CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

An Adaptive Large Neighborhood Search for the Two-Echelon Multiple-Trip Vehicle Routing Problem with Satellite Synchronization

Philippe Grangier
Michel Gendreau
Fabien Lehuédé
Louis-Martin Rousseau

July 2014

CIRRELT-2014-33

Bureaux de Montréal :
Université de Montréal
Pavillon André-Aisenstadt
C.P. 6128, succursale Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince
2325, de la Terrasse, bureau 2642
Québec (Québec)
Canada G1V 0A6
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

An Adaptive Large Neighborhood Search for the Two-Echelon Multiple-Trip Vehicle Routing Problem with Satellite Synchronization

Philippe Grangier[1,2,*], Michel Gendreau[1], Fabien Lehuédé[2], Louis-Martin Rousseau[1]

[1] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT) and Department of Mathematics and Industrial Engineering, Polytechnique Montréal, P.O. Box 6079, Station Centre-Ville, Montréal, Canada H3C 3A7

[2] École des Mines de Nantes, 4 rue Alfred Kastler, F-44307 Nantes Cedex 3, France

Abstract. The two-echelon vehicle routing problem (2E-VRP) consists in making deliveries to a set of customers using two distinct fleets of vehicles. First-level vehicles pick up requests at a distribution center and bring them to intermediate sites. At these locations, the requests are transferred to second-level vehicles, which deliver them. This paper addresses a variant of the 2E-VRP that integrates constraints arising in City Logistics such as time window constraints, synchronization constraints, and multiple trips at the second level. The corresponding problem is called the two-echelon multiple-trip vehicle routing problem with satellite synchronization (2E-MTVRP-SS). We propose an adaptive large neighborhood search to solve this problem. Custom destruction and repair heuristics and an efficient feasibility check for moves have been designed and evaluated on modified benchmarks for the VRP with time windows.

Keywords. Routing, city logistics, synchronization, adaptive large neighborhood search.

Acknowledgements. This work was partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) through its discovery grants program and by the Fonds de recherche du Québec - Nature et technologies (FRQNT) through its team research program.

---

* Corresponding author: Philippe.Grangier@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2014

1 Introduction

The two-echelon vehicle routing problem (2E-VRP) consists in routing freight from a central depot to customers through a set of intermediate sites. The depot is an intermodal logistics site called the distribution center (DC). It has some storage capacity, and it is where consolidation takes place. Intermediate sites, usually called satellites, have little or no storage capacity but are located closer to customers. Two fleets of vehicles are involved: first-level vehicles carry requests from the DC to the satellites, and second-level vehicles carry requests from the satellites to the customers. First-level vehicles are usually significantly larger than second-level vehicles.

Over the last few years freight transportation in urban areas has received much attention [7]. Indeed, because of increasing traffic congestion, environmental issues, and low average truckloads, new policies (e.g., London Congestion Charges, Monaco UDC) and initiatives (Amsterdam City Cargo) have emerged to ban large trucks from city centers. This movement is known as City Logistics and represents a move from independent direct shipping strategies toward integrated logistics systems. In this context, multi-echelon distribution systems and particularly two-tiered systems are often proposed as an alternative to current distribution systems [8].

Several specific constraints arise in the urban context: time windows, multiple use of vehicles, and synchronization. Delivery hours are often restricted because of customer requirements or city regulations. Moreover, second-level vehicles are usually small in order to access every street, so even a full load does not represent an entire work-day. Finally, operating a satellite in a city is expensive, because of labor costs and high rent. More and more cities are allowing transporters to use dedicated or existing infrastructure (reserved parking spaces, bus depots) to unload [7]. No storage capacity is normally available at these locations, thus requiring a synchronization of the two levels.

The contribution of this paper is a solution methodology for a 2E-VRP that integrates constraints that have not yet been addressed in the literature: time windows, synchronization, and multiple trips. Similar problems have been discussed in [19] under the name *two-echelon vehicle routing problem with satellite synchronization* (2E-VRP-SS) and modeled in [8] under the name *two-echelon, synchronized, scheduled, multidepot, multiple-tour, heterogeneous VRPTW* (2SS-MDMT-VRPTW). However, to the best of our knowledge, no implementation has been reported.

Related work includes models for City Logistics, multi-echelon vehicle routing problems with multiple routes and transfer or synchronization constraints. A general model for City Logistics systems is presented by Crainic et al. in [8], while Mancini focuses on multi-echelon systems [13]. The 2E-VRP was introduced by Gonzalez-Feliu and Perboli [10], who proposed a mathematical model. Since then several algorithms have been developed: math-based heuristics [19, 18], clustering-based heuristics [3], GRASP [5, 4, 30], and adaptive large neighborhood search (ALNS) [11]. Exact methods include [12, 22, 24, 25]. Crainic et al. [6] study the impact of satellite location on the cost of a 2E-VRP solution compared to that of a VRP. A similar problem is the two-echelon location routing problem (2E-LRP) [17]. Our problem also integrates some multiple-trip aspects [29] that have been solved with tabu search [16] and ALNS [1]. See [9] for a detailed survey of synchronization in vehicle routing problems. Recent papers on vehicle routing with transfers include [14, 15, 21] for the pickup and delivery problem with transfers.

The remainder of this paper is organized as follows. Section 2 presents a formulation of the problem, and Sections 3 and 4 are devoted to the solution method with a special focus on efficiently solving the timing subproblem. Computational results are presented in Section 5.

2 Problem formulation

In this section we define the problem and discuss the synchronization model at satellites. The complete model is presented in A.

2.1 Problem statement

We introduce the two-echelon multiple-trip vehicle routing problem with satellite synchronization (2E-MTVRP-SS). We consider a city distribution center (CDC), a set of satellites V_s, a set of requests R, and two homogenous fleets of vehicles K_1 and K_2 of capacity q_1 and q_2, based at o_1 and o_2. Each request r is located at the CDC at the beginning of the time horizon and must be delivered within the time window $[e_r, l_r]$ to a customer denoted by d_r (the set of customers is denoted by V_c). The quantity associated with r is q_r. No direct shipping from the CDC is allowed. Second-level vehicles can perform multiple trips, which may start at different satellites. Satellites have no storage capacity, thus requiring an exact synchronization between the vehicles of the two levels.

The 2E-MTVRP-SS is defined on a directed graph $G = (V, A)$, which reflects the two-level system. The first level is defined by $G_1 = (V_1, A_1)$ with $V_1 = \{o_1\} \cup \{CDC\} \cup V_s$ and $A_1 = \{(o_1, CDC)\} \cup \{(CDC, i)|i \in V_s\} \cup \{(i, j)|i, j \in V_s\} \cup \{(i, o_1)|i \in V_s\}$. The second level is defined by $G_2 = (V_2, A_2)$ with $V_2 = \{o_2\} \cup V_c \cup V_s$ and $A_2 = \{(o_2, i)|i \in V_s\} \cup \{(i, j)|i \in V_s, j \in V_c\} \cup \{(i, j)|i, j \in V_c\} \cup \{(i, j)|i \in V_c, j \in V_s\} \cup \{(i, o_2)|i \in V_c\}$. With each arc $(i, j) \in A = A_1 \cup A_2$ is associated a travel time $t_{i,j}$ and a travel cost $c_{i,j}$. Each node i has a known service time s_i. Solving the 2E-MTVRP-SS involves finding $|K_1|$ first-level routes and $|K_2|$ second-level routes, and a schedule for them, such that the capacity and time-related constraints are satisfied.

2.2 Transfer and synchronization at satellites

We define a transfer as the operation during which a first-level vehicle transmits one or more requests to a second-level vehicle at a satellite. Given the time windows on the customer requests and the lack of storage capacity at the satellites, the two vehicles must be at the satellite at the same time. Thus, the first and second levels must be synchronized. Figure 1 illustrates the temporal aspects of a transfer.

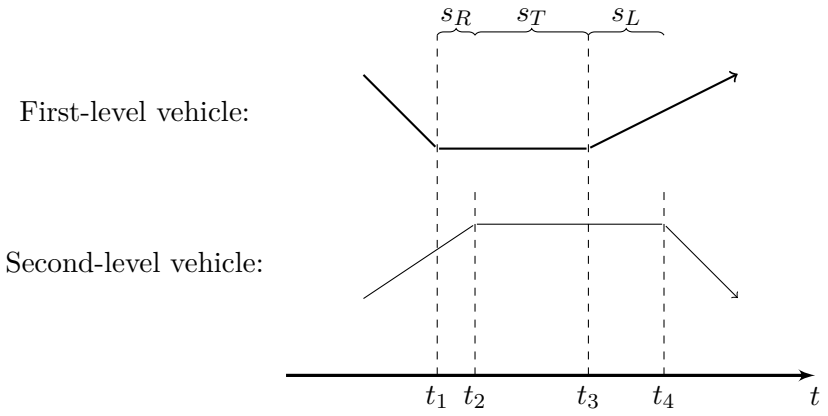


Figure 1: Time chart for a transfer

s_R $[t_1, t_2]$: The first-level vehicle arrives at t_1 and gets ready to transfer. The second-level vehicle is not involved.

s_T $[t_2, t_3]$: Time in common for the vehicles of both levels. For a transfer to occur, the two vehicles should spend at least this time together at the satellite.

s_L $[t_3, t_4]$: The first-level vehicle is not involved. The second-level vehicle loads the requests that were transferred. The time does not depend on the transferred quantity.

If $t_2 \geq t_1 + s_R$, the first-level vehicle must wait. Conversely, if $t_2 \leq t_1 + s_R$ the second-level vehicle must wait for the first-level vehicle. If the first-level vehicle transfers requests to several second-level vehicles, it cannot leave before $\max_{i \in K_2} t_i + s_T$. s_R, s_T, and s_L can be integrated into the travel time from and to the satellites, to simplify the model. Thus, we later consider that all the transfer-related periods (s_R, s_T, s_L) are equal to zero. If a second-level vehicle returns several times to pick up requests from the same first-level vehicle at the same satellite, each visit corresponds to a different transfer.

3 An ALNS for the 2E-MTVRP-SS

In this section we describe the destruction and repair methods used in our ALNS for the 2E-MTVRP-SS. ALNS was created by Ropke and Pisinger [23] as an extension of the large neighborhood search introduced by Shaw [27], which iteratively destroys and repairs the current solution using heuristics. ALNS adds an adaptive layer that chooses among several heuristics based on scores that depend on past successes. ALNS has been widely used to solve vehicle routing problems (see for example [20]); see [23, 20] for an extensive description of the method. In the following, we focus on the components of our method.

3.1 Destruction methods

When partially destroying a solution we select a method and a percentage p of the total number of requests to remove. Unless stated otherwise, this method is reused until p is reached. Following Azi et al. [1], we use three levels of destruction methods: workday, route, and customer.

3.1.1 Workday level

The following operators are used for first- and second-level vehicles.
Random Vehicle Removal: We randomly remove a vehicle.
Least Used Vehicle Removal: We remove the vehicle with the smallest load. For the second level, the total load of a vehicle is defined as the sum of the load of each trip.

3.1.2 Route level

Random Trip Removal: We randomly remove a trip from the solution.
First-level Stop Removal: We randomly remove a first-level stop from the solution. The trips that contain this stop are removed.
Trip Related Removal: This method is similar to that of Azi et al. [1]. Trips are removed based on a proximity measure: we start by randomly selecting a trip and removing it. We then find the trip that contains the nearest customer to any customer in the trip just removed, and we remove that trip.
Synchronization-Based Trip Removal: Intuitively, a *good* synchronization occurs when the vehicles involved arrive at approximately the same time. If a second-level vehicle arrives a long time before (or after) the first-level vehicle there will be a long waiting time; this should be avoided. This method

removes the trip for which the time between the arrival of the second-level vehicle and the arrival of the first-level vehicle is maximum.

3.1.3 Customer level

Random Customer Removal: We randomly remove a customer.
Related Removal Heuristics: These methods aim to remove related requests. Let the relatedness of requests i and j be $R(i, j)$. We use two distinct relatedness measures: distance and time. The distance measure is the distance between the delivery points of i and j. The time measure is the sum of the absolute gap between their earliest delivery times and the absolute gap between their latest delivery times. Each measure is normalized by dividing it by the longest distance (resp. travel time) between two customers. In both cases a lower $R(i, j)$ value indicates a great degree of relatedness.

We ran preliminary tests to compare these two measures with that of Shaw [27], which groups time and distance into a single measure. The methods give similar results, but we have chosen time and distance because they do not require parameter tuning.
History-Based Removal: This is inspired by [14] and removes requests that seem poorly placed in the current solution with regard to the best-known solutions. For requests r and r', let $\xi_{r,r'}$ be the number of solutions among the 50 best-known in which r' is a direct successor of r. For request r and satellite s, let $\chi_{s,r}$ be the number of solutions in which r is delivered via a transfer at s. For each request r, delivered in the current solution via a transfer at s, we define a score ϕ as follows:

$$\phi_r = \xi_{\delta^-(r),r} + \xi_{r,\delta^+(r)} + \chi_{s,r}.$$

Then we remove the p requests with the lowest scores.

3.2 Repair methods

In this section we describe the methods used to repair a solution. We first describe the three different ways to insert a given customer into a given second-level route, and then we describe the repair methods.

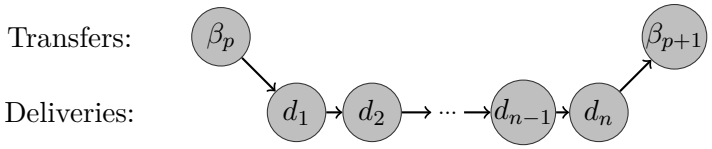
3.2.1 Three insertion operators

In the VRP, the insertion of a customer c into a partially built solution is fully described by giving the route and the position for the insertion. Thus, all possible insertions can be described by the unique set $\{(v, p) : \ v \in \text{Vehicles}, \ 0 \leq p \leq |route(v)| + 1\}$. Given the multiple-trip and two-echelon characteristics of the 2E-MTVRP-SS, there are three distinct ways of inserting a customer into a solution. We call them *insertion operators* and describe them below.

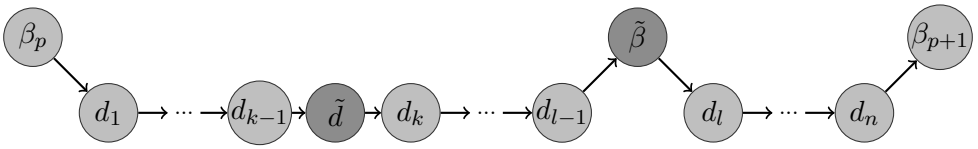
Insertion into an existing trip: The customer is inserted into an existing trip.

Insertion by creation of a new trip: A new trip is created for the customer. This new trip can be connected either to an existing stop or to a newly created stop of a first-level vehicle.

Insertion by trip split: Before inserting c into trip t, we split t into two trips, t_1 and t_2. Trip t_1 is still connected to the same first-level stop as t, but t_2 is connected either to an existing stop or to a newly created stop of a first-level vehicle. This vehicle can be different from that involved in t_1. Then c is inserted into one of the two resulting trips. Figure 2 illustrates the use of a split trip operator.



(a) Example of a second-level trip; β is a transfer and d_i is a delivery point.



(b) Insertion into the first trip

Figure 2: Example of the use of a trip split operator with insertion of the request into the first resulting trip.

3.2.2 Reducing the size of the neighborhoods

The insertion of a request corresponds to two decisions: one at the first level and one at the second level. Thus, the neighborhoods generated by the insertion operators are huge. For example, to test every possible insertion of a request r with the *trip split* operator, we have to test for each pair (*insertion position*, *split position*) in each trip of second-level vehicles, each existing stop of first-level vehicles, and each creation of a stop (i.e., each satellite at each position). To ensure a reasonable runtime, we have created restricted neighborhoods.

For the *trip split* we have introduced two variants: *existing stops* and *customer first*. In the *existing stops* variant, we try to connect t_2 with an existing stop of a first-level vehicle. In the *customer first* variant, we first select the insertion position that leads to the smallest increase in the cost of the global solution. Then we select the best possible way to split t into feasible trips t_1 and t_2, trying both existing and newly created first-level stops.

When we create a new first-level stop to be connected to a second-level trip, it is likely that the best choice for the satellite will be close to the second vehicle. At the beginning of our algorithm we sort the satellites in order of distance for every pair of customers. When creating a new trip connected to a new first-level stop, we consider only the s satellites closest to the pair (predecessor of the trip, first customer in the trip). This is used for the *trip creation* and the variants of the *trip split* operator.

As shown in Section 5.3.4, using these restricted neighborhoods makes our algorithm about 3.3 times faster while maintaining the quality of the solution.

3.2.3 Repair methods

All the unplanned requests are stored in a request bank.

Best insertion: From the requests in the request bank, we insert the one with the cheapest insertion cost considering all possible insertion operators.

K-Regret: For each request in the request bank, let δ_r^i represent the gap between the insertion of r at its best position in its best trip and the insertion at its best position in its i^{th} best trip. We select the request where $\sum_{i=2}^{k} \delta_r^i$ is maximum. In other words, we maximize the sum of the differences of the cost of inserting request r into its best trip and its i^{th} best trip. To control the computational time, we use small values of k (generally less than 5).

3.2.4 First-level routes

Inserting a request r by moving a first-level vehicle to a new satellite generates a large increase in the routing cost compared to transferring the request at a transfer point. Thus, it is rare for repair methods to choose such insertions. However, subsequent insertions may benefit from a new transfer point, because the second-level vehicle may have a smaller distance to travel. Therefore, when an insertion operator creates a new stop, we consider the following *biased* cost:

$$\text{Biased cost} = \text{second-level insertion cost} + \text{first-level insertion cost} \times \max\left(\alpha, \frac{\text{load in second-level trip}}{\text{second-level vehicle capacity}}\right). \quad (1)$$

In this biased cost, we acknowledge that if there is some room in the second-level trip, then it is likely that we will later use it for a customer. For our instances, after some tuning, we have used $\alpha = 0.7$ for the *trip creation* operator and $\alpha = 0$ for the *trip split* operator and its variants.

3.3 Initial solution

We designed a two-phase constructive algorithm to obtain the initial solution. First, we schedule the second level using a best insertion algorithm for a multiple-trip multiple-depot problem. Then we create the first-level routing that corresponds to visits to the satellites created by the second-level.

As shown in the experiments (see Section 5.3.1), although this method creates initial solutions with more vehicles than those obtained by applying a best insertion at both levels, it is a better starting point for the fleet reduction because on average we are able to find better solutions later in the search process.

4 Route scheduling and feasibility algorithm

In this section we describe an efficient way to schedule the routes of a given solution and to test if an insertion is feasible with respect to the time and precedence constraints. We use the notation in Table 1.

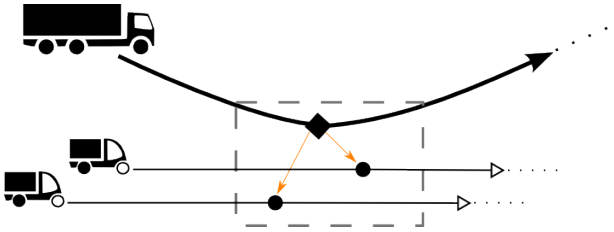
Notation	Definition
$\psi_{u,v}$	ordered set of vertices on path $(u, \ldots, v)$
$\delta^+(i)$	direct successors (resp. predecessors) of a vertex i
$\Gamma^+(i)$	set of all successors of i
$\Omega_{u,v}$	set of paths from u to v

Table 1: Notation in the temporal graph

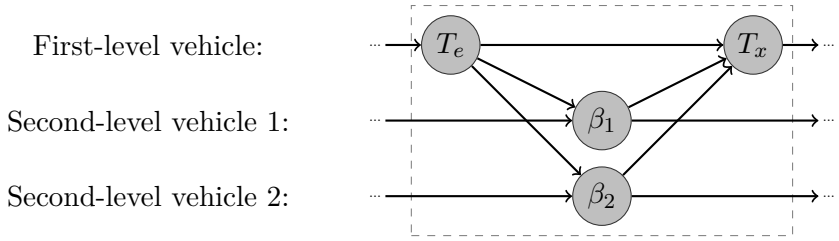
4.1 Modeling of precedence and synchronization constraints

Given the routes in a solution of the 2E-MTVRP-SS, we can represent the time constraints as a directed acyclic graph, G_t. This precedence graph is built as follows: for every operation except transfers, we create a node and add an arc to each of its successors in the given route. Each arc (u, v) has a weight that corresponds to $t_{u,v}$. For a transfer we create three nodes: a transfer entrance node T_e, a transfer exit node T_x for the first-level vehicle, and a pick-up node β for the second-level vehicle.

We create three arcs (T_e, T_x), (T_e, β), (β, T_x) with weight 0. If the first-level vehicle transfers its load to several second-level vehicles, we create only one pair (T_e, T_x). We assume that transfers to second-level vehicles can occur simultaneously. Figure 3 illustrates this transformation.



(a) First-level stop with two transfers



(b) Precedence graph representation.

Figure 3: A first-level stop with two transfers and its precedence graph representation.

4.2 Route scheduling

G_t corresponds to a PERT chart. As mentioned in [2] (p. 657), scheduling tasks in such a diagram can be performed using a shortest-path algorithm, with linear complexity. Furthermore, in an as-early-as-possible schedule, after a change, only the downstream operations have to be rescheduled, thus reducing the number of modifications to be performed.

4.3 Efficient feasibility testing for insertion of requests

When evaluating an insertion of an unplanned request into a feasible solution, we need to ensure that the solution will remain feasible after the insertion. With the insertion operators of Section 3.2.1, infeasibility can occur in three ways: by exceeding the capacity of a vehicle, violating a time window, or creating a cycle of precedence constraints. In the following section we will discuss our method to efficiently check if an insertion violates a time window or creates a cycle in the precedence graph.

4.3.1 Extension of forward time slacks [26]

Inserting a request into a feasible solution of a VRP may postpone several deliveries, thus leading to the violation of a time window. To avoid rescheduling a route for each insertion attempt, which has linear complexity, Savelsbergh introduced a method [26] that checks in constant time if an insertion generates a time-window violation. It is based on the computation of so-called *forward time slacks* (FTSs), which correspond to the maximum possible forward shift of an operation that does not lead to a time-window violation later in the route.

Let the total waiting time on path $(u, \ldots, v)$ be $TWT_{\psi_{u,v}} = \sum_{i \in \psi_{\delta^+(u),v}} w_i$. The FTS at node u is

$$F_u = \min_{i \in \{u\} \cup \Gamma^+(u)} \{TWT_{\psi_{u,i}} + l_i - h_i\}.$$

As pointed out by Drexl [9], there is an *interdependence problem* relating to synchronization. A change in one route may have effects on other routes, potentially making them infeasible. For example, in the pickup and delivery problem with transfers (PDPT), if a vehicle A has to collect a request at a transfer point, then a vehicle B collected the request at its pickup location. Thus, a delay of B will affect A. Masson et al. [15] extended FTS to the PDPT. We present below their main results.

First, the notion of *slack time* is introduced. It is a generalization of the total waiting time between two nodes:

$$ST_{u,v} = \min_{w \in \Omega_{u,v}} TWT_w.$$

Then in the case of the PDPT, the FTS at node u becomes:

$$F_u = \min_{i \in \{u\} \cup \Gamma^+(u)} \{ST_{u,i} + l_i - h_i\}.$$

Masson et al. proved the above result for a precedence graph modeling a PDPT. These results are valid for any precedence graph.

The constant-time feasibility tests provided by FTS come at the expense of recomputing FTS every time an insertion is performed. However, since a local-search-based algorithm usually tests many more insertions than it actually performs, this trade-off is efficient. In contrast to Masson et al., we recompute FTS using the Floyd–Warshall algorithm ([2], p. 693), which is faster in our case than the suggested shortest path method.

4.3.2 Feasibility check for precedence constraints

Some insertion operators create new transfers, which may lead to infeasible precedence relations (see Figure 4 for an illustration). We extend the method of Masson et al. [15] for detecting cycles in constant time.

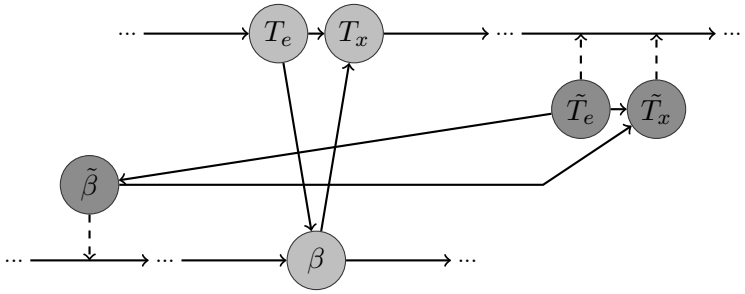


Figure 4: Infeasible insertion: it creates a cycle in the precedence graph.

Proposition 1. *Synchronizing a first-level stop T and a second-level trip ($\beta \to d_1 \ldots d_n$) creates a cycle in the precedence graph if and only if*

$$T_e \in \Gamma^+(\beta) \text{ or } \beta \in \Gamma^+(T_x).$$

Proof. $\Rightarrow$ With the synchronization, only two arcs are created $(T_e \to \beta)$ and $(\beta \to T_x)$. One of them is responsible for the cycle. If it is $(T_e \to \beta)$, previously $T_e \in \Gamma^+(\beta)$. If it is $(\beta \to T_x)$, previously $\beta \in \Gamma^+(T_x)$.
$\Leftarrow$ If $T_e \in \Gamma^+(\beta)$, since $\beta \in \delta^+(T_e)$ by definition, a cycle is created. If $\beta \in \Gamma^+(T_x)$, since $\beta \in \Gamma^+(T_x)$ by definition, a cycle is created. □

Corollary 1. *Synchronizing a new first-level stop inserted after node i and a new second-level trip inserted after node j creates a cycle in G_t if and only if*

$$i \in \Gamma^+(j) \text{ or } j \in \Gamma^+(i).$$

By maintaining a list of successors for each point in the precedence graph, we can check if an insertion violates a precedence constraint.

4.3.3 Efficiency of the method

We compare the running time of the FTS extension versus a check based on an incremental PERT: on average, performing 25,000 iterations of the ALNS in the cost optimization phase is approximately 12 times faster with the FTS extension than with the PERT (see Section 5.3.2).

5 Computational experiments

In this section we first describe the adaptation of some well-known VRPTW instances to the 2E-MTVRP-SS. The parameter configuration is discussed in Section 5.2. In Section 5.3 we show that our custom heuristics are efficient, and we present our results.

5.1 Instances

Since the 2E-MTVRP-SS is a new problem, there are no instances for it. We adapt the well-known Solomon's instances for the VRPTW [28]. We used a subset of these instances to tune our algorithm: the first two of every type, for a total of 12 tuning instances.

5.1.1 Geographical configuration

The customer requests are unchanged. The depot (node 0) is the base of second-level vehicles; the first-level vehicles are based at the CDC.
We adopt the following $X/Y/M/N$ notation to describe the position of the CDC and the satellites. X and Y give the position of the CDC expressed as a percentage of the size of the map. M and N describe the number of rows (resp. columns) of a grid. We locate a satellite at each exterior intersection of the grid. Figure 5 illustrates a -50 / 50 / 3 / 3 configuration.

According to [6], the maximum benefit of the 2E-VRP compared to the VRP occurs when the CDC is *external* (outside the customer's zone), thus saving travel from the depot to the customers and back, and when the satellites are between the CDC and the customers. The appropriate satellite number is between 7 and 10 for instances with between 100 and 200 customers. In all the benchmarks we use a 50 / 150 / 3 / 3 configuration.
Because the CDC is situated farther from the customers than the depot, some orders may be impossible to deliver. We therefore add an offset δ to each time window, with $\delta = \lceil t_{CDC,Depot} \rceil$. Hence, a time window $[e_i, l_i]$ in a Solomon instance becomes $[e_i + \delta, l_i + \delta]$.

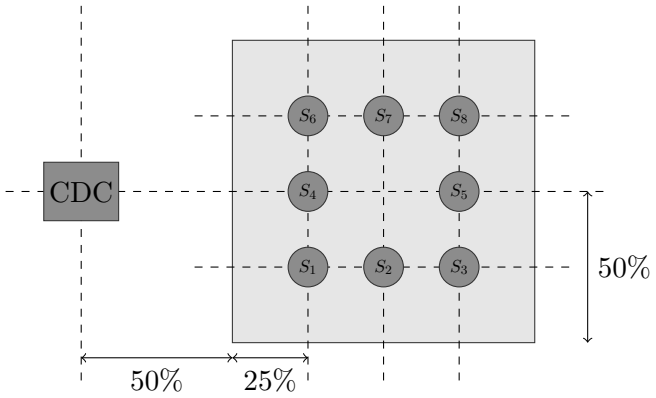


Figure 5: Geometrical layout -50 / 50 / 3 / 3.

5.1.2 Vehicle configuration

According to Savelsbergh, the instances labeled with a 1 (R1, C1, RC1) have short scheduling horizons, whereas instances labeled with a 2 have long scheduling horizons. Thus, the former are more time-constrained, and the latter are more capacity-constrained. To preserve this idea we use the following ratios for first-level vehicles/second-level vehicles: 4/0.5 ratio for instances 1; 2/0.25 ratio for instances 2.

5.1.3 Objective function

The number of vehicles is not fixed, so we consider the following hierarchical objective: (1) number of first-level vehicles, (2) number of second-level vehicles, (3) routing cost.

5.2 Parameter configuration

In this section we discuss the parameters used in our algorithm.

5.2.1 Parameters for fleet optimization phase

We use a sequential optimization scheme to reduce the number of vehicles. We start by reducing the number of first-level vehicles and then we reduce the number of second-level vehicles. When we find a feasible solution with $n+1$ vehicles, we start looking for a feasible solution with n vehicles by calling the least-used-vehicle removal heuristic. $LB_1 = \lceil \sum_{r \in R} d_r / q_1 \rceil$ is a lower bound on the number of first-level vehicles. If we reach it, we switch to the reduction of the number of second-level vehicles. Overall, we perform 25,000 iterations with no more than half of them dedicated to the reduction of the first-level fleet. We use an LNS (without the adaptive layer) in this phase.

5.2.2 Parameters for cost optimization phase

In the cost optimization phase we use the following parameters (w, c, σ_1, σ_2, σ_3, r, ρ_{min}, ρ_{max})=(0.05, 0.99975, 33, 9, 13, 0.1, 10%, 40%). The notation is that of Ropke and Pisinger [23]. We again perform 25,000 iterations, since this gives a compromise between runtime and solution quality.

5.2.3 Heuristics

For both the fleet optimization phase and the cost reduction phase, we use the following heuristics.
Destruction heuristics: random vehicle removal, least-used-vehicle removal, random trip removal, first-level stop removal, trip-related removal, synchronization-based trip removal, random customer removal, distance-related removal, time-related removal, history-based removal.
Repair heuristics: best insertion, 3-Regret, 4-Regret, 5-Regret. Each was used in three variants: without split, existing stops, and customer first (see Section 3.2.2).

5.3 Results

The algorithm was coded in C++, and the experiments were conducted using a single core of an Intel Xeon X5675 @ 3.07 GHz under Linux. We report figures for a subset of 12 instances, and the best and average solutions found for all instances.

5.3.1 Impact of two-phase initialization

For the initial solution, using the two-phase best insertion creates more vehicles than applying a best insertion at both levels. However, as shown in Table 2, after the fleet optimization phase, it gives better results. We therefore use it as a starting point.

	Average initial solution				Average solution after fleet opt.				Best solution after fleet opt.			
	Best Insertion		Two Phase		Best Insertion		Two Phase		Best Insertion		Two Phase	
Instance	FL	SL	FL	SL	FL	SL	FL	SL	FL	SL	FL	SL
c101	3.0	13.0	3.16	13.0	3.0	11.32	*3.0*	*11.28*	**3**	**11**	**3**	**11**
c102	3.0	13.0	3.04	12.0	**3.0**	**10.0**	**3.0**	**10.0**	**3**	**10**	**3**	**10**
c201	2.0	4.0	2.0	4.0	2.0	4.0	*2.0*	*3.72*	2	4	**2**	**3**
c202	2.0	4.0	2.0	4.0	2.0	3.24	*2.0*	*3.04*	**2**	**3**	**2**	**3**
r101	2.0	22.0	2.84	22.0	2.0	19.72	*2.0*	*19.64*	**2**	**19**	**2**	**19**
r102	2.0	21.0	2.88	20.0	2.0	18.08	**2.0**	**18.00**	**2**	**18**	**2**	**18**
r201	1.0	5.0	2.0	5.0	**1.0**	**4.0**	**1.0**	**4.0**	**1**	**4**	**1**	**4**
r202	1.0	5.0	2.43	5.0	*1.0*	*3.88*	1.0	3.90	**1**	**3**	**1**	**3**
rc101	3.0	21.0	4.28	20.0	*3.0*	*16.48*	3.0	16.56	**3**	**16**	**3**	**16**
rc102	3.0	18.0	5.24	17.0	3.0	14.4	*3.0*	*14.2*	**3**	**14**	**3**	**14**
rc201	1.0	6.0	3.0	5.0	**1.0**	**4.0**	**1.0**	**4.0**	**1**	**4**	**1**	**4**
rc202	1.0	5.0	2.0	5.0	**1.0**	**4.0**	**1.0**	**4.0**	**1**	**4**	**1**	**4**

Table 2: Comparison of the results of the fleet optimization phase, using best insertion or two-phase best insertion as initialization methods. The FL (resp. SL) columns indicate the number of first-level (resp. second-level) vehicles. Bold figures indicate best known solution; italics indicate best average solution.

5.3.2 Time spent in each part of algorithm

On average the repair heuristics are far more time consuming than the destruction heuristics (96.1% versus 1.1% of the total runtime). This is because our destruction methods reschedule the entire solution only when p customers have been removed from the current solution, and not after each removal. This reduces the computational effort necessary.

5.3.3 Impact of FTS on reduction of runtime

Using the extended FTS instead of an incremental PERT to check the feasibility of an insertion is key to reducing the computational effort, as shown in Table 3: with FTS our algorithm is 91.9% faster. Moreover, this gain is almost independent of the instance. In the cost optimization phase, about 16.6% of the runtime is spent updating the FTSs, while 21.1% of the runtime is spent checking if an insertion is feasible timewise.

Instance	c101	c102	c201	c202	r101	r102	r201	r202	rc101	rc102	rc201	rc202
PERT (in min)	529.2	527.1	768.1	912.0	330.9	270.4	499.0	937.5	403.0	522.0	687.8	1599.0
FTS (in min)	**42.4**	**56.2**	**55.4**	**77.9**	**26.0**	**32.8**	**42.6**	**72.9**	**26.2**	**42.8**	**50.6**	**63.5**
Difference (in %)	-92.0	-89.3	-92.8	-91.5	-92.1	-87.7	-91.5	-92.2	-93.5	-91.8	-92.6	-96.0

Table 3: Computational time for incremental PERT versus FTS for 25,000 iterations in the cost optimization phase

5.3.4 Impact of reduced neighborhoods on solution quality and runtime

Table 4 shows the impact of the reduced neighborhoods on the solution quality and the runtime. We observe that using these neighborhoods has a limited impact on the solution quality (less than 0.20%), but the runtimes are significantly reduced (by 70%). The runtime reduction is larger for type-2 instances, which are capacity constrained, than for type-1 instances, which are time constrained. This is expected, because the tighter the time constraint the smaller the number of reachable satellites. Based on these results we have chosen the following configuration (neighborhood 4 in Table 4): we use *customer first* and *existing stops* instead of the full *trip split* operator, and we limit the number of satellites explored to three when creating a new stop for a first-level vehicle.

	Neighborhood 1		Neighborhood 2		Neighborhood 3		Neighborhood 4		Neighborhood 5	
Instance	avg. cost	avg. time	avg. cost	avg. time	avg. cost	avg. time	avg. cost	avg. time	avg. cost	avg. time
c101	2049.70	82.6	**2045.41**	64.2	2057.80	39.4	2056.84	41.0	2059.46	49.2
c102	1976.38	255.6	1981.15	87.2	1978.91	48.3	1976.26	54.8	**1974.45**	65.8
c201	1415.90	273.8	1416.55	78.9	**1413.75**	50.8	1414.83	55.6	1416.24	66.9
c202	**1331.92**	692.7	1351.76	103.0	1340.33	70.0	1335.16	77.3	1340.37	87.3
r101	2354.25	38.6	**2342.03**	36.8	2348.75	22.2	2347.57	23.5	2351.79	29.6
r102	**2150.24**	82.9	2156.62	47.7	2154.83	27.8	2157.93	32.3	2158.27	39.6
r201	1608.31	196.6	1620.73	54.0	1611.62	37.9	**1606.58**	42.7	1612.29	48.0
r202	1612.91	697.6	1614.86	94.6	**1598.76**	68.1	1611.59	73.0	1598.96	79.8
rc101	2625.63	46.7	**2607.49**	40.6	2626.38	24.5	2613.35	26.2	2641.05	31.8
rc102	2474.25	130.3	2499.48	61.6	2467.24	35.4	**2461.66**	38.7	2497.12	48.6
rc201	1827.67	221.9	1827.07	66.9	1823.43	42.3	**1819.67**	45.9	1830.01	56.1
rc202	1882.10	991.7	**1869.90**	127.7	1903.28	95.1	1890.92	99.2	1893.47	120.2
Dev. from Neigh. 1	-	-	0.16%	-56.1%	0.08%	-73.0%	-0.05%	-70.8%	0.24%	-64.5%

Table 4: Comparison of the average cost and runtime over 10 runs for 5 different neighborhood configurations. Configuration 1 uses the entire *split trip* neighborhood. In configuration 2 the *split trip* operator is replaced by its variants *customer first* and *existing stops*. In configurations 3 to 5 the number of satellites explored when creating a new first-level stop is limited to s, with $s = 2$ in neighborhood 3, $s = 3$ in neighborhood 4, and $s = 4$ in neighborhood 5. Bold figures indicate best values.

5.3.5 Impact of custom destruction and repair methods on cost optimization phase

We now focus on the contribution of our new heuristics to the solution of the 2E-MTVRP-SS. Figure 6 shows the evolution of the solution cost in the cost optimization phase for four different configurations: with every heuristic, without the synchronization-based removal heuristic, without the split-recreate heuristics, and without biased cost (see Section 3.2.4). Not using synchronization-based removal leads to solutions that are 0.74% more expensive on average (0.62% for the best known solution). Not using split-recreate heuristics is 0.42% more expensive on average (0.65% for the best known solution). Not using biased cost is 0.19% more expensive on average (0.42% for the best known solution). Considering how hard it is to move within the space of feasible solutions because of the interdependence problem, these figures, although small, show that the custom heuristics are effective.

5.3.6 Solutions

In Table 5, we report the best and average results found based on 10 runs for each instance. In the fleet optimization phase, we observe that the number of first-level vehicles is always equal to $LB1$

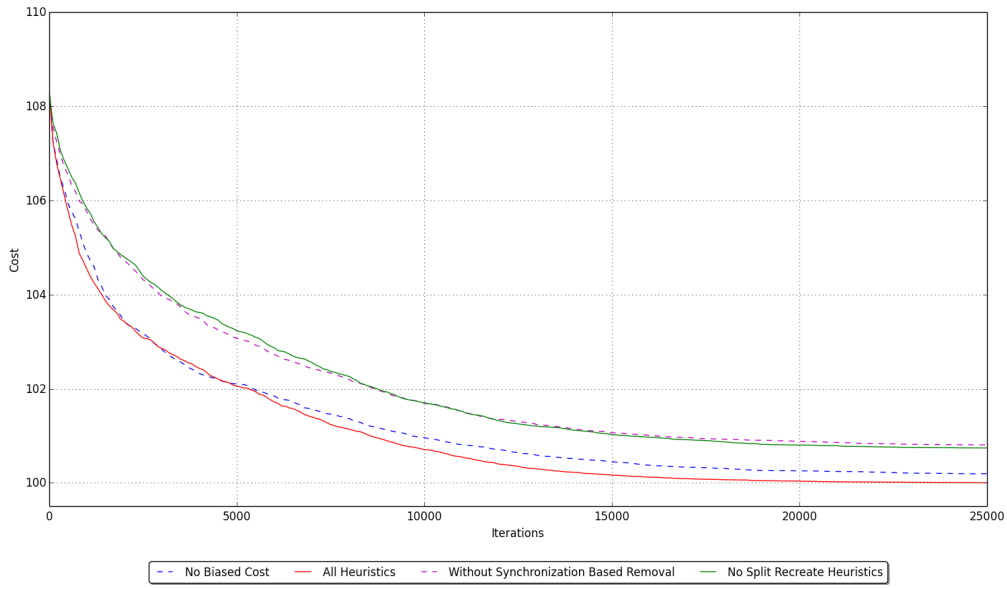


Figure 6: Comparison of the evolution of the average solution quality for 4 different ALNS configurations; 10 runs were performed for each instance. The results have been normalized, with 100 representing the final cost of the best configuration in each case.

and thus optimal. In the cost optimization phase, there is an average gap of 3.75% between the best solution and the average results. In Table 6, we evaluate the impact of the instance characteristics on the routing network used by the vehicles. We notice that there are on average more trips in second-level routes for type-2 instances (capacity-constrained), and that more satellites are used in instances with clustering.

6 Conclusion

We have presented an extension of the well-known 2E-VRP that takes into account constraints arising in City Logistics (time window constraints, synchronization constraints, and multiple trips for some vehicles). We have developed an ALNS to solve the 2E-MTVRP-SS, which has both custom destruction and repair heuristics and an efficient way to check if an insertion is feasible. These contributions help to find good solutions in a reasonable time, thus making it possible to consider this algorithm for other vehicle routing problems with synchronization constraints.

Appendix A Mathematical formulation

We present a mixed integer linear programming formulation for the 2E-MTVRP-SS. In the model, to represent the transfer of one request at one satellite, for each request r and satellite s we create a node $v_{s,r}$, and we denote by $\tilde{V}_s = \{v_{i,r} | v_i \in V_s, r \in R\}$ all the satellites. For each vehicle k we create a start node o_k and an end node o'_k ($\tilde{O}_k = \cup_k o_k$, $\tilde{O}'_k = \cup_k o'_k$).

The mathematical formulation is defined on a graph $G^{\text{math}} = (V^{\text{math}}, A^{\text{math}})$. The first level is $G_1^{\text{math}} = (V_1^{\text{math}}, A_1^{\text{math}})$ with $V_1^{\text{math}} = \tilde{O}_1 \cup \tilde{O}'_1 \cup \tilde{V}_s$ and $A_1^{\text{math}} = \{(o,i) | o \in \tilde{O}_1, i \in \tilde{V}_s\} \cup$

	Average					Best Known Solution				
Instance	FL	SL	T1 (min)	Cost	T2 (min)	FL	SL	T1 (min)	Cost	T2 (min)
c101	**3.0**	11.7	26.2	2035.3	42.4	**3**	11	27.1	2013.7	44.2
c102	**3.0**	*10.0*	36.8	1978.4	56.2	**3**	*10*	32.7	1947.3	63.3
c103	**3.0**	*9.0*	50.6	1942.7	70.4	**3**	*9*	46.1	1893.5	71.9
c104	**3.0**	*9.0*	57.9	1848.1	77.8	**3**	*9*	62.4	1812.8	73.0
c105	**3.0**	10.9	28.6	1945.3	46.6	**3**	10	32.9	1934.0	47.4
c106	**3.0**	10.8	30.2	1987.7	52.0	**3**	10	33.4	1968.7	53.1
c107	**3.0**	*10.0*	31.9	1905.4	49.4	**3**	*10*	27.9	1898.5	41.6
c108	**3.0**	*10.0*	37.0	1907.6	48.8	**3**	*10*	41.3	1888.7	48.9
c109	**3.0**	9.1	43.9	1961.7	64.1	**3**	9	40.4	1920.5	59.9
c201	**2.0**	3.5	27.8	1417.7	55.4	**2**	3	19.4	1417.5	50.8
c202	**2.0**	*3.0*	35.9	1311.3	77.9	**2**	*3*	40.9	1305.0	77.9
c203	**2.0**	*3.0*	49.9	1280.2	98.0	**2**	*3*	56.8	1272.7	86.1
c204	**2.0**	*3.0*	56.7	1270.6	106.1	**2**	*3*	47.2	1238.2	101.2
c205	**2.0**	*3.0*	27.7	1318.9	50.8	**2**	*3*	28.3	1312.1	50.0
c206	**2.0**	*3.0*	33.9	1304.3	58.8	**2**	*3*	29.4	1292.2	56.9
c207	**2.0**	3.1	34.4	1304.1	65.4	**2**	3	32.4	1280.7	61.1
c208	**2.0**	3.1	38.2	1317.1	61.9	**2**	3	37.1	1278.5	60.8
r101	**2.0**	19.7	20.5	2349.5	26.0	**2**	19	19.3	2335.3	24.5
r102	**2.0**	*18.0*	27.6	2153.0	32.8	**2**	*18*	23.8	2136.9	30.4
r103	**2.0**	13.9	40.3	1975.2	46.3	**2**	13	38.0	1949.6	39.9
r104	**2.0**	10.9	53.7	1785.2	71.3	**2**	10	51.8	1764.0	64.7
r105	**2.0**	15.2	22.3	2094.5	24.4	**2**	15	16.9	2075.0	25.2
r106	**2.0**	12.9	35.0	2065.4	38.8	**2**	12	32.8	2030.1	43.3
r107	**2.0**	11.2	45.4	1819.8	46.7	**2**	11	56.0	1796.3	39.5
r108	**2.0**	10.3	49.9	1709.1	61.5	**2**	10	56.2	1655.1	51.6
r109	**2.0**	12.9	30.1	1967.6	32.4	**2**	12	27.6	1915.6	31.2
r110	**2.0**	*12.0*	41.1	1891.3	49.2	**2**	*12*	43.9	1835.2	39.4
r111	**2.0**	*12.0*	42.7	1832.0	49.3	**2**	*12*	34.0	1773.5	38.1
r112	**2.0**	11.1	51.4	1823.8	57.9	**2**	11	45.9	1790.8	61.0
r201	**1.0**	*4.0*	43.2	1615.6	42.6	**1**	*4*	37.6	1594.0	45.4
r202	**1.0**	3.6	59.7	1606.5	72.9	**1**	3	59.6	1540.1	77.1
r203	**1.0**	*3.0*	80.2	1279.1	85.0	**1**	*3*	90.2	1270.8	81.9
r204	**1.0**	2.9	96.0	1192.5	81.7	**1**	2	33.3	1184.0	73.1
r205	**1.0**	*3.0*	56.4	1355.6	52.8	**1**	*3*	47.5	1319.0	54.3
r206	**1.0**	*3.0*	74.8	1253.0	72.0	**1**	*3*	78.2	1232.8	78.0
r207	**1.0**	*3.0*	94.2	1157.5	87.6	**1**	*3*	100.1	1143.3	68.7
r208	**1.0**	*2.0*	31.0	1081.7	86.5	**1**	*2*	37.6	1067.1	80.8
r209	**1.0**	*3.0*	77.8	1270.5	69.3	**1**	*3*	66.3	1263.0	76.2
r210	**1.0**	*3.0*	74.1	1300.7	79.5	**1**	*3*	75.5	1279.8	80.3
r211	**1.0**	*3.0*	103.9	1146.4	90.7	**1**	*3*	89.4	1114.4	84.2
rc101	**3.0**	16.4	18.7	2626.0	26.2	**3**	16	16.2	2577.0	24.1
rc102	**3.0**	14.3	28.9	2451.6	42.8	**3**	14	29.1	2420.5	49.4
rc103	**3.0**	12.0	41.7	2530.3	115.3	**3**	11	48.6	2479.8	120.0
rc104	**3.0**	11.1	55.3	2167.1	72.6	**3**	11	64.2	2103.9	63.2
rc105	**3.0**	15.6	24.3	2595.5	39.3	**3**	15	22.5	2565.3	39.1
rc106	**3.0**	13.9	27.4	2601.4	59.8	**3**	13	36.0	2517.5	49.4
rc107	**3.0**	13.1	36.2	2335.0	49.9	**3**	13	29.8	2267.8	48.4
rc108	**3.0**	12.2	46.3	2258.7	60.1	**3**	12	48.0	2222.3	49.0
rc201	**1.0**	*4.0*	40.5	1831.8	50.6	**1**	*4*	41.5	1803.0	50.9
rc202	**1.0**	*4.0*	70.6	1546.1	63.5	**1**	*4*	55.5	1516.1	71.2
rc203	**1.0**	*3.0*	62.7	1462.8	94.2	**1**	*3*	69.4	1454.1	103.2
rc204	**1.0**	*3.0*	80.5	1228.6	99.6	**1**	*3*	71.6	1203.7	116.8
rc205	**1.0**	*4.0*	58.2	1706.5	57.8	**1**	*4*	63.0	1687.1	51.8
rc206	**1.0**	3.3	56.6	1636.8	72.5	**1**	3	48.3	1579.4	66.8
rc207	**1.0**	3.1	64.3	1510.0	100.3	**1**	3	60.4	1487.2	96.2
rc208	**1.0**	*3.0*	84.7	1310.1	102.2	**1**	*3*	75.6	1278.8	86.0

Table 5: Average and best solutions. FL (resp. SL) columns indicate the number of first-level (resp. second-level) vehicles. $T1$ (resp. $T2$) is the time spent in the fleet optimization (resp. cost optimization) phase. Bold figures indicate optimal values. Italics indicate that the average value is equal to its equivalent in the best known solution.

Type	avg. no. stops in FL routes	avg. no. trips in SL routes	no. of satellites used
C1	2.6	2.5	6.1
C2	2.7	4.8	5.2
R1	2.2	1.5	3.5
R2	2.7	2.7	2.3
RC1	2.6	1.8	5.6
RC2	3.8	2.7	3.3

Table 6: Comparison of the number of stops in first-level routes, the number of trips in second-level routes, and the number of satellites used for each type of instance.

$\{(i,j)|i,j \in \tilde{V}_s\} \cup \{(i,o')|i \in \tilde{V}_s, o' \in \tilde{O}'_1\}$. The second level is $G_2^{\text{math}} = (V_2^{\text{math}}, A_2^{\text{math}})$ with $V_2^{\text{math}} = \tilde{O}_2 \cup \tilde{O}'_2 \cup V_c \cup \tilde{V}_s$ and $A_2^{\text{math}} = \{(o,i)|o \in \tilde{O}_2, i \in \tilde{V}_s\} \cup \{(i,j)|i \in \tilde{V}_s, j \in V_c\} \cup \{(i,j)|i,j \in V_c\} \cup \{(i,j)|i \in V_c, j \in \tilde{V}_s\} \cup \{(i,o')|i \in V_c, o' \in \tilde{O}'_2\}$. We introduce the following variables:

$$x_{i,j}^k = \begin{cases} 1 & \text{if vehicle } k \text{ travels from node } i \text{ to node } j \\ 0 & \text{otherwise} \end{cases}$$

$h_i =$ service time at node i

$u_i =$ load of the second-level vehicle after serving i.

We introduce two constants: M_h corresponds to the end of the planning horizon, and M_u is the sum of all the ordered quantities.

$$\min \sum_{k \in K_1} \sum_{(i,j) \in A_1^{\text{math}}} x_{i,j}^k c_{i,j} + \sum_{k \in K_2} \sum_{(i,j) \in A_2^{\text{math}}} x_{i,j}^k c_{i,j} \tag{2}$$

s.t.

$$\sum_{(o_k,j) \in A_e^{\text{math}}} x_{o_k,j}^k = \sum_{(j,o'_k) \in A_e^{\text{math}}} x_{j,o'_k}^k = 1 \qquad \forall e \in \{1,2\},\ \forall k \in K_e \tag{3}$$

$$\sum_{(i,j) \in A_e^{\text{math}}} x_{i,j}^k = \sum_{(i,j) \in A_e^{\text{math}}} x_{j,i}^k \qquad \forall e \in \{1,2\}, \forall k \in K_e, \forall i \in V_e^{\text{math}} \setminus (\tilde{O}_e \cup \tilde{O}'_e) \tag{4}$$

$$\sum_{k \in K_2} \sum_{(j,d_r) \in A_2^{\text{math}}} x_{j,d_r}^k = 1 \qquad \forall r \in R \tag{5}$$

$$\sum_{k \in K_e} \sum_{j \in V_{s,r}} \sum_{(i,j) \in A_e^{\text{math}}} x_{i,j}^k = 1 \qquad \forall e \in \{1,2\}, \forall r \in R \tag{6}$$

$$\sum_{k \in K_1} \sum_{(i,v_{s,r}) \in A_1^{\text{math}}} x_{i,j}^k = \sum_{k \in K_2} \sum_{(i,v_{s,r}) \in A_2^{\text{math}}} x_{i,j}^k \qquad \forall v \in V_s, \forall r \in R \tag{7}$$

$$\sum_{v \in V_{s,r}} \sum_{(i,v) \in A_2^{\text{math}}} x_{j,v}^k = \sum_{(j,d_r) \in A_2^{\text{math}}} x_{j,d_r}^k \qquad \forall r \in R, \forall k \in K_2 \tag{8}$$

$$h_j \geq h_i + s_j + t_{i,j} - M_h \times (1 - \sum_{k \in K_e} x_{i,j}^k) \qquad \forall e \in \{1,2\}, \forall (i,j) \in A_e^{\text{math}} \tag{9}$$

$$h_{o_k} \geq 0 \qquad \forall k \in K \tag{10}$$

$$e_i \leq h_i \leq l_i \qquad \forall i \in V_c \tag{11}$$

$$h_{o'_k} \leq M_h \qquad \forall k \in K \tag{12}$$

$$\sum_{r \in R} q_r \times \sum_{j \in V_s} \sum_{(i,j) \in A_1^{\text{math}}} x_{i,j_r}^k \leq q_1 \qquad \forall k \in K_1 \tag{13}$$

$$u_j \geq u_i + q_i - M_u \times (1 - \sum_{k \in K_2} \sum_{(i,j) \in A_2^{\text{math}}} x_{i,j}^k) \qquad \forall i \in V_2 \setminus O_2 \tag{14}$$

$$u_j \leq M_u \times (1 - \sum_{k \in K_2} \sum_{i \in V_2^{\text{math}} \setminus \tilde{V}_s} x_{i,v}^k) \qquad \forall v \in \tilde{V}_s \tag{15}$$

$$0 \leq u_i \leq q_2 \qquad \forall i \in V_2^{\text{math}} \tag{16}$$

$$x_{i,j}^k \in \{0,1\} \qquad \forall k \in K, (i,j) \in A^{\text{math}} \tag{17}$$

The objective function (2) minimizes the travel costs. Constraints (3) state that each vehicle must start and end its route at its base. Constraints (4) are flow conservation constraints. Constraints (5) ensure that each request is delivered. Constraints (6) ensure for each request that only one transfer node is used, and (7) ensure that it is visited by both a first- and a second-level vehicle. Constraints (8) ensure for each request that the second-level vehicle that visits the transfer node is the one that delivers the request. Constraints (9) compute the travel time between two nodes if they are visited consecutively by the same vehicle, and constraints (10) handle the special case of bases for which there is no predecessor. Constraints (11) ensure that each request is delivered within its time window. Constraints (12) ensure that each vehicle has completed its route within the time horizon. Constraints (13) (resp. (16)) ensure for each first-level (second-level) vehicle that the load does not exceed the vehicle capacity. Constraints (14) ensure for each second-level vehicle that the load after visiting a node is equal to the load before plus (or minus) the quantity that has been loaded (unloaded). Constraints (15) ensure that each second-level vehicle is empty when arriving at a satellite.

References

[1] Nabila Azi, Michel Gendreau, and Jean-Yves Potvin. An adaptive large neighborhood search for a vehicle routing problem with multiple routes. *Computers & Operations Research*, 41:167–173, January 2014.

[2] Thomas H. Cormen, Charles E. Leiserson, Rivest L. Ronald, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2010.

[3] Teodor Gabriel Crainic, Simona Mancini, Guido Perboli, and Roberto Tadei. Clustering-based heuristics for the two-echelon vehicle routing problem. Technical report, CIRRELT, 2008.

[4] Teodor Gabriel Crainic, Simona Mancini, Guido Perboli, and Roberto Tadei. Multi-start heuristics for the two-echelon vehicle routing problem. In Peter Merz and Jin-Kao Hao, editors, *Evolutionary Computation in Combinatorial Optimization*, pages 179–190. Springer, 2011.

[5] Teodor Gabriel Crainic, Simona Mancini, Guido Perboli, and Roberto Tadei. GRASP with path relinking for the two-echelon vehicle routing problem. *Advances in Metaheuristics*, 53:113–125, 2013.

[6] Teodor Gabriel Crainic, Guido Perboli, Simona Mancini, and Roberto Tadei. Two-echelon vehicle routing problem: a satellite location analysis. *Procedia - Social and Behavioral Sciences*, 2(3):5944–5955, 2010.

[7] Teodor Gabriel Crainic, Nicoletta Ricciardi, and Giovanni Storchi. Advanced freight transportation systems for congested urban areas. *Transportation Research Part C: Emerging Technologies*, 12(2):119–137, April 2004.

[8] Teodor Gabriel Crainic, Nicoletta Ricciardi, and Giovanni Storchi. Models for evaluating and planning city logistics systems. *Transportation Science*, 43(4):432–454, 2009.

[9] Michael Drexl. Synchronization in vehicle routing–A survey of VRPs with multiple synchronization constraints. *Transportation Science*, 46(3):297–316, 2012.

[10] Jesus Gonzalez-Feliu and Guido Perboli. The two-echelon capacitated vehicle routing problem. In *Proceedings of the 22nd European Conference on Operational Research, Prague*, 2007.

[11] Vera C. Hemmelmayr, Jean-François Cordeau, and Teodor Gabriel Crainic. An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. *Computers & Operations Research*, 39(12):3215–3228, 2012.

[12] Mads Jepsen, Simon Spoorendonk, and Stefan Ropke. A branch-and-cut algorithm for the symmetric two-echelon capacitated vehicle routing problem. *Transportation Science*, 47(1):23–37, 2013.

[13] Simona Mancini. Multi-echelon distribution systems in city logistics. *European Transport/Trasporti Europei*, 54, 2013.

[14] Renaud Masson, Fabien Lehuédé, and Olivier Péton. An adaptive large neighborhood search for the pickup and delivery problem with transfers. *Transportation Science*, 47(3):344–355, 2013.

[15] Renaud Masson, Fabien Lehuédé, and Olivier Péton. Efficient feasibility testing for request insertion in the Pickup and Delivery Problem with Transfers. *Operations Research Letters*, 41(3):211–215, 2013.

[16] Phuong Khanh Nguyen, Teodor Gabriel Crainic, and Michel Toulouse. A tabu search for Time-dependent Multi-zone Multi-trip Vehicle Routing Problem with Time Windows. *European Journal of Operational Research*, 231(1):43–56, November 2013.

[17] Viet-Phuong Nguyen, Christian Prins, and Caroline Prodhon. Solving the two-echelon location routing problem by a GRASP reinforced by a learning process and path relinking. *European Journal of Operational Research*, 216(1):113–126, 2012.

[18] Guido Perboli and Roberto Tadei. New families of valid inequalities for the two-echelon vehicle routing problem. *Electronic Notes in Discrete Mathematics*, 36:639–646, 2010.

[19] Guido Perboli, Roberto Tadei, and Daniele Vigo. The two-echelon capacitated vehicle routing problem: models and math-based heuristics. *Transportation Science*, 45(3):364–380, 2011.

[20] David Pisinger and Stefan Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435, 2007.

[21] Yuan Qu and Jonathan F. Bard. A GRASP with adaptive large neighborhood search for pickup and delivery problems with transshipment. *Computers & Operations Research*, 39(10):2439–2456, 2012.

[22] Roberto Roberti. *Exact algorithms for different classes of vehicle routing problems*. PhD thesis, Bologna, June 2012.

[23] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

[24] Fernando Afonso Santos, Alexandre Salles Cunha, and Geraldo Robson Mateus. Branch-and-price algorithms for the two-echelon capacitated vehicle routing problem. *Optimization Letters*, 7(7):1537–1547, October 2012.

[25] Fernando Afonso Santos, Geraldo Robson Mateus, and Alexandre Salles da Cunha. A branch-and-cut-and-price algorithm for the two-echelon capacitated vehicle routing problem. *Transportation Science*, Articles i(April):1–14, 2014.

[26] Martin W. P. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 4(1):285–305, 1985.

[27] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Fourth International Conference on Principles and Practice of Constraint Programming*, pages 417–431, 1998.

[28] Marius M. Solomon. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*, 35(2):254–265, 1987.

[29] Éric D. Taillard, Gilbert Laporte, and Michel Gendreau. Vehicle routing with multiple use of vehicles. *Journal of the Operational Research Society*, 47:1065–1070, 1996.

[30] Zheng-Yang Zeng, W Xu, Zhi-Yu Xu, and Wei-Hui Shao. A Hybrid GRASP+ VND heuristic for the two-echelon vehicle routing problem arising in City Logistics. Technical report, School of Electronics and Information Engineering, Tongji University, 2014.