A Multilayer Variable Neighborhood Search Method for a Two-Echelon Adaptive Location-Distribution Problem

Bernard Gendron
Paul-Virak Khuong
Frédéric Semet

June 2011

CIRRELT-2011-37

A Multilayer Variable Neighborhood Search Method for a Two-Echelon Adaptive Location-Distribution Problem

Bernard Gendron[1,*] **Paul-Virak Khuong**[1], **Frédéric Semet**[2]

[1] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT), and Department of Computer Science and Operations Research, Université de Montréal, P.O. Box 6128, Station Centre-Ville, Montréal, Canada H3C 3J7

[2] Laboratoire d'Automatique, Génie Informatique et Signal (LAGIS), Ecole Centrale de Lille, Avenue Paul Langevin, BP 48, 59651 Villeneuve d'Ascq Cedex, France

Abstract. We consider a location-distribution problem motivated by a case study for a multi-channel retailing company, which sells a wide variety of products via Internet, mail order catalogs, and stores. Since most items to deliver are small or medium-size parcels, consolidation is a major concern which is addressed by designing a multi-echelon distribution system. This system is adaptive, in the sense that terminals and satellites can be opened or closed easily according to demand fluctuations. We develop a heuristic method to solve this problem, which is based on a variant of the variable neighborhood search (VNS) metaheuristic, which we call the multilayer VNS. We present computational results on a large set of instances based on an actual application.

Keywords. Multi-echelon distribution system, adaptive location problem, variable neighborhood search.

Acknowledgements. We gratefully acknowledge financial support for this project provided by Natural Sciences and Engineering Council of Canada (NSERC), by the Nord-Pas-de-Calais Region, the European Community, the Regional Delegation for Research and Technology, the French Ministry of Higher Education and Research, and the National Center for Scientific Research through the International Campus on Safety and Intermodality in Transportation.

---

* Corresponding author: Bernard.Gendron@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2011

1 Introduction

In this paper, we address the approximate solution of a two-echelon location-distribution problem which arises typically in the distribution of goods at a national level. This work is motivated by a case study for a multi-channel retailing company described by Gendron, Semet and Strozyk [11]. The company sells a wide variety of products such as clothes, electronic devices and appliances via Internet, mail order catalogs and stores. One of its main challenges at the logistics level is to adapt the distribution system according to demand variations while guaranteeing service quality and minimizing the overall logistics cost. Indeed, the demand varies quite significantly from one month to another due to seasonal effects and sales, while goods have to be delivered within a preset period of time, 24 hours typically. Since most items to deliver are small or medium-size parcels, consolidation is a major concern which is addressed by designing a multi-echelon distribution system. The first echelon is associated with primary facilities such as central warehouses. The second echelon corresponds to facilities such as cross-docking terminals. The third echelon is associated to facilities such as small cross-docking terminals called satellites from which tours are issued to serve customers. More precisely, from a small set of warehouses (their locations are assumed known and fixed, following a preliminary strategic analysis), a fleet of large-size trucks delivers parcels to cross-docking terminals, where they are transferred on medium-size trucks, and then shipped to satellites, where the parcels are sorted and delivered to the customers. The company owns only the central warehouses. Cross-docking terminals and satellites are neither owned nor rented by the company but by subcontractors such as independent carriers. Satellites can be very basic facilities such as car parks where items are transferred from trucks to vans. The problem is to ensure that customers' requests are satisfied on time at minimum cost, taking into account the transportation costs and the location costs for using the cross-docking terminals and the satellites. The multi-echelon system we consider allows to satisfy time constraints in a cost-effective way. It is also an *adaptive* system, in the sense that satellites can be opened or closed easily according to demand fluctuations.

We model the problem by defining a network for which the only possible connections are those that ensure on time delivery of the parcels to the customers. In addition, we assume that for each satellite, the set of customers and the routes used to satisfy their requests have been determined in a preprocessing phase. Hence, the model does not include any routing aspect. Each customer in our model represents a set of customers served by a single vehicle. The transportation cost between a satellite and a customer thus corresponds to the cost of the best route determined during this preprocessing phase.

Transportation costs between central warehouses and cross-docking terminals, and between terminals and satellites, vary with the distance travelled, but more importantly, with the number of vehicles used on each connection, each type of vehicle (large- or medium-size truck) having an associated volumetric capacity. A fixed cost is incurred when using any terminal, while the satellite location cost increases with the number of batches of products handled at the satellite (a batch corresponds to a fixed number of parcels). Note that this cost structure is more complex than what can be found in most location-distribution problems discussed in the literature, which typically exhibit fixed costs at the nodes and transportation costs that are linear in the number of product units. Location-distribution problems with complex cost functions have been studied in the past. However, as evidenced by recent contributions by Ghiani, Guerriero and Musmano [13], Correia and Captivo [8], Melo, Nickel and Saldanha da Gama [19], and Wu, Zhang and Zhang [28], most works in facility location consider multiple facilities in one site, but do not address the determination of the transportation capacity as well.

Because the locations of the central warehouses are assumed to be fixed, there are no fixed costs associated to them and we can always assign to each terminal its closest warehouse without losing optimality. This simplification is also performed in the preprocessing phase. Note that we still need to determine how many product units, on how many large-size vehicles, need to be transported between any terminal and its closest warehouse, but we are now allowed to associate the corresponding decision variables to the terminals, instead of the connections between warehouses and terminals. The resulting problem can therefore be considered as a *two-echelon* (from terminals to satellites, and from satellites to customers) capacitated location-distribution problem.

There is an abundant literature on two-echelon location-distribution problems as illustrated by the recent surveys by Klose and Drexl [16] and by Sahin and Süral [26]. Two main formulations are considered: arc-based [17, 23, 24] and path-based models [2, 10, 15, 25]. Most authors propose exact solution methods based on strengthening the models with valid inequalities and facets [1, 7] and on Lagrangean relaxation [3, 18, 23, 24]. Some researchers compare relaxations for the arc-based and path-based formulations [5, 6, 7, 18]. Considering the approximate solution of two-echelon location-distribution problems, heuristic methods based on Lagrangean or linear programming relaxations have been developed [4, 23, 24], while other heuristic approaches rely on greedy strategies or on simple neighborhood search methods based on add, drop or exchange moves [2, 20, 22]. To the best of our knowledge, there are very few attempts to solve two-echelon location-distribution problems with metaheuristics. Chardaire *et al.* [7] describe a simulated annealing procedure which produces good solutions, while Vilcapoma Ignacio *et al.* [27] present a tabu search algorithm which is shown to be efficient compared to the solution of the mathematical program using CPLEX.

Gendron and Semet [12] compare the arc-based and the path-based formulations for our problem. In particular, they show that the LP relaxation of the path-based model provides effective lower bounds, but also that large-scale instances derived from an actual application cannot be solved in a reasonable time using CPLEX. This motivates the development of heuristic methods capable of finding provably good solutions to large-scale real instances in short computing times. The present paper introduces such a heuristic algorithm, based on a variant of the variable neighborhood search (VNS) metaheuristic [14, 21], called the *multilayer VNS*.

The contribution of this paper is three-fold. First, the two-echelon location-distribution problem we study differs significantly from the ones considered in the literature, because the objective function in our problem involves facility installation costs at some of the arcs (represented by the number of vehicles used on the arcs) and at some of the nodes (corresponding to the number of product batches at the satellites). Second, this paper introduces the multilayer VNS, a variant of the VNS metaheuristic. It consists in partitioning the neighborhood structures into multiple layers. For each layer l, a VNS defined on the associated neighborhood structures is invoked, each move being evaluated and completed by a recursive call to the multilayer VNS at layer $l-1$. Such a decomposition turns out to be quite natural when neighborhood structures can be classified according to their complexity. Simple neighborhood structures are assigned to the first layers, while complex neighborhood structures are divided between the last layers, given that the search in layer 1 is frequently invoked due to the recursive nature of the approach. Our third contribution is an adaptation of the multilayer VNS to solve efficiently large-scale instances of the two-echelon location-distribution problem we consider.

The paper is organized as follows. In Section 2, we introduce the problem notation and we present the the path-based model, the solution of which will serve as a basis of comparison. In Section 3, we first describe the multilayer VNS and then show how a heuristic method based on multilayer VNS can be devised for the two-echelon capacitated location-distribution problem. Section 4 presents computational results for instances derived from the large-scale application that motivated this study. We conclude with the presentation of some avenues for future research.

2 Notation and Problem Formulation

Gendron and Semet [12] compare two formulations for the problem, arc-based and path-based, showing that the linear programming (LP) relaxation of the path-based model provides a better bound than the LP relaxation of the arc-based model. When the state-of-the-art mixed-integer programming (MIP) solver CPLEX is performed for the two models, the computational results presented in this paper also demonstrate that the path-based model is a better alternative than the arc-based formulation: either better feasible solutions are obtained for large-scale instances, or optimal solutions are found more rapidly for small-scale instances. In this section, we recall the path-based formulation, as its solution by CPLEX will serve as a basis of comparison for our heuristic method. We also present the notation used to represent the different components of the networks (terminals, satellites and customers) and the connections between them.

The following sets define the different types of nodes in the network:

T	$=$	set of potential sites to locate terminals;
S	$=$	set of potential sites to locate satellites;
L	$=$	set of customers;
T_j^S	$=$	set of potential sites to locate terminals connected to satellite $j \in S$;
S_i^T	$=$	set of potential sites to locate satellites connected to terminal site $i \in T$;
S_l^L	$=$	set of potential sites to locate satellites connected to customer $l \in L$;
L_j^S	$=$	set of customers connected to satellite $j \in S$;
L_i^T	$=$	set of customers connected to terminal $i \in T$ from some satellite $j \in S_i^T$.

The data related to the customer demands and the vehicle capacities are defined as follows (all values are assumed to be positive):

n_l	$=$	number of product units to deliver to customer $l \in L$;
v_l	$=$	volume of product units to deliver to customer $l \in L$;
Q	$=$	capacity (in number of product units) of one batch of products handled at any satellite;
P	$=$	volumetric capacity of a large-size vehicle transporting product units to any terminal, from its closest hub;
R	$=$	volumetric capacity of a medium-size vehicle transporting product units from any terminal to any satellite.

The location and transportation costs are defined as follows (all values are assumed to be nonnegative):

f_i	$=$	fixed cost for using and operating terminal $i \in T$;
g_j	$=$	cost per Q product units for using and operating satellite $j \in S$;
d_i	$=$	transportation cost for using one large-size vehicle to transport product units to terminal $i \in T$ from its closest hub;
e_{ij}	$=$	transportation cost for using one medium-size vehicle from terminal $i \in T$ to satellite $j \in S_i^T$;
c_{jl}	$=$	transportation cost between satellite $j \in S$ and customer $l \in L_j^S$.

To derive the path-based model, the following sets of binary variables are introduced:

$$X_{ijl} = \begin{cases} 1, & \text{if some product units are transported on path } (i,j,l), \\ & i \in T,\ j \in S_i^T,\ l \in L_j^S; \\ 0, & \text{otherwise.} \end{cases}$$

$$W_{ij} = \begin{cases} 1, & \text{if some product units are transported between terminal } i \in T \\ & \text{and satellite } j \in S_i^T; \\ 0, & \text{otherwise;} \end{cases}$$

$$Y_i = \begin{cases} 1, & \text{if some product units are transported to terminal } i \in T \\ & \text{from its closest hub;} \\ 0, & \text{otherwise.} \end{cases}$$

We also use the following general integer variables to represent the number of batches handled at any satellite and the number of vehicles used on any terminal-satellite arc or at any terminal:

U_j	=	number of batches of products handled at satellite $j \in S$;
T_i	=	number of large-size vehicles used between terminal $i \in T$ and its closest hub;
H_{ij}	=	number of medium-size vehicles used between terminal $i \in T$ and satellite $j \in S_i^T$.

The formulation of the problem can then be written as follows:

$$\min \sum_{i\in T} f_i Y_i + \sum_{j\in S} g_j U_j + \sum_{i\in T} d_i T_i + \sum_{i\in T}\sum_{j\in S_i^T} e_{ij} H_{ij} + \sum_{i\in T}\sum_{j\in S_i^T}\sum_{l\in L_j^S} c_{jl} X_{ijl} \tag{1}$$

$$\sum_{j\in S_l^L}\sum_{i\in T_j^S} X_{ijl} = 1, \quad \forall l \in L, \tag{2}$$

$$\sum_{i\in T_j^S} W_{ij} \le 1, \quad \forall j \in S, \tag{3}$$

$$\sum_{j\in S_i^T} X_{ijl} \le Y_i, \quad \forall i \in T, l \in L_i^T, \tag{4}$$

$$X_{ijl} \le W_{ij}, \quad \forall i \in T, \forall j \in S_i^T, \forall l \in L_j^S, \tag{5}$$

$$W_{ij} \le Y_i, \quad \forall i \in T, \forall j \in S_i^T, \tag{6}$$

$$\sum_{i\in T_j^S}\sum_{l\in L_j^S} n_l X_{ijl} \le Q U_j, \quad \forall j \in S, \tag{7}$$

$$\sum_{j\in S_i^T}\sum_{l\in L_j^S} v_l X_{ijl} \le P T_i, \quad \forall i \in T, \tag{8}$$

$$\sum_{l\in L_j^S} v_l X_{ijl} \le R H_{ij}, \quad \forall i \in T, \forall j \in S_i^T, \tag{9}$$

$$U_j \ge 0 \text{ and integer}, \quad \forall j \in S, \tag{10}$$

$$T_i \ge 0 \text{ and integer}, \quad \forall i \in T, \tag{11}$$

$$H_{ij} \ge 0 \text{ and integer}, \quad \forall i \in T, \forall j \in S_i^T, \tag{12}$$

$$X_{ijl} \in \{0,1\}, \quad \forall i \in T, \forall j \in S_i^T, \forall l \in L_j^S, \tag{13}$$

$$W_{ij} \in \{0,1\}, \quad \forall i \in T, \forall j \in S_i^T, \tag{14}$$

$$Y_i \in \{0,1\}, \quad \forall i \in T. \tag{15}$$

The objective function, (1), consists in minimizing all costs incurred by using and operating terminals and satellites, as well as transportation costs between hubs and terminals, between terminals and satellites, and between satellites and customers. Constraints (2) ensure that each customer is being served by a single satellite. Constraints (3) ensure that any satellite, when it is used, is connected to a single terminal. The forcing constraints (4) ensure that no flow can circulate through a terminal that is not used to transport product units (note that this constraint is a disaggregated, hence stronger version, of the corresponding constraint introduced in Gendron and Semet [12]). Constraints (5) and (6) are also forcing constraints that link together the different types of binary variables. Constraints (5) ensure that any customer cannot be routed from a satellite that is not connected to some terminal. Similarly, constraints (6) ensure that any satellite cannot be connected to a terminal that is not used to transport product units. Constraints (7) ensure that the number of product units handled at a satellite cannot exceed the capacity of product batches. Constraints (8) and (9) ensure that the total volume of all product units transported on a network element (terminal or terminal-satellite arc) cannot exceed the capacity of the vehicles used on that network element. Other constraints specify the nature of the different types of variables.

3 Multilayer Variable Neighborhood Search

The variable neighborhood search (VNS) metaheuristic [14, 21] provides a systematic approach for exploiting multiple neighborhood structures for problem (P): $\min_{x \in X \subseteq S}\{f(x)\}$, where X is the set of feasible solutions and S is the search space. Starting from the best known feasible solution $x \in X$ and assuming there are k_{max} neighborhood structures, VNS first orders them from 1 to k_{max} and then scans them iteratively in that order. At every iteration k, a new solution $x' \in S$ is generated in the current neighborhood $N_k(x)$

(this solution might then be further optimized using some search algorithm). Then, there are two cases: (1) $f(x') < f(x)$, in which case the search restarts from x', which is now the best known feasible solution, and is re-centered towards the first neighborhood, i.e., we set k to 1; (2) $f(x') \geq f(x)$, in which case the attempt to improve solution x using $N_k(x)$ has failed and we move on to the next neighborhood $N_{k+1}(x)$, unless $k+1$ exceeds k_{max}, in which case the loop over k is stopped. Typically, there are randomized components in the selection of x' in $N_k(x)$. This is why this loop is embedded into an outer loop that controls the overall computational effort. Algorithm 1 summarizes the method.

Algorithm 1 VNS(x: current solution)

```
1: Order the $k_{max}$ neighborhood structures: $N_1, \ldots, N_{k_{max}}$
2: repeat
3:   $k \leftarrow 1$
4:   repeat
5:     Generate $x' \in N_k(x)$
6:     if $f(x') < f(x)$ then
7:       $x \leftarrow x'$
8:       $k \leftarrow 1$
9:     else
10:      $k \leftarrow k + 1$
11:    end if
12:  until $k > k_{max}$
13: until some termination condition is satisfied
```

The solution x given as input to the algorithm is typically obtained by some greedy constructive procedure, but can also be derived from more sophisticated methods. In Step 5, one might generate x' at random (the so-called "shaking" step), or find the best solution in $N_k(x)$. When x' is generated at random, one obtains the so-called "reduced" VNS, while if x' is the best solution in $N_k(x)$, one obtains the variable neighborhood descent (VND) metaheuristic. Between these two extreme approaches, there is a wide spectrum of possibilities: for instance, find the best solution in a subset of $N_k(x)$ (often randomly generated), or scan the solutions in $N_k(x)$ in some order (often randomly generated) until a first improving solution is found. Often, the final x' is obtained after performing a search algorithm, which can be inspired by VNS itself or by any other metaheuristic, or even by an exact method derived from integer programming or constraint programming.

The termination condition of the algorithm typically controls the overall computational effort by imposing limits on the number of iterations or the total time.

The multilayer VNS (MLVNS) is a variant of VNS based on dividing the neighborhood structures into l_{max} layers, each layer l having k^l_{max} associated neighborhood structures. The layers are ordered from 1 to l_{max} and then scanned in that order. Typically, as in the standard VNS, the neighborhood structures grow in complexity with the layer index. For each layer l, a VNS is invoked, with the generation of x' in a neighborhood $N_k(x)$ consisting in scanning the solutions in a subset of $N_k(x)$ in some (randomly generated) order until a first improving solution is found. To define the subset of $N_k(x)$, we use m^k_{max} (randomly ordered) sub-neighborhood structures, resulting from a "natural" decomposition of N_k: $N^1_k, \ldots, N^{m^k_{max}}_k$. At each step m, one move in $N^m_k(x)$ is performed to generate a candidate solution x'. If $l = 1$, we assume the evaluation of this candidate solution is easy. Otherwise, when $l > 1$, further optimization is necessary to evaluate the impact of the move; for that purpose, MLVNS is called recursively up to layer $l - 1$ to produce a final candidate solution x'. The generation of candidate solutions continues until x' improves upon x or $m > m^k_{max}$, i.e., all sub-neighborhood structures have been explored. The remaining steps are identical to those performed by the standard VNS. Algorithm 2 summarizes the approach.

As for VNS, the initial solution x given as input to the algorithm is often obtained by a greedy constructive procedure. The number of layers l_{max} and the order in which they will be traversed is also determined in the initialization phase. The search in layer $l > 1$ makes use of the search for the previous layers to evaluate any candidate solution x'. Such a strategy can only be efficient if the complexity of the neighborhood structures increases with the layer index, since the search in layers 1 to l is repeated for all subsequent layers $l + 1$ to l_{max}. MLVNS formalizes two approaches that are commonly used in the metaheuristics literature. First, when there are neighborhood structures for which the exact evaluation of each move is so complex that it could be advantageous to perform a heuristic evaluation instead. In that case, the MLVNS framework represents the situation where this heuristic evaluation is obtained by calling MLVNS itself on previous layers.

Algorithm 2 MLVNS(x: current solution, l_{max}: number of layers)

```
1:  for $l \leftarrow 1$ to $l_{max}$ do
2:    Order the $k^l_{max}$ neighborhood structures: $N_1, \ldots, N_{k^l_{max}}$
3:    repeat
4:      $k \leftarrow 1$
5:      repeat
6:        Order (randomly) the $m^k_{max}$ sub-neighborhood structures: $N^1_k, \ldots, N^{m^k_{max}}_k$
7:        $m \leftarrow 1$
8:        repeat
9:          Generate $x' \in N^m_k(x)$
10:         MLVNS($x'$, $l - 1$)
11:         $m \leftarrow m + 1$
12:       until $(f(x') < f(x))$ OR $(m > m^k_{max})$
13:       if $f(x') < f(x)$ then
14:         $x \leftarrow x'$
15:         $k \leftarrow 1$
16:       else
17:         $k \leftarrow k + 1$
18:       end if
19:     until $k > k^l_{max}$
20:   until some termination condition is satisfied
21: end for
```

A second situation occurs when a diversification method is used to complement a VNS, where the diversification steps themselves can be represented by a VNS. We will see examples of these two situations in the heuristic method that we propose to solve our problem. It is a three-layer MLVNS approach: layer 1 represents a VNS with simple neighborhood structures; layer 2 is an MLVNS based on complex neighborhood structures for which each move is evaluated using the VNS of layer 1; layer 3 can be seen as a diversification approach that perturbs the current solution with techniques that can be assimilated to particular moves in large-scale neighborhood structures, each move being completed by performing the MLVNS of layer 2.

In order to simplify the description of the MLVNS heuristic for our problem, we present a *generic neighborhood search* (GNS) procedure that is used in all subsequent developments. Given a solution x and a non-empty finite neighborhood $N(x)$, the procedure outputs a solution $x' \in N(x) \cup \{x\}$ according to a search strategy that depends on the values of two input parameters: STOP, which determines when to stop the search in $N(x)$, and Δ, which specifies if x' should improve, or not, upon x. The procedure is described in Algorithm 3.

Algorithm 3 GNS(x: current solution, $N(x)$: neighborhood of x, (STOP, Δ), x': new solution)

```
1: $x' \leftarrow x$
2: repeat
3:     Generate $x'' \in N(x)$ and remove $x''$ from $N(x)$
4:     $\Delta(x'', x) \leftarrow f(x'') - f(x)$
5:     if $\Delta(x'', x) < \Delta$ then
6:         $x' \leftarrow x''$
7:         $\Delta \leftarrow \Delta(x'', x)$
8:     end if
9: until (STOP) OR ($N(x) = \emptyset$)
```

Four main variants of the GNS procedure are defined by the values of the input parameters STOP and Δ:

1. *best*: finding the best, but not necessarily improving, solution in $N(x)$ is obtained by setting STOP = *false* and $\Delta = \infty$;

2. *best improving*: finding the best improving solution in $N(x)$ (and returning x if no such solution exists) is obtained by setting STOP = *false* and $\Delta = 0$;

3. *first improving*: by setting STOP = $(\Delta < 0)$ and $\Delta = 0$, the search stops when a first improving solution in $N(x)$ is found, but if no such solution exists, x is returned;

4. *first improving or best*: by setting STOP = $(\Delta < 0)$ and $\Delta = \infty$, the search stops when a first improving solution in $N(x)$ is found, but if no such solution exists, the best solution in $N(x)$ is returned.

In the remainder of the text, these four terms will be used to designate the appropriate combination of values of the two input parameters STOP and Δ.

Before providing the details of the three layers of our MLVNS heuristic, we now describe the structure of feasible solutions to our problem and then present the greedy constructive procedure used to generate the initial solution.

3.1 Solution Space

The structure of any feasible solution x to our problem is a forest containing all customers, and only the open satellites $S(x)$ and the open terminals $T(x)$. Each tree in this forest is rooted at an open terminal i, which is connected to the open satellites assigned to terminal i, denoted $S_i(x)$, and each of these open satellites, say j, is itself connected to the customers assigned to satellite j, denoted $L_j(x)$. Computing the objective function value $f(x)$ of any such solution x is an easy task, although there are many elements defining the cost. In particular, the number of large- and medium-size vehicles, as well as the number of product batches handled at satellite locations, are easy to compute. We exploit the forest structure of any feasible solution, and the ease in computing the cost of such solution, in the MLVNS method developed to solve our problem.

3.2 Initial Solution by a Greedy Constructive Procedure

At every iteration of the greedy constructive procedure, we are given an infeasible current solution x, i.e., some customers are not connected in x. An iteration starts by selecting one such unconnected customer, say l, and by exploring the ConnectC(l) neighborhood which contains all solutions that connects l to a pair satellite-terminal. The ConnectC(l) neighborhood is explored using the *best* variant; hence, it finds the best way to connect customer l to any pair satellite-terminal, given the cost of the current infeasible solution. To select customer l, we simply use the observation that customers with higher demand volumes have more impact on the objective function value. Hence, we initially order the customers by decreasing volume demands v^l and scan the customers in that order, each time exploring the ConnectC(l) neighborhood. Since the procedure is used only to quickly generate an initial feasible solution, we decided not to further explore other ways to scan the set of customers. The greedy constructive procedure is summarized in Algorithm 4.

Algorithm 4 Greedy(x: solution)

```
1: Let $x$ be the empty solution (no customers are connected yet)
2: for all $l \in L$ (in decreasing order of volume demands $v^l$) do
3:     GNS($x$, ConnectC($l$), best, $x'$)
4:     $x \leftarrow x'$
5: end for
```

3.3 Layer 1: VNS with Exchange and Close Neighborhoods

Four neighborhood structures are used at layer 1 of the MLVNS method: ExchangeC, ExchangeS, CloseS and CloseT.

In the ExchangeC neighborhood structure, the sub-neighborhood structures are defined by scanning the set of customers in random order. Then, for each customer l, the ExchangeCs(l) sub-neighborhood is explored: it consists in reassigning customer l, currently assigned to satellite j, to another open satellite j'. The ExchangeCs(l) sub-neighborhood is explored using the *first improving* variant of GNS, with the candidate open satellites j' being scanned in increasing order of distance to l. As soon as an improving solution is found, it is selected as the new current solution, but if no improving solution is obtained, the current solution remains so. Note that, when reassigning a customer l from satellite j to another satellite, we might close satellite j (as well as the depot connected to j), but this is easily taken into account when computing the impact of the move on the objective function. To summarize, steps 6 to 12 of MLVNS are performed as in Algorithm 5 for the ExchangeC neighborhood structure.

Algorithm 5 ExchangeC(x: current solution)

```
1: $L' \leftarrow L$
2: **repeat**
3:     Randomly select $l \in L'$ and remove $l$ from $L'$
4:     GNS($x$, ExchangeCs($l$), *first improving*, $x'$)
5: **until** $(f(x') < f(x))$ OR $(L' = \emptyset)$
```

In the ExchangeS neighborhood structure, the sub-neighborhood structures are obtained by scanning the set of open satellites $S(x)$ in random order. For each open satellite j, the ExchangeSs(j) sub-neighborhood is explored: it consists in reassigning satellite j, currently assigned to terminal i, to another open terminal i'. The ExchangeSs(j) sub-neighborhood is explored using the *first improving* variant of GNS, with the candidate open terminals being scanned in increasing order of distance to j. Steps 6 to 12 of MLVNS are performed as in Algorithm 6 for the ExchangeS neighborhood structure.

Algorithm 6 ExchangeS(x: current solution)

```
1: $S' \leftarrow S(x)$
2: **repeat**
3:     Randomly select $j \in S'$ and remove $j$ from $S'$
4:     GNS($x$, ExchangeSs($j$), *first improving*, $x'$)
5: **until** $(f(x') < f(x))$ OR $(S' = \emptyset)$
```

The sub-neighborhood structures in the CloseS neighborhood structure are derived from scanning the set of open satellites $S(x)$ in random order. For each open satellite j, the CloseSs(j) sub-neighborhood is explored by applying the ExchangeCs(l) neighborhood search for all $l \in L_j(x)$, i.e., the customers connected to satellite j in solution x. The exploration of the ExchangeCs(l) neighborhood is performed using the *first* *improving or best* variant of the GNS procedure, i.e., as soon as customer l can be reassigned to another open satellite by improving the cost, the move is performed, but otherwise, the best way to reassign customer l to another open satellite is performed. This approach ensures that all customers currently assigned to satellite j are reassigned to another satellite, which effectively closes satellite j. As in the greedy constructive procedure, the customers in $L_j(x)$ are scanned in decreasing order of volume demands v_l. Algorithm 7 summarizes steps 6 to 12 of MLVNS when the CloseS neighborhood structure is explored.

Algorithm 7 CloseS(x: current solution)

```
 1: $S' \leftarrow S(x)$
 2: **repeat**
 3:     Randomly select $j \in S'$ and remove $j$ from $S'$
 4:     $L' \leftarrow L_j(x)$
 5:     $x'' \leftarrow x$
 6:     **for all** $l \in L'$ (in decreasing order of volume demands $v^l$) **do**
 7:         GNS($x''$, ExchangeCs($l$), *first improving or best*, $x'$)
 8:         $x'' \leftarrow x'$
 9:     **end for**
10: **until** $(f(x') < f(x))$ OR $(S' = \emptyset)$
```

The CloseT neighborhood structure is explored in a similar way. The sub-neighborhood structures are obtained by scanning the set of open terminals $T(x)$ in random order. For each open terminal i, the CloseTs(i) sub-neighborhood is explored by searching the ExchangeSs(j) neighborhood for all $j \in S_i(x)$, i.e., the set of satellites connected to terminal i in solution x. Again, the ExchangeSs(j) neighborhood is explored using the *first improving or best* variant of the GNS procedure, in order to make sure that all satellites connected to terminal i are reassigned, so that terminal i is effectively closed. For each satellite $j \in S_i(x)$, we compute the total volume demand $V_j = \sum_{l \in L_j(x)} v_l$ and scan the satellites in decreasing order of these values. Steps 6 to 12 of MLVNS are performed as in Algorithm 8 when the CloseT neighborhood structure is explored.

The four neighborhood structures are explored at layer 1 of the MLVNS algorithm in the following order: ExchangeC, CloseS, CloseT and ExchangeS. The idea is

Algorithm 8 CloseT(x: current solution)

```
1: $T' \leftarrow T(x)$
2: repeat
3:    Randomly select $i \in T'$ and remove $i$ from $T'$
4:    $S' \leftarrow S_i(x)$
5:    $x'' \leftarrow x$
6:    for all $j \in S'$ (in decreasing order of total volume demands $V_j$) do
7:       GNS($x''$, ExchangeSs($j$), first improving or best, $x'$)
8:       $x'' \leftarrow x'$
9:    end for
10: until $(f(x') < f(x))$ OR $(T' = \emptyset)$
```

to perform ExchangeC moves as much as possible, before trying the more computationally intensive CloseS and CloseT moves, which will also significantly modify the current solution. The less effective ExchangeS neighborhood is explored as a last resort in case the other types of moves did not succeed in improving the solution. Preliminary computational experiments have confirmed the effectiveness of this particular order of exploration. Layer 1 is completed as soon as one loop over the four neighborhood structures did not succeed in improving the current solution, i.e., the stopping condition in Step 20 is set to *true*.

3.4 Layer 2: VNS with Open Neighborhoods

At layer 2 of the MLVNS algorithm, two neighbourhood structures are used: OpenT and OpenS.

In the OpenT neighbourhood structure, the sub-neighborhood structures are obtained by scanning the set of closed terminals $T \setminus T(x)$ in random order. For each closed terminal i, it performs the OpenTs(i) move which connects several, but not necessarily all, open satellites in $S(x) \cap S_i^T$ to terminal i. If *all* open satellites in $S(x) \cap S_i^T$ were connected to i, then some open terminals would have no more satellites connected to them. To avoid this situation, all open satellites in $S(x) \cap S_i^T$ are connected to terminal i, but if an open terminal i' would then be "closed," one (arbitrarily chosen) satellite in $S(x) \cap S_i^T$ remains connected to i'. It is only after applying MLVNS at layer 1 to complete the move that the final configuration of open terminals will be determined.

The OpenS neighborhood structure is explored in a similar way. The sub-neighborhood structures are derived by scanning the set of closed satellites $S \setminus S(x)$ in random order. For each closed satellite j, it performs the OpenSs(j) move by connecting all customers in L_j^S to satellite j, but if an open satellite j' would then be "closed," one (arbitrarily chosen) customer in L_j^S remains connected to j'. Steps 6 to 12 of MLVNS are performed as follows when the OpenT (Algorithm 9) and OpenS (Algorithm 10) neighborhoods are explored. Note that OpenTs(i) and OpenSs(j) can both be seen as neighborhoods containing only one solution; hence, we use GNS with the *best* variant to represent the corresponding moves.

Algorithm 9 OpenT(x: current solution)

```
1: $T' \leftarrow T \setminus T(x)$
2: repeat
3:     Randomly select $i \in T'$ and remove $i$ from $T'$
4:     GNS($x$, OpenTs($i$), best, $x'$)
5:     MLVNS($x'$, 1)
6: until $(f(x') < f(x))$ OR $(L' = \emptyset)$
```

Algorithm 10 OpenS(x: current solution)

```
1: $S' \leftarrow S \setminus S(x)$
2: repeat
3:     Randomly select $j \in S'$ and remove $j$ from $S'$
4:     GNS($x$, OpenSs($j$), best, $x'$)
5:     MLVNS($x'$, 1)
6: until $(f(x') < f(x))$ OR $(L' = \emptyset)$
```

The two neighborhood structures are explored in the following order: OpenT and OpenS. The motivation is that OpenT perturbs the solution more significantly than OpenS, i.e., more customers are reassigned when exploring the OpenT neighborhood than when exploring the OpenS neighborhood. Hence, OpenT tends to diversify the search better than OpenS and should therefore be performed first. Preliminary computational experiments have confirmed this intuition. Layer 2 is completed when one loop over the two neighborhood structures did not succeed in improving the current solution, i.e., the stopping condition in Step 20 is set to *true*.

3.5 Layer 3: Diversification Based on Cost Perturbation

The diversification techniques used at layer 3 of the MLVNS algorithm are based on the following steps: (1) increase the (location or transportation) costs associated to subsets of open satellites (or terminals) in solution x, thus attempting to close these satellites (or terminals); (2) starting from x, solve the resulting perturbed problem by performing the MLVNS algorithm at layer 2, thus obtaining a new solution x'; (3) restore the original costs and perform the MLVNS algorithm at layer 2, starting from x'. Steps (1) and (2) correspond to performing one move in a sub-neighborhood structure, i.e., Step 9 of the MLVNS algorithm, while step (3) is the recursive call to MLVNS, i.e., Step 10 of the MLVNS algorithm. Four types of cost perturbations are performed, each type being assimilated to a neighborhood structure. The PERTURBNODESS (PERTURBNODEST) neighborhood structure randomly selects random numbers of open satellites (terminals), then increases all location costs related to these open satellites (terminals). The PERTURBARCSS (PERTURBARCST) neighborhood structure also randomly selects random numbers of open satellites (terminals). Then, all transportation costs from these open satellites (terminals) to connected customers (satellites) are increased. Algorithms 11 to 14 summarize Steps 6 to 12 of MLVNS for the four neighborhood structures.

Algorithm 11 PERTURBNODESS(x: current solution)

```
1: $x' \leftarrow x$
2: Randomly generate a number $NPerturbS$ in $[\lceil p_1\%|S(x)|\rceil, \lceil p_2\%|S(x)|\rceil]$
3: Randomly select $NPerturbS$ open satellites and increase the location costs of these
   satellites by multiplying them by $M(location)$
4: MLVNS($x'$, 2)
5: Restore all costs to their original values
6: MLVNS($x'$, 2)
```

The order in which these four neighborhood structures are explored is: PERTURBARCSS, PERTURBNODESS, PERTURBNODEST and PERTURBARCST. This order is motivated by the same reasons that justified the order selected for layer 1, since the PERTURBARCSS and PERTURBARCST neighborhood structures are similar to the EXCHANGEC and EXCHANGES neighborhood structures, while the PERTURBNODESS and

Algorithm 12 PERTURBNODEST(x: current solution)

```
1: $x' \leftarrow x$
2: Randomly generate a number $NPerturbT$ in $[\lceil p_1\%|T(x)|\rceil, \lceil p_2\%|T(x)|\rceil]$
3: Randomly select $NPerturbT$ open terminals and increase the location costs of these terminals by multiplying them by $M(location)$
4: MLVNS($x'$, 2)
5: Restore all costs to their original values
6: MLVNS($x'$, 2)
```

Algorithm 13 PERTURBARCSS(x: current solution)

```
1: $x' \leftarrow x$
2: Randomly generate a number $NPerturbS$ in $[\lceil p_1\%|S(x)|\rceil, \lceil p_2\%|S(x)|\rceil]$
3: Randomly select $NPerturbS$ open satellites and increase all transportation costs originating at these satellites by multiplying them by $M(transport)$
4: MLVNS($x'$, 2)
5: Restore all costs to their original values
6: MLVNS($x'$, 2)
```

Algorithm 14 PERTURBARCST(x: current solution)

```
1: $x' \leftarrow x$
2: Randomly generate a number $NPerturbT$ in $[\lceil p_1\%|T(x)|\rceil, \lceil p_2\%|T(x)|\rceil]$
3: Randomly select $NPerturbT$ open terminals and increase all transportation costs originating at these terminals by multiplying them by $M(transport)$
4: MLVNS($x'$, 2)
5: Restore all costs to their original values
6: MLVNS($x'$, 2)
```

Problem	$\lvert T\rvert$	$\lvert S\rvert$	$\lvert L\rvert$	$\lvert A_{TS}\rvert$	$\lvert A_{SL}\rvert$	M_f	M_g	M_p
R(M_f, M_g, M_p)	93	320	701	2250	28782	$\{1,2\}$	$\{1,2\}$	$\{1,2\}$
L(M_f, M_g, M_p)	70	240	526	1260	16131	$\{1,2\}$	$\{1,2\}$	$\{1,2\}$
M(M_f, M_g, M_p)	46	160	350	562	6652	$\{1,2\}$	$\{1,2\}$	$\{1,2\}$
S(M_f, M_g, M_p)	23	80	175	167	1807	$\{1,2\}$	$\{1,2\}$	$\{1,2\}$

Table 1: Set of 32 instances (each row contains 8 instances)

PerturbNodesT neighborhood structures are similar to the CloseS and CloseT neighborhood structures. Layer 3 is completed when a CPU time limit is reached.

4 Computational Results

The MLVNS algorithm was performed on the testbed described in [12]. This testbed is derived from data obtained from a major French mail-order company, which provided us with a typical network having 93 potential sites for the terminals, 320 potential sites for the satellites and 701 customers as well as realistic estimates of the costs and the capacities. Based on this real-application data, we have generated 32 instances by specifying:

- subsets of the sets of terminals, satellites and customers, i.e., T, S and L (three subnetworks, large, medium and small, were generated);
- multipliers M_f, M_g and M_p for, respectively, the fixed costs at the terminals, the unit batch costs at the satellites and the capacities of the large-size vehicles at the terminals (two values, 1 and 2, were tested for each multiplier).

Every instance is denoted X(M_f, M_g, M_p), with X = R, L, M or S, standing for real-application, large-scale, medium-scale or small-scale network, M_f, M_g and M_p denoting the multiplier values. Table 1 summarizes the characteristics of the 32 instances. Column 1 gives the problem name, while the next three columns indicate the number of terminals, satellites and customers. The next two columns show, respectively, the number of arcs between terminals and satellites, denoted $\lvert A_{TS}\rvert$, and the number of arcs between satellites and customers, denoted $\lvert A_{SL}\rvert$.

The algorithms were coded in C++ and run on an Intel Xeon X5660 operating at 2.8 GHz and equipped with 24 GB RAM (the operating system is Debian 6.0/Linux 2.6.36). Since all neighborhood structures are explored according to a random order, each instance was solved five times. The stopping criterion in layer 3 is a maximum CPU time of 1440 seconds. This means that we impose a total limit of 2 hours for the five runs on any given instance. Note that for some runs, CPU times might exceed 1440 seconds, since the stopping criterion is only controlled at Step 20 of the MLVNS algorithm. The other parameters of layer 3 are set to the following values, determined after preliminary computational experiments: $p_1 = 5$ and $p_2 = 20$ (interval of percentages of open elements that see their related costs perturbed); $M(location) = 1 \times 10^9$ and $M(transport) = 100$ (amount by which each cost, location or transportation, is multiplied).

The path-based formulation was solved with the state-of-the-art MIP solver CPLEX (version 12) in two modes:

- "Integer feasibility," all other parameters at their default values, except a CPU time limit set to 1440 seconds, the same that we use for MLVNS; this mode drives CPLEX towards finding good feasible solutions quickly, especially for large-scale and real-application instances, and constitute a benchmark with which we compare our heuristic method.

- All parameters at their default value and a CPU time limit set to 7200 seconds; after preliminary experiments, we have found this mode to deliver effective lower bounds in a reasonable time, thus allowing us to estimate how far our heuristic solutions are from optimal ones.

We provide computational results of the MLVNS algorithm including all three layers. To assess the efficiency of each layer, we also report the results obtained with simplified versions including only the first layer or the first two layers. Our computational results are summarized in Tables 2 to 5. The meanings of the column headings are as follows:

- *Layer i* (i=1, 2, 3): Statistics obtained when MLVNS is applied to layers 1 to i.

Instance	Layer 1 gap(min/avg/max)%			Layer 2 gap(min/avg/max)%			Layer 3 gap(min/avg/max)%			Path gap%
	time(s)		sd(s)	time(s)		sd(s)	time(s)		sd(s)	time(s)
S(1,1,1)	9.63	9.72	9.82	1.06	3.01	4.74	0.68	0.71	0.73	0.07
	0.00		0.00	0.6		0.2	1440.3		0.2	1440.0
S(1,1,2)	10.66	10.74	10.85	0.08	2.48	4.30	0.02	0.03	0.04	0.01
	0.00		0.00	0.5		0.1	1440.3		0.2	9.0
S(1,2,1)	9.14	9.18	9.19	0.89	2.92	4.30	0.47	0.48	0.49	0.08
	0.00		0.00	0.4		0.1	1440.3		0.3	1440.0
S(1,2,2)	10.36	10.37	10.37	0.12	3.30	4.69	0.01	0.03	0.04	0.01
	0.00		0.00	0.6		0.2	1440.4		0.2	11.2
S(2,1,1)	11.40	11.47	11.56	0.87	3.52	5.63	0.51	0.57	0.61	0.07
	0.00		0.00	0.6		0.2	1440.4		0.2	1440.0
S(2,1,2)	12.29	12.35	12.45	0.07	3.10	5.30	0.01	0.02	0.04	0.01
	0.00		0.00	0.5		0.1	1440.3		0.2	6.2
S(2,2,1)	10.91	10.94	10.95	0.74	3.42	5.23	0.39	0.40	0.40	0.05
	0.00		0.00	0.2		0.04	1440.4		0.2	1440.0
S(2,2,2)	11.96	11.97	11.97	0.10	4.09	5.57	0.01	0.02	0.03	0.01
	0.00		0.00	0.5		0.2	1440.4		0.2	7.6
S(-,-,-)	10.79	10.84	10.89	0.49	3.23	4.97	0.26	0.28	0.30	0.04
	0.00			0.5			1440.3			724.3

Table 2: Performance analysis for small-scale instances

- *Path*: Statistics obtained when CPLEX is performed in "Integer feasibility" mode for 1440 seconds.

- *gap (min/avg/max) %*: Minimum, average, and maximum gap over the five runs. The gap (%) is computed between the best known lower bound Z^L and a given upper bound Z^* according to the following formula: $100 \times (Z^* - Z^L)/Z^L$. The best known lower bound Z^L is the lower bound obtained by solving the path-based formulation using CPLEX with a time limit of 7200 seconds and all other parameters at their default values.

- *time (s)*: Average CPU time in seconds.

- *sd(s)*: Standard deviation for CPU time in seconds.

Our results indicate that the MLVNS algorithm performed on the three layers outperforms the solution of the path-based formulation using CPLEX on all instances, except

Instance	Layer 1 gap(min/avg/max)% time(s)		sd(s)	Layer 2 gap(min/avg/max)% time(s)		sd(s)	Layer 3 gap(min/avg/max)% time(s)		sd(s)	Path gap% time(s)
M(1,1,1)	6.47	6.99	7.30	2.39	3.46	5.10	1.68	1.92	2.22	2.49
	0.01		0.00	10.7		3.7	1444.2		4.4	1440.0
M(1,1,2)	8.94	9.19	9.51	2.97	3.07	3.20	0.77	0.80	0.85	2.05
	0.01		0.00	9.0		1.0	1444.5		3.3	1440.0
M(1,2,1)	6.37	7.02	7.97	3.09	3.54	3.81	1.45	1.68	2.23	3.83
	0.01		0.00	12.3		2.7	1446.1		1.6	1440.1
M(1,2,2)	8.79	9.00	9.36	2.58	3.18	3.73	1.03	1.06	1.09	2.91
	0.01		0.00	7.3		2.7	1445.7		2.7	1440.0
M(2,1,1)	8.89	9.34	9.62	2.76	3.64	3.94	1.17	1.35	1.53	2.04
	0.01		0.00	7.5		1.0	1445.6		4.0	1440.0
M(2,1,2)	12.22	12.44	12.72	4.70	4.97	5.84	1.09	1.48	1.60	2.40
	0.01		0.00	7.9		1.2	1445.2		4.6	1440.1
M(2,2,1)	7.66	8.65	10.12	2.53	3.33	4.38	1.35	1.51	1.74	1.72
	0.01		0.00	9.9		1.1	1447.5		3.9	1440.1
M(2,2,2)	11.80	11.99	12.30	2.09	3.69	4.83	1.14	1.40	1.51	1.54
	0.01		0.00	7.1		1.2	1446.9		3.0	1440.1
M(-,-,-)	8.87	9.31	9.85	2.88	3.61	4.35	1.21	1.40	1.60	2.37
	0.01			9.0			1445.7			1440.1

Table 3: Performance analysis for medium-scale instances

Instance	Layer 1			Layer 2			Layer 3			Path
	gap(min/avg/max)%			gap(min/avg/max)%			gap(min/avg/max)%			gap%
	time(s)		sd(s)	time(s)		sd(s)	time(s)		sd(s)	time(s)
L(1,1,1)	12.63	12.98	13.34	4.64	5.83	6.82	2.89	3.11	3.42	16.14
	0.04		0.01	45.3		15.5	1454.8		21.2	1440.1
L(1,1,2)	11.98	12.36	12.71	3.40	6.27	12.27	2.60	2.89	3.32	9.31
	0.05		0.00	34.9		21.4	1456.1		7.2	1440.2
L(1,2,1)	12.22	12.97	13.37	5.05	5.59	6.13	2.95	3.53	4.40	19.49
	0.05		0.01	62.2		7.8	1462.7		17.1	1440.3
L(1,2,2)	12.96	13.15	13.31	5.31	5.79	6.32	3.34	3.46	3.84	18.29
	0.04		0.00	36.1		13.0	1454.8		11.3	1440.1
L(2,1,1)	13.04	13.36	13.68	4.63	5.51	6.59	2.78	2.91	3.10	7.80
	0.03		0.01	37.9		5.2	1450.6		7.4	1440.1
L(2,1,2)	13.16	13.50	13.81	4.02	5.48	7.17	3.00	3.28	3.57	6.29
	0.03		0.00	31.5		12.5	1463.6		16.8	1440.2
L(2,2,1)	11.32	13.15	13.80	4.78	6.35	8.03	3.40	3.52	3.64	9.54
	0.04		0.01	44.5		9.9	1455.1		7.3	1440.1
L(2,2,2)	13.98	14.14	14.29	4.92	5.81	6.62	3.17	3.83	4.52	15.89
	0.02		0.00	34.1		10.0	1460.8		5.4	1440.1
L(-,-,-)	12.66	13.20	13.54	4.59	5.83	7.48	3.02	3.32	3.72	12.74
	0.04			40.8			1457.3			1440.2

Table 4: Performance analysis for large-scale instances

Instance	Layer 1 gap(min/avg/max)% time(s)		sd(s)	Layer 2 gap(min/avg/max)% time(s)		sd(s)	Layer 3 gap(min/avg/max)% time(s)		sd(s)	Path gap% time(s)
R(1,1,1)	12.72	13.37	14.90	5.80	6.26	6.58	3.87	4.21	4.73	28.44
	0.2		0.03	82.9		24.0	1474.4		27.1	1440.2
R(1,1,2)	16.69	17.22	17.67	7.51	7.95	8.26	6.53	6.75	7.14	26.32
	0.1		0.01	129.2		65.7	1475.1		31.4	1440.8
R(1,2,1)	12.92	13.61	14.63	6.46	6.94	7.48	4.34	4.60	5.08	26.30
	0.1		0.02	49.3		13.9	1456.0		3.5	1440.7
R(1,2,2)	17.17	17.47	17.87	8.45	9.51	11.79	6.86	7.24	7.74	39.17
	0.1		0.02	51.5		20.7	1488.6		18.5	1440.2
R(2,1,1)	14.22	14.62	15.25	6.35	6.88	7.32	4.25	5.13	5.87	22.13
	0.1		0.01	82.7		19.4	1474.4		36.9	1441.0
R(2,1,2)	17.65	18.03	18.33	9.27	9.78	10.13	6.99	7.86	8.49	36.64
	0.1		0.01	91.5		24.2	1488.5		32.4	1440.2
R(2,2,1)	13.81	14.36	14.95	7.05	7.76	8.49	5.33	5.79	6.19	22.95
	0.2		0.05	47.1		16.9	1556.9		73.1	1440.2
R(2,2,2)	17.60	17.97	18.60	9.18	9.60	9.93	8.03	8.73	9.53	28.03
	0.1		0.02	104.2		40.2	1491.0		21.2	1440.2
R(-,-,-)	15.33	15.82	16.51	7.50	8.08	8.74	5.77	6.28	6.84	28.63
	0.1			79.8			1488.1			1440.4

Table 5: Performance analysis for real-life instances

the small-scale ones (for which the MLVNS solutions are always within 0.75% of optimality and 0.28% on average). On real-application instances, CPLEX is not able to identify any solution of interest (with solutions showing more than 20% gap with respect to the best lower bound), while the MLVNS algorithm identifies solutions always within 9.5% of the best lower bound. In particular, for instance R(1,1,1), the actual application obtained from the French mail-order company, the solutions obtained by the MLVNS algorithm are always within 5% of the best lower bound, while CPLEX exhibits a solution with a gap of 28.44%.

We note that the gaps obtained for instances of classes M and L are generally not sensitive to the multiplier values. This is not the case for the real-application instances where the performance worsens when $M_p = 2$, i.e., when the large-size vehicle capacity is increased; it is interesting to note that this tendency is exactly the opposite for class S, as instances in this class with larger vehicle capacity are easier to solve.

Each additional layer included in the MLVNS method proves to be effective in improving the solution. Indeed, when layers 1 and 2 are included, the solution values are around 8% better than when only layer 1 is considered. This percentage increases up to 10% when the three layers are used. It is noteworthy that the results are relatively stable from one run to another, especially when the three layers are performed.

5 Conclusion

In this paper, we have presented a heuristic method for solving a multi-echelon location-distribution problem arising from an actual application in fast delivery service. The heuristic algorithm is based on a variant of the variable neighborhood search (VNS) metaheuristic, which we call the multilayer VNS (MLVNS). Three layers of neighborhood structures are used and ordered in increasing complexity. In addition, each time a move is performed in a neighborhood at layer $l > 1$, this move is completed and evaluated by a recursive call to MLVNS up to layer $l - 1$. The computational results on a large set of instances derived from an actual application show the efficiency of our adaptation of the MLVNS approach to the multi-echelon location-distribution problem.

These results also confirm the quality of the lower bounds obtained by solving the path-based formulation. A promising research avenue is to explore the development of exact algorithms based on decomposition approaches applied to the path-based model. These algorithms would benefit from the integration of the heuristic methods developed in this paper. Other research avenues include the study of extensions to our problem. In particular, both the location-distribution and the routing decisions could be handled simultaneously within a multi-period version of the problem.

Acknowledgments

We gratefully acknowledge financial support for this project provided by NSERC (Canada), by the Nord-Pas-de-Calais Region, the European Community, the Regional Delegation for Research and Technology, the French Ministry of Higher Education and Research, and the National Center for Scientific Research through the International Campus on Safety and Intermodality in Transportation.

References

[1] Aardal, K., Labbé, M., Leung, J.M.Y., Queyranne, M. (1996). On the Two-Level Uncapacitated Facility Location Problem. *INFORMS Journal on Computing* 8, 289-301.

[2] Barros, I., Labbé, M. (1994). A General Model for the Uncapacitated Facility and Depot Location Problem. *Location Science* 2, 173-191.

[3] Barros, I. (1998). Discrete and Fractional Programming Techniques for Location Models. *Combinatorial optimization vol. 3* Kluwer Academic Publishers, Dordrecht, Boston, London.

[4] Barros, A.I., Dekker, R., Scholten, V. (1998). A Two-Level Network for Recycling Sand: A Case Study. *European Journal of Operational Research* 110, 199-214.

[5] Bloemhof-Ruwaard, J.M., Salomon, M., Van Wassenhove, L.N. (1994). On the Coordination of Product and By-Product Flows in Two-Level Distribution Networks: Model Formulations and Solution Procedures. *European Journal of Operational Research* 79, 325-339.

[6] Bloemhof-Ruwaard, J.M., Salomon, M., Van Wassenhove, L.N. (1996). The Capacitated Distribution and Waste Disposal Problem. *European Journal of Operational Research* 88, 490-503.

[7] Chardaire, P., Lutton, J.-L., Sutter, A. (1999). Upper and Lower Bounds for the Two-Level Simple Plant Location Problem. *Annals of Operations Research* 86, 117-140.

[8] Correia, I., Captivo, M.E. (2003). A Lagrangean Heuristic for a Modular Capacitated Location Problem. *Annals of Operations Research* 122, 141-161.

[9] Erlenkotter, D. (1978). A Dual Procedure for Uncapacitated Facility Location. *Operations Research* 26, 992-1009.

[10] Gao, L.-L., Robinson, E.P. (1992). A Dual-Based Optimization Procedure for the Two-Echelon Uncapacitated Facility Location Problem. *Naval Research Logistics* 39, 191-212.

[11] Gendron, B., Semet, F., Strozyk, C. (2002). Adaptive Distribution Systems. *TRANSTECH: Transport Technology Product & Process Innovation Management*, 36-42, Presses Universitaires de Valenciennes.

[12] Gendron, B., Semet, F. (2009). Formulations and Relaxations for a Multi-Echelon Capacitated Location-Distribution Problem. *Computers & Operations Research* 36, 1335-1355.

[13] Ghiani, G., Guerriero, F., Musmanno, R. (2002). The Capacitated Plant Location Problem with Multiple Facilities in the Same Site. *Computers & Operations Research* 29, 1903-1912.

[14] Hansen, P., Mladenovic, N. (2003). Variable Neighborhood Search. *Handbook of Metaheuristics*, International Series in Operations Research and Management Science 57, 145-184.

[15] Kaufman, L., Van Eede, M.V., Hansen, P. (1977). A Plant and Warehouse Location Problem. *Operational Research Quarterly* 28, 547-554.

[16] Klose, A., Drexl, A. (2005). Facility Location Models for Distribution System Design. *European Journal of Operational Research* 162, 4-29.

[17] Marin, A. (2006). Lower Bounds for the Two-Stage Uncapacitated Facility Location Problem. *European Journal of Operational Research*, forthcoming.

[18] Marin, A., Pelegrin, B. (1999). Applying Lagrangian Relaxation to the Resolution of Two-Stage Location Problems. *Annals of Operations Research* 86, 179-198.

[19] Melo, M.T., Nickel, S., Saldanha da Gama, F. (2006). Dynamic Multi-Commodity Capacitated Facility Location: A Mathematical Modeling Framework for Strategic Supply Chain Planning. *Computers & Operations Research* 33, 181-208.

[20] Mitropoulos, P., Giannikos, I., Mitropoulos, I. (2009). Exact and Heuristic Approaches for the Locational Planning of an Integrated Solid Waste Management System. *Operational Research* 9, 329-347.

[21] Mladenovic, N., Hansen, P. (1997). Variable Neighborhood Search. *Computers and Operations Research* 24, 1097-1100.

[22] Narula, S.C., Ogbu,U.I. (1979) An hierarchal location–allocation problem. *Omega* 7, 137-143.

[23] Pirkul, H., Jayaraman, V. (1996). Production, Transportation and Distribution Planning in a Multi-Commodity Tri-Echelon System. *Transportation Science* 30, 291-302.

[24] Pirkul, H., Jayaraman, V. (1998). A Multi-Commodity Multi-Plant Capacitated Facility Location Problem: Formulation and Efficient Heuristic Solution. *Computers & Operations Research* 25, 869-878.

[25] Ro, H.-B., Tcha, D.-W. (1984). A Branch and Bound Algorithm for the Two-Level Uncapacitated Facility Location Problem with Some Side Constraints. *European Journal of Operational Research* 18, 349-358.

[26] Sahin, G., Süral, H. (2007). A Review of Hierarchical Facility Location Models. *Computers & Operations Research* 34, 2310-2331.

[27] Vilcapoma Ignacio, A.A., Martins Ferreira Filho, V.J., Dieguez Galvao, R. (2008). Lower and Upper Bounds for a Two-Level Hierarchical Location Problem in Computer Networks. *Computers & Operations Research* 35, 1982-1998.

[28] Wu, L.Y., Zhang, X.S., Zhang, J.L. (2006). Capacitated Facility Location Problem with General Setup Cost. *Computers & Operations Research* 33 , 1226-1241.