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

Trung Hieu Tran,[a] Maria Paola Scaparra,[b*] Jesse R. O'Hanley[b]

[a]*Department of Statistics, University of Warwick, Coventry, CV4 7AL, U.K.*

[b]*Kent Business School, University of Kent, Canterbury, CT2 7PE, U.K.*

Abstract

In this paper, we introduce a large-scale neighborhood search procedure for solving the single-source capacitated facility location problem (SSCFLP). The neighborhood structures are induced by innovative *split multi-customer multi-exchanges,* where clusters of customers assigned to one facility can be moved simultaneously to multiple destination facilities and vice versa. To represent these exchanges, we use two types of *improvement hypergraphs.* The improvement hypergraphs are built dynamically and the moving customers associated with each hyperedge are selected by solving heuristically a suitably defined mixed-integer program. We develop a hypergraph search framework, including forward and backward procedures, to identify improving solutions efficiently. Our proposed algorithm can obtain improving moves more quickly and even find better solutions than a traditional multi-exchange heuristic (Ahuja et al., 2004). In addition, when compared with the Kernel Search algorithm (Guastaroba and Speranza, 2014), which at present is the most effective for solving SSCFLP, our algorithm is not only competitive but can find better solutions or even the best known solution to some very large scale benchmark instances from the literature.

Keywords: facility location; large-scale neighborhood search; multi-exchange heuristic; hypergraphs

1 Introduction and Background

Location science is a very active area of research (Laporte et al., 2015). Facility location problems naturally arise in many different application settings, including transportation, supply chain management, emergency logistics, telecommunications, and healthcare, to name a few, and play a crucial role in strategic planning for both the public and private sectors. One of the best known location problems is the single-source capacitated facility location problem (referred to as SSCFLP). SSCFLP determines where to locate facilities out of n possible candidate sites (denoted by set I) so as to serve a set of m customers (denoted by J) at minimum cost. Each customer $j \in J$ has an associated demand w_j that must be supplied by a single facility $i \in I$. The cost for supplying all of the demand of customer j from a facility located at i is denoted by c_{ij}. Each

[*]Correspondence email: `m.p.scaparra@kent.ac.uk`

facility $i \in I$ has a maximum capacity s_i and a fixed setup cost f_i. To state the problem mathematically, the following decision variables are introduced:

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

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

The SSCFLP can then be formulated as follows.

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

$s.t.$

$$\sum_{i \in I} x_{ij} = 1 \qquad \forall j \in J \tag{2}$$

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

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

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

The objective (1) minimizes transportation costs plus setup costs. Constraints (2) ensure that each customer $j \in J$ is assigned to exactly one facility $i \in I$. Constraints (3) state that the total customer demand assigned to a facility $i \in I$ can not exceed its maximum capacity. Constraints (4) and (5) enforce binary restrictions on the allocation and location variables, respectively.

SSCFLP belongs to the class of NP-hard problems (Yang et al., 2012) and is much harder to solve than its multiple allocation counterpart, where customer demands can be split among several open facilities (Klose and Drexl, 2005). In spite of recent advances in computer technology and the performance of modern commercial optimization software, large instances of the SSCFLP remain difficult to solve. Over the decades, researchers have proposed several specialized algorithms to identify optimal or near-optimal solutions to SSCFLP. The solution approaches can be broadly classified into two categories: exact methods and heuristics. One of the first attempts at solving SSCFLP exactly is the branch-and-bound scheme proposed by Neebe and Rao (1983). Holmberg et al. (1999) propose a different branch-and-bound framework, which integrates a Lagrangean dual heuristic for the computation of lower bounds and a strong primal heuristic for the computation of upper bounds. Diaz and Fernandez (2002) use a column generation method to determine lower and upper bounds within a branch-and-price framework. The most recent exact method for SSCFLP is the cut-and-solve algorithm developed by Yang et al. (2012). The authors report optimal solutions for large problem instances with up to 80 candidate sites and 400 customers.

Among heuristic approaches, the most widely studied are those based on Lagrangean relaxation. These approaches mainly differ in terms of the constraints that are relaxed (e.g., assignment constraints (2), capacity constraints (3), or both). Examples of Lagrangean heuristics based upon the relaxation of constraints (2) can be found in Barcelo and Casanova (1984), Pirkul (1987), Sridharan (1991), Hindi and Pienkosz (1999), and Ronnqvist et al. (1999). The relaxation of constraints (3) is considered by Klincewicz and Luss (1986) and Cortinhal and Captivo (2003), whereas Beasley (1993) and Agar and Salhi (1998) relax both sets of constraints. Finally, Chen and Ting (2008) propose a hybrid approach where a Lagrangean heuristic based on the relaxation of the demand constraints is used to identify the facility locations, while customer allocations are determined by a multiple ant colony system metaheuristic. For an overview of Lagrangean based solution approaches for SSCFLP, the reader is referred to Galvao and Marianov (2011).

Other heuristics for SSCFLP include: Tabu search (Filho and Galvao, 1998); variants of reactive GRASP and Tabu search (Delmaire et al., 1999); very large-scale neighborhood search (Ahuja et al., 2004); scatter search (Contreras and Diaz, 2008); and iterated Tabu search (Ho, 2015). To date, the most effective heuristic for solving SSCFLP is the Kernel Search procedure recently introduced by Guastaroba and Speranza (2014).

In this paper, we propose a very large-scale neighborhood search (VLNS) heuristic for solving SSCFLP. Our heuristic extends and improves the multi-exchange heuristic proposed by Ahuja et al. (2004) by introducing several novel elements in the design of VLNS approaches. The premise behind VLNS is to explore very large neighborhoods using a local search framework in the hope of finding high-quality, locally-optimal solutions. Clearly, the larger the neighborhood, the more time is required to explore and identify improving solutions. VLNS techniques typically exploit fast heuristics and network flow based algorithms to detect improving solutions efficiently (Ahuja et al., 2002).

The idea of building very large-scale neighborhood structures and exploring them using network flow algorithms originated with Thompson and Orlin (1989) for solving hard combinatorial optimization problems with a set-partitioning structure. The authors consider a large neighborhood structure induced by the cyclical exchange of elements among partition subsets, thus extending the single-swap idea characterizing traditional local search algorithms. An improving neighbor in the cyclic exchange neighborhood is obtained by finding a negative cost subset-disjoint cycle in a suitably defined *improvement graph*. Although detecting such a cycle is itself NP-hard, heuristics can be employed for searching the graph efficiently (Ahuja et al., 2001).

VLNS algorithms based on the idea of searching for special paths or cycles in an improvement graph have been widely studied and successfully applied to several hard combinatorial optimization problems. These include: vehicle routing problems (Thompson and Psaraftis, 1993); airline fleet assignment problems (Talluri, 1996); capacitated minimum spanning tree problems (Ahuja et al., 2001); machine scheduling problems (Frangioni et al., 2004); capacitated facility location problems, such as SSCFLP and the p-center problem (Ahuja et al., 2004; Scaparra et al., 2004); k-constraint multiple knapsack problems (Ahuja and Cunha, 2005); covering assignment and single source transportation problems (Öncan et al., 2008); location routing problems (Ambrosino et al., 2009); location problems with flexible demand (Rainwater et al., 2012); and hierarchical facility location problems (Addis et al., 2013).

Among the aforementioned papers, of particular relevance to our work is the paper by Ahuja et al. (2004). This paper introduces a novel aspect in the development of VLNS techniques and significantly improves the performance of traditional multi-exchange heuristics. Previous multi-exchange neighborhood structures only considered cyclic transfer of single elements. These transfers are represented on an improvement graph where each node corresponds to an element of the partition that can be moved (e.g., a customer) and each arc denotes the transfer of a “tail” element toward the partition subset (e.g., a facility) containing the “head” element. Ahuja et al. (2004) extend this scheme by allowing the cyclic movement of element clusters. Since there is an exponential number of possible clusters that can be considered for movement, the authors propose an alternative way of building an improvement graph, in which each node corresponds to a partition subset and each arc denotes the movement of a cluster of elements from the tail subset to the head subset. The improvement graph is built dynamically and only promising element clusters, selected in a greedy manner, are considered for movement between any pair of nodes. In the case of SSCFLP, the improvement graph, called a *facility improvement graph*, is built in such a way that the dynamic selection of the moving clusters along arcs guarantees the feasibility of the move with respect to the capacity constraints. Indeed, it is possible that the movement of a cluster of customers towards the partition subset associated with an open facility could result in the violation of its capacity. If this is the case, the selection of customers that are moved out to another partition subset in the cycle must ensure that enough customers are removed to restore capacity. This is done by solving heuristically a knapsack-type problem. The authors demonstrate that feasible and profitable multi-customer multi-exchanges can be identified by searching for negative-cost, subset-disjoint paths or cycles in the dynamic facility improvement graph. To perform the search efficiently, they propose a few variants of the well-known label-correcting algorithm for shortest path problems.

Although the multi-customer multi-exchange heuristic proved to be extremely effective in solving large scale instances of SSCFLP, there are two drawbacks that limit its full potential: 1) in some cases, the procedure cannot reach an optimal solution because some feasible and profitable multi-customer moves cannot be modeled and detected in the facility improvement graph; 2) the multi-exchange heuristic mainly reassigns customers among already open facilities but only rarely leads to the opening of new facilities or the closing of facilities. As a consequence, finding improving configurations of open facilities requires using alternative, more traditional facility moves (e.g., swaps).

The multi-exchange heuristic introduced in this paper aims at redressing these two shortcomings by proposing new and more flexible multi-customer multi-exchanges. We do this by employing *hypergraph theory* for defining and searching the improvement graph. The use of this methodology is quite general in scope and can be applied to other hard combinatorial optimization problems with a set-partitioning structure.

The rest of this paper is organized as follows. In Section 2, we provide a practical example of the main shortcomings of the multi-customer multi-exchange heuristic and briefly outline the rational behind our new approach. Some basic definitions and properties of hypergraphs are presented in Section 3. The hypergraph multi-exchange heuristic is described in detail in Section 4. In Section 5, we present some computational results from a large suite of benchmark problems, together with a performance comparison with the multi-exchange heuristic of Ahuja et al. (2004) and the Kernel Search algorithm of Guastaroba and Speranza (2014). Finally, some concluding remarks and suggestions for future research are provided in Section 6.

2 Limitation of Traditional Multi-Customer Multi-Exchanges

As mentioned in the previous section, there are some cases where the multi-customer multi-exchange heuristic (Ahuja et al., 2004) fails to detect feasible and profitable moves. One such case is displayed in Fig. 1. The figure shows a solution to a simple SSCFLP with 4 facilities (squares) and 16 customers (circles). Assume that during the search for a multi-customer multi-exchange, the heuristic has identified the cluster of customers to be moved from facility 1 to facility 2 and the cluster of customers to be moved from facility 2 to facility 3 (selected clusters are shown within dashed rectangles). The reassignment of the customer to facility 3 violates its capacity constraint. To restore capacity, a knapsack type problem is solved to select the best subset of customers that must leave facility 3 and go to another facility.

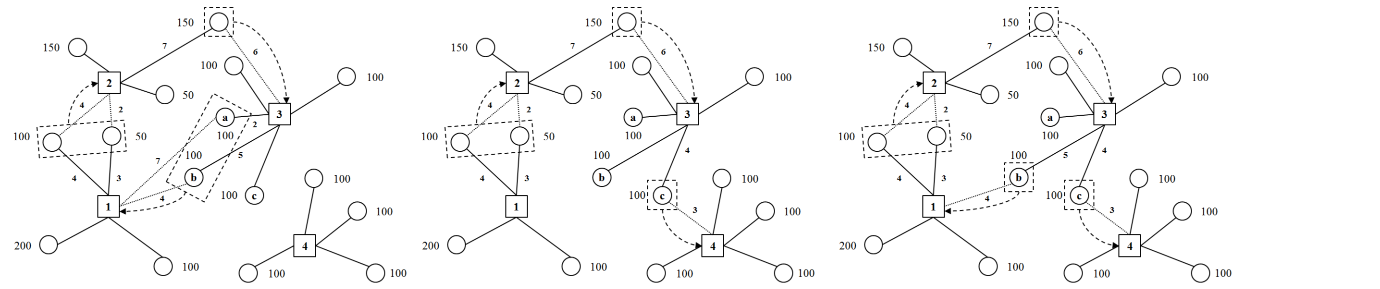

(a) (b) (c)

Figure 1: Illustrative example of a feasible but non-profitable move (a), a profitable but infeasible move (b), and a profitable and feasible split move (c). Each facility has a capacity of 500 units. Customer demands and assignment costs are shown next to each node and arc, respectively.

A solution, involving the reassignment of customers a and b to facility 1, is displayed in Fig. 1a. Although feasible, the resulting move (cyclic transfer) is not profitable. By forcing two customers to be relinquished from facility 3 and moving them to the same destination facility (facility 1), the new solution corresponding to this move would have a higher cost than the initial solution (the overall assignment cost increases by 2 units). Profitable moves could be obtained by moving either customer c to facility 4 (Fig.1b) or customer b to facility 1 (not shown). However, neither of these moves is feasible, as not enough demand is removed from facility 3 to restore its capacity. As a result, the search for a profitable multi-customer multi-exchange starting from the node associated with facility 1 fails and the algorithm either stops or restarts from a different root node.

A feasible and profitable cyclic transfer, however, does exist if customers allocated to facility 3 can be reassigned to *different* destination facilities. An example of a feasible and profitable transfer is displayed in Fig. 1c, with customers b and c being moved simultaneously to facilities 1 and 4, respectively. The cost of the solution induced by this transfer is 4 units less than the cost of the initial solution. Our proposed heuristic enhances previous multi-exchange heuristics by considering this type of *split* multi-customer multi-exchanges. As a result, not only are cost improving solutions detected more quickly and more often, but there is no need to design specialized facility moves for opening and closing facilities. By allowing customers to move towards different facilities, it is more likely that a facility may reassign all of its existing customers and be closed. Similarly, if a facility can receive customers from several other facilities, it is more likely that a currently closed facility may be opened.

3 Directed Hypergraphs

Hypergraphs are a generalization of standard graphs where edges can connect any number of nodes (Berge, 1993). Hypergraphs are often used to model complex interactions which go beyond single pair relationships. Example applications can be found in computer science (Torres and Araoz, 1988), transportation (Nguyen and Pallottino, 1988; Borndörfer and Heismann, 2015), telecommunication (Zorzi and Rao, 2003), social networks (Wasserman and Faust, 1999), and molecular biology (Rahman et al., 2013) among others.

In this section, we introduce some basic hypergraph definitions and concepts which lay the groundwork for

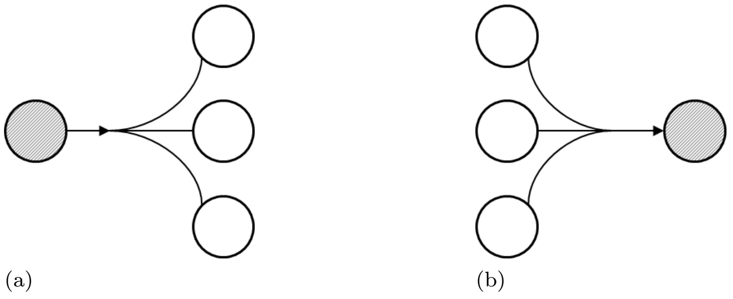


Figure 2: Forward and backward hyperarcs.

the presentation of our hypergraph multi-exchange heuristic. A more comprehensive treatment of hypergraph theory and applications can be found in Gallo et al. (1993) and Cambini et al. (1997).

A *directed hypergraph* is a pair $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of nodes, and $E = \{E_1, E_2, \ldots, E_m\}$ is the set of hyperarcs. A hyperarc E_i is a pair $\{T(E_i), H(E_i)\}$, where $T(E_i) \subseteq V$ is the *tail* of E_i and $H(E_i) \in V \setminus T(E_i)$ is its *head*. Clearly, when $|T(E_i)| = |H(E_i)| = 1$ for each $E_i \subseteq E$, the hypergraph corresponds to a standard directed graph.

A *directed path* P_{sd} from a source s to a destination d in G is a sequence of nodes and hyperarcs $P_{sd} = (v_1 = s, E_{i_1}, v_2, E_{i_2}, \ldots, E_{i_q}, v_{q+1} = d)$, where $s \in T(E_{i_1})$, $d \in H(E_{i_q})$, and $v_j \in H(E_{i_{j-1}}) \cap T(E_{i_j})$ for $j = 2, \ldots, q$. If $d \in T(E_{i_1})$, P_{sd} is a *directed cycle*. A path is *simple* if all hyperarcs are distinct. A path is said to be *cycle-free* if it does not contain any subpath which is a cycle. By extension, a hypergraph is cycle-free when no directed cycle exists.

Two types of hyperarcs will be considered in this paper: *forward* hyperarcs and *backward* hyperarcs. A forward hyperarc (or simply *F-arc*), denoted by E_i, is a hyperarc with $|T(E_i)| = 1$ (Fig. 2a), whereas a backward hyperarc (or simply *B-arc*), denoted by E'_i, is a hyperarc with $|H(E'_i)| = 1$ (Fig. 2b). A hypergraph whose hyperarcs are all F-arcs is an *F-graph*. A hypergraph whose hyperarcs are all B-arcs is a *B-graph*. A *mono F-graph* is a special F-graph where each node has at most one exiting hyperarc (Fig. 3a). Similarly, a *mono B-graph* is a special B-graph where each node has at most one entering hyperarc (Fig. 3b).

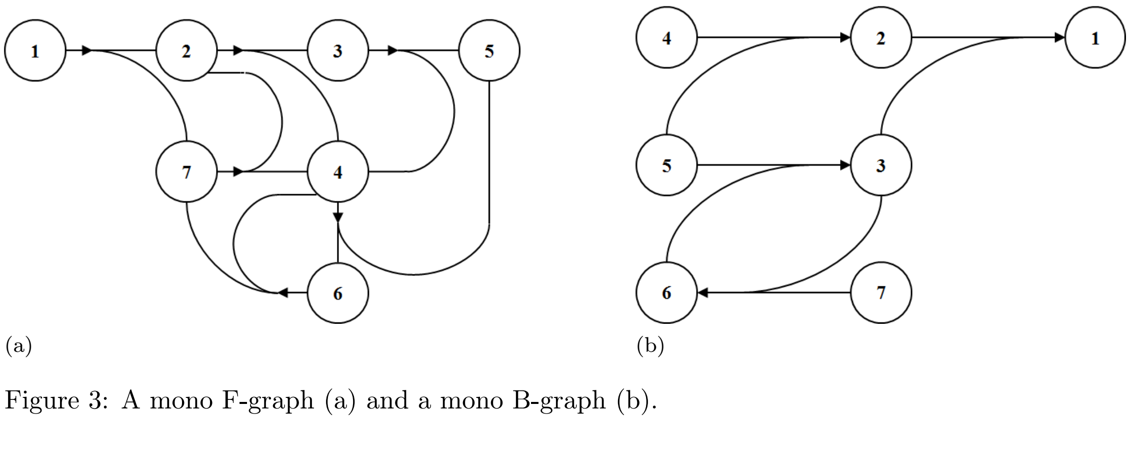


Figure 3: A mono F-graph (a) and a mono B-graph (b).

4 Hypergraph Multi-Exchange Heuristic

The proposed hypergraph multi-exchange heuristic is a multi-start local search procedure, which alternatively explores two very large-scale neighborhoods: the *forward split multi-customer multi-exchange neighborhood* and the *backward split multi-customer multi-exchange neighborhood.* For the sake of brevity, we will refer to these two neighborhoods hereafter as the *forward split* and the *backward split.*

Given their very large size, the split neighborhoods are explored only partially by using search heuristics that detect special cost-negative mono F-subgraphs or mono B-subgraphs within suitably defined improvement hypergraphs. In this section, we first define the VLSN structures. We then briefly describe the improvement hypergraphs and show how the problem of finding improving feasible solutions in the split neighborhoods can be translated into the problem of finding special hyper-structures in the improvement hypergraphs. Finally, we provide a detailed description of the search heuristics.

4.1 Split Multi-Customer Multi-Exchange Neighborhoods

Let $S = \{S_1, S_2, ..., S_n\}$ be a solution to problem (1)-(5), where each subset S_i, $i = 1, 2, \ldots, n$, corresponds to a potential facility site and contains the set of customers assigned to that facility in S. If no facility is established at i, $S_i = \emptyset$. The cost of each subset S_i, $i = 1, 2, ..., n$, is determined by:

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

Then, the cost of the solution S is:

$$C(S) = \sum_{i \in I} C(S_i)$$

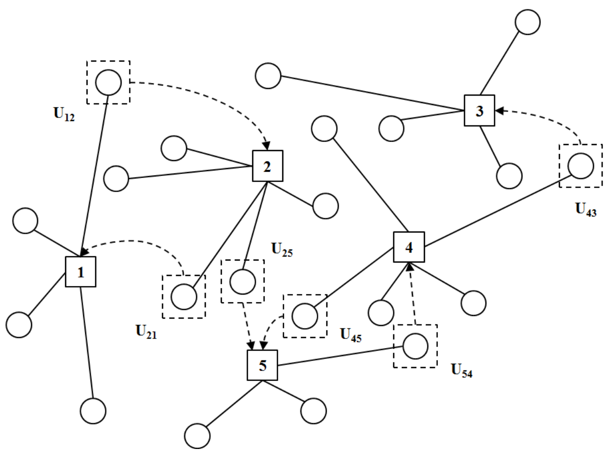
The solution S is feasible if the capacity constraints are satisfied for each facility i:

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

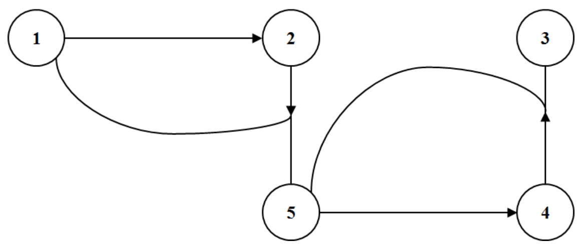
The residual capacity r_i of each subset S_i is defined as:

$$r_i = s_i - W(S_i) \geq 0$$

Given a solution S to SSCFLP, the split neighborhood of S is defined as the set of new solutions that can be obtained from S by exchanging clusters of customers among facilities in a sequential way. More precisely, the forward split neighborhood is induced by *forward split exchange*s, where customers can be moved from one facility towards multiple destination facilities. Conversely, the backward split neighborhood is induced by *backward split exchanges,* where customers assigned to multiple other facilities can be moved to a single destination facility. More formally, the split neighborhoods are defined in terms of forward and backward split exchanges as follows.



(a)



(b)

Figure 4: Forward split exchange (a) and corresponding F-graph (b).

Let $Q = \{i_1, i_2, \ldots, i_q\}$ denote a sequence of indexes of q subsets of the partition $S = \{S_1, S_2, ..., S_n\}$. Let $U_{i_r i_p} \subseteq S_{i_r}$, with $i_r, i_p \in Q$, be a subset of customers that can be moved from S_{i_r} to S_{i_p}. A forward split exchange is a sequence $U = \left\{U_{i_1}, U_{i_2}, ..., U_{i_q}\right\}$ of the q sets of customer subsets such that:

- $U_{i_r} = \left\{U_{i_r i_p} : i_p \in Q \setminus \{i_r\}; \cup U_{i_r i_p} \subseteq S_{i_r}; U_{i_r i_p} \cap U_{i_r i_s} = \emptyset, \forall i_p, i_s \in Q \setminus \{i_r\}\right\}$ for all $i_r \in Q$;
- For each $i_p \in \{i_{2,...,} i_q\}$, there must be a $U_{i_r i_p} \neq \emptyset$ for some i_r in $\{i_1, ..., i_{p-1}\}$.

In the following, we denote by $\bar{U} = \bigcup_{i_r \in Q} U_{i_r}$ the union of all the customer subsets in the sequence. An example of a forward split exchange is shown in Fig. 4a. In this example, $Q = \{1, 2, 5, 4, 3\}$, $U = \{U_1, U_2, U_5, U_4, U_3\} = \{\{U_{12}\} \{U_{21}, U_{25}\}, \{U_{54}\}, \{U_{43}, U_{45}\}, \emptyset\}$, and $\bar{U} = \{U_{12}, U_{21}, U_{25}, U_{54}, U_{43}, U_{45}\}$.

A *backward split exchange* is a sequence $U = \left\{U_{i_1}, U_{i_2}, ..., U_{i_q}\right\}$ of q customer subsets such that

- For all $i_r \in Q$, $U_{i_r} = \{U_{i_s i_r} : i_s \in Q \setminus \{i_r\}; U_{i_s i_r} \subseteq S_{i_s}\}$ and $U_{i_s i_r} \cap U_{i_s i_p} = \emptyset, \forall U_{i_s i_r}, U_{i_s i_p} \in \bar{U}$;
- For each $i_s \in \{i_{2,...,} i_q\}$, there must be a $U_{i_s i_r} \neq \emptyset$ for some i_r in $\{i_1, ..., i_{s-1}\}$.

An example of a backward split exchange is shown in Fig. 5a. In this example, $Q = \{5, 2, 3, 4\}$, $U = \{U_5, U_2, U_3, U_4\} = \{\{U_{25}, U_{45}\}, \{U_{32}, U_{42}\}, \{U_{43}\}, \emptyset\}$, and $\bar{U} = \{U_{25}, U_{45}, U_{32}, U_{42}, U_{43}\}$. Note that by definition $U_{45} \cap U_{42} = \emptyset$, meaning that the customer clusters leaving the same facility cannot have any element in common.

Any forward or backward split exchange U uniquely defines a new solution $S' = \{S'_1, S'_2, ..., S'_n\}$ in the neighborhood of S obtained by shifting customer subsets along the sequence. More precisely,

$$S'_{i_r} = \begin{cases} S_{i_r} & \text{if } i_r \notin Q \\ \left(S_{i_r} \bigcup_{U_{i_s i_r} \in \bar{U}} U_{i_s i_r}\right) \setminus \bigcup_{U_{i_r i_p} \in \bar{U}} U_{i_r i_p} & \text{if } i_r \in Q \end{cases} \tag{6}$$

A split exchange U is *feasible* if the capacity constraint of each partition set of the new solution S' is satisfied and *profitable* if the cost of the new solution S' is lower than the cost of the initial solution S (i.e.,

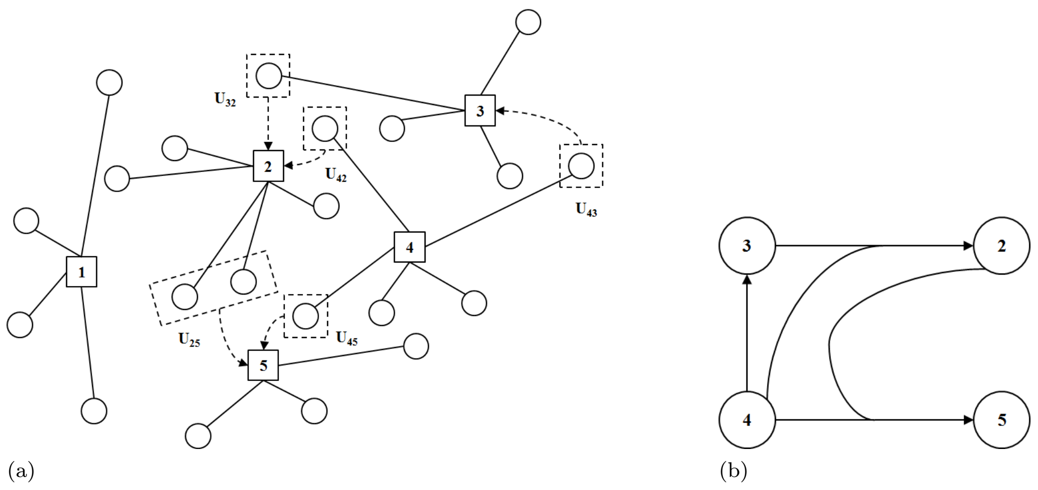


Figure 5: Backward split exchange (a) and corresponding B-graph (b).

$C(S') < C(S)$). The set of all the new solutions obtained from S through a feasible forward split exchange defines the *forward split neighborhood*. Similarly, the set of all the new solutions obtained from S through a feasible backward split exchange defines the *backward split neighborhood*.

Note that the multi-customer multi-exchange neighborhood described in Ahuja et al. (2004) is a special case of a forward split neighborhood obtained when clusters of customers can only move to a single facility. The multi-customer multi-exchange can be derived from the forward split exchange definition by imposing $|U_{i_r}| = 1$, $r = 1, \ldots, q$. In addition, if $\left|U_{i_r i_p}\right| = 1$ for each $r = 1, \ldots, q$, $p = 1, \ldots, q$, and $r \neq p$, the neighborhood induced by a forward split exchange reduces to a traditional single-customer multi-exchange neighborhood.

4.2 Improvement Hypergraphs

It should be clear that there is a geometric number of clusters that could be considered for movement along forward and backward split exchanges. Modeling all such exchanges in a hypergraph is impractical. We therefore construct the improvement hypergraphs dynamically, meaning that only promising hyperarcs (corresponding to the movement of clusters of customers) are generated during the search phase. In the following, we denote by $I(S)$ the set of open facilities in solution S and by $I^c(S)$ the set of closed facilities, where $I(S) \cup I^c(S) = I$.

The *forward improvement hypergraph* used to represent forward split exchanges is an F-graph $G(S) = (V(S), E(S))$, where the set of nodes $V(S)$ includes a node for each facility $i \in I$ and the set of hyperarcs $E(S)$ includes only F-arcs. An F-arc $E_a = \{\{i\}, H(E_a)\}$, with $i \in I(S)$, indicates that the facility established at location i relinquishes a subset of customers $U_{i\ell}$ toward each facility $\ell \in H(E_a)$. The process of generating promising hyperarcs in the forward hypergraph and selecting the subset of customers $U_{i\ell}$ involves solving a *Forward Selection Problem*, which is explained in the next section. The cost v_a of each F-arc E_a included in the set of hyperarcs is defined as:

$$v_a = \sum_{\ell \in H(E_a)} \sum_{j \in U_{i\ell}} (c_{\ell j} - c_{ij}) \tag{7}$$

If all customers assigned to facility i in S are reallocated (i.e., $\bigcup_{\ell \in H(E_a)} U_{i\ell} = S_i$), then the setup cost f_i is subtracted from the right hand side of equation (7). Conversely, if a facility $\ell \in H(E_a)$ is currently closed (i.e., $\ell \in I^c(S)$), then the setup cost f_ℓ is added to the right hand side of equation (7). Note that because the arcs are generated dynamically during the search phase, it is always possible to determine whether fixed costs need to be added to or subctrated from the cost of the newly generated F-arc. During runtime, a check is made to see if a move would result in facility i having no assigned customers (fixed cost subtracted) or if some facility ℓ that is currently closed needs to be opened due the reassignment of customers (fixed cost added).

The *backward improvement hypergraph* used to represent backward split exchanges is a B-graph $G(S) = (V(S), E(S))$, where the set of nodes $V(S)$ includes a node for each facility $i \in I$ and the set of hyperarcs $E(S)$ includes only B-arcs. A B-arc $E'_b = \{T(E'_b), \{i\}\}$ indicates that facility i receives a subset of customers U_{ki} from each facility $k \in T(E'_b)$, such that $k \in I(S)$. The process of generating promising hyperarcs in the backward hypergraph and selecting subsets of customers U_{ki} involves solving a *Backward Selection Problem*, which is explained in the next section. The cost v_b of each B-arc E'_b included in the hyperarc set is defined as:

$$v_b = \sum_{k \in T(E'_b)} \sum_{j \in U_{ki}} (c_{ij} - c_{kj}) \tag{8}$$

If no facility is established at location i in S ($i \in I^c(S)$), then the set up cost f_i is added to the right hand side of equation (8). If all the customers are removed from a facility $k \in T(E'_b)$ (i.e., $U_{ki} = S_k$), then the set up cost f_k is subtracted from the right hand side of equation (8). As for the costs of the F-arcs, these conditions are checked at running time when the B-arcs are generated.

The problem of finding feasible and profitable forward (backward) split exchanges for S can be translated into the problem of detecting negative mono F-graphs (mono B-graphs) in the forward (backward) improvement graph. The correspondence between mono F-graphs (mono B-graphs) and forward (backward) split exchanges is based on the fact that our neighborhood search algorithm (described in Sec. 4.4) not only generates F-graphs (B-graphs), where each node has at most one exiting (entering) hyperarc, but also produces a sequence of customer subsets that move among the facilities associated with the nodes of the mono F-graphs (mono B-graphs). In addition, the method used to identify the subsets of moving customers guarantees that their transfer only produces feasible exchanges. If the mono F-graph (mono B-graphs) is negative (i.e., the sum of the costs of its hyperarcs is negative), the corresponding split forward (backward) exchange is profitable. An example forward split exchange and corresponding mono F-graph are displayed in Fig. 4a and Fig. 4b, respectively. Fig. 5a and Fig. 5b show an example backward split exchange and corresponding mono B-graph.

4.3 Selection of Moving Customers

In this section, we present the customer selection problems whose solutions are used to generate the hyperarcs E_a and E'_b in the forward and backward improvement hypergraphs, respectively. Given the large number of possible hyperarcs and corresponding customer subsets that can be generated, our hyperarc generation policy makes the assumption that each forward (backward) hyperarc has at most L heads (tails). To further improve the computational time of the heuristic, we also assume that customers will only be served by facilities within

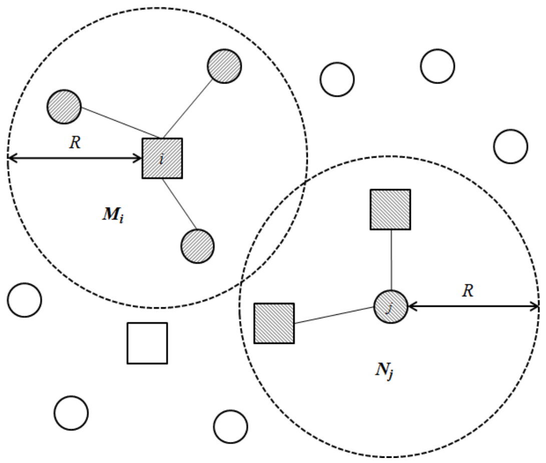


Figure 6: An illustration of the set of customers M_i and the set of facilities N_i within radius R.

a given distance or cost radius R. We denote by $M_i = \{j \in J : c_{ij} \leq R\}$ the set of customers that can be served by facility i and by $N_j = \{i : c_{ij} \leq R\}$ the set of facilities that can serve customer j (see Fig. 6). Note that the value of R is chosen in such a way that problem feasibility is guaranteed, as explained in Sec. 5. In addition, we denote by k_j the index of the facility serving customer j in the current solution S and by RC the set of customers that have been reassigned. This set is initially empty but is updated dynamically as customer moves are performed. RC is used to define $M'_i = \{j \in J \setminus RC : c_{ij} \leq R\}$ (i.e., the set of customers within distance R from i that have not been reassigned) and $S'_i = S_i \setminus RC$ (i.e., the subset of customers assigned to i in the current solution and not yet reassigned).

For the forward selection problem, we define $F_i = \cup_{j \in S_i} N_j$. Set F_i specifies all of the possible destination facilities for customers currently allocated to facility i. For the backward selection problem, we define $F'_i = \cup_{j \in M'_i} \{k_j\}$. Set F'_i specifies all of the possible source facilities for customers moving into i (i.e., the facilities currently serving customers within distance radius R of facility i that have not already been reassigned).

4.3.1 Forward Selection Problem

In the search for improving solutions contained in the forward improvement hypergraph, we first determine the hyperarcs (and customer subsets) candidate to enter a given node i. The transfer of customers into facility i is allowed to temporarily violate i's capacity constraint. The Forward Selection Problem (FSP) is then solved to identify the sets of customers leaving i for destination nodes $\ell \in F_i$ in such a way that the total cost of the reassignment is minimized and the capacity of facility i is restored if currently violated.

Assume that during the search for a mono F-graph, node i is reached and the hyperarc exiting i must be determined. Given that for each node $i \in N$ in the forward improvement graph we only generate one exiting arc, without loss of generality we denote by E_i the F-arc whose tail node is i. Let $BS(i) = \{E_k \in E : i \in H(E_k)\}$ be the backward star of node i (i.e., the set of previously generated F-arcs entering node i).

To state FSP mathematically, we define the following sets of decision variables.

$$\hat{x}_{\ell j} = \begin{cases} 1 & \text{if customer } j \in S_i \text{ is reassigned to facility } \ell \in N_j \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{y}_\ell = \begin{cases} 1 & \text{if facility } \ell \in F_i \text{ is a destination facility} \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{z} = \begin{cases} 1 & \text{if all customers allocated to facility } i \text{ are reassigned} \\ 0 & \text{otherwise} \end{cases}$$

FSP is given as follows.

$$\text{[FSP]} \;\; \min Z_F(E_i) = \sum_{j \in S_i} \sum_{\ell \in N_j} (c_{\ell j} - c_{ij})\, \hat{x}_{\ell j} - f_i \hat{z} + \sum_{\ell \in F_i \cap I^c(S)} f_\ell \hat{y}_\ell \tag{9}$$

s.t.

$$\sum_{\ell \in F_i} \hat{y}_\ell \leq L \tag{10}$$

$$\hat{x}_{\ell j} \leq \hat{y}_\ell \qquad \forall j \in S_i, \ell \in N_j \tag{11}$$

$$\sum_{\ell \in N_j} \hat{x}_{\ell j} \leq 1 \qquad \forall j \in S_i \tag{12}$$

$$\sum_{j \in S_i} w_j \hat{x}_{\ell j} \leq \alpha s_\ell + (1-\alpha) r_\ell \qquad \forall \ell \in F_i \tag{13}$$

$$\sum_{j \in S_i} \sum_{\ell \in N_j} w_j \hat{x}_{\ell j} \geq \sum_{E_k \in BS(i)} W(U_{ki}) - r_i \tag{14}$$

$$|S_i|\, \hat{z} \leq \sum_{j \in S_i} \sum_{\ell \in N_j} \hat{x}_{\ell j} \tag{15}$$

$$\hat{x}_{\ell j} \in \{0,1\} \qquad \forall j \in S_i, \ell \in N_j \tag{16}$$

$$\hat{y}_\ell \in \{0,1\} \qquad \forall \ell \in F_i \tag{17}$$

$$\hat{z} \in \{0,1\} \tag{18}$$

In FSP, the objective function (9) minimizes the total cost of reassignment. This includes the potential cost savings from closing facility i if all of its customers are reassigned and the set up cost of opening any destination facility ℓ which is currently closed. Constraints (10) require the number of destination facilities to be less than or equal to the specified limit L. Inequalities (11) ensure that customers are only transferred to facilities which are selected as destinations. Constraints (12) prevent customers from being reassigned to more than one destination facility. Although we allow the capacity of the destination facilities to be violated, we impose a limit on the extent of the violation. This is accomplished by constraints (13) and is parameterized by $\alpha \in [0, 1]$, whose value can vary during the search for profitable mono F-graphs. The maximum capacity violation is obtained when $\alpha = 1$, in which case the total capacity of the destination facility ℓ is used. If $\alpha = 0$, a capacity violation is not allowed and priority is given to identifying feasible solutions quickly. The value of α can be increased in an attempt to identify profitable moves but at the expense of satisfying capacity restrictions. The search strategy based on adjusting α plays an important role in the forward procedure, as detailed in the next section. Inequality (14) guarantees that the selection of moving customers restores the capacity constraint of facility i if currently violated (due to the transfer of customer subsets along the F-arcs in the backward star of node i). Constraint (15) forces the value of the variable $\hat{z}$ to zero unless all of facility i's customers have been relinquished. Given the nature of the objective function, values of $\hat{z} = 1$ are preferred, meaning that the fixed cost of opening facility i can be saved. Finally, constraints (16)-(18) impose binary restrictions on the decision variables.

Note that if node i is the root node of the search, constraint (14) is always satisfied. We refer to problem FSP without constraint (14) as FSP-Root. Only nodes $i \in I(S)$ (i.e., open facilities) are considered as root nodes for the search.

The solution to each FSP is used to construct an F-arc in the forward improvement hypergraph. More specifically, the F-arc exiting node i is defined as $E_i = \{\{i\}; H(E_i)\}$ with $H(E_i) = \{\ell \in F_i : \hat{y}_\ell = 1\}$. A solution to FSP also uniquely determines the subset $U_{i\ell}$ for each $\ell \in H(E_i)$ by setting $U_{i\ell} = \{j \in S_i | \hat{x}_{\ell j} = 1\}$.

4.3.2 Backward Selection Problem

In the search for improving solutions contained in the backward improvement hypergraph, we first determine the hyperarcs (and customer subsets) exiting a given node i. The Backward Selection Problem (BSP) is then solved to identify the sets of customers entering i from multiple source nodes $k \in F'_i$ in such a way that the total cost of the reassignment is minimized and the capacity of facility i is not exceeded.

Assume that during the search for a mono B-graph, node i is reached and the B-arc entering i must be determined. Given that for each node $i \in N$ in the backward improvement graph we only generate one entering arc, without loss of generality we denote by E'_i the B-arc whose head node is i. Let $FS(i) = \{E'_\ell \in E : i \in T(E'_\ell)\}$ be the forward star of node i (i.e., the set of already generated B-arcs exiting node i).

To state BSP mathematically, we define the following sets of decision variables.

$$\check{x}_j = \begin{cases} 1 & \text{if customer } j \in M'_i \text{ is reassigned from its current facility to facility } i \\ 0 & \text{otherwise} \end{cases}$$

$$\check{y}_k = \begin{cases} 1 & \text{if facility } k \text{ is a source facility} \\ 0 & \text{otherwise} \end{cases}$$

$$\check{z}_k = \begin{cases} 1 & \text{if all customers of facility } k \text{ are reassigned to facility } i \\ 0 & \text{otherwise} \end{cases}$$

BSP is then given as follows.

$$[\text{BSP}] \ \min Z_B(E'_i) = \sum_{j \in M'_i} \left(c_{ij} - c_{k_j j}\right) \check{x}_j - \sum_{k \in F'_i} f_k \check{z}_k \tag{19}$$

s.t.

$$\sum_{k \in F'_i} \check{y}_k \le L \tag{20}$$

$$\check{x}_j \le \check{y}_{k_j} \qquad j \in M'_i \tag{21}$$

$$\sum_{j \in M'_i} w_j \check{x}_j \le r_i + \sum_{E'_\ell \in FS(i)} W(U_{i\ell}) \tag{22}$$

$$|S'_k|\,\check{z}_k \leq \sum_{j \in S'_k} \check{x}_j \qquad \forall k \in F'_i \tag{23}$$

$$\check{x}_j \in \{0,1\} \qquad \forall j \in M'_i \tag{24}$$

$$\check{y}_k \in \{0,1\} \qquad \forall k \in F'_i \tag{25}$$

$$\check{z}_k \in \{0,1\} \qquad \forall k \in F'_i \tag{26}$$

In BSP, the objective function (19) minimizes the total cost of reassignment, including the potential cost savings from closing one or more facilities due to reassigning of all customers. Constraints (20) ensure that the number of source facilities do not exceed L. Constraints (21) state that customers can only be transferred to facility i from facilities that are selected as sources. Constraint (22) guarantees that the capacity of facility i is not exceeded after new customers are reassigned to it. The right-hand side of this constraint represents the updated residual capacity of facility i after the removal of customers moving along the B-arcs in the forward star of node i. Constraints (23) state that a variable $\check{z}_k$ can take value one only if all of its customers are reallocated to facility i. Constraints (24)-(26) impose binary restrictions on the decision variables.

One of the aims of performing backward split exchanges is to facilitate the opening of new facilities. Therefore, we always start the search using a closed facility as the root node. When solving BSP with root node $i \in I^c(S)$, the right-hand side of constraint (22) is simply s_i. In addition, the fixed cost f_i of opening facility i must be added to the objective function. We refer to BSP solved at the root node as BSP-Root.

The solutions to each BSP is used to define a B-arc in the backward improvement hypergraph. Specifically, the B-arc entering node i is defined as $E'_i = \{T(E'_i), \{i\}\}$ with $T(E'_i) = \{k \in F'_i : \check{y}_k = 1\}$. A solution to BSP also uniquely determines the subset U_{ki} for each $k \in T(E'_i)$ by setting $U_{ki} = \{j \in M'_i \cap S'_k | \check{x}_j = 1\}$.

Both FSP and BSP are solved using a simple greedy heuristic that involves sorting the customers in non-decreasing order of reassignment costs. The greedy selection of customers guarantees that all problems constraints are satisfied. The possibility of closing and opening facilities is also taken into account. Pseudocode for the greedy heuristics is provided in Appendix A.

4.4 Neighborhood Search Algorithm

In this section, we first outline the procedures for detecting mono F-graphs in the forward improvement hypergraph and mono B-graphs in the backward improvement hypergraph. We refer to these as the *forward* and *backward procedures*, respectively. We then present the overall structure of our Hypergraph Multi-Exchange Heuristic (HMEH), which consists of a multi-start algorithm for exploring different neighborhoods.

4.4.1 Forward Procedure

The algorithm used to dynamically search for mono F-graphs relies on the following labels associated with each node i in the forward improvement hypergraph.

- $out(i)$: the F-arc $E_i = \{\{i\}, H(E_i)\}$ exiting node i
- $MC(i)$: the customer subsets moving from facility i toward each node $\ell \in H(E_i)$

- $BStar(i)$: the backward star of node i
- $rc(i)$: the residual capacity at node i

The labels at node i are initialized as follows: $out(i) = E_i = \{\{i\}, \emptyset\}$, $MC(i) = \emptyset$, $BStar(i) = \emptyset$, and $rc(i) = r_i$. Solution of FSP is used to generate an F-arc E_i exiting node i. When the hyperarc is generated, the labels associated with node i are updated as follows: $out(i) = E_i = \{\{i\}, H(E_i)\}$, $MC(i) = \{U_{i\ell} : \ell \in H(E_i)\}$, and $rc(i) = rc(i) + \sum_{\ell \in H(E_i)} W(U_{i\ell})$. The following labels associated with each node $\ell \in H(E_i)$ are also updated: $BStar(\ell) = BSstar(\ell) \cup \{E_i\}$ and $rc(\ell) = rc(\ell) - W(U_{i\ell})$.

Besides the node labels, the forward procedure makes use of the following additional notation: F^0 is the set of candidate root nodes, Q_N is the set of nodes still to be examined, P is the set of nodes already processed, and C_F is the cost of the mono F-graph.

The forward procedure solves a sequence of FSP problems to build a feasible, cost-negative mono F-graph. It starts by selecting an open facility $i \in I(S)$ (the root node) and tries to reassign subsets of customers currently allocated to i to other open or closed facilities (the destination nodes) by solving FSP-Root with $\alpha = 0$. If a cost-negative F-arc is found, an improving move has been detected, the solution S is updated, and the procedure terminates. Otherwise, problem FSP-Root is resolved with an increased value of α to allow some capacity violation at the destination nodes. Parameter α is increased repeatedly until either it exceeds a maximum value of 1, in which case the procedure is restarted with a new root, or a cost-negative F-arc E_i is generated. In the latter case, all nodes $\ell \in H(E_i)$ whose capacity have been violated are inserted into the queue of nodes Q_N to be examined. When a node ℓ is extracted from the queue, problem FSP with $\alpha = 0$ is first solved to generate an F-arc E_ℓ, which restores the capacity at ℓ without generating additional capacity violations at the head nodes. If the cost of the overall mono F-graph generated so far is negative, another node is extracted from the queue. Otherwise, FSP is solved for node ℓ with an increased value of α. The procedure iterates in this fashion until capacity has been restored at each node (i.e., when Q_N becomes empty). If the resulting mono F-graph has negative cost, an improving move has been detected, the solution S is updated, and the procedure terminates. Otherwise, the search is restarted from a different root. If all of the candidate root nodes have been considered, the procedure terminates without detecting an improving solution.

Note that in the forward procedure, we allow the search to return to a node ℓ that has already been processed (i.e., $\ell \in P$). An already examined node ℓ is included in the set of possible destinations when solving FSP for a particular node i. However, for each node $\ell \in P$, the right hand side of constraints (13) is replaced by the residual capacity $rc(\ell)$. This guarantees that the capacity at node ℓ is not exceeded and prevents ℓ from being inserted into the queue again. Pseudo-code for the forward procedure is given in Algorithm 1.

Note that the forward procedure only generates at most one F-arc exiting from a node (revisited nodes are not inserted into Q_N again). Consequently, the procedure is guaranteed to generate a mono F-graph. At termination, if a cost-improving F-graph is detected, the set of processed nodes P and the labels $MC(i)$ for each node $i \in P$ uniquely identify a split forward exchange. Solution S can then be updated using formula (6) where $Q = P$ and $\bar{U} = \cup_{i \in P} MC(i)$. Based on some preliminary results, the forward procedure uses a breadth-first search strategy when extracting nodes from Q_N.

Algorithm 1 Pseudo-code for the forward procedure.

```
Step 0:  Input an initial solution $S$ and set $F^0 = I(S)$
Step 1:  If $F^0 = \emptyset$, then STOP
         Extract at random $i \in F^0$, initialize nodes labels, and set $P = \emptyset$, $Q_N = \emptyset$, and $C_F = 0$
         Solve FSP-Root at root node $i$ with $\alpha = 0$ to generate $E_i$
         If $Z_F(E_i) < 0$, then update label $MC(i)$, set $P = \{i\}$, update solution $S$ and STOP
Step 2:  Set $\alpha = \alpha + \varepsilon$, such that $\varepsilon \in (0, 1]$
         If $\alpha > 1$, then go to Step 1
         Else solve FSP-Root at root node $i$ with increased $\alpha$ to generate $E_i$
Step 3:  If $Z_F(E_i) \geq 0$, then go to Step 2
         Else
          (i) Update labels $out(i)$, $MC(i)$, and $rc(i)$ at node $i$ and $BStar(\ell)$ and $rc(\ell)$ at each node $\ell \in H(E_i)$
          (ii) Update the cost of the mono F-graph: $C_F = C_F + Z_F(E_i)$
          (iii) Update $P = P \cup \{i\}$ and $Q_N = Q_N \cup \{\ell : \ell \in H(E_i), W(U_{i\ell}) > r_\ell\}$
Step 4:  If $Q_N \neq \emptyset$, then go to Step 6
Step 5:  If $C_F < 0$, then update solution $S$ and STOP, else go to Step 1
Step 6:  Extract $\ell$ from $Q_N$ and solve FSP at node $\ell$ with $\alpha = 0$ to generate $E_\ell$
         If $C_F + Z_F(E_\ell) < 0$ then
          (i) Update labels $out(\ell)$, $MC(\ell)$, and $rc(\ell)$ at node $\ell$ and $BStar(k)$ and $rc(k)$ at each node $k \in H(E_\ell)$
          (ii) Update $C_F = C_F + Z_F(E_\ell)$ and $P = P \cup \{\ell\}$
          (iii) Go to Step 4
Step 7:  Set $\alpha = \alpha + \varepsilon$, such that $\varepsilon \in (0, 1]$
         If $\alpha > 1$, then go to Step 1, else solve FSP at node $\ell$ with increased $\alpha$ to generate $E_\ell$
         If $C_F + Z_F(E_\ell) < 0$ then
          (i) Update labels $out(\ell)$, $MC(\ell)$, and $rc(\ell)$ at node $\ell$ and $BStar(k)$ and $rc(k)$ at nodes $k \in H(E_\ell)$
          (ii) Update $C_F = C_F + Z_F(E_\ell)$, $P = P \cup \{\ell\}$, and $Q_N = Q_N \cup \{k : k \in H(E_\ell), W(U_{\ell k}) > r_k\}$
          (iii) Go to Step 4
```

4.4.2 Backward Procedure

The algorithm used to dynamically search for mono B-graphs relies on the following labels associated with each node i in the backward improvement hypergraph.

- $in(i)$: the B-arc $E'_i = \{T(E'_i); \{i\}\}$ entering node i
- $MC(i)$: the customer subsets moving into facility i from each facility $k \in T(E'_i)$
- $FStar(i)$: the forward star of node i
- $close(i)$: true if facility i is closed during the search, false otherwise
- $RC(i)$: the subset of customer reassigned from facility i to some other facility.

The labels at node i are initialized as follows: $in(i) = E'_i = \{\emptyset, \{i\}\}$, $MC(i) = \emptyset$, $FStar(i) = \emptyset$, $close(i) = false$, and $RC(i) = \emptyset$. Solution of BSP is used to generate a B-arc E'_i entering node i. When the hyperarc is generated, the labels associated with node i are updated as follows: $in(i) = E'_i = \{T(E'_i), \{i\}\}$ and $MC(i) = \{U_{ki} : k \in T(E'_i)\}$. The following labels associated with each node $k \in T(E'_i)$ are also updated: $FStar(k) = FStar(k) \cup \{E'_i\}$ and $RC(k) = RC(k) \cup \{U_{ki}\}$.

In the backward procedure, the sets F^0, Q_N and P are defined as in the forward procedure. C_B denotes the cost of the mono B-graph.

Algorithm 2 Pseudo-code for the backward procedure.

```
Step 0: Input an initial solution $S$ and set $F^0 = I^c(S)$
Step 1: If $F^0 = \emptyset$, then STOP
        Extract at random $i \in F^0$, initialize nodes labels, and set $RC = \emptyset$, $P = \emptyset$, $Q_N = \emptyset$, and $C_B = 0$
        Solve BSP-Root at root $i$ to obtain $E'_i$
Step 2: If $Z_B(E'_i) = f_i$, then go to Step 1
        Else
         (i) Update labels $in(i)$ and $MC(i)$ at node $i$
         (ii) Update labels $FStar(k)$ and $RC(k)$ at each node $k \in T(E'_i)$. If $RC(k) = S_k$ then $close(k) = true$.
         (iii) Update the cost of the mono B-graphs: $C_B = C_B + Z_B(E'_i)$
         (iv) Update $RC = RC \cup MC(i)$, $P = P \cup \{i\}$ and $Q_N = Q_N \cup \{k : k \in T(E'_i)\}$
Step 3: If $Q_N \neq \emptyset$, go to Step 5
Step 4: If $C_B < 0$, then update solution $S$ and STOP, else go to Step 1
Step 5: Extract $k \in Q_N$ and solve BSP-Root if $close(k) = true$, else solve BSP to obtain $E'_k$ at node $k$
        If $Z_B(E'_k) = f_k$ given $close(k) = true$ or $Z_B(E'_k) = 0$ given $close(k) = false$, then go to Step 3
        Else
         (i) If $close(k) = true$, set $close(k) = false$
         (ii) Update labels $in(k)$ and $MC(k)$ at node $k$
         (iii) Update labels $FStar(\ell)$ and $RC(\ell)$ at each node $\ell \in T(E'_k)$. If $RC(\ell) = S_\ell$ then $close(\ell) = true$.
         (iv) Update the cost of the mono B-graph: $C_B = C_B + Z_B(E'_k)$
         (v) Update $RC = RC \cup MC(k)$, $P = P \cup \{k\}$ and $Q_N = Q_N \cup \{\ell : \ell \in T(E'_k) \setminus P\}$
         (vi) Go to Step 3
```

The backward procedure solves a sequence of BSP problems to build a feasible, cost-negative mono B-graph. It starts by selecting a closed facility $i \in I^c(S)$ (the root node) and tries to reassign to i subsets of customers currently allocated to other open facilities (the source facilities) by solving BSP. If no profitable reassignment is detected (i.e., $Z_B(E'_i) = f_i$), the procedure is restarted with a different root. If a cost-negative reassignment is found, even if it does not offset the cost of opening facility i, the B-arc E'_i is generated and each node $k \in T(E'_i)$ is inserted into the queue of nodes Q_N to be examined. Note that if all the customers served by facility k are reassigned (i.e., $RC(k) = S_k$), facility k is closed. Every time a node k is extracted from Q_N, the procedure checks if k has been previously closed or not ($close(k) = true$ or $false$). BSP-Root or BSP is solved accordingly in an attempt to find a transfer of customer to facility k that makes the total cost of the mono B-graph negative. If a cost-negative reassignment cannot be found, node k is simply disregarded. Otherwise, the B-arc E'_k is generated and the nodes in $T(E'_k)$ are added to the queue if they have not already been examined.

The procedure terminates when the queue Q_N is empty. If the resulting mono B-graph has negative cost, an improving solution has been detected, the solution S is updated, and the procedure terminates. Otherwise, the search is restarted from a different root. If all of the candidate root nodes have been considered, the procedure terminates without detecting an improving solution. Throughout the procedure, set RC is updated to track the customers already involved in a transfer so that they are not selected again when solving BSP or BSP-Root for subsequent nodes. Note that unlike the forward procedure, capacity restrictions are never violated in the backward procedure. Pseudo-code for the backward procedure is given in Algorithm 2.

The Backward Procedure generates at most one B-arc entering each node (nodes already processed are not inserted in Q_N again). Consequently, the procedure is guaranteed to generate a mono B-graph. At termination, if a cost-improving mono B-graph is detected, the set of processed nodes P and the labels $MC(i)$ of each node $i \in P$ uniquely identify a split backward exchange. Solution S can be updated using

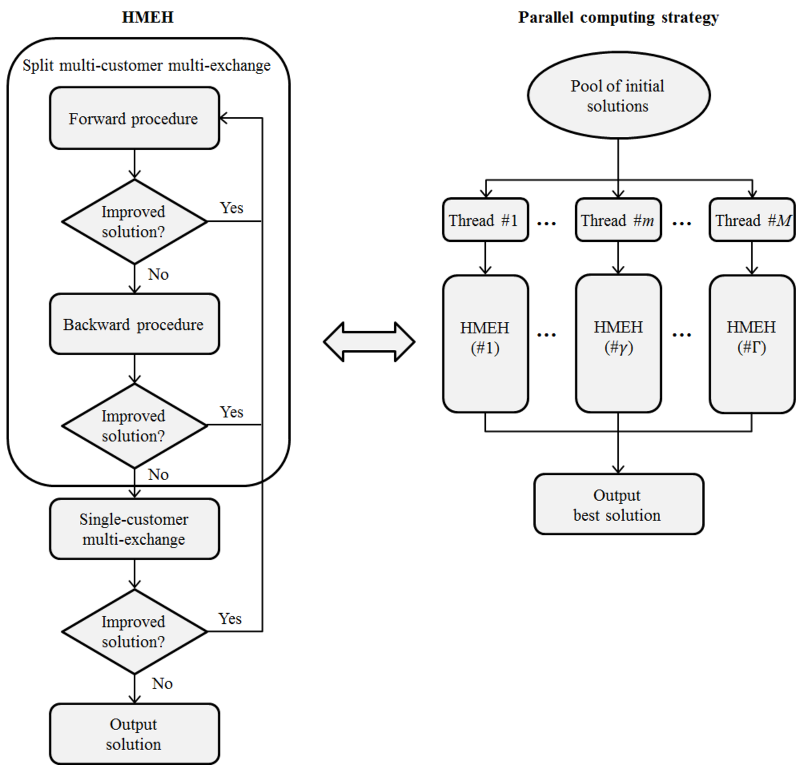


Figure 7: Flowchart of the HMEH heuristic.

formula (6) where $Q = P$ and $\bar{U} = \cup_{i \in P} MC(i)$. As in the forward procedure, a breadth-first search strategy is used for extracting nodes from Q_N .

4.4.3 Hypergraph Multi-Exchange Heuristic (HMEH)

The hypergraph multi-exchange heuristic (HMEH) starts with a feasible solution S and explores the forward and backward split neighborhoods sequentially until no improving moves can be found. When no more profitable split exchanges can be detected, a traditional single-customer multi-exchange (Ahuja et al., 2004) is attempted to further improve the solution. Every time an improving solution is found, the algorithm goes back to the exploration of the forward neighborhood. The HMEH is started multiple times with different initial feasible solutions. These are generated using a Lagrangean relaxation procedure analogous to the one described in Holmberg et al. (1999) and implemented in Ahuja et al. (2004). Finally, to speed up the algorithm, a parallel computing strategy is utilized. Fig. 7 shows a flowchart of HMEH with single or multiple initial solutions.

5 Experimental Results

In this section, we investigate the computational efficiency of our HMEH algorithm for solving SSCFLP. The evaluation of the algorithm's performance is tested using three well-known benchmark datasets of various sizes (10-1,000 facilities and 50-1,000 customers). We first compare our algorithm with the original multi-exchange heuristic (MEH) algorithm on the same instances used in Ahuja et al. (2004). Note that all such

Table 1: A summary of parameter values for HMEH and MEH.

Dataset	L	#InitSoln
Holmberg		
p1-24, p41-55	2	30
p25-40, p56-71	4	30
Beasley	10	10
Avella and Boccia		
i300	50	10
i3001500	50	10
i500	100	10
i700	100	10
i1000	200	10

instances can now be solved optimally using the general purpose mixed integer linear programming solver CPLEX. We further test HMEH on some very large instances and compare with the standard version of Kernel Search (Guastaroba and Speranza, 2014).

- Holmberg: This dataset comprises five sets of small-to-medium sized instances (71 instances in total) randomly generated by Holmberg et al. (1999). The number of facilities ranges from 10 to 30, while the number of customers ranges from 50 to 200.
- Beasley: A total of 12 large-sized instances with 100 facilities and 1,000 customers were taken from the OR-Library (Beasley, 1990).
- Avella and Boccia: The dataset consists of a total 100 instances with 300 to 1,000 facilities and 300 to 1,000 customers (Avella and Boccia, 2009). Only some of the instances ($\leq$ 700 facilities/customers) can be solved to optimality by CPLEX. For the remainder, the best known value is reported in Guastaroba and Speranza (2014).

The HMEH algorithm was implemented in C++ and run on a desktop PC with an Intel Core i7-3770 processor (8 cores, 3.40 GHz per chip) and 24 GB of RAM. Optimal solutions for all small-to-medium and some large sized instances were obtained with CPLEX. Reported computation times are given in CPU seconds.

The parallel computing strategy was not used to solve the problems in the Holmberg dataset, which are quite small. For the instances in the two larger datasets (Beasley dataset and Avella and Boccia dataset), parallel computing was implemented with 5 cores ($\Gamma = 5$). On average, the use of parallel computing reduced the solution time of the Beasley instances by a factor of around 1.5 and the Avella and Boccia instances by a factor of around 2.

A summary of parameter values used for HMEH on the different datasets is provided in Table 1. Columns L give values of the parameters which limit the number of hyperarc heads/tails when selecting subsets of moving customers. Column #InitSoln gives the number of initial solutions that HMEH (and MEH) were run on. The distance radius R used to define sets M_i and N_j was set to $R = 1.5 \times d_{\max}$, where $d_{\max}$ is the maximum distance between a customer and facility in the initial feasible solution to the problem. The parameter ϵ used in the forward procedure was set to 1.

To evaluate the performance of the algorithms, we calculated the percentage deviation (denoted $\triangle$) between the objective value found by a given algorithm (denoted Obj) and the optimal value (denoted Opt) or the

Table 2: Comparison of HMEH and MEH on the Holmberg instances.

				HMEH			MEH		
Prob Set	$\lvert I\rvert$	$\lvert J\rvert$	#Inst	Δ_{Avg}	#Opt	Time	Δ_{Avg}	#Opt	Time
p1-p24	10-20	50	24	0.0	24	0.42	0.00	24	0.49
p25-p40	30	150	16	0.0	16	4.08	0.07	10	9.38
p41-p55	10-30	70-100	15	0.0	15	1.08	0.01	10	1.37
p56-p71	30	200	16	0.0	16	15.53	0.03	9	13.60
Avg				0.0		5.28	0.03		6.21

lower bound (denoted LB) found by CPLEX.

$$\Delta = \begin{cases} \dfrac{(\text{Obj} - \text{Opt})}{\text{Opt}} \times 100 & \text{if a known optimal solution exists} \\ \dfrac{(\text{Obj} - \text{LB})}{\text{LB}} \times 100 & \text{otherwise} \end{cases} \tag{27}$$

The value Δ_{Avg} is defined as the average percentage deviation for all instances of a particular dataset.

In Tables 2 and 3, we report results for HMEH and MEH on the Homberg and Beasley instances, respectively. The results clearly demonstrate the superiority of HMEH over MEH. Specifically, for the Holmberg dataset, HMEH finds an optimal solution (#Opt) in every case. MEH, in contrast, finds an optimal solution in just 53 of the 71 instances. Solution times are comparable.

For the Beasley instances, HMEH again outperforms MEH. Whereas MEH has an average optimality gap of 0.07% and an average run time of 64 seconds, HMEH has an average gap of 0.04% and an average run time of 43 seconds. What is more, HMEH finds more optimal solutions than MEH.

We carried out additional experiments comparing HMEH against standard Kernel Search on the much larger Avella and Boccia instances. Kernel Search was implemented by Guastaroba and Speranza (2014) on a computer with similar specifications as ours. Results are reported in Table 4. For each of the two heuristics (HMEH and Kernel Search), the table displays the average percentage deviation from the best known solution, the number of optimal solutions found, and the average time. Overall, the performance of HMEH and Kernel Search are comparable. Average optimality gap for HMEH is slightly lower: 0.60% versus 0.64%. The

Table 3: Comparison of HMEH and MEH on the Beasley instances.

		HMEH		MEH	
Inst	Opt	$\triangle$	Time	$\triangle$	Time
capa1	19,241,056.93	0.0052	58.41	0.0074	137.13
capa2	18,438,329.78	0.0001	45.18	0.0080	127.32
capa3	17,765,201.95	0.0000	49.86	0.0000	89.32
capa4	17,160,612.23	0.0000	13.88	0.0011	14.61
capb1	13,657,464.23	0.0027	63.76	0.0027	82.20
capb2	13,362,529.34	0.0041	58.74	0.0188	89.45
capb3	13,199,213.19	0.2962	64.49	0.2980	63.99
capb4	13,083,203.74	0.0000	19.13	0.0143	18.34
capc1	11,647,410.50	0.0333	46.81	0.3241	60.54
capc2	11,570,437.68	0.0000	36.41	0.0000	33.42
capc3	11,519,169.78	0.1783	42.05	0.1808	37.82
capc4	11,505,861.86	0.0000	13.36	0.0307	9.48
Avg		0.0433	42.67	0.0738	63.63

Table 4: Comparison of HMEH and standard Kernel Search on the Avella and Boccia instances.

				HMEH				Kernel Search		
Prob Set	$\|I\|$	$\|J\|$	#Inst	Δ_{Avg}	#Opt	#Better	Time	Δ_{Avg}	#Opt	Time
i300	300	300	20	0.54	10	4	428.03	0.56	12	2,206.70
i3001500	300	1,500	20	0.01	14	0	1,159.33	0.00	20	334.71
i500	500	500	20	0.52	4	8	2,982.72	0.66	6	4,190.27
i700	700	700	20	0.82	0	9	4,992.14	0.90	2	5,244.69
i1000	1,000	1,000	20	1.10	0	11	8,582.74	1.07	0	6,533.15
Avg				0.60			3,628.99	0.64		3,701.90

number of verified optimal solution found by HMEH and Kernel Search are 28 and 40, respectively. In terms of solution times, the two algorithms are similar, about 1 hour on average.

One interesting finding is that HMEH finds better solutions than Kernel Search in 32 cases. These specific instances are reported in Table 5. On average, HMEH improves by 0.70 percentage points for these 32 instances. More importantly, in 27 instances, HMEH even improves on the best known value (BKV) reported in Guastaroba and Speranza (2014). The new best values found by our heuristic are highlighted in bold. Overall, what the results in Tables 4 and 5 suggest is that HMEH is highly effective at solving SSCFLP. It is competitive with Kernel Search, currently the best performing algorithm for SSCFLP, and is capable of finding better solutions or even the best known solution to some very large benchmark instances.

Finally, it is worth mentioning that without using all the procedures of the HMEH heuristic (i.e., the forward procedure, backward procedure, and single-customer multi-exchange), the best known solutions to these instances could not be found. To provide evidence of this, we show in Table 6 the minimum, maximum, and average improvement over the initial solutions due to the three procedures for the 32 problem instances shown in Table 5. As can be seen, each procedure of the heuristic provides a distinct improvement over the initial solution. On average, HMEH improved the initial solutions for this data set by 3.52%. When only the forward procedure (FP) was used, the average improvement was 1.18%. This increased to 2.66% when the backward procedure (BP) was also used. Table 6 also shows that all of the initial solutions found with the Lagrangean heuristic were improved by HMEH. The smallest improvement was obtained in one of the restarts for instance i700_17 (0.57%). The average improvement across the 10 restarts for this instance was 0.93%. The maximum improvement obtained in a single restart was 7.29% for instance i500_2.

6 Conclusions and Future Work

In this paper, we developed an efficient very-large neighborhood search heuristic for solving SSCFLP and other types of partitioning problems. The proposed neighborhood structures are induced by novel backward and forward split exchanges. Profitable split multi-exchanges are detected by searching for special structures within dynamically-built improvement hypergraphs. The use of split transfers represents a notable improvement over previous VLNS approaches based on cyclic transfers and multi-exchange moves. It also overcomes some limitations of existing VLNS heuristics for solving facility location problems. In particular, there is no need to devise special facility add/drop moves within the search, as these are automatically built into the forward and backward split exchanges. As the results clearly demonstrate, our heuristic can obtain better solution quality in less computation time when compared with the traditional multi-exchange heuristic of

Table 5: Improved solutions found by HMEH on the Avella and Boccia instances.

			HMEH			Kernel Search		
Inst	LB	BKV	Obj	△	Time	Obj	△	Time
i300_1	16,350.66	16,629.65	**16,598.65**	1.52	830.28	16,630.26	1.71	5,512.00
i300_4	17,989.97	18,389.35	**18,387.06**	2.21	761.02	18,389.35	2.22	4,602.00
i300_7	11,392.52	11,470.31	**11,422.07**	0.26	412.43	11,473.41	0.71	2,799.00
i300_10	11,232.77	11,324.34	**11,323.33**	0.81	355.76	11,324.88	0.82	3,777.00
i500_2	28,130.74	28,651.48	**28,412.62**	1.00	5,715.20	28,651.16	1.85	4,702.95
i500_4	28,159.03	28,732.60	**28,518.40**	1.28	4,769.20	28,733.47	2.04	6,002.19
i500_5	24,702.77	25,080.66	**24,928.78**	0.91	4,019.56	25,130.13	1.73	5,803.31
i500_6	15,756.82	15,853.35	**15,807.15**	0.32	3,170.29	15,857.66	0.64	6,319.43
i500_7	16,109.28	16,205.15	**16,128.97**	0.12	3,279.96	16,205.94	0.60	6,321.43
i500_9	16,327.71	16,399.40	**16,378.42**	0.31	3,979.32	16,399.55	0.44	5,050.18
i500_10	15,815.13	15,931.14	**15,877.48**	0.39	3,007.45	15,930.58	0.73	6,474.08
i500_13	13,666.25	13,715.45	**13,699.32**	0.24	2,041.22	13,715.45	0.36	4,866.34
i700_1	36,905.93	37,652.76	**37,616.61**	1.93	6,145.34	37,751.08	2.29	6,102.50
i700_5	37,802.10	38,458.45	**38,138.79**	0.89	7,009.14	38,459.86	1.74	5,530.16
i700_6	19,910.67	20,100.24	**19,965.07**	0.27	5,569.45	20,099.82	0.95	6,828.50
i700_7	21,297.30	21,447.62	**21,437.82**	0.66	5,216.23	21,495.36	0.93	6,672.66
i700_9	20,979.88	21,103.89	**21,007.41**	0.13	5,057.04	21,103.66	0.59	7,526.15
i700_10	22,055.41	22,303.66	**22,274.57**	0.99	5,915.74	22,317.87	1.19	6,589.16
i700_14	17,337.63	17,384.12	**17,383.99**	0.27	3,876.69	17,387.91	0.29	6,349.25
i700_15	18,145.49	18,237.48	**18,203.29**	0.32	6,288.32	18,238.03	0.51	5,469.25
i700_17	16,199.55	16,199.55*	16,204.17	0.03	3,845.20	16,228.71	0.18	2,968.71
i1000_1	49,509.81	50,734.33	50,950.51	2.91	10,883.00	51,049.57	3.11	5,977.81
i1000_6	27,823.84	28,096.36	**28,051.58**	0.82	10,534.94	28,096.51	0.98	7,461.93
i1000_8	27,375.37	27,638.39	27,660.13	1.04	10,576.12	27,665.55	1.06	9,025.49
i1000_9	26,857.09	27,164.81	**27,127.70**	1.01	7,579.05	27,187.43	1.23	7,612.12
i1000_14	22,312.01	22,407.30	**22,387.37**	0.34	8,716.35	22,407.95	0.43	6,461.07
i1000_15	22,629.44	22,722.58	**22,706.59**	0.34	7,518.65	22,726.75	0.43	7,385.74
i1000_16	21,331.81	21,389.82	21,417.10	0.40	7,528.09	21,417.14	0.40	6,420.71
i1000_17	21,188.89	21,234.62	**21,198.30**	0.04	8,457.18	21,235.51	0.22	5,787.05
i1000_18	20,713.43	20,752.95	20,768.22	0.26	8,520.34	20,796.28	0.40	5,665.80
i1000_19	20,537.45	20,609.43	**20,564.79**	0.13	6,942.33	20,625.76	0.43	5,642.94
i1000_20	21,560.86	21,619.98	**21,618.06**	0.27	5,289.36	21,619.98	0.27	4,327.69

*Verified optimum found by CPLEX.

Table 6: Percentage improvement due to each procedure of HMEH for instances shown in Table 5.

	FP			FP+BP			HMEH*		
Inst	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
i300_1	0.53	3.87	1.95	2.17	5.49	3.24	3.97	5.96	4.97
i300_4	1.04	4.14	2.23	3.48	5.13	4.08	4.84	5.64	5.17
i300_7	1.15	1.88	1.40	2.15	3.53	2.55	3.36	4.37	3.76
i300_10	0.87	2.19	1.66	2.44	3.35	2.92	3.11	3.79	3.45
i500_2	2.30	4.62	3.20	3.86	5.87	4.73	5.65	7.29	6.54
i500_4	1.03	3.12	1.82	2.19	4.76	3.30	4.49	5.96	5.22
i500_5	0.05	1.71	0.80	2.63	4.55	3.79	3.77	5.99	4.96
i500_6	0.40	1.89	0.98	2.22	3.68	3.03	2.86	4.73	3.68
i500_7	0.35	2.07	0.98	3.17	4.16	3.58	3.47	4.78	4.31
i500_9	0.76	2.26	1.38	2.28	3.50	2.98	3.31	4.29	3.88
i500_10	0.81	2.29	1.38	2.24	3.34	2.91	2.87	3.94	3.55
i500_13	0.14	1.76	0.92	1.43	2.52	1.87	1.63	2.83	2.34
i700_1	0.11	1.54	0.82	2.87	4.29	3.58	4.71	5.39	5.15
i700_5	1.17	3.20	2.33	3.65	5.38	4.48	5.57	6.03	5.75
i700_6	0.57	1.51	1.03	2.12	3.17	2.79	3.17	4.51	3.69
i700_7	0.57	1.64	0.98	1.76	3.20	2.52	2.93	3.71	3.38
i700_9	0.42	1.48	0.86	2.06	3.19	2.71	3.09	4.28	3.60
i700_10	0.68	1.74	1.16	2.10	2.99	2.63	2.94	4.52	3.66
i700_14	0.78	1.37	1.01	1.82	2.09	1.90	1.95	2.25	2.06
i700_15	0.39	2.02	1.12	2.28	3.50	2.86	3.00	4.54	3.82
i700_17	0.14	0.40	0.27	0.37	1.14	0.71	0.57	1.20	0.93
i1000_1	0.28	1.05	0.64	2.77	3.52	3.28	3.79	4.52	4.23
i1000_6	1.03	2.31	1.69	2.67	3.72	3.46	3.72	4.13	4.00
i1000_8	0.18	0.88	0.50	1.51	2.38	1.84	2.46	3.21	2.88
i1000_9	0.44	1.33	0.76	1.64	2.72	2.24	2.61	3.41	3.16
i1000_14	0.76	1.76	1.24	1.77	2.37	2.03	2.13	2.85	2.60
i1000_15	0.55	1.78	0.94	1.18	1.93	1.57	2.06	3.09	2.50
i1000_16	0.44	1.48	0.85	0.85	2.27	1.65	1.52	2.71	2.27
i1000_17	0.35	0.91	0.64	0.96	1.74	1.33	1.16	1.81	1.57
i1000_18	0.41	0.63	0.52	0.72	1.23	0.95	0.88	1.35	1.13
i1000_19	0.63	1.56	1.17	1.68	2.65	2.08	2.06	2.88	2.50
i1000_20	0.21	1.21	0.65	1.04	1.73	1.36	1.46	2.13	1.85
Min	0.05	-	-	0.37	-	-	0.57	-	-
Max	-	4.62	-	-	5.87	-	-	7.29	-
Avg	-	-	1.18	-	-	2.66	-	-	3.52

*HMEH includes FP, BP, and the single-customer multi-exchange procedure.

Ahuja et al. (2004) on a number of benchmark instances. It can also find a new best known solution to 27 large-scale problems from the literature.

As for future research, there are several ways we believe that our approach could be improved upon. Firstly, the forward and backward selection problems were solved in a very simply greedy manner. More sophisticated solution approaches (e.g., specialized heuristics for multi-knapsack problems) could likely be adapted to solve these problems, resulting in a more efficient search and potentially higher quality solutions if more restarts can be performed for the same level of computational effort. Secondly, alternative search strategies (e.g., other hypergraph structures) for finding improving split multi-exchanges could also be investigated. We searched for cost-negative mono F-subgraphs and mono B-subgraphs, but other types of specialized hypergraphs may prove to be better. Finally, based on the success of using hypergraph multi-exchange to solve SSCFLP, it would be interesting to see it applied to other capacitated facility location problems (e.g., dynamic shelter location and problems with bottleneck objectives), other types of partitioning problems (e.g., school districting and nurse scheduling problems), routing problems, and other complex combinatorial optimization problems.

Acknowledgments

We thank four anonymous referees and the handling editor for very helpful comments made on earlier drafts of this paper.

References

Addis, B., Carello, G., Ceselli, A., 2013. Combining very large scale and ILP based neighborhoods for a two-level location problem. European Journal of Operational Research 231 (3), 535–546.

Agar, M., Salhi, S., 1998. Lagrangean heuristics applied to a variety of large capacitated plant location problems. Journal of Operational Research Society 49 (10), 1072–1084.

Ahuja, R. K., Cunha, C. B., 2005. Very large-scale neighborhood search for the k-constraint multiple knapsack problem. Journal of Heuristics 11 (5-6 SPEC. ISS.), 465–481.

Ahuja, R. K., Ergun, O., Orlin, J. B., Punnen, A. P., 2002. A survey of very large-scale neighborhood search techniques. Discrete Applied Mathematics 123 (1-3), 75–102.

Ahuja, R. K., Orlin, J. B., Pallottino, S., Scaparra, M. P., Scuttellà, M. G., 2004. A multi-exchange heuristic for the single-source capacitated facility location problem. Management Science 50 (6), 749–760.

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

Ambrosino, D., Sciomachen, A., Scutellà, M. G., 2009. A heuristic based on multi-exchange techniques for a regional fleet assignment location-routing problem. Computers and Operations Research 36 (2), 442–460.

Avella, P., Boccia, M., 2009. A cutting plane algorithm for the capacitated facility location problem. Computational Optimization and Applications 43 (1), 39–65.

Barcelo, J., Casanova, J., 1984. A heuristic lagrangean algorithm for the capacitated plant location problem. European Journal of Operational Research 15, 212–226.

Beasley, J. E., 1990. OR-Library: Distributing test problems by electronic mail. Journal of Operational Research Society 41 (11), 1069–1072.

Beasley, J. E., 1993. Lagrangean heuristics for location problems. European Journal of Operational Research 65 (3), 383–399.

Berge, C., 1993. Graphs and Hypergraphs. Elsevier.

Borndörfer, R., Heismann, O., 2015. The hypergraph assignment problem. Discrete Optimization 15, 15 – 25.

Cambini, R., Gallo, G., Scutellà, M. G., 1997. Flows on hypergraphs. Mathematical Programming 78 (2), 195–217.

Chen, C. H., Ting, C. J., 2008. Combining lagrangean heuristic and ant colony system to solve the single source capacitated facility location problem. Transportation Research Part E: Logistics and Transportation Review 44 (6), 1099–1122.

Contreras, I. A., Diaz, J. A., 2008. Scatter search for single source capacitated facility location problem. Annals of Operations Research 157 (1), 73–89.

Cortinhal, M. J., Captivo, M. E., 2003. Upper and lower bounds for the single source capacitated location problem. European Journal of Operational Research 151 (2), 333–351.

Delmaire, H., Diaz, J. A., Fernandez, E., Ortega, M., 1999. Reactive GRASP and tabu search based heuristics for the single source capacitated plant location problem. Infor-Information Systems and Operational Research 37 (3), 194–225.

Diaz, J. A., Fernandez, E., 2002. A branch-and-price algorithm for the single source capacitated plant location problem. Journal of Operational Research Society 53 (3), 728–740.

Filho, V. J. M. F., Galvao, R. D., 1998. A tabu search heuristic for the concentrator location problem. Location Science 6 (1-4), 189–209.

Frangioni, A., Necciari, E., Scutellà, M. G., 2004. A multi-exchange neighborhood for minimum makespan parallel machine scheduling problems. Journal of Combinatorial Optimization 8 (2), 195–220.

Gallo, G., Longo, G., Pallottino, S., 1993. Directed hyperpaths and applications. Discrete Applied Mathematics 42 (2-3), 177–201.

Galvao, R. D., Marianov, V., 2011. Lagrangean relaxation-based techniques for solving facility location problems. In H.A. Eiselt and V. Marianov (Eds.), Foundations of Location Analysis, International Series in Operations Research & Management Science 155, 391–420.

Guastaroba, G., Speranza, M. G., 2014. A heuristic for BILP problems: The single source capacitated facility location problem. European Journal of Operational Research 238 (2), 438–450.

Hindi, H., Pienkosz, K., 1999. Efficient solution of large scale, single-source, capacitated plant location problems. Journal of Operational Research Society 50 (3), 268–274.

Ho, S. C., 2015. An iterated tabu search heuristic for the single source capacitated facility location problem. Applied Soft Computing Journal 27, 169–178.

Holmberg, K., Ronnqvist, M., Yuan, D., 1999. An exact algorithm for the capacitated facility location problems with single sourcing. European Journal of Operational Research 113 (3), 544–559.

Klincewicz, J., Luss, H., 1986. A lagrangean relaxation heuristic for capacitated facility location with single-source constraints. Journal of Operational Research Society 37 (5), 495–500.

Klose, A., Drexl, A., 2005. Facility location models for distribution system design. European Journal of Operational Research 162 (1), 4–29.

Laporte, G., Nickel, S., Saldanha da Gama, F., 2015. Location Science. Springer.

Neebe, A. W., Rao, M. R., 1983. An algorithm for the fixed-charge assigning users to sources problem. Journal of the Operational Research Society 34 (11), 1107–1113.

Nguyen, S., Pallottino, S., 1988. Equilibrium traffic assignment for large scale transit networks. European Journal of Operational Research 37 (2), 176–186.

Öncan, T., Kabadi, S., Nair, K. P. K., Punnen, A. P., 2008. VLSN search algorithms for partitioning problems using matching neighbourhoods. Journal of the Operational Research Society 59 (3), 388–398.

Pirkul, H., 1987. Efficient algorithm for the capacitated concentrator location problem. Computers & Operations Research 14 (3), 197–208.

Rahman, A., Poirel, C. L., Badger, D. J., Estep, C., Murali, T. M., 2013. Reverse engineering molecular hypergraphs. IEEE/ACM Transactions on Computational Biology and Bioinformatics 10 (5), 1113–1124.

Rainwater, C., Geunes, J., Romeijn, H. E., 2012. A facility neighborhood search heuristic for capacitated facility location with single-source constraints and flexible demand. Journal of Heuristics 18 (2), 297–315.

Ronnqvist, M., Holt, J., Tragantalerngsak, S., 1999. A repeated matching heuristic for the single-source capacitated facility location problem. European Journal of Operational Research 116 (1), 51–68.

Scaparra, M. P., Pallottino, S., Scutellà, M. G., 2004. Large-scale local search heuristics for the capacitated vertex p-center problem. Networks 43 (4), 241–255.

Sridharan, R., 1991. A lagrangean heuristic for the capacitated plant location problem with single source constraints. European Journal of Operational Research 66 (3), 305–312.

Talluri, K. T., 1996. Swapping applications in a daily airline fleet assignment. Transportation Science 30 (3), 237–248.

Thompson, P., Orlin, J. B., 1989. The theory of cyclic transfers. Report 200-89, Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA.

Thompson, P. M., Psaraftis, H. N., 1993. Cyclic transfer algorithms for multivehicle routing and scheduling problems. Operations Research 41 (5), 935–946.

Torres, A. F., Araoz, J. D., 1988. Combinatorial models for searching in knowledge bases. Acta Cient, Venezolana 39, 387–394.

Wasserman, S., Faust, K., 1999. Social Network Analysis: Methods and Applications. New York: Cambridge University Press.

Yang, Z., Chu, F., Chen, H., 2012. A cut-and-solve based algorithm for the single-source capacitated facility location problem. European Journal of Operational Research 221 (3), 521–532.

Zorzi, M., Rao, R. R., 2003. Geographic random forwarding (GeRaF) for ad hoc and sensor networks: Multihop performance. IEEE Transactions on Mobile Computing 2 (4), 337–348.

Appendix A Greedy Algorithms

Algorithm A1 Pseudo-code of the greedy heuristic for solving FSP at node i.

```
Step 0: Set $\hat{x}_{\ell j} = 0$ for each $j \in S_i$ and $\ell \in N_j$, $\hat{y}_\ell = 0$ for each $\ell \in F_i$, and $Z_F(E_i) = 0$
        For each $j \in S_i$ and $\ell \in N_j$, compute the reassignment cost $c'_{j\ell} = c_{\ell j} - c_{ij}$
        Sort the pairs $(j, \ell)$ in ascending order of reassignment costs and store them in a list $Q$
Step 1: If $c'_{j\ell} \geq 0$ for each pair $(j, \ell)$, then
          (i) Try to reassign each customer $j \in S_i$ to the best suitable open facility and close facility $i$
          (ii) If the move has negative cost, then update $\hat{x}_{\ell j}$, $\hat{y}_\ell$, and $Z_F(E_i)$ accordingly
          (iii) STOP
Step 2: Extract the first pair $(j, \ell)$ from $Q$
        If assigning $j$ to $\ell$ does not violate constraints (10) and (13), perform the reassignment:
          $\hat{x}_{\ell j} = 1$ ,$\hat{y}_\ell = 1$, and $Z_F(E_i) = Z_F(E_i) + c_{\ell j} - c_{ij}$
          If facility $\ell$ is closed, then $Z_F(E_i) = Z_F(E_i) + f_\ell$; mark facility $\ell$ as open
          Remove all pairs containing customer $j$ from $Q$
        If $Q = \emptyset$ is empty or $c'_{j\ell} \geqslant 0$ for each pair $(j, \ell)$, then go to Step 3
        Else go back to Step 2
Step 3: If $i$ is a root node and all the customers in $S_i$ are reassigned, then $Z_F(E_i) = Z_F(E_i) - f_i$
        If constraint (14) is met, then STOP
        Else
          Do
            If $Q = \emptyset$ , then set $Z_F(E_i) = 0$ and STOP (no solution is found)
            Extract the first pair $(j, \ell)$ from $Q$
            If moving $j$ from $i$ to $\ell$ does not violate constraints (10) and (13), perform the reassignment:
              (i) Set $\hat{x}_{\ell j} = 1$ ,$\hat{y}_\ell = 1$, and $Z_F(E_i) = Z_F(E_i) + c_{\ell j} - c_{ij}$
              (ii) If facility $\ell$ is closed, then $Z_F(E_i) = Z_F(E_i) + f_\ell$; mark facility $\ell$ as open
              (iii) Remove all pairs containing customer $j$ from $Q$
          Loop until constraint (14) is met
          Improve the solution by undoing any assignment of $j$ to $\ell$ (set $\hat{x}_{\ell j} = 0$) if
          $c'_{j\ell} > 0$ and constraint (14) is still met. Update $\hat{y}_\ell$, and $Z_F(E_i)$ accordingly.
```

Algorithm A2 Pseudo-code of the greedy heuristic for solving BSP at node i.

```
Step 0:  Set $\check{x}_j = 0$ for each $j \in M'_i$ and $\check{y}_k = 0$ for each $k \in F'_i$
         If solving BSP-Root, then initialize $Z_B(E'_i) = f_i$, otherwise $Z_B(E'_i) = 0$
Step 1:  For each customer $j \in M'_i$ compute the reassignment cost per unit of demand $b_j = \frac{c_{ij} - c_{k_j j}}{w_j}$
         Sort the customers in ascending order of $b_j$ and store them in a list $Q$
Step 2:  While $Q \neq \emptyset$ and $b_j < 0$ for some $j \in Q$
             Extract the first element $j$ from $Q$
             If moving $j$ from $k_j$ to $i$ does not violate constraints (20) and (22), perform the reassignment:
               (i) Set $\check{x}_j = 1$, $\check{y}_{k_j} = 1$ , $Z_B(E'_i) = Z_B(E'_i) + c_{ij} - c_{k_j j}$
               (ii) Set $S'_{k_j} = S'_{k_j} \setminus \{j\}$. If $S'_{k_j} = \emptyset$, then $Z_B(E'_i) = Z_B(E'_i) - f_{k_j}$
         STOP
```

Appendix B Notation Summary

Table B1: General notation.

$S = \{S_1, ..., S_n\}$	Solution to problem SSCFLP
S_i	Set of customers assigned to facility i in solution S
$C(S_i)$	Cost of subset S_i
$W(S_i)$	Total demand of subset S_i
r_i	Residual capacity of facility i
$I(S)$	Set of open facilities in solution S
$I^c(S)$	Set of closed facilities in solution S
$U_{i\ell}$	Subset of customers moving from facility i to facility ℓ
$Q = \{i_1, ..., i_q\}$	Sequence of indexes of q subsets of the partition S
U_{i_p}	Set of all customer subsets leaving facility i_p
$U = \{U_{i_1}, ..., U_{i_q}\}$	Sequence of q customer subsets defining a split exchange
$\bar{U}$	Union of all the customer subsets in the sequence U

Table B2: Hypergraph notation.

$BS(i)$	Backward star of node i
$FS(i)$	Forward star of node i
E_i	F-arc exiting node i
E'_i	B-arc entering node i
$H(E_i)$	Head of hyperarc E_i
$T(E'_i)$	Tail of hyperarc E'_i

Table B3: Notation used in the forward and backward selection problems.

L	Maximum number of heads (tails) of a forward (backward) hyperarc
R	Distance or cost radius
α	Parameter controlling the capacity violation in the Forward Selection Problem
M_i	Set of customers that can be served by facility i
N_j	Set of facilities that can serve customer j
k_j	Index of the facility serving customer j in the current solution S
RC	Set of customers that have been reassigned
M_i'	Set of customers within distance R from facility i that have not been reassigned
S_i'	Subsets of customers assigned to facility i in solution S and not yet reassigned
F_i	Set of possible destination facilities for customers currently allocated to facility i
F_i'	Set of possible source facilities for customers moving into facility i

Table B4: Notation used in the forward and backward procedures.

F^0	Set of root nodes for the search
Q_N	Queue of nodes to be examined
P	Set of nodes already processed
C_F	Cost of the mono F-graph
C_B	Cost of the mono B-graph
$Z_F(E_i)$	Cost of the F-arc E_i
$Z_B(E_i')$	Cost of the B-arc E_i'
ϵ	Parameter used to calibrate α in the forward procedure