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

Philippe Grangier[a,b], Michel Gendreau[b], Fabien Lehuédé[a], Louis-Martin Rousseau[b]

[a] *L'UNAM, Ecole des Mines de Nantes, IRCCyN UMR CNRS 6597, 4 Rue Alfred Kastler, 44307 Nantes Cedex 3, France*
[b] *Department of Mathematics and Industrial Engineering and CIRRELT, Ecole Polytechnique de Montréal and CIRRELT, C.P 6079, Succursale Centre-ville, Montreal, QC, Canada H3C 3A7*

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, Two-Echelon VRP, Synchronization, city logistics, Adaptive Large Neighborhood Search.

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 [4]. 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 [13].

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 rents. More and more cities allow transporters to use dedicated or existing infrastructures (reserved parking spaces, bus depots) to unload [12]. No storage capacity is normally available at these locations, thus requiring a synchronization of the two levels.

Email addresses: philippe.grangier@mines-nantes.fr (Philippe Grangier), michel.gendreau@cirrelt.ca (Michel Gendreau), fabien.lehuede@mines-nantes.fr (Fabien Lehuédé), louis-martin.rousseau@polymtl.ca (Louis-Martin Rousseau)

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 [27] under the name *two-echelon vehicle routing problem with satellite synchronization* (2E-VRP-SS) and modeled in [13] under the name *two-echelon, synchronized, scheduled, multi-depot, 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 [13], while Mancini focuses on multi-echelon systems [19]. The 2E-VRP was introduced by Perboli et al.[26], who proposed a mathematical model. Since then several algorithms have been developed: math-based heuristics [26, 27], clustering-based heuristics [8], GRASP [9, 10, 38], adaptive large neighborhood search (ALNS) [16], and a large neighbourhood search combined with a local search [3]. Exact methods include [17, 30, 32, 33]. Crainic et al. [11] study the impact of satellite location on the cost of a 2E-VRP solution compared to that of a VRP. Cuda et al. [14] recently published a survey on two-echelon routing problems. A similar problem is the two-echelon location routing problem (2E-LRP) [25]. Our problem also integrates some multiple-trip aspects [37] that have been solved with tabu search [24], ALNS [1] and iterated local search [6]. Synchronization of multiple trips supplied by a single bus line as been studied in [23] in a city logistics context. We refer to [15] for a detailed survey of synchronization in vehicle routing problems and to [5] for a recent survey of vehicle routing problems in city logistics.

The 2E-MTVRP-SS can be considered as a particular Pickup and Delivery Problem with Transfers (PDPT) which has been recently studied in [20, 21, 29]. The major differences are that in the 2E-MTVRP-SS transfers are mandatory, routes should be designed for two types of vehicles which do not share the same network, and that two vehicles must be simultaneously present at a satellite during a transfer. In this paper, we extend the previous approaches to integrate those differences and we exploit the specificities of the 2E-MTVRP-SS to propose a better exploration of the search space, as well as a more compact graph representation of temporal constraints.

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, followed by the conclusion in Section 6.

2. Problem formulation

In this section we define the problem and discuss the synchronization model at satellites.

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 homogeneous 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. As second-level vehicles are small we assume that they are empty every time they arrive at a satellite, a similar assumption is made in [13]. 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 duration 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 transfers one or more requests to a second-level vehicle at a satellite. Given 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. In details, for a transfer to happen, the two vehicles:

- may need time to get ready for the transfer (for example if the first-level vehicle has a lift gate to open),
- should spend some time transferring the items,
- should get ready to leave the satellite (for example the second-level vehicle may have to sort items).

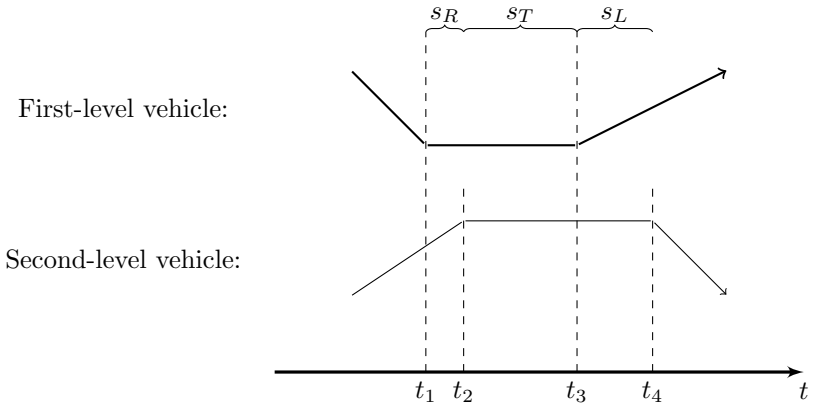


Figure 1: Time chart for a transfer

s_R $[t_1, t_2]$: Preparation time for the first-level vehicle.

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 $[t_3, t_4]$: Sorting time for the second-level vehicle.

Figure 1 illustrates the temporal aspects of a transfer. In this example, if $t_2 > t_1 + s_R$, the first-level vehicle must wait. Conversely, if $t_2 < t_1 + s_R$ 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 $\max_{i \in K_2} t_i + s_T$. In this paper, we assume that s_R, s_T, and s_L can reasonably be considered independent of the transferred quantity, without inducing a significant imprecision. This simplifying hypothesis allow to integrate those times into the travel times from and to the satellites. 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.

2.3. 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 $v_{s,r}$ whose associated demand is $-q_r$, 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 ($\tilde{O} = \cup_k o_k$, $\tilde{O'} = \cup_k o'_k$).

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 $A_1^{\text{math}} = \{(o,i) | o \in \tilde{O_1}, i \in \tilde{V_s}\} \cup \{(i,j) | i,j \in \tilde{V_s}\} \cup \{(i,o') | i \in \tilde{V_s}, o' \in \tilde{O'_1}\} \cup O_1 \times \tilde{O_1}$. 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

$A_2^{\text{math}} = \{(o,i)|o \in \tilde{O}_2, i \in \tilde{V}_s\} \cup \{(i,j)|i \in \tilde{V}_s, j \in V_c\} \cup \{(i,j)|i,j \in V_c\} \cup \{(i,j)|i \in V_c, j \in \tilde{V}_s\} \cup \{(i,o')|i \in V_c, o' \in \tilde{O}_2'\} \cup O_2 \times \tilde{O}_2$. 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 (point in time when service at node i starts)

$u_i =$ 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.

We use a classical approach in vehicle routing problems with time windows, which consists in lexicographically minimizing the fleet-size and the routing cost. The first objective is the number of first-level vehicles, the second objective is the number of second-level vehicles and the third objective is the sum of arc costs.

$$lex - min(\sum_{k \in K_1} \sum_{j \in \tilde{V}_s} x_{o_k,j}, \sum_{k \in K_2} \sum_{j \in \tilde{V}_s} x_{o_k,j}, \sum_{k \in K_1} \sum_{(i,j) \in A_1^{\text{math}}} c_{i,j} \times x_{i,j}^k + \sum_{k \in K_2} \sum_{(i,j) \in A_2^{\text{math}}} c_{i,j} \times x_{i,j}^k) \quad (1)$$

s.t.

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

$$\sum_{(i,j) \in A_e^{\text{math}}} x_{i,j}^k = \sum_{(j,i) \in A_e^{\text{math}}} x_{j,i}^k \qquad \forall e \in \{1,2\}, \forall k \in K_e, \forall i \in V_e^{\text{math}} \setminus (\tilde{O}_e \cup \tilde{O}_e') \quad (3)$$

$$\sum_{k \in K_2} \sum_{(j,d_r) \in A_2^{\text{math}}} x_{j,d_r}^k = 1 \qquad \forall r \in R \quad (4)$$

$$\sum_{k \in K_e} \sum_{j \in V_{s,r}} \sum_{(i,j) \in A_e^{\text{math}}} x_{i,j}^k = 1 \qquad \forall e \in \{1,2\}, \forall r \in R \quad (5)$$

$$\sum_{k \in K_1} \sum_{i \in V_1^{\text{math}}} x_{i,v_{s,r}}^k = \sum_{k \in K_2} \sum_{i \in V_2^{\text{math}}} x_{i,v_{s,r}}^k \qquad \forall v \in V_s, \forall r \in R \quad (6)$$

$$\sum_{v \in V_{s,r}} \sum_{(i,v) \in A_2^{\text{math}}} x_{i,v}^k = \sum_{(j,d_r) \in A_2^{\text{math}}} x_{j,d_r}^k \qquad \forall r \in R, \forall k \in K_2 \quad (7)$$

$$h_j \geq h_i + s_i + t_{i,j} - M_h \times (1 - \sum_{k \in K_e} x_{i,j}^k) \qquad \forall e \in \{1,2\}, \forall (i,j) \in A_e^{\text{math}} \quad (8)$$

$$h_{o_k} \geq 0 \qquad \forall k \in K \quad (9)$$

$$e_i \leq h_i \leq l_i \qquad \forall i \in V_c \quad (10)$$

$$h_{o_k'} \leq M_h \qquad \forall k \in K \quad (11)$$

$$\sum_{r \in R} q_r \times \sum_{j \in V_{s,r}} \sum_{(i,j) \in A_1^{\text{math}}} x_{i,j}^k \leq Q_1 \qquad \forall k \in K_1 \quad (12)$$

$$u_j \geq u_i - q_i - M_u \times (1 - \sum_{k \in K_2} x_{i,j}^k) \qquad \forall (i,j) \in A_2^{math} \quad (13)$$

$$u_j \leq M_u \times (1 - \sum_{k \in K_2} \sum_{i \in V_2^{\text{math}} \setminus \tilde{V}_s} x_{i,j}^k) \qquad \forall j \in \tilde{V}_s \quad (14)$$

$$0 \leq u_i \leq Q_2 \qquad \forall i \in V_2^{\text{math}} \quad (15)$$

$$x_{i,j}^k \in \{0,1\} \qquad \forall k \in K, (i,j) \in A^{\text{math}} \quad (16)$$

The objective function (1) lexicographically minimizes the fleet-size and the travel costs. Constraints (2) state that each vehicle must start and end its route at its base. Constraints (3) are flow conservation

constraints. Constraints (4) ensure that each request is delivered. Constraints (5) ensure for each request that only one transfer node is used, and (6) ensure that it is visited by both a first and a second-level vehicle. Constraints (7) ensure for each request that the second-level vehicle that visits the transfer node is the one that delivers the request. Constraints (8) compute the travel time between two nodes if they are visited consecutively by the same vehicle, and constraints (9) handle the special case of bases for which there is no predecessor. Constraints (10) ensure that each request is delivered within its time window. Constraints (11) ensure that each vehicle has completed its route within the time horizon. Constraints (12) (resp. (15)) ensure for each first-level (second-level) vehicle that the load does not exceed the vehicle capacity. Constraints (13) 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 (14) ensure that each second-level vehicle is empty when arriving at a satellite.

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 proposed by Ropke and Pisinger [31] as an extension of the large neighborhood search introduced by Shaw [35]. The general principle of ALNS is described in Algorithm 1: it iteratively destroys and repairs the current solution using heuristics, which are selected based on their past successes. Destroying here means removing a number p of requests from the current solution (p is bounded by some parameters), while repairing means inserting unplanned requests in the solution. Note that a solution s may be *incomplete*, if a set of requests $\mathcal{L}_s$ remained unplanned. The size of this set is then penalized in the objective function. The solution obtained after the destroy and repair operations is accepted if it satisfies an acceptance criterion. As in [31], we use a roulette-wheel mechanism as adaptive layer for the selection of destruction and repair methods, and a simulated annealing as acceptance criterion.
Since its introduction, ALNS has been used to solve the 2E-VRP [16] as well as many complex vehicle routing problems [1, 2, 18, 20, 22, 28].

Algorithm 1: Adaptive Large Neighborhood Search

Data: Candidate solution s, destroy operators $\mathcal{N}^-$, repair operators $\mathcal{N}^+$
Result: The best found solution $s^\star$

$s^\star \leftarrow s$
while *stop-criterion not met* **do**
  $s' \leftarrow s$
  Destroy quantity: select a number p of requests to remove from the candidate solution
  Operator selection: select a destruction method $or \in \mathcal{N}^-$ and a repair method $oi \in \mathcal{N}^+$
  Removal: $\mathcal{L} \leftarrow \mathcal{L}_{s'} \cup or(s', \text{p})$
  Insertion: $s' \leftarrow oi(s', \mathcal{L})$
  if $f(s') \leq f(s^\star)$ **then**
    $s^\star \leftarrow s'$
  Acceptance criterion: set the candidate solution s to s'
return $s^\star$

We refer to [31] for a more detailed explanation of ALNS. In the following, we focus on the specific components of our method, namely the construction of the initial solution, and the destroy and repair methods.

3.1. Initial solution

We design a two-phase constructive algorithm to obtain the initial solution. First, we design the second level routes using a best insertion algorithm for a multiple-trip multiple-depot problem. In this heuristic the possible insertion of a customer are all the insertions in existing trips and all insertions by creation of a new trip. Then we create the first-level routes to supply the satellites according to the second-level routing plan previously created.

3.2. Destroy methods

When partially destroying a solution we select a method and a number p 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.

3.2.1. 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.

3.2.2. 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 get their requests from 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.

3.2.3. Customer level

Random Customer Removal: we randomly remove a customer.
Worst Removal: this operator is the same as in [31]. For each request we compute the difference in cost of the solution with and without this request. We then sort the requests from the largest to the smallest difference. And we remove the *worst* (largest difference) request. Some randomization is introduced to avoid repeatedly removing the same requests.
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 start of service and the absolute gap between their latest delivery times. 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 [35], which groups time and distance into a single measure. The methods gives similar results, but we chose time and distance because they do not require parameter tuning.
History-based Removal: This is inspired by [20] 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 each request, let $\delta^-(r)$ (resp. $\delta^+(r)$ be its direct predecessor (resp. successor). 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.

3.3. 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 that we use, and then we describe the repair methods.

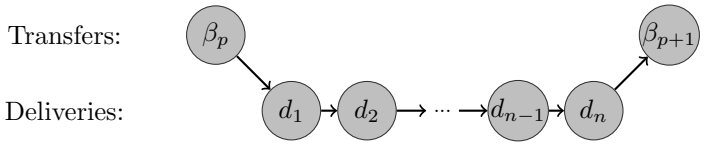
3.3.1. Three insertion operators

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 $\{(k, i) : \ k \in \text{Vehicles}, \ 0 \leq i \leq |route(k)| + 1\}$. Given the multiple-trip and two-echelon characteristics of the 2E-MTVRP-SS, we consider three distinct greedy ways of inserting a customer into a solution. We call them *insertion operators* and describe them below.

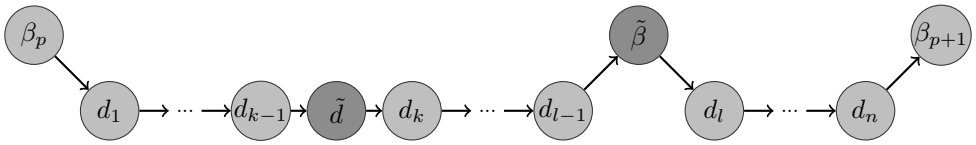
Insertion into an existing trip: The customer is inserted into an existing trip.

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 trip split operator.



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



(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.

3.3.2. 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 only 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 5.3.3, using these restricted neighborhoods makes our algorithm about 2.3 times faster while maintaining the quality of the solution.

3.3.3. 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.

3.3.4. 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 an existing 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:

$$\begin{aligned}\text{Biased cost} = {} & \text{second-level insertion cost} \\ & + \text{first-level insertion cost} \times \max\left(\alpha, \frac{\text{load in second-level trip}}{\text{second-level vehicle capacity}}\right). \quad (17)\end{aligned}$$

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.

4. Route scheduling and feasibility algorithm

For each performed insertion, repair methods evaluate thousands of insertions both in terms of profitability and feasibility. In this section we describe an efficient way to test if an insertion is feasible with respect to the temporal constraints of the problem. The proposed method is an adaptation of the feasibility algorithm designed by Masson et al [21] for the PDPT, which was based on forward time slacks [34]. The main idea behind forward time slacks is first introduced in 4.1, then a way to model time constraints is presented in 4.2, and the efficient feasibility tests are detailed in 4.3. All along this section, we use the notation in Table 1.

Notation	Definition
$\psi_{u,v}$	ordered set of vertices on path $(u, \dots, v)$
$\delta^+(i)$	direct successors 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. Efficient feasibility test for the vehicle routing problem with time windows

Inserting a request into a route of a feasible VRP solution may postpone several deliveries later in this route, potentially violating time windows. For a given insertion, a way to check if such a violation occurs would be to reschedule downstream operations while taking into account the postponement created by the insertion. Such method has a linear complexity in the size of the route, but Savelsbergh [34] introduced a method that checks in constant time if an insertion generates a time-window violation in an *as early as possible* route schedule. It comes at the price of recomputing some coefficients every time an insertion is performed, but this proved to be a very efficient trade off, as insertion based algorithms usually test many insertions before actually performing one.

Savelsbergh's method computes for each delivery the maximum possible forward shift, its *forward time slack* (FTS), that does not lead to a time-window violation later in the route.

In detail, let u and v be two nodes in the same route, with v being delivered after u. For all vertices i in the route, let h_i be the current time of service at i, w_i be the waiting time before service and l_i be the latest time for service at vertex i. We define the total waiting time on path $(u, \ldots, v)$ as

$$TWT_{\psi_{u,v}} = \sum_{i \in \psi_{\delta^+(u),v}} w_i. \tag{18}$$

The FTS at node u is

$$F_u = \min_{i \in \{u\} \cup \Gamma^+(u)} \{TWT_{\psi_{u,i}} + l_i - h_i\} \tag{19}$$

An intuitive explanation of formula (19) is the following. For every successor i of u, the path $(u, \ldots, i)$ can be decomposed into a working time and an idle/waiting time. If u is postponed, first the idle time will be reduced and then the time of service at node v will be postponed. So the margin for postponement of v that does not violate the time window at i is the total waiting time between u and i plus the difference between the current start time at i and the end of its time window. This should be true for every indirect successor of v thus the min.
And if the start time of service of u is postponed by δ then the new start time of service of v, $\tilde{h_v}$, can be computed

$$\tilde{h_v} = h_v + \max(0, \delta - TWT_{u,v}) \tag{20}$$

By simply comparing the shift of the delivery right after the insertion with its forward time slack, we get a constant time feasibility check.

4.2. Modeling time constraints in the 2E-MTVRP-SS

Contrary to the VRP, in problems with synchronization, a change in the schedule of one route may have effects on other routes, potentially making them timewise infeasible. For example, in the 2E-MTVRP-SS, if we insert a delivery in a trip of a second-level vehicle A, it may arrive later to its next transfer, thus delaying the first level-vehicle it is synchronized with. In turn, the first-level vehicle will spread its delay to second-level vehicles at its next transfer, and so on, eventually causing a time windows violation for a second-level vehicle B whose link with A was not obvious at first sight. This is known as *interdependence problem* [15], and we model it through a precedence graph.

4.2.1. Precedence graph definition

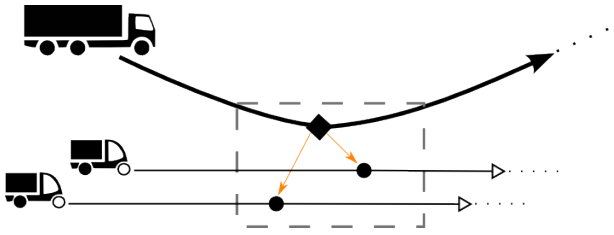
Given the routes in a solution of the 2E-MTVRP-SS, we can represent the time constraints as a directed acyclic graph, G_t. We refer to it as a *precedence graph* and it is built as follows: for every operation except transfers, we create a node and add an arc to each of its direct successors in the given 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.

4.2.2. Route scheduling

G_t corresponds to a PERT chart. As mentioned in [7] (p. 657), scheduling tasks in such a diagram can be performed using a shortest-path algorithm, with linear complexity in the number of vertices. 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.

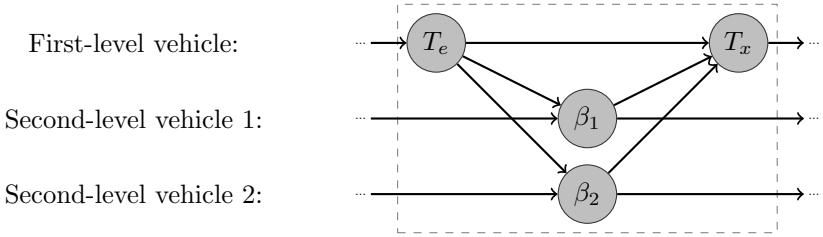
4.3. Using the precedence graph for efficient feasibility testing

With G_t we would be able to check if an insertion is timewise feasible by rescheduling downstream operations. But this would be even more time consuming than in the VRP, as it would be linear in the number of operations in the solution (versus number of operations in one route in the VRP). Masson et al. [21] were faced with a similar problem for the PDPT. They introduced a directed acyclic graph to model their precedence constraints and extended the FTS obtaining a constant time feasibility check for time constraints. Hereafter we present their main results and how they can be applied to the 2E-MTVRP-SS.



"/usr/local/texlive/2016/bin/universal-darwin/

(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.

4.3.1. Extension of FTS to directed acyclic graph modeling precedence constraints [21]

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

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

Then the FTS at node u becomes:

$$F_u = \min_{i \in \{u\} \cup \Gamma^+(u)} \{ST_{u,i} + l_i - h_i\}. \tag{21}$$

The intuitive explanation given for equation (19) still holds for equation (21), with slack times accounting for the fact they may exist several time paths between two vertices. Masson et al. proved the above result in their Proposition 2. Their proof does not rely on the particular structure of their precedence graph for the PDPT, it is thus valid for any directed acyclic graph. As such, this result (as well as the one hereafter on start of service) is valid for our precedence graph.
Similarly to the VRP, if the start of service time of node u is postponed by δ, the new start of service time $\tilde{h}$ of any of its sucessor v, is

$$\tilde{h_v} = h_v + max(0, \delta - ST_{u,v}) \tag{22}$$

Note that in contrast to Masson et al., we recompute FTS using the Floyd–Warshall algorithm ([7], p. 693), which is faster in our case than the suggested shortest path method.

4.3.2. Efficient feasibility test for the 2E-MTVRP-SS

When evaluating an insertion of an unplanned request into a feasible solution, we need to ensure that the solution will remain timewise feasible after the insertion. With the insertion operators of Section 3.3.1, timewise infeasibility can occur in two ways: creating a cycle of precedence constraints or violating a time window.

Cycle detection. 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. [21] for detecting cycles in constant time.

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

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

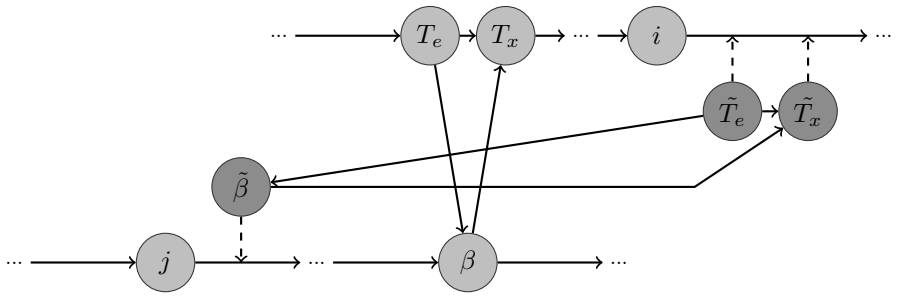


Figure 4: Infeasible insertion: it creates a cycle in the precedence graph.

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 $T_x \in \Gamma^+(\beta)$ 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*

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

Provided that we maintain a successor matrix, which can easily be computed together with slack times, we can check in constant time if an insertion creates a cycle in G_t.

Use of FTS. The use of FTS for checking the feasibility with respect to time windows of an *insertion into an existing trip*, or an *insertion by creation of a new trip* is straightforward. For an *insertion into an existing trip*, we check if the shift of the operations of the second-level vehicle right after the insertion is smaller than its forward time slacks. For an *insertion by creation of a new trip*, we check if the shift of the operations of the second-level vehicle right after the new delivery, and the shift of the operations of the first level vehicle right after the new transfer exit are smaller than their respective forward time slacks. The case of an *insertion by trip split* is more complex, it is illustrated in Figure 5 and its details are given in Algorithm2.
Still, for all insertion operators, with FTS we get a constant time feasibility check.

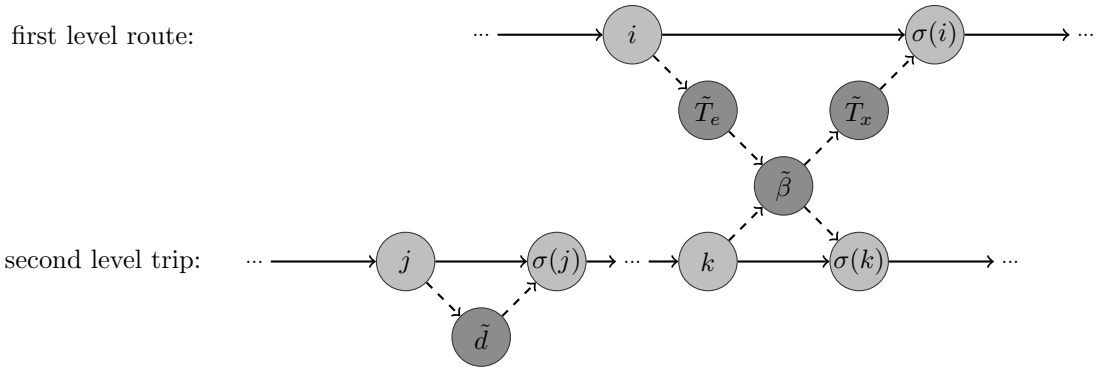


Figure 5: Example of insertion with split trip in the precedence graph. Customer $\tilde{d}$ is inserted between vertices j and $\sigma(j)$ in a second level trip. This trip is split and connected at a new satellite $\tilde{\beta}$ between vertices k and $\sigma(k)$. The new trip is supplied at a new stop in a first level route, which is inserted between vertices i and $\sigma(i)$.

4.3.3. Efficiency of the method

We compare the runtime of the extension of FTS versus a check based on an incremental PERT: on average our algorithm is approximately 12 times faster with the FTS extension than with the PERT (see Section 5.3.2).

Algorithm 2: Evaluation procedure for the feasibility of an insertion with trip split

Result: return **true** if the insertion illustrated in Figure 5 is feasible, **false** otherwise

$\bar{h}_{\tilde{T}_e} \leftarrow \max(h_i + s_i + t_{i,\tilde{T}_e}, e_{\tilde{T}_e})$
$\bar{h}_{\tilde{d}} \leftarrow \max(h_j + s_j + t_{j,\tilde{d}}, e_{\tilde{d}})$
if $\bar{h}_{\tilde{T}_e} > l_{\tilde{T}_e}$ **or** $\bar{h}_{\tilde{d}} > l_{\tilde{d}}$ **then**
    return false
$\bar{h}_{\sigma(j)} \leftarrow \max(\bar{h}_{\tilde{d}} + s_{\tilde{d}} + t_{\tilde{d},\sigma(j)}, e_{\sigma(j)})$
$\delta_{\sigma(j)} \leftarrow \bar{h}_{\sigma(j)} - h_{\sigma(j)}$
if $\delta_{\sigma(j)} > F_{\sigma(j)}$ **then**
    return false
$\bar{h}_k \leftarrow h_k + \max(\delta_{\sigma(j)} - ST_{\sigma(j),k}, 0)$
$\bar{h}_{\tilde{\beta}} \leftarrow \max(\bar{h}_{\tilde{T}_e}, \bar{h}_k + s_k + t_{k,\tilde{\beta}})$
$\bar{h}_{\sigma(k)} \leftarrow \max(\bar{h}_{\tilde{\beta}} + t_{\tilde{\beta},\sigma(k)}, e_{\sigma(k)})$
$\delta_{\sigma(k)} \leftarrow \bar{h}_{\sigma(k)} - h_{\sigma(k)}$
if $\delta_{\sigma(k)} > F_{\sigma(k)}$ **then**
    return false
$\bar{h}_{\tilde{T}_x} \leftarrow \bar{h}_{\tilde{\beta}}$
$\bar{h}_{\sigma(i)} \leftarrow \max(\bar{h}_{\tilde{T}_x} + t_{\bar{h}_{\tilde{T}_x},\sigma(i)}, e_{\sigma(i)})$
$\delta_{\sigma(i)} \leftarrow \bar{h}_{\sigma(i)} - h_{\sigma(i)}$
if $\delta_{\sigma(i)} > F_{\sigma(i)}$ **then**
    return false
return true

5. Computational experiments

In this section, we first describe the adaptation of some well-known VRPTW instances to the 2E-MTVRP-SS. Parameters 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 [36]. We use 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 6 illustrates a -50 / 50 / 3 / 3 configuration.

According to [11], 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 $[e_i + \delta, l_i + \delta]$.

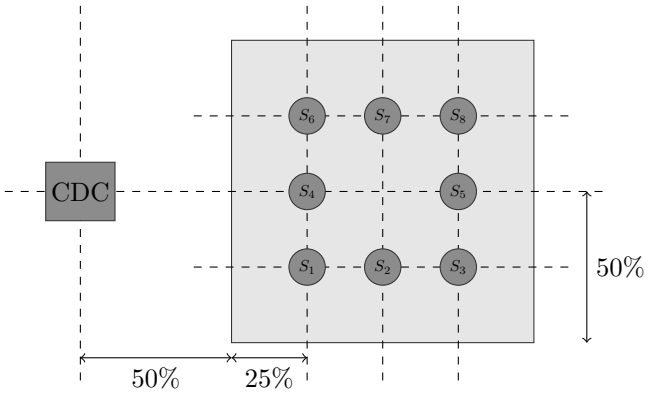


Figure 6: Geometrical layout -50 / 50 / 3 / 3.

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 capacity/second-level vehicles capacity: 4/0.5 ratio for instances 1; 2/0.25 ratio for instances 2.

5.1.3. Objectives

As the number of vehicles is not fixed, our algorithm is used twice. In a first phase we try to minimize the number of vehicles used by minimizing first the number of first level vehicles and then the number of second level vehicles. In a second phase, given the fleet obtained in the first phase, we minimize the routing cost.

5.2. Parameters 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 a maximum of 25,000 iterations with no more than 12,500 dedicated to the reduction of the first-level fleet. As Ropke and Pisinger [31], we stop the search if 5 or more requests are unplanned and no improvement in the number of unplanned requests has been found in the last 2,000 iterations.

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 [31]. We 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, worst 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 3.3.2).

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 solutions values and average values found for all instances out of ten runs.

5.3.1. Impact of two-phase initialization

For constructing the initial solution, we compare the best insertion method of Section 3.3.3 and the dedicated two phase best insertion method of Section 3.1. As shown in Table 2, we observe that although the latter alone produce worst results, it is a better starting point for the fleet reduction.

	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 solutions; italics indicate best average values.

5.3.2. Impact of FTS on reduction of runtime

On average the repair heuristics are far more time consuming than the destruction heuristics (96.1% versus 1.1% of the total runtime). This partly comes from the fact that the repair methods have to reschedule the solution every time an insertion is performed while the destruction methods reschedule the entire solution only when p customers have been removed from the current solution, and not after each removal.
In table 3, we compare the runtime of our algorithm using the extended FTS versus an incremental PERT to check the feasibility of an insertion. This clearly shows that FTS are keys to reducing the computational effort, as our algorithm is 91.9% faster with FTS. With FTS, time-related functions account for 37.7% of the total runtime of our algorithm.

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

5.3.3. 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, but the runtimes are significantly reduced (by 56.3%). 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.

5.3.4. 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.
Introducing split-recreate heuristics leads to solution that are 0.5% cheaper on average, showing that this

	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	2061.0	59.4	**2048.1**	48.3	2060.0	28.7	2053.7	33.5	2065.1	35.3
C102	1981.3	119.2	1984.0	61.7	1983.3	36.3	**1974.4**	42.7	1977.6	45.1
C201	1290.6	79.6	**1290.4**	38.4	1291.6	26.1	1293.8	27.7	1290.6	31.0
C202	1316.1	120.0	1311.3	64.8	**1309.4**	45.1	1311.4	48.9	1319.4	50.0
R101	**2343.5**	28.7	2352.0	27.8	2355.5	17.6	2353.2	20.8	2355.6	21.7
R102	2156.0	46.7	**2148.9**	31.7	2150.4	19.1	2158.1	23.0	2158.4	24.7
R201	1602.8	88.7	**1598.3**	33.1	1605.9	23.7	1604.5	26.8	1605.7	26.5
R202	1550.1	120.0	**1544.5**	49.9	1547.2	37.2	1546.1	40.0	1545.9	40.6
RC101	2624.0	34.2	2637.1	33.5	2616.3	20.7	**2609.0**	23.3	2613.0	23.3
RC102	2447.3	79.7	2451.2	46.9	2451.2	29.3	2446.9	33.9	**2435.1**	36.0
RC201	1821.3	104.8	1820.7	40.2	1812.5	29.2	**1809.3**	30.6	1811.0	32.0
RC202	1531.6	120.0	1534.8	49.5	**1525.0**	31.1	1527.1	37.8	1529.8	39.0
Dev. from Neigh. 1	-	-	-0.05%	-40.4%	-0.10%	-61.6%	-0.16%	-56.3%	-0.07%	-54.5%

Table 4: Comparison of the average cost and runtime over 10 runs for 5 different neighborhood configurations. Configuration 1 uses the entire *trip split* neighborhood. In configuration 2 the *trip split* 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.

neighborhood, although time consuming, is worth considering.
Biased cost and synchronization based removal do not make a significant difference on the quality of the solutions found. But introducing them makes our method more stable, this can be shown by comparing the standard deviation. When normalizing the average results for each instance to 100, average standard deviations on all test instances are the following: 0.93 with all heuristics, 0.97 without the synchronization based removal and 1.10 without biased costs.

5.3.5. 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 $LB1$ and thus optimal. In the cost optimization phase, there is an average gap of 4.68% 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.

5.4. Impact of time windows and synchronization on the cost of solutions

In what follows, we assess the impact of time windows and exact synchronization on the cost of solutions. For testing the impact of time windows on the cost of solutions, we run our algorithm on the same instances while removing the time windows of customers. As Solomon's instances from the same type only differ in their time windows, when removing them, there are only six instances.
To evaluate the impact of exact synchronization on the cost of solutions, we run our algorithm on all instances of the 2E-MTVRP-SS while only imposing a precedence constraint for transfers at satellites: a second-level vehicle can pick up goods iff a first-level vehicle has already dropped them. This corresponds to the case in which satellites have a storage capacity.
For creating Table 7, we run our algorithm ten times for each instance. We select the best result for each instance and then compute the average of the second-level fleet size and cost for each type of instance. We do not report the results for first-level fleet size since for the 2E-MTVRP-SS we already reached the lower bound.

From table 7, we observe that considering a precedence constraint instead of the exact synchronization constraint only marginally reduces the second-level fleet size and the average solution cost. This is not very surprising since we observe in the solution for the 2E-MTVRP-SS that first-level vehicles tend to stay for rather long periods of time at the same satellite, acting as depot. A realistic perspective here would be to enforce a maximum stay at satellites and to add waiting stations. A second observation is that removing time windows drastically reduces the second-level fleet size and also the cost of the solution.

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).

	Average					Best solution found				
Instance	FL	SL	T1 (min)	Cost	T2 (min)	FL	SL	T1 (min)	Cost	T2 (min)
c101	**3.0**	11.3	21.9	2057.4	30.5	**3**	11	8.3	2022.4	27.5
c102	**3.0**	*10.0*	48.4	1974.9	40.1	**3**	*10*	54.0	1947.6	45.6
c103	**3.0**	*9.0*	9.8	1946.8	46.9	**3**	*9*	13.4	1880.7	51.6
c104	**3.0**	*9.0*	8.4	1857.3	50.5	**3**	*9*	6.9	1811.1	51.5
c105	**3.0**	10.9	26.6	1959.2	35.1	**3**	10	26.4	1934.0	33.5
c106	**3.0**	10.8	32.2	1974.5	39.7	**3**	10	17.0	1945.0	34.0
c107	**3.0**	*10.0*	8.2	1902.8	35.5	**3**	*10*	5.8	1888.9	35.5
c108	**3.0**	*10.0*	8.8	1910.5	38.6	**3**	*10*	8.1	1875.3	34.1
c109	**3.0**	9.2	23.7	1921.3	46.1	**3**	9	5.2	1863.1	47.4
c201	**2.0**	3.5	27.8	1414.2	37.2	**2**	3	19.4	1389.3	33.9
c202	**2.0**	*3.0*	6.3	1317.9	45.4	**2**	*3*	3.6	1305.0	46.6
c203	**2.0**	3.1	9.2	1282.6	52.5	**2**	3	14.3	1272.7	51.8
c204	**2.0**	*3.0*	6.5	1261.3	57.5	**2**	*3*	9.1	1237.9	55.0
c205	**2.0**	3.1	6.8	1322.8	33.0	**2**	3	2.9	1312.1	33.0
c206	**2.0**	*3.0*	4.4	1328.6	36.1	**2**	*3*	4.8	1312.6	31.6
c207	**2.0**	*3.0*	3.9	1306.6	38.2	**2**	*3*	3.4	1280.4	39.5
c208	**2.0**	*3.0*	4.2	1300.0	37.3	**2**	*3*	3.2	1278.3	37.2
r101	**2.0**	19.6	28.9	2352.2	19.3	**2**	19	31.3	2333.5	17.0
r102	**2.0**	18.1	34.4	2152.8	21.7	**2**	18	30.5	2136.8	22.1
r103	**2.0**	13.8	45.2	1955.3	25.5	**2**	13	40.2	1942.7	22.2
r104	**2.0**	10.8	51.7	1804.8	47.2	**2**	10	44.4	1777.2	42.0
r105	**2.0**	14.9	27.7	2138.5	17.2	**2**	14	29.3	2096.8	11.5
r106	**2.0**	12.9	35.9	2028.0	26.8	**2**	12	23.8	1992.4	21.3
r107	**2.0**	11.2	37.7	1817.0	29.6	**2**	11	7.1	1779.2	29.9
r108	**2.0**	10.2	23.8	1702.2	36.6	**2**	10	17.8	1654.3	31.6
r109	**2.0**	12.9	30.3	1982.8	30.1	**2**	12	9.9	1925.9	26.6
r110	**2.0**	12.1	36.9	1888.9	33.6	**2**	12	39.6	1833.6	29.7
r111	**2.0**	*12.0*	44.3	1812.5	31.9	**2**	*12*	38.2	1770.8	29.7
r112	**2.0**	11.1	31.4	1800.9	46.9	**2**	11	54.5	1746.0	39.3
r201	**1.0**	*4.0*	8.2	1614.4	25.5	**1**	*4*	6.8	1587.8	26.6
r202	**1.0**	3.3	32.8	1548.3	39.9	**1**	3	36.6	1530.8	42.8
r203	**1.0**	*3.0*	12.6	1278.3	43.5	**1**	*3*	18.3	1255.1	44.4
r204	**1.0**	2.8	73.4	1200.1	49.3	**1**	2	6.2	1191.7	51.2
r205	**1.0**	*3.0*	6.7	1358.0	29.8	**1**	*3*	8.6	1319.1	31.0
r206	**1.0**	*3.0*	13.0	1251.5	39.1	**1**	*3*	17.5	1228.3	41.4
r207	**1.0**	*3.0*	20.7	1157.7	44.3	**1**	*3*	10.7	1140.2	43.7
r208	**1.0**	*2.0*	10.2	1082.5	49.6	**1**	*2*	5.3	1050.2	50.2
r209	**1.0**	*3.0*	9.2	1275.6	37.1	**1**	*3*	7.8	1258.7	46.8
r210	**1.0**	*3.0*	11.1	1300.4	42.7	**1**	*3*	11.5	1279.8	43.3
r211	**1.0**	*3.0*	15.9	1149.0	49.8	**1**	*3*	7.2	1118.2	44.6
rc101	**3.0**	16.4	26.5	2613.4	20.3	**3**	16	25.7	2577.0	19.7
rc102	**3.0**	14.3	34.8	2460.6	31.4	**3**	14	33.8	2407.1	29.2
rc103	**3.0**	12.0	41.7	2541.9	80.1	**3**	11	48.6	2476.9	72.7
rc104	**3.0**	11.1	41.1	2163.5	41.5	**3**	11	41.4	2125.9	39.7
rc105	**3.0**	15.4	33.1	2602.4	28.3	**3**	15	28.9	2542.6	30.0
rc106	**3.0**	13.9	27.4	2584.7	47.0	**3**	13	36.0	2494.9	47.8
rc107	**3.0**	13.1	41.1	2311.1	39.4	**3**	13	38.0	2271.1	40.4
rc108	**3.0**	12.2	38.9	2229.5	45.8	**3**	12	52.6	2202.9	41.5
rc201	**1.0**	*4.0*	6.8	1834.4	29.5	**1**	*4*	8.2	1787.6	25.4
rc202	**1.0**	*4.0*	65.2	1544.2	37.3	**1**	*4*	68.1	1513.8	36.9
rc203	**1.0**	*3.0*	8.3	1450.7	45.4	**1**	*3*	9.1	1416.2	44.3
rc204	**1.0**	*3.0*	10.0	1211.8	50.1	**1**	*3*	8.6	1188.2	42.8
rc205	**1.0**	*4.0*	27.4	1714.7	34.3	**1**	*4*	5.9	1693.7	30.1
rc206	**1.0**	3.4	34.8	1627.1	41.4	**1**	3	4.3	1583.1	35.3
rc207	**1.0**	*3.0*	10.0	1509.2	50.7	**1**	*3*	11.5	1449.8	49.8
rc208	**1.0**	*3.0*	9.8	1301.4	56.5	**1**	*3*	6.0	1257.3	55.9

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.

Type	2E-MTVRP-SS		No TW		Precedence	
	SL	Cost	SL	Cost	SL	Cost
C1	9.8	1911.0	9	1701.7	9.7	1911.8
C2	3.1	1284.2	3	1185.5	3	1268.4
R1	12.8	1913.3	9	1595.6	13	1911.5
R2	2.9	1268.3	2	975.1	2.9	1265.4
RC1	13.3	2348.6	10	2018.5	13.1	2343.2
RC2	3.4	1476.3	2	1061.2	3.25	1525.6

Table 7: Comparison of the average best results for each instance type for the original 2E-MTVRP-SS, removing time windows and enforcing only precedence at transfers

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. As for city logistics, our comparison in 5.4, cleary outlines that time windows have a greater influence than exact synchronization on the cost of solutions.

Acknowledgements

This work was partially supported by the Canadian Natural Science and Engineering Research Council through its Discovery Grants program and by the Fonds de recherche du Québec - Nature et technologies through its Team research Program.

References

[1] N. Azi, M. Gendreau, J.-Y. Potvin, An adaptive large neighborhood search for a vehicle routing problem with multiple routes, Computers & Operations Research 41 (2014) 167–173.

[2] A. Bortfeldt, T. Hahn, D. Männel, L. Mönch, Hybrid algorithms for the vehicle routing problem with clustered backhauls and 3D loading constraints, European Journal of Operational Research 243 (2015) 82–96.

[3] U. Breunig, V. Schmid, R. F. Hartl, T. Vidal, A fast large neighbourhood based heuristic for the Two-Echelon Vehicle Routing Problem, Working Paper (2015).

[4] D. Cattaruzza, N. Absi, D. Feillet, J. González-Feliu, Vehicle routing problems for city logistics, EURO Journal on Transportation and Logistics 1–29.

[5] D. Cattaruzza, N. Absi, D. Feillet, J. González-Feliu, Vehicle routing problems for city logistics, EURO Journal on Transportation and Logistics 1–29.

[6] D. Cattaruzza, N. Absi, D. Feillet, D. Vigo, An iterated local search for the multi-commodity multi-trip vehicle routing problem with time windows, Computers & Operations Research 51 (2014) 257–267.

[7] T. H. Cormen, C. E. Leiserson, R. L. Ronald, C. Stein, Introduction to Algorithms, 3rd ed., The MIT Press, 2010.

[8] T. G. Crainic, S. Mancini, G. Perboli, R. Tadei, Clustering-based heuristics for the two-echelon vehicle routing problem, Tech. Rep. CIRRELT-2008-46, CIRRELT (2008).

[9] T. G. Crainic, S. Mancini, G. Perboli, R. Tadei, Multi-start heuristics for the two-echelon vehicle routing problem, in: P. Merz, J.-K. Hao (eds.), Evolutionary Computation in Combinatorial Optimization, Springer, 2011, pp. 179–190.

[10] T. G. Crainic, S. Mancini, G. Perboli, R. Tadei, GRASP with path relinking for the two-echelon vehicle routing problem, Advances in Metaheuristics 53 (2013) 113–125.

[11] T. G. Crainic, G. Perboli, S. Mancini, R. Tadei, Two-echelon vehicle routing problem: a satellite location analysis, Procedia - Social and Behavioral Sciences 2 (3) (2010) 5944–5955.

[12] T. G. Crainic, N. Ricciardi, G. Storchi, Advanced freight transportation systems for congested urban areas, Transportation Research Part C: Emerging Technologies 12 (2) (2004) 119–137.

[13] T. G. Crainic, N. Ricciardi, G. Storchi, Models for evaluating and planning city logistics systems, Transportation Science 43 (4) (2009) 432–454.

[14] R. Cuda, G. Guastaroba, M. Speranza, A survey on two-echelon routing problems, Computers & Operations Research (2014) 1–15.

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

[16] V. C. Hemmelmayr, J.-F. Cordeau, T. G. Crainic, An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics, Computers & Operations Research 39 (12) (2012) 3215–3228.

[17] M. Jepsen, S. Spoorendonk, S. Ropke, A branch-and-cut algorithm for the symmetric two-echelon capacitated vehicle routing problem, Transportation Science 47 (1) (2013) 23–37.

[18] A. Kovacs, S. Parragh, K. Doerner, R. Hartl, Adaptive large neighborhood search for service technician routing and scheduling problems, Journal of Scheduling 15 (5) (2012) 579–600.

[19] S. Mancini, Multi-echelon distribution systems in city logistics, European Transport/Trasporti Europei 54.

[20] R. Masson, F. Lehuédé, O. Péton, An adaptive large neighborhood search for the pickup and delivery problem with transfers, Transportation Science 47 (3) (2013) 344–355.

[21] R. Masson, F. Lehuédé, O. Péton, Efficient feasibility testing for request insertion in the Pickup and Delivery Problem with Transfers, Operations Research Letters 41 (3) (2013) 211–215.

[22] R. Masson, F. Lehuédé, O. Péton, The Dial-A-Ride Problem with Transfers, Computers & Operations Research 41 (2014) 12–23.

[23] R. Masson, A. Trentini, F. Lehuédé, N. Malhéné, O. Péton, H. Tlahig, Optimization of a city logistics transportation system with mixed passengers and goods, EURO Journal on Transportation and Logistics (2015) 1–29.

[24] P. K. Nguyen, T. G. Crainic, M. Toulouse, A tabu search for Time-dependent Multi-zone Multi-trip Vehicle Routing Problem with Time Windows, European Journal of Operational Research 231 (1) (2013) 43–56.

[25] V.-P. Nguyen, C. Prins, C. 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) (2012) 113–126.

[26] G. Perboli, R. Tadei, New families of valid inequalities for the two-echelon vehicle routing problem, Electronic Notes in Discrete Mathematics 36 (2010) 639–646.

[27] G. Perboli, R. Tadei, D. Vigo, The two-echelon capacitated vehicle routing problem: models and math-based heuristics, Transportation Science 45 (3) (2011) 364–380.

[28] D. Pisinger, S. Ropke, A general heuristic for vehicle routing problems, Computers & Operations Research 34 (8) (2007) 2403–2435.

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

[30] R. Roberti, Exact algorithms for different classes of vehicle routing problems, Ph.D. thesis, Bologna (Jun. 2012).

[31] S. Ropke, D. Pisinger, An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows, Transportation Science 40 (4) (2006) 455–472.

[32] F. A. Santos, A. S. Cunha, G. R. Mateus, Branch-and-price algorithms for the two-echelon capacitated vehicle routing problem, Optimization Letters 7 (7) (2012) 1537–1547.

[33] F. A. Santos, G. R. Mateus, A. S. da Cunha, A branch-and-cut-and-price algorithm for the two-echelon capacitated vehicle routing problem, Transportation Science Articles i (April) (2014) 1–14.

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

[35] P. Shaw, Using constraint programming and local search methods to solve vehicle routing problems, in: Fourth International Conference on Principles and Practice of Constraint Programming, 1998, pp. 417–431.

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

[37] E. D. Taillard, G. Laporte, M. Gendreau, Vehicle routing with multiple use of vehicles, Journal of the Operational Research Society 47 (1996) 1065–1070.

[38] Z.-Y. Zeng, W. Xu, Z.-Y. Xu, W.-H. Shao, A Hybrid GRASP+ VND heuristic for the two-echelon vehicle routing problem arising in City Logistics, Tech. rep., School of Electronics and Information Engineering, Tongji University (2014).