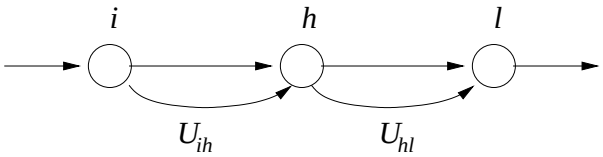

Fig. 2. Forward Search

The first strategy that we consider for selecting the customers to be moved to facility l consists in choosing k elements among the ones currently assigned to h in such a way that minimizes the added costs due to the assignment of these customers to facility l, provided that the chosen subset is a candidate leaving subset with respect to S_h and U_{ih}. This subset will be denoted as the *most profitable* with respect to facility l since it attempts to keep the cost of subset S_l as low as possible. The corresponding exchange is referred to as the *most profitable forward exchange*.

The second strategy selects the candidate leaving subset of cardinality k which guarantees the maximum cost decrease of subset S_h with no consideration for the cost increment of the receiving subset S_l. Thus, the leaving subset will be indicated as the *least profitable subset* with respect to S_l, and the exchange as the *least profitable forward exchange*.

Finally, the third strategy seeks for a tradeoff between the cost decrease of S_h and the cost increase of S_l due to the transfer of a candidate leaving subset of k elements from h to l. The moving subsets selected with this strategy are

referred to as *profitable subsets* and their transfer originates *profitable forward exchanges*.

The problem of finding a candidate leaving subset with respect to S_h and U_{ih} to be moved from facility h to facility l can be mathematically stated as a combinatorial optimization problem, denoted by (P_f), as follows.

$$\min \quad \sum_{j \in S_h} \hat{c}_{hl} x_j \tag{18}$$

$$\text{s.t.} \quad \sum_{j \in S_h} x_j = k \tag{19}$$

$$\sum_{j \in S_h} w_j x_j \geq gl(U_{ih}, S_h) \tag{20}$$

$$x_j \in \{0,1\} \qquad \forall\ j \in S_h \tag{21}$$

where the decision variable x_j takes value 1 if customer j is moved from S_h to S_l, 0 otherwise. Constraint (19) ensures that U_{hl} is a subset of exactly k users; constraint (20) guarantees that the removal of U_{hl} from S_h restores the capacity limit of facility h which might have been violated after the insertion of U_{ih}. In other words, U_{hl} must be a candidate leaving subset with respect to the entering subset U_{ih} and S_h.

Finally, the cost $\hat{c}_{hl}$ can be defined in three different ways, according to the three strategies described above. Namely,

$$\hat{c}_{hl} = \begin{cases} c_{lj} & (\textit{most profitable}) \\ -c_{hj} & (\textit{least profitable}) \\ c_{lj} - c_{hj} & (\textit{profitable}) \end{cases}$$

A solution to problem P_f 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 problem P_f. The first occurs when $i = s$ since in this case there is no subset moving from the source node into node h. This implies that $gl(U_{ih}, S_h) = -r(S_h) \leq 0$ and the capacity constraint is always satisfied. P_f can then be simplified by removing constraint (20). The second situation occurs when, during the search phase, we detect a subset-disjoint cycle and must select the customer subset $U_{i_q i_1}$ in order to close the cycle. In this case we must guarantee not only that the removal of this subset from $S_h = S_{i_q}$ restores the capacity constraint for facility h, according to (20), 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 P_f the additional constraint:

$$\sum_{j \in S_{i_q}} w_j x_j \leq ge(U_{i_1 i_2}, S_{i_1}) \tag{22}$$

Inequality (22) states that $U_{i_q i_i}$ must be a candidate entering subset with respect to the leaving subset $U_{i_1 i_2}$ and the partition set S_{i_1}.

Additional constraints of the form $x \in X$ might also be added to problem P_f in the general case, for example to include restrictions to customer movements based upon distance considerations.

C.2 Subset Selection in Backward Neighborhood Search

Similarly to the case of forward search, we allow three strategies to identify the subset of customers entering a node h of the improvement graph during the backward search for negative subset-disjoint exchange cycles. When the backward search reaches node h from node l, we know that the set U_{hl} leaves subset S_h and is added to S_l (see fig. 3). We need to determine the "best" subset U_{ih} moving into facility h for each node i such that $(i,h) \in A^c$.

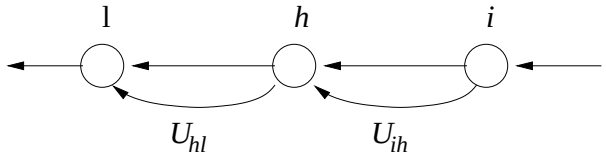

Fig. 3. Backward Search

The three strategies for backward search correspond to the selection of: *the most profitable subset* U_{ih} if the exchange aims at minimizing the cost increment of S_h due to the assignment of the customers in U_{ih} to facility h; *the least profitable subset* U_{ih} if the exchange aims at maximizing the cost reduction due to the removal of U_{ih} from S_i; *the profitable subset* U_{ih} if we seek for a combination of the two previous goals. The use of these three strategies produces *most profitable backward exchanges*, *least profitable backward exchanges*, and *profitable backward exchanges*, respectively.

The subset U_{ih} of entering users can be determined by solving the following combinatorial optimization problem (P_b).

$$\max \quad \sum_{j \in S_i} \hat{c}_{ih} x_j \tag{23}$$

$$\text{s.t.} \quad \sum_{j \in S_i} x_j = k \tag{24}$$

$$\sum_{j \in S_i} w_j x_j \leq ge(U_{hl}, S_h) \tag{25}$$

$$x_j \in \{0,1\} \qquad \forall\ j \in S_i \tag{26}$$

where the decision variable x_j takes value 1 if customer j is moved from S_i to S_h, 0 otherwise. Constraint (24) ensures that $|U_{ih}| = k$; the set of constraints (25) guarantees that after the insertion of U_{ih} in S_h, the capacity limit of facility h is not exceeded, i.e. U_{ih} is a candidate entering subset with respect to S_h and U_{hl}.

The cost $\hat{c}_{ih}$ for each strategy is given by:

$$\hat{c}_{ih} = \begin{cases} -c_{hj} & (\textit{most profitable}) \\ c_{ij} & (\textit{least profitable}) \\ c_{ij} - c_{hj} & (\textit{profitable}) \end{cases}$$

The subset U_{ih} is obtained by setting $U_{ih} = \{j \in S_i | x_j = 1\}$.

Problem P_b needs to be modified under two circumstances. The first occurs when computing the customer subset to be transfer into a dummy node d_h. In this case there is no subset exiting S_h and we must guarantee that the total demand of the candidate entering subset does not exceed the residual capacity of facility h, instead of the capacity gap value for entering subsets. Condition (25) is then replaced by:

$$\sum_{j \in S_i} w_j x_j \leq r(S_h) \tag{27}$$

The second situation occurs when a cycle $\{i_1, i_2, \ldots, i_q\}$ is detected and we must select the subset $U_{i_1 i_2}$ which closes the cycle (note that we are working backward so that the last arc found during the search of the cycle is (i_1, i_2)). In this case we must guarantee that not only the insertion of this subset in S_{i_2} $(h = i_2)$ does not violate the capacity limit of facility i_2 according to (25), but also that its removal from S_{i_1} $(i = i_1)$ restores the capacity constraint for facility i_1. Hence we must add to problem P_b the additional constraint:

$$\sum_{j \in S_{i_1}} w_j x_j \geq gl(U_{i_q i_1}, S_{i_1}) \tag{28}$$

Inequality (28) states that $U_{i_1 i_2}$ must be a candidate leaving subset with respect to the entering subset $U_{i_q i_i}$ and the partition set S_{i_1}.

Problems P_f and P_b have the structure of a knapsack problem with an additional constraint fixing the number of objects to be selected. Such a problem can be solved using a heuristic method based on the ideas contained in Martello and Toth [13] or through the approximation algorithms described in [6].

IV. Search of Negative Subset-Disjoint Cycles

As already mentioned in section III-B, we use two different strategies to explore the composite improvement graph in the attempt to detect negative subset-disjoint cycles. Both approaches are modifications of the well-known label-correcting algorithm for the shortest path problem. The major variation concerns the check for subset disjointness.

In practice, the forward and backward search strategies characterize two main families of algorithms which we will consider next. The algorithms in each family differ for the particular data structure used to implement the set of candidate nodes to be examined during the algorithm execution.

A relevant feature of our improvement graph is that its peculiar dynamic nature invalidates the shortest path optimality conditions on which label-correcting algorithms are based. In fact, in our particular case, we cannot state that if P is an optimal path from a node s to a node t in H^c, 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 needs to maintain multiple labels for each node, each one corresponding to a different subpath through which the given node can be reached.

However, since our heuristic approach only aims at finding a negative cycle (not necessarily the best), we choose to maintain for each node i^k only one cost label $d(i^k)$ 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 satisfied. This particular case occurs when the arcs emanating from [terminating at] the current node have already been examined during the forward [backward] 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 clarified and detailed in the next paragraph.

A. Forward Search Algorithm

To detect a negative subset-disjoint cycle in a forward fashion, we start from the generic label correcting framework by maintaining a cost label $d(i)$ and a *predecessor* $pred(i)$ for each node i in N^c. In addition we add 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.

We start the search at the source node and, at each iteration of the algorithm, we extract a node, denoted by i^k, from Q, the set of candidate nodes still to be examined. We then perform the following operations until either Q is empty or we find a subset-disjoint negative cycle.

We first check if the path from the source node to i^k is subset disjoint. If not, we skip node i^k and proceed to the extraction of another node from Q. Otherwise, we examine all the arcs emanating from i^k. For each arc $(i^k, h) \in FS(i^k)$, we check if the path from s to h is a subset-disjoint path. We have two possibilities:

1. If the path from s to h is a subset-disjoint path, we execute the following steps.

(a) We determine the subset of customers U_{ih} such that $|U_{ih}| = k$ by solving problem P_f according to one of the three selection strategies described in III-C.1. (Note that if we use the *least profitable strategy* the subset U_{ih} will be the same for each arc emanating from i^k, so that only one problem P_f need be solved for all the arcs in the forward star of i^k).

(b) We compute the cost $\gamma_{i^k h}$ of arc (i^k, h)using the subset U_{ih} determined at step (1a).

(c) We check the Bellman conditions for (i^k, h). If $d(h) \leq d(i^k) + \gamma_{i_k h}$, we just skip the arc under consideration and examine another arc. Otherwise, one of the following three

cases can occur:

i. $h = h^w$, for some $w \in \{1, \ldots, K\}$, i.e. 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. $pred(h) = i^k$, $d(h) = d(i^k) + \gamma_{i^k h}$, $U(h) = U_{ih}$. Further, if the total demand of U_{ih} is smaller than the demand of the other subsets associated with the arcs in $FS(i)$, then $min\_w(i^k) = W(U_{ih})$.

ii. $h = h^w$, for some $w \in \{1, \ldots, K\}$, i.e. h is a *regular node*, and its forward star has already been examined, i.e. h has been extracted from Q at a previous iteration. In this case, the customer subsets U_{hl} associated with the arcs (h, l) emanating from h have already been determined by solving instances of problem P_f. If we then update the labels of h, those subsets might no longer be candidate leaving subset with respect to S_h and the new entering subset U_{ih}, i.e. the capacity constraint (20) in P_f might be violated. This would lead to the need for recomputing some subset U_{hl} which in turn might cause some other capacity constraints to be violated and so on. To avoid this problem, we only update the labels of h if we can guarantee that after the update all the leaving subsets U_{hl} are still feasible. To perform this control, we just need to check if the following condition involving the label $min\_w(h)$ is satisfied:

$$W(U_{ih}) \leq r(S_h) + min\_w(h). \tag{29}$$

If (29) holds, then the labels of node h can be updated as in step (1(c)i), otherwise we skip to the next arc in $FS(i^k)$.

iii. $h = d_h$, i.e. h is a *dummy node*. In this case a path exchange has been found. In fact, by adding arc (d_h, s), we can close a cycle in H^c which includes a dummy node and the source node. If the cost of the cycle is negative, the algorithm returns the path exchange and terminates, otherwise we examine the next arc in $FS(i^k)$.

2. If the path from s to h is not a subset-disjoint path, i.e. along the path a node h^z associated with facility h already exists, then we must consider the two following cases.

(a) $h = h^w$, for some $w \in \{1, \ldots, K\}$, i.e. h is a *regular node*. In this case a cycle has been found. We execute steps (1a)-(1b) as for the case of subset-disjoint paths, taking into account that the computation of U_{ih} at step (1a) requires the solution of problem P_f with the additional constraint (22) necessary to close the cycle while maintaining feasibility. If $d(i^k) + \gamma_{i^k h^w} - d(h^z) < 0$, a profitable cyclic exchange has been found. Otherwise, we skip the arc.

(b) $h = d_h$, i.e. h is a *dummy node*. In this case, we can skip arc (i^k, h) since it would produce the same cycle which is generated when considering the arcs from i^k to the other regular nodes h^w, with $w \in \{1, \ldots, K\}$.

B. Backward Search Algorithm

The backward search algorithm is similar to the forward search algorithm, but it works by scanning the arcs in reverse order and maintaining for each node h in N^c a label $succ(h)$ (instead of the label *pred* used by the previous algorithm), which indicates the successor of h along the path. Additionally, the label $U(h)$ and $max\_w(h)$ are associated with each regular node which store, respectively, the subset of customers which moves from h into its successor along the path and the maximum demand of the customer subsets associated with the arcs (i, h) terminating at node h. Formally, $max\_w(h) = \max_{(i,h) \in BS(h)} W(U_{ih})$, where $BS(h)$ denotes the backward star of node h. Labels $d(h)$ and $succ(h)$ are initially set to ∞, $max\_w(h)$ to zero and $U(h)$ to the empty set.

The backward search starts at the source node and scans its backward star which only contains arcs coming from dummy nodes. It updates the distance labels associated with them according to the cost definition given in section III-B.1, sets their successor to s and inserts them in Q. After this initialization phase, the generic iteration of the algorithm, which is repeated until either Q is empty or we find a subset-disjoint negative cycle, can be described as follows.

We extract a generic node h from Q and check if the path from h to the source node is subset disjoint. If not, we skip node h and proceed to the extraction of another node. Otherwise, we examine the incoming arcs of h. For each arc $(i, h) \in BS(h)$, we check if the path from i to s is a subset-disjoint path. We have two possibilities:

1. If the path from i to s is a subset-disjoint path (note that in this case i can only be a regular node, so that $i = i^k$ for some $k \in \{1, \ldots, K\}$), we execute the following steps.

(a) We determine the subset of customers U_{ih} such that $|U_{ih}| = k$ by solving problem P_b according to one of the three selection strategies for backward search described in (III-C.2).

(b) We compute the cost $\gamma_{i^k h}$ of the arc (i^k, h) using the subset U_{ih} determined at step 1a.

(c) We check the Bellman conditions for (i^k, h). If $d(i^k) \leq d(h) + \gamma_{i_k h}$, we skip the arc under consideration and examine another arc. Otherwise, we can have two cases:

i. The backward star of i^k has not been examined yet. In this case i^k is inserted in Q and its labels are updated, i.e. $succ(i^k) = h$, $d(i^k) = d(h) + \gamma_{i^k h}$, $U(i^k) = U_{ih}$. Further, if the total demand of U_{ih} is greater than the demand of the other subsets associated with the arcs in $BS(h)$, then $max\_w(h) = W(U_{ih})$.

ii. The backward star of i^k has already been examined, i.e. i^k has been extracted from Q at a previous iteration. In this case, the customer subsets U_{li} associated with the arcs (l, i^k) terminating at i^k have already been determined by solving instances of problem P_b. If we then update the labels of i^k, those subsets might no longer be candidate entering subset with respect to S_{i^k} and the new leaving subset U_{ih}, i.e. the capacity constraint (25) in P_b might be violated. This would lead to the need for recomputing some subset U_{li}, which in turn might cause some other capacity constraints to be violated and so on.

To avoid this problem, we only update the labels of i^k if we can guarantee that after the update all the entering subsets U_{li} are still feasible. To perform this control, we just need to check if the following condition involving the label $max\_w(i^k)$ is satisfied:

$$W(U_{ih}) \geq max\_w(i^k) - r(S_{i^k}), \qquad (30)$$

If (30) holds, then the labels of node h can be updated as in step (1(c)i), otherwise we skip to the next arc in $BS(i^k)$.

2. If the path from i to s is not a subset-disjoint path, we can have two cases:

(a) $i = s$, i.e. i is the *source node*. In this case a path exchange has been found. If the cost of the path is negative, the algorithms terminates, otherwise we extract a new node from Q.

(b) $i = i^k$, for some $k \in \{1, \ldots, K\}$, i.e. i is a *regular node*: in this case a cyclic exchange has been found. Assume that node i^z is the node associated with facility i already present in the path. We execute steps (1a)-(1b) as for the case of subset-disjoint paths, taking into account that the computation of U_{ih} at step (1a) requires the solution of problem P_b with the additional constraint (28). If $d(h) + \gamma_{i^k h} - d(i^z) < 0$, a profitable cyclic exchange has been found. Otherwise, we examine another arc.

V. Neighborhood Search Algorithms

The multi-exchange neighborhood structure developed in this paper is embedded in a local improvement algorithm which starts with a feasible solution and performs profitable multi-exchanges to improve it until it gets a locally optimal solution. To avoid early stopping at poor local optima, the neighborhood search algorithm includes a restart procedure, meaning that a single run is repeated several times starting from different initial solutions. The restart procedure requires a mechanism to generate feasible initial solutions. This is accomplished by randomly selecting subsets of facilities to be opened and assigning customers to them by solving heuristically a *generalized assignment problem*.

Another implementation issue of the local improvement algorithm concerns the rules to manage the set of candidate nodes Q during the negative cycle search phase. 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 preliminary investigation we consider three different implementations of Q corresponding to the first in-first out policy, last in-first out policy and to the selection of the node with minimum label.

VI. Conclusions

In this paper we outlined the main issues characterizing a new kind of neighborhood search algorithms to solve the capacitated facility location problem with single source constraints (SSCFLP). The innovative aspect of the proposed approach lies in the design of a very large scale neighborhood and the development of a network optimization based methodology to efficiently search cost-decreasing neighbors in it. Further, whereas existing cyclic neighborhood structures generally only consider transfers of single elements among partition subsets, our neighborhood structure includes the possibility of moving clusters of elements simultaneously. Since there are an exponential number of possible clusters, we suggested a method to select a priori promising clusters to be moved. The method consists in building a dynamic composite improvement graph with a peculiar significance associated to each arc.

We intend to exploit the proposed method to solve large instances of SSCFLP and extend it to the solution of other location problems.

References

[1] R.K. Ahuja, O. Ergun, J.B. Orlin, and A.P. Punnen. A survey of very large-scale neighborhood search techniques. Working paper, July 1999.

[2] R.K. Ahuja, J.B. Orlin, and D. Sharma. New neighborhood search structures for the capacitated minimum spanning tree problem. Research Report 99-2, University of Florida, Department of Industrial and Systems Engineering, 1999.

[3] R.K. Ahuja, S. Pallottino, M.G. Scutellà, and M.P. Scaparra. New neighborhood search structures for the single source capacitated facility location problem. Working paper, 2001.

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

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

[8] 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. (Submitted to Journal on Combinatorial Optimization).

[9] M. Gendreau, F. Guertin, J.Y. Potvin, and R. Seguin. Neighborhood search heuristics for a dynamic vehicle dispaching problem with pick-ups and deliveries. CRT-98-10, Université de Montréal, 1998.

[10] A.M. Geoffrion and G.W. Graves. Multicommodity distribution system design by Benders decomposition. *Mathematical programming*, 2:82–114, 1974.

[11] F. Glover. Ejection chains, reference structures, and alternating path algorithms for the traveling salesman problem. *Discrete Applied Mathematics*, 65:223–253, 1996.

[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] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, New York, 1990.

[14] 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.

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

[16] M. Ronnqvist and J. Tragantalerngsak, S. an 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] K.T. Talluri. Swapping application in a daily airline fleet assignment. *Transportation Science*, 30, 1996.

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