

Alma Mater Studiorum Università di Bologna  
Archivio istituzionale della ricerca

A column generation based heuristic for the generalized vehicle routing problem with time windows

This is the final peer-reviewed author's accepted manuscript (postprint) of the following publication:

*Published Version:*

Yuan Y., Cattaruzza D., Ogier M., Semet F., Vigo D. (2021). A column generation based heuristic for the generalized vehicle routing problem with time windows. TRANSPORTATION RESEARCH PART E-LOGISTICS AND TRANSPORTATION REVIEW, 152, 1-25 [10.1016/j.tre.2021.102391].

*Availability:*

This version is available at: <https://hdl.handle.net/11585/855775> since: 2024-04-18

*Published:*

DOI: <http://doi.org/10.1016/j.tre.2021.102391>

*Terms of use:*

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (<https://cris.unibo.it/>).  
When citing, please refer to the published version.

(Article begins on next page)

# A column generation based heuristic for the generalized vehicle routing problem with time windows

Yuan Yuan<sup>1</sup>, Diego Cattaruzza<sup>1</sup>, Maxime Ogier<sup>1</sup>, Frédéric Semet<sup>1</sup>, Daniele Vigo<sup>2</sup>

1: Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRISTAL, F-59000 Lille, France

yuanyuannpu@163.com

{diego.cattaruzza, maxime.ogier, frederic.semet}@centralelille.fr

2: DEI, University of Bologna, Italy

daniele.vigo@unibo.it

**Abstract:** The generalized vehicle routing problem with time windows (GVRPTW) is defined on a directed graph  $\mathcal{G} = (\mathcal{V}, \mathcal{A})$  where the vertex set  $\mathcal{V}$  is partitioned into clusters. One cluster contains only the depot, where is located a homogeneous fleet of vehicles, each with a limited capacity. The other clusters represent customers. A demand is associated with each cluster. Inside a cluster, the vertices represent the possible locations of the customer. A time window is associated with each vertex, during which the visit must take place if the vertex is visited. The objective is to find a set of routes such that the total traveling cost is minimized, exactly one vertex per cluster is visited, and all the capacity and time constraints are respected. This paper presents a set covering formulation for the GVRPTW which is used to provide a column generation based heuristic to solve it. The proposed solving method combines several components including a construction heuristic, a route optimization procedure, local search operators and the generation of negative reduced cost routes. Experimental results on benchmark instances show that the proposed algorithm is efficient and high-quality solutions for instances with up to 120 clusters are obtained within short computation times.

**Keywords:** generalized vehicle routing problem; time windows; last mile delivery; delivery options; trunk/in-car delivery.

## 1 Introduction

Nowadays, e-commerce is becoming more and more popular and the growing e-commerce poses a huge challenge for the last mile delivery since ordered items need to be delivered to individual customers. Currently, there exist several last mile delivery services. The most common one is home delivery. Customers wait at home to get their packages. Besides, the delivery can be made to pick-up points such as dedicated lockers or stores (Morganti et al., 2014; Janjevic et al., 2019; Lin et al.,

2020). In this case, customers can retrieve their packages after delivery has been accomplished. This reduces the fragmentation of the deliveries in the last mile, thereby helping to reduce the congestion and environmental pollution caused by urban freight trips (Morganti et al., 2014). In recent years, a new concept called *trunk/in-car delivery*, has been proposed. Here, customers' packages can be delivered to the trunks of cars. Volvo launched its world-first in-car delivery in Sweden in 2016 (Kirsten, 2016). Amazon provides the in-car delivery to its prime members in more than 50 cities in the US (Hawkins, 2019). In-car delivery is different from home delivery and pick-up points delivery since a car moves and may be in different locations during different periods of time, e.g., parked at the workplace during the morning and at the commercial center during the afternoon. As a consequence, synchronization between the car and the courier is required to make the delivery.

All these delivery services can be combined and proposed to customers, and instead of selecting one delivery location when purchasing online, a customer can propose a set of delivery locations (home, pick-up points and car trunk) with the associated time constraints. To deliver a package to a specific customer, the courier only needs to choose one of the locations provided by the customer. Customers thus benefit from greater flexibility according to their own convenience. In addition, it could increase the rate of successful first-time deliveries and decrease delivery costs. In Figure 1, we give an example of the application. Six customers are represented with their associated locations grouped into a dotted circle, representing the different clusters. Every location is associated with a TW during which delivery should occur. In the case of home or in-car delivery, the TW represents the period during which the customer or the customer's car is present at that location. In the case of a pick-up point, the TW represents the period during which the courier can deliver the package before the customer arrives at the location and picks it up. The problem is to jointly determine the location visited to serve customer, and for each vehicle, the customers delivered and the sequence of visits while satisfying TW and capacity restrictions. The feasible solution given in the example in Figure 1 involves two vehicles, each of them serving three customers.

The underlying routing problem in the above application can be modeled as the Generalized Vehicle Routing Problem with Time Windows (GVRPTW), where clusters represent possible delivery locations associated with a customer. The GVRPTW is defined on a directed graph where the vertex set is partitioned into clusters. One cluster contains only the depot, where is located a homogeneous fleet of vehicles, each with a limited capacity. The other clusters represent customers. Then, the vertices of a cluster are the possible delivery locations of the customer, each with a time window during which the visit must take place if the vertex is visited. Each customer is associated with a demand. The objective is to find a set of routes such that the total traveling cost is minimized, only one vertex per cluster is visited, and all the capacity and time constraints are respected.

To the best of our knowledge, the GVRPTW has barely been studied before. When TWs are not considered, the GVRPTW reduces to the Generalized Vehicle Routing Problem (GVRP) (Ghiani and Improtà, 2000). When only one vehicle is available, the GVRPTW reduces to the Generalized

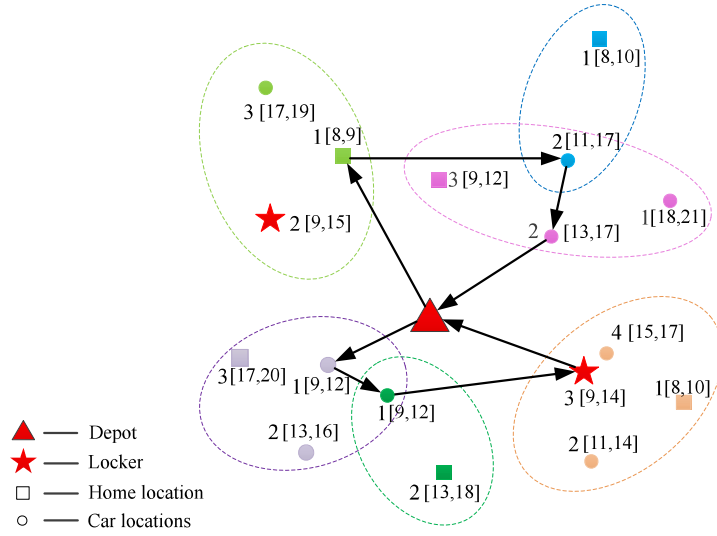


Figure 1: An example of the application: a routing problem in the context of last mile delivery with multiple delivery options.

Traveling Salesman Problem with Time Windows (GTSPTW) (Yuan et al., 2020). One special case of the GVRPTW is called the Vehicle Routing Problem with Roaming Delivery Locations (VRPRDL) (Reyes et al., 2017). In the VRPRDL, the TWs of locations provided by the same customer do not overlap. Moreover, the time between windows of two roaming locations of one customer is at least the travel time between them.

In this paper, a set covering model is built for the GVRPTW and an efficient column generation based heuristic is proposed to solve it. This matheuristic relies on a construction heuristic, a route optimization procedure, local search operators, and a heuristic procedure to provide negative reduced cost routes. All the procedures mentioned above take into account the main characteristic of the GVRPTW: given a customer served on a route, we can choose the location to visit. The proposed solution method is tested on different sets of instances from the literature and their variants. Results proved the efficiency of the solution method.

The remainder of this paper is organized as follows. Section 2 presents the related literature. The formal definition of the problem is described in Section 3. The description of the matheuristic is given in Section 4, including the set covering model, the construction heuristic, the route optimization, the local search operators, and the negative reduced cost route generation. Section 5 provides details about the experiments and reports the computational results. Finally, conclusions are drawn in Section 6.

## 2 Related literature

To the best of our knowledge, there is very limited literature on the GVRPTW. However, there exist works addressing related problems as the Generalized Traveling Salesman Problem (GTSP),

Generalized Vehicle Routing Problem (GVRP) and Vehicle Routing Problem with Roaming Delivery locations (VRPRDL). The GVRP is a special case of the GVRPTW where the TWs are not considered. The GVRP reduces to the GTSP when there is only one vehicle in the fleet. In the VRPRDL, TWs of locations within the same cluster have a specific structure.

The GTSP was first introduced by [Srivastava et al. \(1969\)](#) and [Saskena \(1970\)](#), and dynamic programming was used to solve it. Several papers proposed to transform the GTSP to the well-studied TSP, and then solve the latter by applying existing exact or heuristic approaches for the TSP ([Noon and Bean, 1993](#); [Laporte and Semet, 1999](#)). [Fischetti et al. \(1997\)](#) proposed an efficient branch-and-cut algorithm to solve the symmetric GTSP. They developed exact and heuristic separation procedures for some classes of facet-defining inequalities. [Gutin and Karapetyan \(2010\)](#) proposed a memetic algorithm combining genetic and powerful local search algorithms. They reported very good performance of this algorithm on benchmark instances. [Helsgaun \(2015\)](#) extended the Lin-Kernighan-Helsgaun TSP heuristic ([Helsgaun, 2000](#)) to the GTSP. The proposed algorithm could find high-quality solutions for large-scale instances. [Smith and Imeson \(2017\)](#) presented a GTSP solver based on adaptive large neighborhood search. Results showed that this algorithm performed consistently well across different problem libraries and outperformed other approaches on harder instances.

For the GTSP with time windows (GTSP<sub>TW</sub>), [Yuan et al. \(2020\)](#) developed a branch-and-cut algorithm and proposed several valid inequalities. Instances with up to 30 clusters are solved to optimality within one hour of computation time.

The GVRP was introduced by [Ghiani and Improta \(2000\)](#). Multiple service locations and a given demand are associated with each customer. The GVRP consists of determining a set of routes for a given number of vehicles with limited capacity such that exactly one location of each customer is visited. The objective is to minimize the total traveling cost. The GVRP has many applications in urban waste collection ([Bautista et al., 2008](#)), ship routing in maritime transportation, and healthcare logistics ([Bektaş et al., 2011](#)). Moreover, [Baldacci et al. \(2010\)](#) showed that several problems like the TSP with profits, the VRP with selective backhauls, the covering VRP, and the windy routing problem can be modeled as GVRPs.

[Kara and Bektas \(2003\)](#) proposed a compact integer linear programming formulation for the GVRP, adapting the well-known Miller-Tucker-Zemlin (MTZ) constraints for the TSP to the GVRP. [Bektaş et al. \(2011\)](#) proposed four integer linear programming formulations for the GVRP and developed an efficient branch-and-cut algorithm to solve it to optimality. They also developed an adaptive large neighborhood search (ALNS) heuristic to compute upper bounds.

[Ha et al. \(2014\)](#) and [Afsar et al. \(2014\)](#) studied a variant of the GVRP where the size of the fleet is not fixed. [Ha et al. \(2014\)](#) proposed a branch-and-cut algorithm, that provides better results than the one presented by [Bektaş et al. \(2011\)](#). [Afsar et al. \(2014\)](#) developed a very efficient iterated local search (ILS) which is able to find near optimal solutions in a few seconds.

[Reihaneh and Ghoniem \(2018\)](#) developed a branch-cut-and-price algorithm for the GVRP. Their computational study indicated that the proposed algorithm is competitive with respect to the

branch-and-cut algorithm proposed by [Bektaş et al. \(2011\)](#). Moreover, it solved eight benchmark instances to optimality that were previously unsolved.

[Zhou et al. \(2018\)](#) introduced a city logistics problem called the multi-depot two-echelon VRP with delivery options. In the second level of the distribution network, two delivery options are considered: customer location and pick-up points. The second level can be formulated as a GVRP.

[Moccia et al. \(2012\)](#) studied the GVRPTW and proposed a tabu search heuristic. However, the instances used to test the algorithm have a special structure and are such that all the vertices that form a cluster share the same TW.

The VRPRDL was introduced by [Reyes et al. \(2017\)](#), inspired by the trunk/in-car delivery. The objective is to find a minimum-cost set of routes for a fleet of capacitated vehicles in which the order of a customer has to be delivered to the trunk of the customer's car. Deliveries have to take place when the car is parked at one of the locations visited on the customer's travel itinerary. The VRPRDL is a special case of the GVRPTW in which one cluster contains all possible car locations for one customer. The TWs of the locations within a cluster have a specific structure since they are non-overlapping. The authors proposed a dynamic programming procedure to optimize the delivery costs for a fixed customer sequence. Based on this procedure, they developed construction and improvement heuristics. The results highlighted the economic benefits for delivery companies to consider trunk deliveries instead of the traditional home delivery. The core procedure of this heuristic is based on the dynamic programming, which takes advantage of the special structure of the non-overlapping TWs in the problem and, as a consequence, results to be very efficient.

[Ozbaygin et al. \(2017\)](#) developed a branch-and-price algorithm for the VRPRDL. This algorithm was able to solve to optimality instances with up to 60 clusters in few minutes. For most of the large instances with 120 clusters, the algorithm ended with an optimality gap after 6 hours of computation. [Ozbaygin et al. \(2017\)](#) also provided another set of instances for a hybrid delivery strategy combining trunk delivery and home delivery. This instance set also has a specific TW structure. In each cluster the TW associated with the home location corresponds to the planning horizon and overlaps all other TWs, while the other TW associated with trunk locations are non-overlapping. The results revealed that employing this combined strategy led to an average cost savings of nearly 20% compared to the classical delivery system when only home delivery is available. Note that the special structure of the TWs of the VRPRDL well fits a branch-and-price resolution scheme. The non-overlapping TW structure allows to eliminate arcs from the graph in the preprocessing phase as well as limiting the combinatorial explosion during the pricing phase.

Another type of problems worth mentioning is the close-enough TSP (CETSP) and VRP (CEVRP). In the CETSP, a set of nodes is given. Each node is associated with a neighborhood that is usually defined as a disk centered on the node. A node is served if the vehicle passes through its neighborhood. Note that the CETSP intrinsically is a continuous optimization problem. However, it is common to propose a discretization of the neighborhood of each node. Thus, only one of the points resulting from the discretization in the neighborhood of a node needs to be visited to perform the service ([Carrabs et al. \(2017, 2020\)](#)). It follows that after determining a discretization,



the GTSP must be solved in order to obtain a feasible solution for the CETSP.

With regard to the column generation based heuristic/metaheuristic approach, to the best of our knowledge, there is no related research having been done for the GVRP, VRPRDL or GVRPTW. However, such algorithms have already been proposed in the literature and proved to provide good results for various problems, e.g., VRPTW (Prescott-Gagnon et al., 2009; Beheshti and Hejazi, 2015), multi-period VRP (Mourgaya and Vanderbeck, 2007), CVRP with loading constraints (Mahvash et al., 2017), heterogeneous fleet VRP (Taillard, 1999; Salhi et al., 2013), dial-a-ride problem (Parragh and Schmid, 2013), waste collection routing problem (Hauge et al., 2014), etc. For most of these approaches, heuristics/metaheuristics are integrated within the column generation and used to solve the subproblems. A route pool is initialized with a feasible solution and columns of negative reduced cost are generated by means of tabu search (Hauge et al., 2014; Beheshti and Hejazi, 2015), variable neighborhood search (Parragh and Schmid, 2013), heuristic procedure based on dynamic programming (Furini et al., 2012), specific designed heuristics (Mahvash et al., 2017; Salhi et al., 2013), etc. For the method proposed in this work, the difference from the above approaches is that columns are not only generated by exploiting the dual information, moreover they are identified starting with routes with a zero reduced cost.

### 3 Problem description and notation

The generalized vehicle routing problem with time windows (GVRPTW) can be formally defined as follows. Given a directed graph  $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ , the set of vertices  $\mathcal{V} = \{0, 1, \dots, N\}$  is partitioned into  $\mathcal{C}_0 = \{0\}, \mathcal{C}_1, \dots, \mathcal{C}_K$  clusters, where  $\mathcal{K} = \{0, 1, \dots, K\}$  denotes the cluster index set. Hence, we have  $\bigcup_{k=0}^K \mathcal{C}_k = \mathcal{V}$  and  $\mathcal{C}_k \cap \mathcal{C}_{k'} = \emptyset, \forall k, k' \in \mathcal{K}, k \neq k'$ . Cluster  $\mathcal{C}_0$  contains only the depot 0 where a fleet of  $M$  homogeneous vehicles is located. Each vehicle has a capacity  $Q$ . Cluster  $\mathcal{C}_k, k \in \mathcal{K} \setminus \{0\}$ , represents the set of alternative locations in which customer  $k$  can be delivered. Moreover, each customer has a demand  $Q_k \geq 0, k \in \mathcal{K} \setminus \{0\}$ . We suppose that the demand at the depot  $Q_0$  is 0. A time window (TW)  $[E_i, L_i]$  is associated with each vertex  $i \in \mathcal{V}$  with  $[E_0, L_0] = [0, T]$  representing the overall time horizon. A visit can only be made to a vertex during its TW, and an early arrival leads to a waiting time while a late arrival causes infeasibility. Without loss of generality, we suppose that the loading and service times are equal to zero. The arc set contains arcs that link vertices belonging to different clusters, that is,  $\mathcal{A} = \{(i, j) | i \in \mathcal{C}_k, j \in \mathcal{C}_l, k \neq l, k, l \in \mathcal{K}\}$ . A traveling cost  $C_{ij}$  and a traveling time  $T_{ij}$  are associated with each arc  $(i, j) \in \mathcal{A}$ .

The GVRPTW consists in finding a set of  $M$  vehicle routes on  $\mathcal{G}$  such that the traveling cost is minimized and: (i) every route starts and ends at the depot; (ii) exactly one vertex from each cluster is visited by a single vehicle; (iii) the sum of the customer demands served by the same vehicle does not exceed  $Q$ ; (iv) the service at vertex  $i$  starts during its TW  $[E_i, L_i]$ , and (v) every vehicle leaves and returns to the depot during  $[0, T]$ .

The GVRPTW can be modeled using a set covering formulation. Let  $\Omega$  denote the set of all feasible routes, i.e., all the routes respecting the capacity and time constraints. Let  $W_r$  be the cost

of route  $r \in \Omega$  and let  $A_{kr}, \forall k \in \mathcal{K} \setminus \{0\}, r \in \Omega$  indicate whether cluster  $k$  is visited on route  $r$  ( $A_{kr} = 1$ ) or not ( $A_{kr} = 0$ ). This formulation makes use of binary variables  $z_r, \forall r \in \Omega$ , that indicate whether route  $r$  is selected or not in the solution. The set covering formulation of the GVRPTW is as follows:

$$\text{minimize} \quad \sum_{r \in \Omega} W_r z_r \quad (1)$$

$$\text{s.t.} \quad \sum_{r \in \Omega} A_{kr} z_r \geq 1 \quad \forall k \in \mathcal{K} \setminus \{0\}, \quad (2)$$

$$\sum_{r \in \Omega} z_r \leq M, \quad (3)$$

$$z_r \in \mathbb{N} \quad \forall r \in \Omega. \quad (4)$$

The objective function (1) minimizes the overall delivery costs. Constraints (2) ensure that each cluster is visited at least once. Constraints (3) ensure that the number of routes in the solution is smaller than the size of the fleet. Constraints (4) are the variable definitions.

Note that the definition of the GVRPTW requires that each cluster is visited exactly once, while Constraints (2) require that each cluster is visited at least once. On the one hand, if the cost structure satisfies the triangle inequality, it is never optimal to visit a cluster twice, i.e., all optimal solutions are such that  $z_r \leq 1, \forall r \in \Omega$ . On the other hand, in the LP relaxation of the set covering model, the dual variables associated with Constraints (2) are non-negative, which typically leads to a faster convergence of the column generation procedure.

## 4 A column generation based heuristic for the GVRPTW

In this section we present the matheuristic to tackle the GVRPTW which is based on the set covering formulation presented in Section 3. This formulation relies on an exponential number of variables that represent all the feasible routes  $\Omega$ , however, generating all feasible routes is not tractable. Therefore, we maintain a subset  $\Omega_1$  of  $\Omega$  that is iteratively populated along the algorithm, referred as the route pool. A restricted version of the set covering model is solved considering the route pool only. All the routes inserted into the pool are optimized by applying a procedure described in Section 4.1. Given a set of clusters to be visited, the route optimization procedure seeks for the best sequence of clusters, and the best location to visit in each cluster, to minimize the cost of the route. The management of the route pool is detailed in Section 4.7.

The algorithm has two phases. Phase 1 aims: 1) to construct feasible solutions, and 2) to initialize the pool with promising routes. Phase 1 relies on a construction heuristic (Section 4.3) that embeds the route optimization procedure. If the construction heuristic finds a feasible solution, the set covering model is solved in the hope of finding a good combination of the routes generated so far and improving the current best solution. The solution obtained after solving the set covering model is improved by applying local search moves (see Section 4.5). The above procedure is repeated until the stopping criterion of Phase 1 is reached.



Phase 2 exploits dual information of the linear programming (LP) relaxation of the set covering model to generate additional routes. In particular, the LP relaxation is solved on the current route pool. Based on the dual variables, negative reduced cost routes are generated and added to the pool. Then, the set covering model is solved and the local search method is applied, as in Phase 1, to improve the current best solution. This procedure is repeated until the stopping criterion of Phase 2 is reached.

In the following, we introduce the main components of the proposed heuristic, i.e., the route optimization procedure in Section 4.1, the construction heuristic in Section 4.3, the local search procedure in Section 4.5, and the negative reduced cost route generation in Section 4.6.

#### 4.1 The route optimization procedure

In the following, we explain how the route optimization procedure works to optimize a route visiting a set of clusters. This procedure is adapted from the method proposed in Yuan et al. (2020) to obtain an initial solution for the branch-and-cut algorithm that is proposed for the GTSPTW. This procedure can also be viewed as an extension, by taking into account time windows, of the refinement procedure *RP2* proposed in Fischetti et al. (1997) for the GTSP.

The route optimization procedure works as follows. Let  $\tilde{\mathcal{K}} \subset \mathcal{K} \setminus \{0\}$  be the set of  $n = |\tilde{\mathcal{K}}|$  different clusters to be visited. We are seeking for a feasible route with a minimum cost that visits all these  $n$  clusters. A procedure to generate sequences of clusters is repeated  $N_{seq}$  times. Starting from an empty sequence, the sequence of clusters is iteratively generated based on the best insertion concept. Given a sequence  $(h_1, \dots, h_p)$  of  $p$  ( $p \leq n$ ) different clusters of  $\tilde{\mathcal{K}}$  to visit, a labeling algorithm is applied to determine the optimal locations to be visited for each cluster of the sequence  $(h_1, \dots, h_p)$ . Note that Reyes et al. (2017) proposed an efficient dynamic programming to optimize the route visiting a fixed sequence of clusters with non-overlapping TWs. Since in the GVRPTW, the TWs overlap, the following labeling algorithm is developed.

Given a sequence  $(h_1, \dots, h_p)$  of  $p$  different clusters of  $\tilde{\mathcal{K}}$  to visit, a layered network (LN) is constructed as depicted in Figure 2. This network has  $p+2$  layers corresponding to clusters  $\mathcal{C}_{h_0} = \mathcal{C}_0$ ,  $\mathcal{C}_{h_1}, \dots, \mathcal{C}_{h_p}$ ,  $\mathcal{C}_{h_{p+1}} = \mathcal{C}_0$ , with their respective vertices. Clusters  $\mathcal{C}_{h_0}$  and  $\mathcal{C}_{h_{p+1}}$  both represent the depot. The LN contains all arcs  $(i, j)$  such that  $E_i + T_{ij} \leq L_j$ ,  $i \in \mathcal{C}_{h_f}$ ,  $j \in \mathcal{C}_{h_{f+1}}$ ,  $f = 0, \dots, p$ . The objective is to find a path in the LN that starts at  $\mathcal{C}_{h_0}$  and arrives at  $\mathcal{C}_{h_{p+1}}$  visiting exactly one vertex in each layer, i.e., one vertex from each cluster. If a vertex is visited, the visit must take place during its TW. The solution can be obtained by determining the shortest path with TWs from  $\mathcal{C}_{h_0}$  to  $\mathcal{C}_{h_{p+1}}$ .

To compute the shortest path with TWs on the LN, a labeling algorithm is applied. A label  $L_i$  associated with a vertex  $i$  consists of a pair  $(c_i, t_i)$  representing respectively the cost and service time of a feasible partial path that starts at  $\mathcal{C}_{h_0}$  and arrives at vertex  $i$ . Let  $\mathcal{L}(i)$  be the set containing all the labels associated with vertex  $i$ . Suppose that  $\mathcal{C}_{cur}$  is the current cluster and  $\mathcal{C}_{pre}$  is the previous one. First, we calculate the label set  $\mathcal{L}(i)$ , for all  $i \in \mathcal{C}_{cur}$  by extending labels in  $\mathcal{L}(j)$ , for all  $j \in \mathcal{C}_{pre}$ . Extending a label  $L_j \in \mathcal{L}(j)$  towards a vertex  $i \in \mathcal{C}_{cur}$  consists in creating another label  $L_i \in \mathcal{L}(i)$

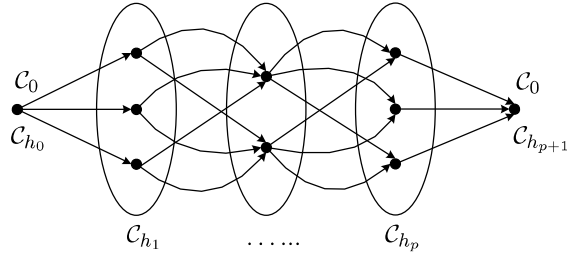


Figure 2: The layered network.

such that:

$$c_i = c_j + C_{ij}, \quad (5)$$

$$t_i = \max\{E_i, t_j + T_{ji}\}. \quad (6)$$

If  $t_i > L_i$ , the partial path associated with the label is infeasible and this label is then discarded. In order to make the algorithm efficient, we only keep non-dominated labels. We say that a label  $L_i^1$  dominates a label  $L_i^2$  if and only if  $c_i^1 \leq c_i^2$  and  $t_i^1 \leq t_i^2$ . It is easy to see that extending  $L_i^1$  across the same arcs toward the last vertex of the LN would always produce a better solution than extending  $L_i^2$  in the same way.

Hence, given a fixed visiting sequence of clusters  $(h_1, \dots, h_p)$  of cluster set  $\tilde{\mathcal{K}}$ , by applying the labeling algorithm, we can get its corresponding optimal solution. Then, in order to generate good visiting sequences of clusters, we develop a cluster sequence construction procedure based on the best insertion concept. The initial sequence is empty, hence the corresponding LN contains two layers:  $\mathcal{C}_{h_0} = \mathcal{C}_{h_1} = \mathcal{C}_0$ . Then, at each step, we randomly pick a cluster from  $\tilde{\mathcal{K}}$  that is not yet inserted into the sequence. Suppose the current sequence is  $(h_1, \dots, h_p), p < n$ , and cluster  $\mathcal{C}_{h_i}$  is chosen to be inserted next. It is obvious that there are  $p + 1$  possible insertion positions for index  $h_i$  into the sequence. To determine the best insertion position, the labeling algorithm described above is applied  $p + 1$  times, one for each possible insertion. The sequence that is kept is the one that provides the lowest cost, if such sequence exists. If not, the sequence construction procedure is stopped. The cluster insertion procedure is repeated until  $p = n$  to obtain a feasible solution visiting all the clusters in  $\tilde{\mathcal{K}}$ . In this process, the labeling algorithm is applied at most  $n(n + 1)/2$  times.

The sequence construction procedure is repeated  $N_{seq}$  times and the best solution is recorded. Note that we always keep the current best solution during the process. During the labeling algorithm, if a label  $L_i$  has a cost greater than the cost of the current best solution, then label  $L_i$  is discarded. Moreover, note that if we provide an initial feasible sequence of clusters to the route optimization procedure, the current best solution is then initialized with this sequence.

Preliminary experiments lead us to set parameter  $N_{seq} = 30$ . Note that our algorithm, in the worst case, calls  $n(n + 1)N_{seq}/2$  times the labeling algorithm. When  $n = |\tilde{\mathcal{K}}| = 5$ , the number of all possible cluster sequences is 120, which is less than  $6(6 + 1)30/2 = 630$ . It is then more efficient to enumerate all the sequences and to compute the shortest path with TWs on each of them. In

this case, the route optimization procedure provides an optimal route that visits the clusters in  $\tilde{\mathcal{K}}$ . When  $|\tilde{\mathcal{K}}| = 6$ , the number of all possible sequences is 720, which is just a little greater than  $6(6 + 1)30/2 = 630$ . Therefore, when  $|\tilde{\mathcal{K}}| < 7$ , we choose to proceed with complete enumeration, otherwise we apply the algorithm described above.

## 4.2 Speed-up the route optimization procedure

Since the route optimization procedure need to be called many times through the heuristic, we use dedicated data structure to speed-up computation. In particular, in order to avoid to repeat computations we maintain two pools of cluster sets:

- $\mathcal{CS}_{opt}$  that contains all the cluster sets on which the route optimization procedure has already found a feasible route;
- $\mathcal{CS}_{infea}$  that contains all the cluster sets on which the route optimization procedure has failed to find a feasible solution.

Note that for every route  $r$  in the route pool  $\Omega_1$ , the corresponding cluster set  $\mathcal{K}_r$  is in  $\mathcal{CS}_{opt}$ .

During the construction heuristic (see Section 4.3), every time before calling the route optimization procedure for a cluster set  $\mathcal{K}_r$ , we first detect if the computation has been already performed in the previous iterations. Thus, we check if: 1)  $\mathcal{K}_r$  is in  $\mathcal{CS}_{opt}$ , 2)  $\mathcal{K}_r$  is in  $\mathcal{CS}_{infea}$ , 3) one of the cluster set in  $\mathcal{CS}_{infea}$  is the subset of  $\mathcal{K}_r$ . If condition 1) is true, the best route visiting the clusters in  $\mathcal{K}_r$  is already in  $\Omega_1$ , and we can retrieve the corresponding route. If condition 2) or 3) is true, we know that the route optimization procedure did not find a feasible route that can visit this cluster set. If none of the three conditions is true, then we apply the route optimization procedure to determine the best route visiting the cluster set  $\mathcal{K}_r$ . Note that in order to be efficient, we use hashing techniques for this implementation.

For the calls to the route optimization procedure outside the construction heuristic, we just detect if the cluster set is already in  $\mathcal{CS}_{opt}$ . If yes, we just get the corresponding route; otherwise, the route optimization procedure is called.

## 4.3 The construction heuristic

The construction heuristic we developed falls into the category of the parallel insertion heuristics. The number of available vehicles at the depot is denoted as  $M$ . The proposed heuristic iteratively constructs a set of at most  $M$  feasible routes that serve all the customers. Let us denote by  $\mathcal{R} = \{r_1, r_2, \dots, r_M\}$  the potentially partial routes that we are iteratively constructing. Let  $\mathcal{K}_r$  be the index set of customers/clusters served by each route  $r \in \mathcal{R}$ . Hence, at any step of the construction heuristic, we have  $\bigcup_{r \in \mathcal{R}} \mathcal{K}_r \subseteq \mathcal{K}$ , and  $\mathcal{K}_r \cap \mathcal{K}_{r'} = \emptyset$ , for all  $r, r' \in \mathcal{R}$ . At the end of the algorithm, if  $\bigcup_{r \in \mathcal{R}} \mathcal{K}_r = \mathcal{K}$ , we found a feasible solution for the GVRPTW.

First, we choose at most  $M$  customers relatively difficult to serve. These customers are called *pivot* customers and can be identified with different criteria. Three criteria to select pivot customers

are described in Appendix A. For each pivot customer  $k$ , the route optimization procedure (Section 4.1) is called to determine the best route  $r$  that visits only customer  $k$ . This route  $r$  is added to the set of partial routes  $\mathcal{R}$ . If necessary, empty routes are added to  $\mathcal{R}$  so that  $\mathcal{R}$  contains  $M$  partial routes.

Then, an unrouted customers (which is not yet assigned to a route) is selected, as explained hereafter, to be inserted next. For each unrouted customer  $k \in \mathcal{K} \setminus (\bigcup_{r \in \mathcal{R}} \mathcal{K}_r)$ , we try to insert it into all the partial routes of  $\mathcal{R}$ . The insertion cost of customer  $k$  into a route  $r$  is evaluated using the route optimization procedure. This means that the route may deeply change when inserting a customer. Then we have a precise evaluation of the new cost of the route. The advantage of using the route optimization procedure is detailed in Section 4.3.1. The choice of the unrouted customer to insert is performed with a regret strategy. For an unrouted customer  $k$ , the regret is defined as the difference between the best and the second best insertion. Note that if for a customer, there exists only one feasible insertion in all the partial routes of  $\mathcal{R}$ , its regret equals to infinity. Then, the unrouted customer  $k^*$  with the maximum regret is chosen to be inserted in the route  $r \in \mathcal{R}$  with the minimum insertion cost.

The procedure is repeated until all customers are inserted or insertions are not possible anymore due to time and/or vehicle capacity constraints.

#### 4.3.1 Interest in using the route optimization procedure

In the construction heuristic, each time we need to compute the insertion cost of a customer in a route, we apply the route optimization procedure described in Section 4.1. In the following, we illustrate the advantage of using this optimization procedure instead of a classical best insertion in the context of the GVRPTW.

An example is provided in Figure 3. Let us suppose that during the construction heuristic there is a partial route  $r = (0, 2, 4, 7, 0)$  depicted in Figure 3(a), visiting a set of clusters  $\mathcal{K}_r = \{1, 2, 3\}$ . When trying to insert an unrouted cluster  $\mathcal{C}_4$ , the route optimization method can obtain a best route visiting all the clusters in the new set  $\mathcal{K}_{r'} = \mathcal{K}_r \cup \{4\} = \{1, 2, 3, 4\}$  as  $r' = (0, 4, 1, 9, 6, 0)$ , depicted in Figure 3(b). By comparing Figure 3(a) and Figure 3(b), we can see that the visiting sequence of clusters changes from  $(1, 2, 3)$  to  $(2, 1, 4, 3)$ , meanwhile the vertex of each cluster visited in the route  $r'$  may be different from the vertices visited in the previous route  $r$ , e.g., the vertices visited in clusters  $\mathcal{C}_1, \mathcal{C}_3$  change from vertices 2, 7 (in green) to 1, 6 (in red) respectively.

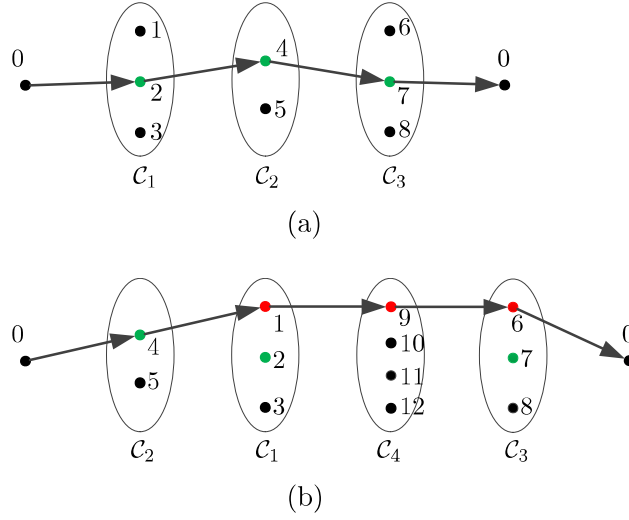


Figure 3: Example of route optimization during the construction heuristic.

In general, in the process of the construction heuristic, when trying to insert an unrouted cluster  $C_k$  into an incumbent route  $r$ , the route optimization is used to identify the new best route  $r'$  visiting clusters in  $\mathcal{K}_r \cup \{k\}$ . Then, the cost change when inserting  $C_k$  into  $r$  is  $\Delta C = W_{r'} - W_r$ , where  $W_{r'}$  and  $W_r$  are the costs of route  $r'$  and  $r$  respectively.

#### 4.4 Recovering infeasibility

When the set covering model is solved on the set of routes  $\Omega_1$ , the solution obtained may visit some clusters more than once. This may occur since if a route  $r$  visiting a set of clusters  $\mathcal{K}_r$  is added to the route pool  $\Omega_1$ , there is no guarantee that all the routes visiting subsets of  $\mathcal{K}_r$  are also in  $\Omega_1$ . However, a solution where a cluster is visited more than once is not a feasible solution according to the definition of the GVRPTW.

When this occurs, we eliminate the repeated visits of clusters. Let us note  $\Omega_{SC}$  the set of routes in the solution of the set covering model solved with the route set  $\Omega_1$ . Let  $\mathcal{K}'$  be the set of clusters served in more than one route in set  $\Omega_{SC}$ . Let us note  $\Omega_{SP}$  a set of routes generated from the routes in  $\Omega_{SC}$  as follows. First,  $\Omega_{SP}$  is initialized with all the routes in  $\Omega_{SC}$ . For each route  $r$  in  $\Omega_{SC}$ , let  $\mathcal{K}_r$  be the set of clusters visited in  $r$ , and  $\tilde{\mathcal{K}}_r = \mathcal{K}_r \cap \mathcal{K}'$  the set containing the repeated clusters. Then, for each subset of clusters  $\mathcal{S} \subseteq \tilde{\mathcal{K}}_r$ , we consider the cluster set  $\mathcal{K}_r \setminus \mathcal{S}$ . The route optimization procedure is applied to this cluster set  $\mathcal{K}_r \setminus \mathcal{S}$ , and the resulting route is stored in  $\Omega_{SP}$ . Note that, by only keeping vertices in  $r$  belonging to clusters in  $\mathcal{K}_r \setminus \mathcal{S}$ , we may obtain a feasible route  $\tilde{r}$  visiting cluster set  $\mathcal{K}_r \setminus \mathcal{S}$ . This route  $\tilde{r}$  can be used as an initial solution for the route optimization procedure.

To obtain a feasible solution for the GVRPTW, we solve the set covering model where we replace Constraints (2) by  $\sum_{r \in \Omega} A_{kr} z_r = 1, \forall k \in \mathcal{K} \setminus \{0\}$ . This makes the set covering model a set partitioning model. The new model is then solved on  $\Omega_{SP}$ . By construction of  $\Omega_{SP}$ , we have the

guarantee to obtain a feasible solution for the GVRPTW.

#### 4.5 Local search

A local search method is applied to improve the current best solution obtained after solving the set partitioning model and recovering infeasibility if necessary (Section 4.4). The GVRPTW differs from classical vehicle routing problems since one customer has multiple locations that can be chosen to be visited. In order to improve the flexibility of the local search moves for the GVRPTW, we adopt the so-called enhanced insertion and deletion concept proposed by Reyes et al. (2017), and extend it to the swap move.

Figure 4 gives an example of the enhanced insertion. A part of an original route is depicted in Figure 4(a). The vertices visited on the route are colored in green. When we try to insert a new cluster  $\mathcal{C}_u$  between two consecutive clusters  $\mathcal{C}_{h_k}$  and  $\mathcal{C}_{h_{k+1}}$ , a new route can be obtained using the enhanced insertion, as depicted in Figure 4(b). The dotted line in Figure 4(b) indicates the new route if the classical insertion is used. By comparison, we can see that the enhanced insertion allows to change the visited vertices in the predecessor cluster  $\mathcal{C}_{h_k}$  and in the successor cluster  $\mathcal{C}_{h_{k+1}}$  (from green vertices to red vertices).

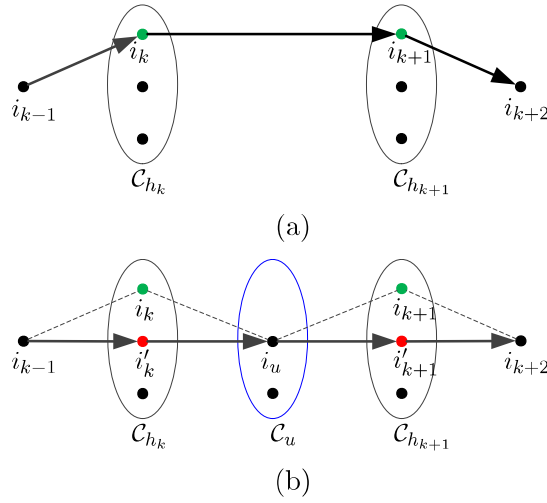


Figure 4: An example of enhanced insertion.

The enhanced moves are based on the concept of effective time window. Assume that we have a feasible fixed sequence of clusters  $(h_1, \dots, h_p)$  and a corresponding incumbent feasible route  $r = (i_0, i_1, \dots, i_p, i_{p+1})$ , where  $i_0 = 0$ ,  $i_k \in \mathcal{C}_{h_k}$ ,  $\forall k \in \{1, \dots, p\}$ , and  $i_{p+1} = 0$  is a duplication of the depot. For each  $i_k$  in  $r$ , we compute its effective TW,  $[\xi_{i_k}^r, \zeta_{i_k}^r]$ , specifying the earliest and latest times that the vertex  $i_k$  can be visited on this route. Given a route  $r$ , these effective TWs are computed using the following recursion rules:

$$\xi_{i_0}^r = E_{i_0}, \xi_{i_k}^r = \max\{E_{i_k}, \xi_{i_{k-1}}^r + T_{i_{k-1}i_k}\} \quad \forall k \in \{1, \dots, p+1\}, \quad (7)$$

$$\zeta_{i_{p+1}}^r = L_{i_{p+1}}, \zeta_{i_k}^r = \min\{L_{i_k}, \zeta_{i_{k+1}}^r - T_{i_k i_{k+1}}\} \quad \forall k \in \{0, \dots, p\}. \quad (8)$$



In addition, for each cluster we also keep similar TWs for every vertex  $i \in \mathcal{C}_{h_k} \setminus \{i_k\}$ :

$$\xi_i^r = \max\{E_i, \xi_{i_{k-1}}^r + T_{i_{k-1}i}\}, \quad (9)$$

$$\zeta_i^r = \min\{L_i, \zeta_{i_{k+1}}^r - T_{ii_{k+1}}\}. \quad (10)$$

Assuming  $\xi_i^r \leq \zeta_i^r$ , this TW represents the earliest and latest times to visit  $\mathcal{C}_{h_k}$  if the visit happens at the alternate vertex  $i$  instead of  $i_k$ , with the other vertices of the route unchanged.

With these effective TWs available, we can check whether a new customer  $u$  can be inserted on route  $r$  at location  $i_u \in \mathcal{C}_u$  between customers  $h_k$  and  $h_{k+1}$  while simultaneously switching customer  $h_k$  to delivery location  $i \in \mathcal{C}_{h_k}$  and customer  $h_{k+1}$  to delivery location  $j \in \mathcal{C}_{h_{k+1}}$  if

$$\max\{E_{i_u}, \xi_i^r + T_{ii_u}\} \leq \min\{L_{i_u}, \zeta_j^r - T_{i_u j}\}. \quad (11)$$

Accordingly, the cost increase of the enhanced insertion  $\Delta C(h_k, i_u, h_{k+1})$  are computed as follows. Given the incumbent route  $r = (0, i_1, \dots, i_k, i_{k+1}, \dots, i_p, 0)$ , suppose  $i'_k \in \mathcal{C}_{h_k} \setminus \{i_k\}$ ,  $i'_{k+1} \in \mathcal{C}_{h_{k+1}} \setminus \{i_{k+1}\}$ . The reader can refer to Figure 4 for an easier understanding of the computation. When a feasible insertion is obtained without changing the predecessor or successor,

$$\Delta C(h_k, i_u, h_{k+1}) = C_{i_k i_u} + C_{i_u i_{k+1}} - C_{i_k i_{k+1}}. \quad (12)$$

When a feasible insertion is obtained by only changing the predecessor  $i_k$  to  $i'_k$ ,

$$\Delta C(h_k, i_u, h_{k+1}) = C_{i_{k-1} i'_k} + C_{i'_k i_u} + C_{i_u i_{k+1}} - C_{i_{k-1} i_k} - C_{i_k i_{k+1}}. \quad (13)$$

When a feasible insertion is obtained by only changing the successor  $i_{k+1}$  to  $i'_{k+1}$ ,

$$\Delta C(h_k, i_u, h_{k+1}) = C_{i_k i_u} + C_{i_u i'_{k+1}} + C_{i'_{k+1} i_{k+2}} - C_{i_k i_{k+1}} - C_{i_{k+1} i_{k+2}}. \quad (14)$$

When a feasible insertion is obtained by changing both the predecessor  $i_k$  to  $i'_k$  and the successor  $i_{k+1}$  to  $i'_{k+1}$ ,

$$\begin{aligned} \Delta C(h_k, i_u, h_{k+1}) &= C_{i_{k-1} i'_k} + C_{i'_k i_u} + C_{i_u i'_{k+1}} + C_{i'_{k+1} i_{k+2}} \\ &\quad - C_{i_{k-1} i_k} - C_{i_k i_{k+1}} - C_{i_{k+1} i_{k+2}}. \end{aligned} \quad (15)$$

The enhanced deletion is illustrated in Figure 5. It consists in removing a cluster from the current route allowing to change the vertices visited in the predecessor and successor clusters. The cost variation is computed as previously.

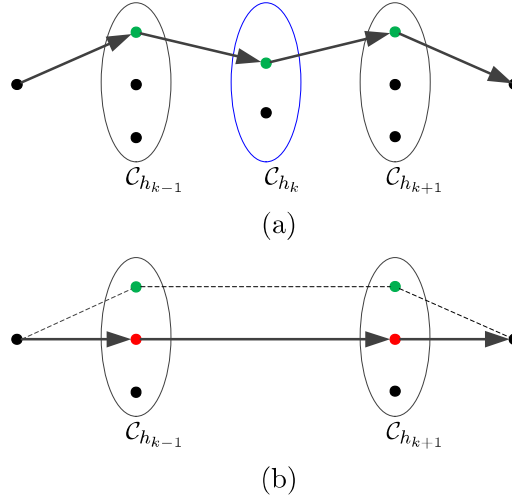


Figure 5: An example of enhanced deletion.

An enhanced relocation is obtained by combining the enhanced insertion and deletion. Moreover, when the relocations of 1) cluster  $C_u$  in the current position of cluster  $C_v$  and 2) cluster  $C_v$  in the current position of cluster  $C_u$  are simultaneously considered, the enhanced swap move is obtained.

In our local search we consider two types of moves: inter-route relocations and swaps. They are sequentially applied to the current best solution. The neighborhoods are fully explored and the best improvement strategy is applied.

Let us indicate by  $r_1$  and  $r_2$  the routes involved in the move, and  $r'_1$  and  $r'_2$  the routes obtained once the move has been applied. Moreover, let us indicate by  $\mathcal{K}_{r_1}$ ,  $\mathcal{K}_{r_2}$ ,  $\mathcal{K}_{r'_1}$ ,  $\mathcal{K}_{r'_2}$  the cluster sets associated with routes  $r_1$ ,  $r_2$ ,  $r'_1$  and  $r'_2$  respectively.

Before computing the cost increase due to the implementation of a move, we first check if  $\mathcal{K}_{r'_1}$  or  $\mathcal{K}_{r'_2}$  are present in  $\mathcal{CS}_{opt}$ . If so, we already know the best route that can be obtained from the corresponding cluster set. Then, we just consider the cost of this best route. If the cluster sets  $\mathcal{K}_{r'_1}$  or  $\mathcal{K}_{r'_2}$  are not present in  $\mathcal{CS}_{opt}$ , the enhanced local search move is conducted, and the cost increase is calculated accordingly. If the total cost increase is negative (the move improves the current solution), then each new route,  $r'_1$  and  $r'_2$ , is improved by applying the route optimization procedure if the corresponding cluster set was not present in  $\mathcal{CS}_{opt}$ .

#### 4.6 Negative reduced cost route generation

Let us first introduce some definitions and notations that will be used in this section. The linear relaxation of the set covering model defined by (1)~(4) is called the master problem (MP). As the size of the set  $\Omega$  grows exponentially with the number of customers  $K$ , the set covering model is usually solved on a subset  $\Omega_1$  of  $\Omega$  and the linear relaxation of the resulting model is called the restricted master problem and denoted as  $RMP(\Omega_1)$ .

Let  $\lambda_k$  be the nonnegative dual variable associated with the visit of cluster  $C_k$  (Constraints (2)), and let  $\lambda_0$  be the nonpositive dual variable associated with the fleet size constraint (Constraint (3)).

The dual program  $D(\Omega)$  of  $MP$  is as follows:

$$\text{maximize} \quad \sum_{k \in \mathcal{K} \setminus \{0\}} \lambda_k + M\lambda_0 \quad (16)$$

$$\text{s.t.} \quad \sum_{k \in \mathcal{K} \setminus \{0\}} A_{kr} \lambda_k + \lambda_0 \leq W_r \quad \forall r \in \Omega, \quad (17)$$

$$\lambda_k \geq 0 \quad \forall k \in \mathcal{K} \setminus \{0\}, \quad (18)$$

$$\lambda_0 \leq 0. \quad (19)$$

The reduced cost of a route  $r \in \Omega$  as a value

$$W_r - \sum_{k \in \mathcal{K} \setminus \{0\}} A_{kr} \lambda_k - \lambda_0. \quad (20)$$

A route associated with a negative reduced cost may reduce the value of the objective function of the  $RMP(\Omega_1)$ . It is then potentially beneficial to introduce such a route in  $\Omega_1$ . To generate routes with negative reduced costs, we proceed as follows.

**Step 1.** For all routes in the route pool  $\Omega_1$ , we compute their corresponding reduced costs. The routes with reduced costs equal to 0 are collected in a set  $\Omega_1^{rc0}$ .

**Step 2.** We apply the deletion, insertion and relocation moves on every route in  $\Omega_1^{rc0}$ . These moves are based on the enhanced insertion and deletion presented in Section 4.5. The routes obtained by applying a move that decreases the reduced cost are stored in a set  $\Gamma_{neg}$ . Let us consider a route  $r \in \Omega_1^{rc0}$  and its corresponding cluster set  $\mathcal{K}_r$ .

- **Deletion.** For each cluster  $k$  visited in  $r$ , we check if the cluster set  $\mathcal{K}_r \setminus \{k\}$  is in  $\mathcal{CS}_{opt}$ . If not, we apply the enhanced deletion of cluster  $k$  from  $\mathcal{K}_r$ . If we obtain a new route  $r'$ , which decreases the reduced cost of  $r$ , we add  $r'$  to set  $\Gamma_{neg}$ .
- **Insertion.** For each cluster  $k$  not visited by route  $r$ , that is,  $k \in \mathcal{K} \setminus \{\mathcal{K}_r \cup \{0\}\}$ , we first check if  $\mathcal{K}_r \cup \{k\}$  is in  $\mathcal{CS}_{opt}$ . If not, we apply the enhanced insertion of cluster  $k$  into all the possible positions in  $r$ . If a new route  $r'$  decreases the reduced cost of  $r$ , we put  $r'$  in  $\Gamma_{neg}$ , and we stop trying to insert cluster  $k$  in the remaining positions of route  $r$ .
- **Relocation.** For each cluster  $k$  that is not visited by route  $r$ , i.e.,  $k \in \mathcal{K} \setminus \{\mathcal{K}_r \cup \{0\}\}$ , and for each cluster  $k'$  visited by route  $r$ , i.e.,  $k' \in \mathcal{K}_r$  we first check if  $\mathcal{K}_r \cup \{k\} \setminus \{k'\}$  is in  $\mathcal{CS}_{opt}$ . If not, we apply the enhanced relocation of cluster  $\mathcal{C}_{k'}$  in the current position of cluster  $\mathcal{C}_k$ . If a new route  $r'$  decreases the reduced cost of  $r$ , we add  $r'$  to  $\Gamma_{neg}$ .

**Step 3.** First, we rank all the routes in set  $\Gamma_{neg}$  in ascendant order with respect to their reduced costs. Moreover, all routes in  $\Gamma_{neg}$  are optimized using the route optimization procedure with the original routes as initial solutions. All these optimized routes are added to  $\Omega_1$ . Then, in order to generate other routes with lower reduced costs, we update  $\Omega_1^{rc0}$  and repeat the above procedure.  $\Omega_1^{rc0}$  is firstly emptied and then filled up with the first  $N_{neg}^{best}$  routes in  $\Gamma_{neg}$ .  $\Gamma_{neg}$  is emptied afterwards.

**Step 4.** We repeat Step 2 and Step 3 for  $Iter_{RC}$  iterations.

#### 4.7 Management of the route pool $\Omega_1$

The route pool  $\Omega_1$  is a subset of the whole route set  $\Omega$ . It is populated in the course of the procedure. More specifically, each time the route optimization procedure (explained in Section 4.1) finds a feasible route, it is inserted in  $\Omega_1$ . This means that, in the construction heuristic, routes obtained during the evaluation of potential cluster insertions are added to  $\Omega_1$  (after a call to the route optimization procedure), even if the insertion is not implemented. Similarly, in the local search method and in the generation of negative reduced cost routes, whenever a move has a negative cost increase, the new routes are added in  $\Omega_1$  (after a call to the route optimization procedure), even if the move is not implemented on the current solution.

#### 4.8 Overall procedure

Here, we explain the overall procedure of the algorithm and provide some details about the stopping criteria for each phase. In Phase 1, we use the construction heuristic (Section 4.3) to build a feasible solution. If a feasible solution is obtained, based on the route pool  $\Omega_1$  built so far, we solve the set covering model to improve the current best solution. A time limit of 30 seconds is set to solve the set covering problem. If the solution visits some clusters more than once, we eliminate the repeated visits to the same cluster (Section 4.4). Then, we apply local search moves (Section 4.5) to improve the current best solution. If the stopping criteria are not reached, the above procedure is repeated. The stopping criterion of Phase 1 depends on two parameters *Iter1* and *nbFeaS*. Phase 1 stops after *Iter1* iterations, or when *nbFeaS* feasible solutions have been obtained from the construction heuristic.

In Phase 2, the LP relaxation of the set covering model based on the route pool  $\Omega_1$  (the restricted master problem  $RMP(\Omega_1)$ ) is solved. Based on the dual information, routes with negative reduced cost are generated (Section 4.6). Then, the set covering model and the local search moves are applied as described in Phase 1 trying to improve the current best solution. If the stopping criterion of Phase 2 is not reached, the above procedure is repeated. Phase 2 stops after *Iter2* iterations, or if the current best solution has not changed in the last *nbNonImprS* iterations, or if no route with negative reduced cost has been found.

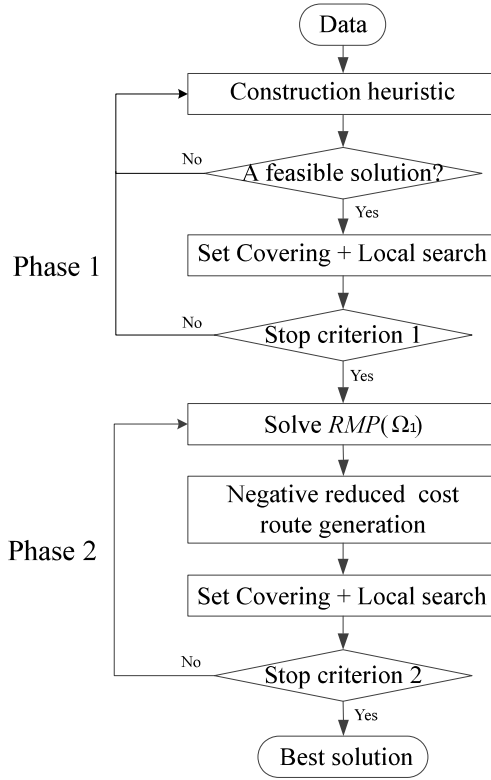


Figure 6: Algorithm scheme

## 5 Computational experiments

In this section, we report the results we obtained with the column generation based heuristic presented in Section 4. The algorithm is implemented in C++ and CPLEX 12.6.3 is used to solve linear programs through Concert Technology. All experiments are performed on a machine with Intel(R) Core(TM) i5-6200U CPU, 2.30GHz, 8G RAM.

### 5.1 Instances

In our computational experiments, we consider different types of instances for different purposes.

- Three sets of instances used by [Ozbaygin et al. \(2017\)](#), with 100 instances in total. The first set includes 40 VRPRDL instances. For these instances, the number of customers ranges from 15 to 120 and the number of vertices from 50 to 471. Note that due to the presence of TW some vertices cannot be reached. Thus, the number of vertices in the instances can be reduced, ranging from 28 to 293. Each customer has at most 5 delivery locations, with the first and last being home location. The TWs associated with these locations are non-overlapping, and the time between windows of two roaming locations of one customer is at least the travel time between them. The second set of instances consists of 40 VRPHRDL (VRP with home and roaming delivery locations) instances. Each VRPHRDL instance is generated from a

VRPRDL instance by keeping one home location for each customer and replacing its TW with the overall time horizon  $[0, T]$ . The third set of instances contains two groups of 10 medium-size VRPRDL instances with 40 customers. The number of vertices ranges from 142 to 174, and it can be reduced due to the presence of TW ranging from 90 to 117.

- To better account the influence of the pick-up points (e.g., lockers, stores), we modify the VRPRDL and VRPHRDL instances as follows. In a VRPRDL/VRPHRDL instance, when a cluster contains more than one location, we randomly choose one of its roaming locations and treat it as a pick-up point. Let us suppose that the TW of this location is  $[E, L]$ , then in the modified instance it is enlarged to  $[0, L]$  due to the nature of the pick-up point.
- We recall that the VRPRDL is characterized by non-overlapping TWs for the locations in the same cluster. Moreover, the time between windows of two locations in the same cluster is at least equal to the travel time between them. In order to analyze the impact of the TWs on the solution cost, we modify the VRPRDL instances as follows. Note that the locations in each cluster are ordered with respect to the increasing values of the opening time of their TWs. Following this order, the TW of the first location in each cluster is not modified. The TWs of the other locations are changed from  $[E, L]$  to  $[\max\{0, E - \delta\}, L]$ . Since the planning horizon in the VRPRDL instances is 720 minutes, we set  $\delta = 30, 60, 90, 120, 150, 180, 240, 300, 360$ . For each VRPRDL instance, we create one instance for each value of  $\delta$ , thus 9 new instances are obtained from one VRPRDL instance. We refer to these new instances as VRPRDL( $\delta$ ) instances. Table 1 shows the percentage ranges of overlapping TWs in the 10 large VRPRDL( $\delta$ ) instances with 120 customers according to different values of  $\delta$ . For example, as shown in the table, when  $\delta = 90$ , between 40% and 50% of the TWs are overlapping.

By making this TW change, we add more flexibility on the delivery planning. From a problem-oriented perspective, a high value of  $\delta$  means that the customer is flexible about when and where to pick up the package. A low value of  $\delta$  increases the likelihood that the customer is actually available at a given location at that time.

Table 1: Percentage ranges of overlapping TWs in 10 large VRPRDL( $\delta$ ) instances.

$\delta$	Overlapping TWs/%
0	0
30	10-20
60	20-40
90	40-50
120	50-70
150	70-80
180	80-90
240	90-100
300	90-100
360	100



- To test the influence of the route length on the solution, i.e., the number of customers that a vehicle can deliver, on the performance of the proposed algorithm, we create instances VRPRDL-variant and VRPHRDL-variant. For an instance of VRPRDL, we first reduce the size of the fleet, the demand of every customer, and the traveling time between every two locations by 2, respectively. Whenever the value is not integer, we round it to the smallest integer. Then we modify the TWs of the locations for each customer. The TW of the first location does not change, as well as the upper bounds of the TWs of all the other locations. The lower bound of a location is set to the upper bound of the previous location in the customer's itinerary plus the traveling time to it. The VRPHRDL-variant is generated similarly to the VRPRDL-variant. For these two sets of instances, the number of vertices ranges from 50 to 471, and it can be reduced due to the presence of TWs ranging from 39 to 392.
- Besides the above instances, in order to test the genericity and flexibility of the algorithm, we perform computational experiments on GVRPTW instances proposed by Moccia et al. (2012) (see Section 5.5) and on benchmark GVRP instances proposed by Bektaş et al. (2011) (see Section 5.6).

## 5.2 Parameters

From the description of the algorithm in Section 4, there are six parameters to set.

For the stop criterion of Phase 1, there are: *Iter1* the maximal number of iterations, and *nbFeaS* the minimal number of feasible solutions to obtain from the construction heuristic. In Phase 1, we set *nbFeaS* = 10 since this usually provides a sufficient large pool of routes of good quality. The resolution of the set covering model may then provide good solutions. On the other hand we set *Iter1* = 100 to avoid consuming too much time when the construction heuristic struggles to find *nbFeaS* feasible solutions.

For the stop criterion of Phase 2, there are: *Iter2* the maximal number of iterations, and *nbNonImprS* the maximal number of consecutively non-improved solutions. Preliminary experiments showed that Phase 1 of the algorithm could already give good quality solutions. Phase 2 exploits the dual information to try to further improve solutions. To this end we set *Iter2* = 10. Preliminary tests showed that this is a good compromise between solution improvement and computational efficiency. We set *nbNonImprS* = 5 to avoid to repeatedly try to improve a solution that is not likely to be ameliorated. In the negative reduced cost routes generation, there are:  $N_{neg}^{best}$  the number of negative reduced cost routes used to iterate the procedure, and *Iter<sub>RC</sub>* the number of iterations of the procedure. In this phase we look for negative reduced cost routes by applying simple local search operators to routes that are already associated with negative reduced cost. Preliminary tests showed that after few iterations it becomes difficult to find new routes with negative reduced cost and it is then beneficial to start Phase 2 again. Thus, we limit  $N_{neg}^{best}$  = 50 and *Iter<sub>RC</sub>* = 3 to iterate the procedure.

In conclusion, we conducted the experiments using *Iter1* = 100, *nbFeaS* = 10, *Iter2* = 10,

$nbNonImprS = 5$ ,  $N_{neg}^{best} = 50$ ,  $Iter_{RC} = 3$ . Note that we use this parameter setting for all types of instances tested below.

### 5.3 Preprocessing

The TW width can be reduced by taking into account the earliest and the latest arrival and departure times at each vertex of the graph from or to another vertex. In particular, we consider the following conditions proposed by Desrochers et al. (1992):

- earliest arrival time from predecessors:  $E_i = \max\{E_i, \min\{L_i, \min_{(j,i) \in \mathcal{A}}(E_j + T_{ji})\}\}$ ;
- earliest departure time to successors:  $E_i = \max\{E_i, \min\{L_i, \min_{(i,j) \in \mathcal{A}}(E_j - T_{ij})\}\}$ ;
- latest arrival time from predecessors:  $L_i = \min\{L_i, \max\{E_i, \max_{(j,i) \in \mathcal{A}}(L_i + T_{ji})\}\}$ ;
- latest departure time to successors:  $L_i = \min\{L_i, \max\{E_i, \max_{(i,j) \in \mathcal{A}}(L_j - T_{ij})\}\}$ .

These conditions are applied iteratively to all vertices until no TW can be reduced.

Moreover, we eliminate from graph  $\mathcal{G}$  vertices and arcs that cannot be part of any feasible solution. To this end, we:

- eliminate a vertex  $i \in \mathcal{V}$  if a round trip from the depot to the vertex, i.e., route  $0 - i - 0$  leads to a time window violation:  $E_0 + T_{0i} > L_i$ , or  $\max\{E_0 + T_{0i}, E_i\} + T_{i0} > L_0$ ;
- eliminate an arc  $(i, j) \in \mathcal{A}$  if it is not possible to go from  $i$  to  $j$ :  $E_i + T_{ij} > L_j$ , or if the route that starts from the depot, visits location  $i$ , then location  $j$  then goes back to the depot (i.e., route  $0 - i - j - 0$ ) violates at least one TW:  $\max\{E_0 + T_{0i}, E_i\} + T_{ij} > L_j$ , or  $\max\{\max\{E_0 + T_{0i}, E_i\} + T_{ij}, E_j\} + T_{j0} > L_0$ .

### 5.4 Computational results on VRPRDL, VRPHRDL instances (Ozbaygin et al. (2017)) and on the modified instances

The results on VRPRDL and VRPHRDL instances are reported in Tables 2 and 3 respectively. We compare the results of the branch-and-price (BP) algorithm presented in Ozbaygin et al. (2017) and the column generation based heuristic (CGBH) proposed in this paper. Instances with 15 to 20 customers are called small instances, instances with 30 to 60 customers are called medium-size instances, while instances with 120 customers are large instances.

In Tables 2 and 3, column *Instance* represents the name of the instance, column *K* is the number of customers, and column *M* is the number of available vehicles at the depot. The next two columns labeled *BP* present the results obtained by the branch-and-price algorithm of Ozbaygin et al. (2017). Column *best-known* is the value of the best solution obtained from different parameter settings of the BP. Column *time/s* is the computation time in seconds to obtain the best solution. Note that it does not include the time for the heuristic of Reyes et al. (2017) to find an initial solution. The time limit has been set to 2 hours for small and medium-size instances and 6 hours

Table 2: Results on VRPRDL instances.

Instance	$K$	$M$	BP		CGBH						
			best-known	time/s	Obj	time/s	GAP/%	nbRoute	minL	maxL	averL
0		5	901	0.26	901	0.18	0	4	2	6	3.8
1		6	1286	0.04	1286	0.13	0	5	2	4	3.0
2	15	5	991	0.07	991	0.16	0	4	3	4	3.8
3		6	1062	0.04	1062	0.11	0	5	1	4	3.0
4		7	1832	0.02	1832	0.14	0	6	1	6	2.5
5		6	1294	1.08	1294	0.42	0	5	2	6	4.0
6		5	1155	2.87	1155	1.04	0	4	3	9	5.0
7	20	7	1455	0.07	1455	0.20	0	6	2	5	3.3
8		6	1260	0.52	1260	0.32	0	5	1	7	4.0
9		8	1684	0.03	1684	0.18	0	7	1	5	2.9
10		8	1922	1.13	1922	1.03	0	7	3	7	4.3
11		9	2324	14.61	2324	0.86	0	8	2	6	3.8
12		8	1747	0.68	1747	0.95	0	6	2	10	5.0
13		7	1273	0.64	1273	1.28	0	6	2	6	5.0
14		7	1694	0.50	1694	0.78	0	6	3	7	5.0
15	30	8	1938	0.75	1938	0.69	0	7	1	7	4.3
16		9	1965	0.73	1965	1.86	0	8	1	11	3.8
17		8	1827	0.23	1827	0.43	0	7	2	6	4.3
18		9	2083	11.13	2083	0.83	0	7	2	7	4.3
19		8	1822	1.53	1822	1.09	0	6	1	8	5.0
20		14	3761	4.13	3761	3.87	0	13	2	7	4.6
21		11	2828	10.74	2828	5.82	0	10	3	8	6.0
22		17	4440	1.10	4440	1.78	0	16	2	5	3.8
23		13	3378	11.62	3378	4.81	0	11	2	8	5.5
24		13	3161	643.79	3161	7.46	0	11	2	9	5.5
25	60	17	4536	1.87	4536	3.03	0	16	1	8	3.8
26		11	2865	7.08	2865	6.05	0	10	2	8	6.0
27		15	4173	43.90	4173	6.19	0	14	1	8	4.3
28		16	3964	38.25	3964	3.42	0	14	2	7	4.3
29		15	4107	1.80	4107	2.11	0	14	2	7	4.3
30		19	4935	1629.82	4935	53.79	0	17	2	13	7.1
31		21	5278	21600.00	5267	130.64	<b>-0.21</b>	18	3	13	6.7
32		19	5083	21600.00	5061	69.09	<b>-0.43</b>	18	1	11	6.7
33		20	5218	8547.16	5218	43.43	0	17	5	11	7.1
34	120	22	5519	21600.00	5500	62.62	<b>-0.34</b>	20	1	14	6.0
35		25	6498	168.13	6498	36.31	0	22	1	9	5.5
36		20	4845	21600.00	4830	38.94	<b>-0.31</b>	17	2	11	7.1
37		21	5608	21600.00	5605	51.76	<b>-0.05</b>	21	1	12	5.7
38		24	5849	21600.00	5848	61.27	<b>-0.02</b>	20	2	9	6.0
39		21	5048	21600.00	5006	73.31	<b>-0.83</b>	18	2	11	6.7
Average			<b>5388.10</b>	<b>16154.51</b>	<b>5376.80</b>	<b>62.11</b>	<b>-0.22</b>	<b>18.8</b>	<b>2.0</b>	<b>11.4</b>	<b>6.4</b>

Table 3: Results on VRPHRDL instances.

Instance	$K$	$M$	BP		CGBH						
			best-known	time/s	Obj	time/s	GAP/%	nbRoute	minL	maxL	averL
0		5	773	0.62	773	0.34	0	3	2	7	5.0
1		6	1065	0.08	1065	0.15	0	4	2	6	3.8
2	15	5	988	0.11	988	0.33	0	3	3	8	5.0
3		6	914	0.17	914	0.33	0	3	4	6	5.0
4		7	1710	0.04	1710	0.09	0	6	1	5	2.5
5		6	1099	2.84	1099	0.95	0	4	2	7	5.0
6		5	996	11.02	996	1.47	0	3	4	12	6.7
7	20	7	1346	0.33	1346	0.29	0	5	1	6	4.0
8		6	997	0.56	997	0.65	0	4	1	7	5.0
9		8	1166	0.18	1166	0.30	0	4	2	8	5.0
10		8	1587	8.54	1587	1.76	0	5	5	9	6.0
11		9	1808	4.7	1808	1.35	0	6	3	9	5.0
12		8	1563	3.38	1563	1.92	0	6	1	10	5.0
13		7	1058	3.26	1058	2.21	0	4	6	9	7.5
14	30	7	1347	155.93	1347	3.49	0	5	3	8	6.0
15		8	1517	7200.00	1517	2.43	0	5	1	9	6.0
16		9	1445	2.14	1445	2.06	0	5	4	10	6.0
17		8	1627	26.67	1627	2.15	0	5	5	7	6.0
18		9	1461	1.59	1461	1.27	0	5	1	11	6.0
19		8	1715	2.09	1715	1.78	0	6	1	8	5.0
20		14	2580	396.47	2580	12.48	0	8	4	9	7.5
21		11	2213	7200.00	2207	35.37	<b>-0.27</b>	7	5	14	8.6
22		17	3363	194.98	3363	5.16	0	10	3	8	6.0
23		13	2569	7200.00	2569	17.72	0	8	4	14	7.5
24	60	13	2400	7200.00	2378	22.45	<b>-0.92</b>	8	1	13	7.5
25		17	2845	7200.00	2845	10.91	0	9	3	10	6.7
26		11	2518	33.85	2518	10.33	0	8	4	11	7.5
27		15	2758	3392.94	2758	26.33	0	8	3	15	7.5
28		16	2892	7200.00	2892	15.53	0	9	4	10	6.7
29		15	2691	41.77	2691	6.91	0	8	5	11	7.5
30		19	3984	21600.00	3666	123.60	<b>-7.98</b>	12	1	16	10.0
31		21	3958	21600.00	3897	308.24	<b>-1.54</b>	14	1	15	8.6
32		19	3630	21600.00	3554	319.05	<b>-2.09</b>	12	1	15	10.0
33		20	3891	21600.00	3701	194.26	<b>-4.88</b>	12	1	18	10.0
34	120	22	3255	21600.00	3174	277.10	<b>-2.49</b>	10	3	16	12.0
35		25	4525	21600.00	4251	267.24	<b>-6.06</b>	13	1	14	9.2
36		20	3395	21600.00	3218	153.57	<b>-5.21</b>	10	10	16	12.0
37		21	3976	21600.00	3935	241.12	<b>-1.03</b>	14	1	13	8.6
38		24	4316	21600.00	4313	132.94	<b>-0.07</b>	17	1	15	7.1
39		21	3680	21600.00	3586	392.87	<b>-2.55</b>	11	5	17	10.9
Average			<b>3861.00</b>	<b>21600.00</b>	<b>3729.50</b>	<b>241.00</b>	<b>-3.39</b>	<b>12.5</b>	<b>2.5</b>	<b>15.5</b>	<b>9.8</b>

for large instances. If the computation time for one instance is less than the time limit, then it means that the instance was solved to optimality by the BP algorithm. The next columns labeled *CGBH* show the results obtained using the CGBH proposed in this work. Column *obj* represents the objective value obtained, and *time/s* is the computation time in seconds. In the column *GAP/%* we provide the relative gap in percentage between the results obtained from the CGBH and the best-known results provided by the BP algorithm of [Ozbaygin et al. \(2017\)](#). It is calculated as  $GAP/\% = 100\% \times (obj - best-known)/best-known$ . Column *nbRoute* represents the number of routes in the best solution obtained by the CGBH. Columns *minL* and *maxL* represent the length of the shortest and longest route in the best solution respectively. Column *averL* is the average length of all the routes in the best solution. Here, the length of a route corresponds to the number of vertices visited on the route. Note that in the last rows of Tables 2 and 3 denoted as *Average*, we show the average results restricted to the 10 large instances.

Table 2 shows that the proposed CGBH is able to obtain the optimal values for all the small and medium-size instances in less than 10 seconds. For the large instances that were solved to optimality by the BP algorithm (instance 30, 33 and 35), the CGBH gets the optimal values in less than 1 minute. For the large instances that were not solved to optimality, the CGBH always improve the best-known values. For the ten large instances, the CGBH is able to improve by 0.22% the solutions found by the BP algorithm, using around one minute of computation time.

Table 3 shows that the CGBH can get the optimal values or even improve the best-known results for all the instances. For instances 30, 35, and 36, the CGBH improves the best-known values by 7.98%, 6.06% and 5.21% respectively. For the ten large instances, the CGBH improves the best-known results by 3.39% on average using less than 5 minutes of computation time.

In conclusion, the proposed CGBH is very efficient. It is able to obtain very high-quality solutions within short computation times.

Furthermore, from Tables 2 and 3, we can see that the average costs of the 10 large VRPRDL and VRPHRDL instances with 120 customers are 5376.80 and 3729.50 respectively. This shows that the combination of home delivery and in-car delivery is cost-effective.

The results on the variants of the 10 large VRPRDL and VRPHRDL instances with 120 customers considering pick-up points are reported in Table 4 (see Section 5.1 for the generation of these instances). From results presented in Tables 2, 3 and 4, we can see that the solutions of the VRPRDL instances are more costly than those for the VRPRDL + pick up points instances. The average delivery cost decreases when home delivery is available (VRPHRDL) while the cheapest solutions are obtained when all delivery options are available (VRPHRDL + pick up points). As expected, adding flexibility reduces the average delivery costs. Our results show the benefits for delivery companies to propose multiple delivery options to their customers.

Next, we study the impact of the TWs by analyzing the solutions of the 10 large VRPRDL( $\delta$ ) instances (see Section 5.1 for the generation of these instances). Figure 7 shows the evolution of the average solution cost for the 10 large VRPRDL( $\delta$ ) instances with different  $\delta$  values. As can be observed in the figure, the average cost gradually decreases when  $\delta$  increases. Specifically, when

Table 4: Results on variants of the VRPRDL and VRPHRDL considering pick-up points.

Instance	VRPRDL + pick-up points		VRPHRDL + pick-up points	
	Obj	time/s	Obj	time/s
30	4594	101.90	3533	210.29
31	4633	222.86	3792	274.56
32	4538	83.43	3508	316.12
33	4791	67.12	3596	239.20
34	4457	146.88	3036	267.19
35	6301	37.00	4094	118.12
36	4432	60.13	3063	264.26
37	4704	56.62	3565	283.81
38	4622	304.62	3722	163.94
39	4260	290.94	3259	413.78
<b>Average</b>	<b>4733.20</b>	<b>137.15</b>	<b>3516.80</b>	<b>255.13</b>

$\delta = 90$ , the average cost is 4173.3, a 22.4% reduction from the average cost of the original VRPRDL instances with  $\delta = 0$ . When  $\delta = 360$ , the average cost is 3360.0 which is a 37.5% reduction.

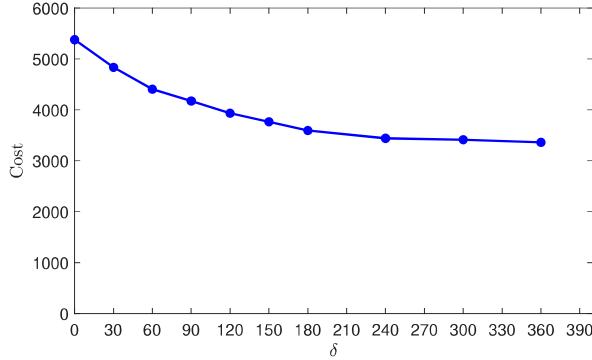


Figure 7: Evolution of the average solution cost for the 10 large VRPRDL( $\delta$ ) instances.

In addition, we analyze the locations selected to visit the customers in the best known solutions. The computational results for the 10 large VRPRDL instances with 120 customers are considered as a reference. First of all, we compute the following three values for each instance:

$$d_{\min} = \frac{1}{K} \sum_{k=1}^K \min_{i \in C_k} \{D_{0i}\}, \quad d_{\text{avg}} = \frac{1}{K} \sum_{k=1}^K \left( \frac{1}{|C_k|} \sum_{i \in C_k} D_{0i} \right), \quad d_{\text{sol}}(y) = \frac{1}{K} \sum_{k=1}^K \sum_{j \in C_k} D_{0j} y_j. \quad (21)$$

$d_{\min}$  is the mean of the shortest distances from the depot to every cluster and  $d_{\text{avg}}$  is the mean of the average distances from the depot to every cluster. Let  $y$  be a vector such that  $y_j$  equals 1 if and only if the location  $j$  is visited in the current solution  $j \in \mathcal{V} \setminus \{0\}$ . Thus,  $d_{\text{sol}}$  is the mean of distances from the depot to the selected locations in the solution.  $d_{\min}$  and  $d_{\text{avg}}$  are measures that depend on the instance itself, while  $d_{\text{sol}}$  is a measure that depends on a solution of the instance. In addition, we calculate the percentages of customers served at the first, second, and third closest location from the



depot, respectively denoted as  $p(1^{st}), p(2^{nd}), p(3^{rd})$ .  $p(\text{other})$  is the percentage of visited locations that are not among the three closest to the depot, i.e.,  $p(\text{other}) = 100\% - p(1^{st}) - p(2^{nd}) - p(3^{rd})$ . The results are presented in Table 5. It can be seen that for all the instances, we have  $d_{\min} < d_{\text{sol}} < d_{\text{avg}}$  and  $p(1^{st}) > p(2^{nd}) > p(3^{rd})$ . On average, 65% of the customers are visited in their closest location to the depot and 19% of the customers are visited in their second closest location to the depot.

Table 5: Results related to the locations selected to visit in the solution.

Instance	$d_{\min}$	$d_{\text{avg}}$	$d_{\text{sol}}$	$p(1^{st})/\%$	$p(2^{nd})/\%$	$p(3^{rd})/\%$	$p(\text{other})/\%$
30	63.3	81.2	73.4	66.7	22.5	4.2	6.7
31	70.6	88.3	79.9	65.8	15.0	11.7	7.5
32	59.6	77.1	71.7	55.0	28.3	10.8	5.8
33	68.4	86.2	76.8	74.2	14.2	3.3	8.3
34	62.3	82.5	75.3	61.7	17.5	10.0	10.8
35	77.8	94.7	89.6	63.3	17.5	12.5	6.7
36	67.4	89.3	80.7	60.0	21.7	11.7	6.7
37	74.0	91.6	81.0	67.5	20.0	6.7	5.8
38	73.4	92.2	84.2	65.8	18.3	9.2	6.7
39	67.1	86.9	75.4	70.0	15.0	9.2	5.8
<b>Average</b>	<b>68.4</b>	<b>87.0</b>	<b>78.8</b>	<b>65.0</b>	<b>19.0</b>	<b>8.9</b>	<b>7.1</b>

In the following, we analyze the influence of the route length on the solution. The last four columns of Tables 2 and 3, i.e., columns  $nbR$ ,  $minL$ ,  $maxL$ ,  $averL$ , report average statistics on the lengths of the routes in the best solutions. From column  $maxL$  in Table 2 and 3, it can be seen that the maximum number of customers that a vehicle serves are 14 and 18 for the VRPRDL and VRPHRDL instances respectively. By comparing the average results for the large instances, it can be seen that the average length of routes in the best solution is 6.4 and 9.8 for VRPRDL and VRPHRDL instances respectively. In general, the number of customers that one vehicle serves is not large enough compared with real-life cases. In real life, one courier could make around 30 deliveries per day. In order to be more consistent with real-life cases and enable one vehicle to deliver more customers, we test the CGBH algorithm on the VRPRDL-variant and VRPHRDL-variant instances generated in Section 5.1, and the results are reported in Table 6.

In Table 6, from the average results, for the VRPRDL-variant and VRPHRDL-variant instances with 30 customers, the CGBH procedure takes less than half a minute and 1 minute respectively to terminate. For instances with 60 customers, it takes around 3 minutes and 6 minutes. It can be concluded that the CGBH is still very efficient for solving medium-size VRPRDL-variant and VRPHRDL-variant instances. But for large instances with 120 customers, the computation time for CGBH reaches 20 minutes and 40 minutes respectively. However, the length of the routes in the best solution increases compared with the VRPRDL and VRPHRDL instances. For the medium-size instances, the maximum number of customers that a vehicle delivers is 24 and 26 for the VRPRDL-variant and VRPHRDL-variant. For the large instances, the maximum number of customers served by a vehicle is 30 and 32 for the VRPRDL-variant and VRPHRDL-variant respectively, while the

Table 6: Results on variants of VRPRDL and VRPHRDL instances.

Instance	$K$	$M$	CGBH VRPRDL-variant						CGBH VRPHRDL-variant					
			Obj	time/s	nbRoute	minL	maxL	averL	Obj	time/s	nbRoute	minL	maxL	averL
0		3	736	2.90	2	7	8	7.5	675	4.21	2	1	14	7.5
1		3	798	1.98	3	1	10	5.0	745	3.06	2	5	10	7.5
2		3	802	1.94	2	7	8	7.5	796	1.46	2	7	8	7.5
3		3	726	1.20	2	4	11	7.5	708	3.75	2	1	14	7.5
4		4	1125	0.90	2	6	9	7.5	1083	1.70	2	6	9	7.5
5		3	951	3.45	3	6	8	6.7	869	6.31	2	6	14	10.0
6		3	831	9.40	2	6	14	10.0	792	11.96	2	7	13	10.0
7		4	869	5.97	2	4	16	10.0	858	9.21	2	6	14	10.0
8		3	893	5.82	2	7	13	10.0	839	8.62	2	8	12	10.0
9		4	929	2.95	2	9	11	10.0	863	5.58	3	1	11	6.7
10		4	1129	18.58	2	15	15	15.0	1035	41.96	2	15	15	15.0
11		5	1175	22.17	3	4	13	10.0	1129	30.15	3	4	15	10.0
12		4	1176	17.30	3	7	13	10.0	1121	33.48	3	9	11	10.0
13		4	975	28.45	3	9	12	10.0	953	37.43	3	7	14	10.0
14		4	1079	25.27	2	15	15	15.0	989	45.22	2	14	16	15.0
15		4	1144	23.70	2	13	17	15.0	1023	47.70	2	13	17	15.0
16		5	1224	17.39	3	7	15	10.0	1138	41.21	3	2	18	10.0
17		4	1315	16.26	3	6	12	10.0	1182	44.17	2	12	18	15.0
18		5	1178	21.27	3	8	13	10.0	1021	26.81	2	9	21	15.0
19		4	1181	30.83	2	9	21	15.0	1181	34.68	2	9	21	15.0
20		7	1874	173.74	4	11	18	15.0	1747	371.10	3	15	23	20.0
21		6	1724	152.21	5	2	20	12.0	1471	343.70	4	1	23	15.0
22		9	1892	106.60	4	14	18	15.0	1717	234.85	4	6	24	15.0
23		7	1976	132.04	5	9	14	12.0	1674	311.15	3	16	22	20.0
24		7	1720	190.23	4	12	18	15.0	1587	457.07	4	2	23	15.0
25		9	1852	192.28	4	13	19	15.0	1632	547.83	3	13	26	20.0
26		6	1749	156.16	4	5	24	15.0	1604	247.60	3	16	24	20.0
27		8	1779	275.09	4	9	22	15.0	1504	437.36	3	18	22	20.0
28		8	1913	121.34	5	1	18	12.0	1742	365.97	4	2	24	15.0
29		8	2017	111.38	4	12	20	15.0	1806	305.60	3	14	25	20.0
30		10	2471	1243.55	8	1	27	15.0	2244	3016.68	7	1	31	17.1
31		11	2554	1280.57	8	1	23	15.0	2352	2077.37	5	22	29	24.0
32		10	2362	1624.80	7	1	27	17.1	2150	2715.41	6	1	29	20.0
33		10	2702	1340.18	6	14	27	20.0	2468	2798.17	5	16	28	24.0
34		11	2364	1125.76	7	8	24	17.1	2073	2753.88	6	4	29	20.0
35		13	2772	1166.24	7	1	29	17.1	2518	2615.98	6	1	29	20.0
36		10	2535	1123.38	7	6	27	17.1	2250	2739.07	5	14	30	24.0
37		11	2644	1304.54	8	1	24	15.0	2271	2190.87	7	1	31	17.1
38		12	2649	1071.00	12	1	22	10.0	2336	1778.69	9	1	28	13.3
39		11	2529	1305.95	8	1	30	15.0	2255	3085.74	6	1	32	20.0
Average			2558.20	1258.60	7.8	3.5	26.0	15.9	2291.70	2577.19	6.2	6.2	29.6	20.0

average number becomes 15.9 and 20.0.

We can conclude that if the number of customers that a vehicle serves becomes large on average, the problem becomes, as expected, more difficult to solve. This is mainly due to the route optimization method, which is frequently invoked in the course of the algorithm and becomes time consuming when the routes are longer.

## 5.5 Computational results on GVRPTW instances (Moccia et al. (2012))

In this section, we further test the proposed CGBH on 20 instances proposed by Moccia et al. (2012). The number of clusters in these instances ranges from 30 to 120 and the number of vertices ranges from 188 to 1198. We solve them using the CGBH. Detailed results are reported in Table 7.

Table 7: Results on GVRPTW instances proposed by Moccia et al. (2012)

Instance	Number of vertices	Tabu search		CGBH		GAP/%
		UB	time/s	UB	time/s	
i-030-04-08	188	3498	87.00	3498	8.47	0.00
i-030-08-12	307	2866	142.00	2797	15.37	-2.41
i-040-04-08	250	3811	111.00	3811	6.46	0.00
i-040-08-12	391	3759	171.00	3768	7.80	0.24
i-050-04-08	296	5447	127.00	5439	10.08	-0.15
i-050-08-12	513	4034	249.00	4054	12.35	0.50
i-060-04-08	376	5919	196.00	5908	49.10	-0.19
i-060-08-12	614	4303	383.00	4303	26.26	0.00
i-070-04-08	413	6205	206.00	6228	24.78	0.37
i-070-08-12	690	4645	433.00	4694	45.27	1.05
i-080-04-08	486	7425	266.00	7420	179.33	-0.07
i-080-08-12	795	5734	521.00	5613	43.65	-2.11
i-090-04-08	560	7110	362.00	7108	328.34	-0.03
i-090-08-12	928	5810	711.00	5893	307.66	1.43
i-100-04-08	607	7455	380.00	7339	85.78	-1.56
i-100-08-12	1011	6703	884.00	6788	419.38	1.27
i-110-04-08	672	8719	443.00	8618	128.12	-1.16
i-110-08-12	1115	6281	842.00	6343	120.46	0.99
i-120-04-08	708	8512	525.00	8455	190.67	-0.67
i-120-08-12	1198	6833	1066.00	6772	151.67	-0.89
Average		5753.45	405.25	5742.45	108.05	-0.17

Column *Instance* represents the name of the instance, defined as follows:  $i - p - v_{min} - v_{max}$ , where  $p$ ,  $v_{min}$  and  $v_{max}$  represent the number of clusters, the minimal number of vertices per cluster and the maximal number of vertices per cluster respectively. The second column lists the number of vertices. The next two columns report the results obtained by the tabu search heuristic proposed by Moccia et al. (2012) running on an Intel Core Duo computer (1.83GHz). *UB* and *time/s* represent the upper bound and the computation time in seconds respectively. The next two columns show the results obtained from the proposed CGBH. Column *GAP/%* provides the relative gap between upper bounds obtained by these two heuristics. It is calculated as  $GAP/\% =$

$100\% \times (UB_{CGBH} - UB_{ts})/UB_{ts}$ , where  $UB_{CGBH}$  and  $UB_{ts}$  represent the upper bounds obtained by the CGBH and the tabu search respectively.

From Table 7, it can be seen that compared with the tabu search heuristic, using the proposed CGBH, the upper bound obtained can be improved by 0.17% on average.

Regarding the computation time, for a fair comparison, we use some benchmarks ([https://hwbot.org/compare/processors#672\\_1,523\\_1,4452\\_1-6,3,7,15,14](https://hwbot.org/compare/processors#672_1,523_1,4452_1-6,3,7,15,14)) to compare the average running times of the two processors. The results show that our processor is usually 2~3 times faster. Then, from the average results reported in Table 7, we can conclude that the CGBH consumes less computation time on average. As a rule, the CGBH is competitive with the tabu search heuristic.

Note that in the problem studied by Moccia et al. (2012), there are constraints that restrict the maximum duration of each route to a given upper bound  $D$ , whereas in our problem definition, there is no such restriction. To handle this difference, since there is no time window for the depot in the GVRPTW instances (Moccia et al. (2012)), we simply attach a time window  $[0, D]$  to the depot. The CGBH is then run as described in Section 4. Handling route duration constraints as we proposed may cut off some feasible solutions. However, the obtained results are comparable and sometimes better with respect to those proposed in Moccia et al. (2012).

## 5.6 Computational results on GVRP instances (Bektaş et al. (2011))

We test the CGBH on the benchmark instances of the GVRP proposed by Bektaş et al. (2011). By assigning a non-binding time window to every vertex, we can solve GVRP instances by the CGBH. The GVRP instances are generated using the  $A, B, P$  instances from CVRP library, with the number of vertices ranging from 16 to 101. For a CVRP instance with  $N$  vertices, a GVRP instance is constructed with  $K = \lceil N/\theta \rceil$  clusters, where  $\theta = 2, 3$ . We test on 139 instances that were solved to optimality using the branch-and-cut algorithm proposed by Bektaş et al. (2011).

Preliminary computations showed that in Phase 1 of the CGBH, the construction heuristic could not find any feasible solution for 2 out of 139 instances of this set. Therefore, we develop a simple procedure to produce feasible solutions for the GVRP before calling the CGBH.

We first solve a bin packing problem to obtain a feasible solution with respect to vehicle capacity constraints. Let  $\mathcal{F}$  be the vehicle index set. Binary variables  $\chi_{kf}$  are introduced and equal 1 if and only if cluster  $k \in \mathcal{K} \setminus \{0\}$  is visited by vehicle  $f$ , 0 otherwise.

$$\text{minimize } 0 \tag{22}$$

$$\text{s.t. } \sum_{f \in \mathcal{F}} \chi_{kf} = 1 \quad \forall k \in \mathcal{K} \setminus \{0\}, \tag{23}$$

$$\sum_{k \in \mathcal{K}} Q_k \chi_{kf} \leq Q, \quad \forall f \in \mathcal{F}, \tag{24}$$

$$\chi_{kf} \in \{0, 1\} \quad \forall k \in \mathcal{K} \setminus \{0\}, f \in \mathcal{F}. \tag{25}$$

Since the goal of this model is to find a feasible assignment of clusters to vehicles, we minimize

a constant function. Constraints (23) make sure that each cluster is served by one vehicle. Constraints (24) impose to respect the vehicle capacity. Constraints (25) are variable definitions.

By using the *Populate* method with its default settings in CPLEX 12.6.3, we can obtain multiple feasible solutions for the bin packing problem. Note that if CPLEX populates more than 20 feasible solutions, we randomly choose 20 in the solution pool. Then for each solution, the route optimization procedure is applied to optimize the routes of vehicles visiting their corresponding sets of clusters. Therefrom, feasible solutions for the GVRP are obtained.

The results are presented in Table 8 (see Appendix B for detailed results). Column *Instance set* indicates the instance set. Columns labeled as  $\theta = 2$  and  $\theta = 3$  provide results for GVRP instances with  $\theta = 2$  and  $\theta = 3$  respectively. Column *#Opt* has the format  $a/b$ , where  $a$  reports the number of instances for which the optimal solutions have been obtained by the CGBH and  $b$  reports the number of instances that were solved to optimality by Bektaş et al. (2011) in the corresponding set. Column *GAP/%* reports the average optimality gap on all the instances.

Table 8: Results on GVRP instances proposed by Bektaş et al. (2011)

Instance set	$\theta = 2$		$\theta = 3$	
	#Opt	GAP/%	#Opt	GAP/%
A	11/25	1.08	21/25	0.48
B	14/23	0.53	21/23	0.04
P	19/20	0.05	21/23	0.04
Average	44/68	0.59	63/71	0.20

For instances with  $\theta = 2$ , the optimal solutions are obtained by the CGBH for 44 instances out of 68 in total. The average optimality gap for 68 instances in three sets is 0.59%. For instances with  $\theta = 3$ , optimal solutions can be obtained for most of the instances using the CGBH, i.e., 63 instances out of 71 in total. The average optimality gap for 71 instances is 0.2%. Even though the CGBH is not designed for solving the GVRP, it can be seen from Table 8 that it still produces good quality solutions for GVRP instances.

## 6 Conclusions

E-commerce is used daily and allows customers to purchase their products online. New last mile delivery services do not require customers to be at a specific location to receive the products they bought online. Goods can be delivered at home, but as well into lockers, pick-up points or in the trunk of the cars. As a result, and unlike classical vehicle routing problems, several delivery locations are associated with a customer. This new family of delivery problems can be modeled as generalized vehicle routing problems.

In this paper, we have presented the Generalized Vehicle Routing Problem with Time Windows (GVRPTW), and we propose a set covering formulation. Based on this set covering formulation,

we have developed a column generation based heuristic for the GVRPTW. It combines several components including a construction heuristic, a route optimization procedure, a local search method, and a procedure to generate negative reduced cost routes. Computational results on benchmark instances show that the proposed algorithm is very efficient and high-quality solutions can be obtained within very short computation times for VRPRDL instances with up to 120 clusters.

One perspective of this work is to investigate the dynamic version of routing problems for the last mile delivery services. When some locations or time windows change, a new solution has to be computed again. We believe that the proposed column generation based heuristic could be adapted to such cases since it includes several components to build or optimize solutions, and computation times are short. Another perspective is to combine the GVRPTW with split delivery. One constraint of the GVRPTW is that each customer is served once by a single vehicle. In practice, customers may purchase different commodities from different platforms and the vehicles available for delivery at the distribution center may have small capacities (e.g., a cargo bike). In this case, instead of delivering multiple packages for one customer with a single vehicle, there may be more cost-effective distribution plans if some deliveries are split, i.e., some customers are served by more than one vehicle (Archetti et al. (2008); Gu et al. (2019)).

## Acknowledgments

The funding body will be acknowledged following peer review.

## References

- Afsar, H. M., Prins, C., and Santos, A. C. (2014). Exact and heuristic algorithms for solving the generalized vehicle routing problem with flexible fleet size. *International Transactions in Operational Research*, 21(1):153–175.
- Archetti, C., Savelsbergh, M. W., and Speranza, M. G. (2008). To split or not to split: That is the question. *Transportation Research Part E: Logistics and Transportation Review*, 44(1):114–123.
- Baldacci, R., Bartolini, E., and Laporte, G. (2010). Some applications of the generalized vehicle routing problem. *Journal of the operational research society*, 61(7):1072–1077.
- Bautista, J., Fernández, E., and Pereira, J. (2008). Solving an urban waste collection problem using ants heuristics. *Computers & Operations Research*, 35(9):3020–3033.
- Beheshti, A. K. and Hejazi, S. R. (2015). A novel hybrid column generation-metaheuristic approach for the vehicle routing problem with general soft time window. *Information Sciences*, 316:598–615.
- Bektaş, T., Erdoğan, G., and Røpke, S. (2011). Formulations and branch-and-cut algorithms for the generalized vehicle routing problem. *Transportation Science*, 45(3):299–316.



- Carrabs, F., Cerrone, C., Cerulli, R., and Gaudioso, M. (2017). A novel discretization scheme for the close enough traveling salesman problem. *Computers & Operations Research*, 78:163–171.
- Carrabs, F., Cerrone, C., Cerulli, R., and Golden, B. (2020). An adaptive heuristic approach to compute upper and lower bounds for the close-enough traveling salesman problem. *INFORMS Journal on Computing*, 32(4):1030–1048.
- Carraghan, R. and Pardalos, P. M. (1990). An exact algorithm for the maximum clique problem. *Operations Research Letters*, 9(6):375 – 382.
- Desrochers, M., Desrosiers, J., and Solomon, M. (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations research*, 40(2):342–354.
- Fischetti, M., Salazar González, J. J., and Toth, P. (1997). A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45(3):378–394.
- Furini, F., Malaguti, E., Durán, R. M., Persiani, A., and Toth, P. (2012). A column generation heuristic for the two-dimensional two-staged guillotine cutting stock problem with multiple stock size. *European Journal of Operational Research*, 218(1):251–260.
- Ghiani, G. and Improta, G. (2000). An efficient transformation of the generalized vehicle routing problem. *European Journal of Operational Research*, 122(1):11–17.
- Gu, W., Cattaruzza, D., Ogier, M., and Semet, F. (2019). Adaptive large neighborhood search for the commodity constrained split delivery vrp. *Computers & Operations Research*, 112:104761.
- Gutin, G. and Karapetyan, D. (2010). A memetic algorithm for the generalized traveling salesman problem. *Natural Computing*, 9(1):47–60.
- Ha, M. H., Bostel, N., Langevin, A., and Rousseau, L.-M. (2014). An exact algorithm and a metaheuristic for the generalized vehicle routing problem with flexible fleet size. *Computers & Operations Research*, 43:9–19.
- Hauge, K., Larsen, J., Lusby, R. M., and Krapp, E. (2014). A hybrid column generation approach for an industrial waste collection routing problem. *Computers & Industrial Engineering*, 71:10–20.
- Hawkins, A. J. (2019). Amazon expands in-car delivery service to ford and lincoln vehicles. <https://www.theverge.com/2018/4/24/17261744/amazon-package-delivery-car-trunk-gm-volvo>. Online, accessed September 2019.
- Helsgaun, K. (2000). An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130.
- Helsgaun, K. (2015). Solving the equality generalized traveling salesman problem using the Lin–Kernighan–Helsgaun algorithm. *Mathematical Programming Computation*, 7(3):269–287.

- Janjevic, M., Winkenbach, M., and Merchán, D. (2019). Integrating collection-and-delivery points in the strategic design of urban last-mile e-commerce distribution networks. *Transportation Research Part E: Logistics and Transportation Review*, 131:37–67.
- Kara, I. and Bektas, T. (2003). Integer linear programming formulation of the generalized vehicle routing problem. In *EURO/INFORMS Joint International Meeting, Istanbul, July*, pages 06–10.
- Kirsten, K. (2016). Volvo’s solution for the package theft epidemic: Your car’s trunk. <http://fortune.com/2016/05/10/volvo-urb-it-delivery/>. Online, accessed March 2019.
- Laporte, G. and Semet, F. (1999). Computational evaluation of a transformation procedure for the symmetric generalized traveling salesman problem. *INFOR: Information Systems and Operational Research*, 37(2):114–120.
- Lin, Y. H., Wang, Y., He, D., and Lee, L. H. (2020). Last-mile delivery: Optimal locker location under multinomial logit choice model. *Transportation Research Part E: Logistics and Transportation Review*, 142:102059.
- Mahvash, B., Awasthi, A., and Chauhan, S. (2017). A column generation based heuristic for the capacitated vehicle routing problem with three-dimensional loading constraints. *International Journal of Production Research*, 55(6):1730–1747.
- Moccia, L., Cordeau, J. F., and Laporte, G. (2012). An incremental tabu search heuristic for the generalized vehicle routing problem with time windows. *Journal of the Operational Research Society*, 63(2):232–244.
- Morganti, E., Seidel, S., Blanquart, C., Dablanc, L., and Lenz, B. (2014). The impact of e-commerce on final deliveries: alternative parcel delivery services in france and germany. *Transportation Research Procedia*, 4:178–190.
- Mourgaya, M. and Vanderbeck, F. (2007). Column generation based heuristic for tactical planning in multi-period vehicle routing. *European Journal of Operational Research*, 183(3):1028–1041.
- Noon, C. E. and Bean, J. C. (1993). An efficient transformation of the generalized traveling salesman problem. *INFOR: Information Systems and Operational Research*, 31(1):39–44.
- Ozbaygin, G., Karasan, O. E., Savelsbergh, M., and Yaman, H. (2017). A branch-and-price algorithm for the vehicle routing problem with roaming delivery locations. *Transportation Research Part B: Methodological*, 100:115–137.
- Parragh, S. N. and Schmid, V. (2013). Hybrid column generation and large neighborhood search for the dial-a-ride problem. *Computers & Operations Research*, 40(1):490–497.
- Prescott-Gagnon, E., Desaulniers, G., and Rousseau, L.-M. (2009). A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. *Networks: An International Journal*, 54(4):190–204.

- Reihaneh, M. and Ghoniem, A. (2018). A branch-cut-and-price algorithm for the generalized vehicle routing problem. *Journal of the Operational Research Society*, 69(2):307–318.
- Reyes, D., Savelsbergh, M., and Toriello, A. (2017). Vehicle routing with roaming delivery locations. *Transportation Research Part C: Emerging Technologies*, 80:71–91.
- Ropke, S. and Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472.
- Salhi, S., Wassan, N., and Hajarati, M. (2013). The fleet size and mix vehicle routing problem with backhauls: Formulation and set partitioning-based heuristics. *Transportation Research Part E: Logistics and Transportation Review*, 56:22–35.
- Saskena, J. (1970). Mathematical model of scheduling clients through welfare agencies. *Journal of the Canadian Operational Research Society*, 8:185–200.
- Smith, S. L. and Imeson, F. (2017). Glms: An effective large neighborhood search heuristic for the generalized traveling salesman problem. *Computers & Operations Research*, 87:1–19.
- Srivastava, S., Kumar, S., Garg, R., and Sen, P. (1969). Generalized traveling salesman problem through  $n$  sets of nodes. *CORS journal*, 7(2):97.
- Taillard, É. D. (1999). A heuristic column generation method for the heterogeneous fleet vrp. *RAIRO-Operations Research*, 33(1):1–14.
- Yuan, Y., Cattaruzza, D., Ogier, M., and Semet, F. (2020). A branch-and-cut algorithm for the generalized traveling salesman problem with time windows. *European Journal of Operational Research*, 286(3):849–866.
- Zhou, L., Baldacci, R., Vigo, D., and Wang, X. (2018). A multi-depot two-echelon vehicle routing problem with delivery options arising in the last mile distribution. *European Journal of Operational Research*, 265(2):765–778.

## A Choice of pivot customers

The construction heuristic starts with the choice of pivot customers. These customers may be identified with different criteria. Here we present the three criteria we use in this work. The construction heuristic is called several times during Phase 1. In the first call, Criterion 1 is chosen, while Criterion 2 or Criterion 3 are chosen for the next calls of the construction heuristic. At each call, the choice between Criterion 2 and Criterion 3 is done randomly.

**Criterion 1.** Due to TWs, some customers cannot be visited on the same route. If customers  $h$  and  $k$  cannot be served on the same route, we call this pair of customers  $\langle h, k \rangle$  an *incompatible pair*. Formally,  $h$  and  $k$  are incompatible if  $E_i + T_{ij} > L_j$  and  $E_j + T_{ji} > L_i$ , for all  $i \in \mathcal{C}_h$ ,  $j \in \mathcal{C}_k$ .

Based on all the incompatible pairs, we build a graph  $\bar{\mathcal{G}} = (\bar{\mathcal{V}}, \bar{\mathcal{E}})$ , where the vertex set  $\bar{\mathcal{V}}$  contains  $K$  vertices, one per customer and  $(h, k) \in \bar{\mathcal{E}}$  if and only if  $\langle h, k \rangle$  is an incompatible pair. We then look for a maximum clique in  $\bar{\mathcal{G}} = (\bar{\mathcal{V}}, \bar{\mathcal{E}})$ . By construction of  $\bar{\mathcal{G}}$ , a clique represents a set of customers such that none of them be visited in the same route since they are all incompatible. Hence, all the customers represented by a clique in  $\bar{\mathcal{G}}$  have to be served in different routes. Therefore, we can choose the customers belonging to a maximum clique in  $\bar{\mathcal{G}}$  as pivots. Here we use a recursive backtracking algorithm (Carraghan and Pardalos, 1990) that searches for all maximal cliques in graph  $\bar{\mathcal{G}}$ . It is an enumeration algorithm that backtracks when the size of the current clique plus the size of the set of potential nodes to add is lower than the size of the current maximum clique. Since the algorithm returns all maximal cliques, one of them is randomly chosen to initialize the pivots customers.

Note that the size of a maximum clique in  $\bar{\mathcal{G}}$  can be smaller than  $M$ , the number of available vehicles at the depot. In this case, the remaining routes of  $\mathcal{R}$  are initialized with empty routes.

**Criterion 2.** For each vertex  $i \in \mathcal{V}$ , we determine a vertex set  $\mathcal{B}_i = \{j_1, j_2\}$  which includes two vertices compatible with  $i$ , and such that  $j_1$  and  $j_2$  do not belong to the same cluster.  $j_1$  is the nearest vertex from which  $i$  can be reached (i.e., it satisfies  $E_{j_1} + T_{j_1 i} \leq L_i$ ) and  $j_2$  is the nearest vertex that can be reached from  $i$  (i.e., it satisfies  $E_i + T_{ij_2} \leq L_{j_2}$ ). Then, we calculate the average cost  $\bar{C}_i$  between vertex  $i$  and the vertices in  $\mathcal{B}_i$ ,  $\bar{C}_i = \frac{1}{2} \sum_{j \in \mathcal{B}_i} C_{ij}$ .

We then define a score  $w_k$  for each customer  $k \in \mathcal{K} \setminus \{0\}$  as follows:

$$w_k = \frac{1}{|\mathcal{C}_k|} \sum_{i \in \mathcal{C}_k} (C_{0i} + \bar{C}_i). \quad (26)$$

By using this score, we select the customers with the highest value of  $w_k$ , i.e., customers that are either far away from the depot and/or far away from other customers. However, using only this score has the disadvantage of selecting as pivots some nearby customers which are relatively far from the depot, but that could be served on the same route. Therefore, it is appropriate to *spread* the pivots so that they belong to different spatial regions. To this end, we define  $\mathcal{A}_{hk} = \{(i, j) \in \mathcal{A} | i \in \mathcal{C}_h, j \in \mathcal{C}_k\}$  as the set of arcs from  $\mathcal{C}_h$  to  $\mathcal{C}_k$ , and  $\bar{C}_{hk}$  as the average traveling cost from cluster  $\mathcal{C}_h$  to cluster  $\mathcal{C}_k$ .  $\bar{C}_{hk}$  is calculated as:

$$\bar{C}_{hk} = \frac{1}{|\mathcal{A}_{hk}|} \sum_{(i,j) \in \mathcal{A}_{hk}} C_{ij} \quad \forall h, k \in \mathcal{K} \setminus \{0\}. \quad (27)$$

Let us denote by  $\mathcal{P}$  the set of selected pivots. At the beginning, this set  $\mathcal{P}$  is empty, and pivots are added one by one. To this end, we define a score  $w'_k$  for each customer  $k \in \mathcal{K} \setminus \{\mathcal{P} \cup \{0\}\}$  as follows:

$$w'_k = w_k + \min_{h \in \mathcal{P}} \{\min\{\bar{C}_{hk}, \bar{C}_{kh}\}\}. \quad (28)$$

The selection of the pivots customers is performed as follows. At first, set  $\mathcal{P}$  is empty and we sort all the customers in  $k \in \mathcal{K} \setminus \{\mathcal{P} \cup \{0\}\}$  by descending values of  $w'_k$  and store them in a list  $\mathcal{I}$ . Then, we random select a number  $\theta$  in the interval  $[0, 1)$  and calculate  $\theta^\rho$ . The customer in position  $\theta^\rho |\mathcal{I}|$  is chosen as a pivot, and added to  $\mathcal{P}$ . Then, the scores  $w'_k$  of the other customers are updated,

and the procedure is repeated until  $M$  pivots are selected. Here, we choose  $\rho = 6$  as in [Ropke and Pisinger \(2006\)](#). Note that when  $\rho = 1$ , the selection becomes purely random. When  $\rho = \infty$ , the customer associated with the best score  $w'_k$  is selected.

**Criterion 3.** As with Criterion 2, this criterion computes the scores for each cluster in order to iteratively select the clusters with the highest scores. This criterion is based on a score related to the compactness of the clusters. Suppose that vertex  $i \in \mathcal{V}$  has coordinates  $(a_i, b_i)$ . Here, we consider the barycenter  $(a_k^c, b_k^c)$  of a cluster  $k \in \mathcal{K} \setminus \{0\}$ :

$$a_k^c = \frac{1}{|\mathcal{C}_k|} \sum_{i \in \mathcal{C}_k} a_i, \quad (29)$$

$$b_k^c = \frac{1}{|\mathcal{C}_k|} \sum_{i \in \mathcal{C}_k} b_i. \quad (30)$$

Then, for each cluster  $k \in \mathcal{K} \setminus \{0\}$  we define  $\overline{C_{inc_k}}$  as the average traveling distance of the vertices in  $\mathcal{C}_k$  to its barycenter  $(a_k^c, b_k^c)$ :

$$\overline{C_{inc_k}} = \frac{1}{|\mathcal{C}_k|} \sum_{i \in \mathcal{C}_k} \sqrt{(a_i - a_k^c)^2 + (b_i - b_k^c)^2}. \quad (31)$$

The score  $w_k$  of cluster  $k \in \mathcal{K} \setminus \{0\}$  is then defined as:

$$w_k = \sqrt{(a_k^c - a_0)^2 + (b_k^c - b_0)^2} - \overline{C_{inc_k}}. \quad (32)$$

By using this score, we favor compact clusters far from the depot.

Similarly with Criterion 2, we try to spread the pivots. We denote by  $\mathcal{P}$  the set of pivots already selected, and we use an updated score  $w'_k$  for each cluster  $k \in \mathcal{K} \setminus \{\mathcal{P} \cup \{0\}\}$ :

$$w'_k = w_k + \min_{h \in \mathcal{P}} \{\min\{\overline{C_{hk}}, \overline{C_{kh}}\}\}. \quad (33)$$

Based on  $w'_k$ , the selection of pivot customers is performed as described for Criterion 2.

## B Supplementary results

### B.1 Results on VRPRDL instances with 40 customers

Table 9 shows that the CGBH gets the best-known values for all the instances.

### B.2 Detailed results on the GVRP instances

Tables 10 and 11 report the results on GVRP instances for  $\theta = 2$  and  $\theta = 3$  respectively. Column *Instance* represents the name of the instance, which has the format  $X - nY - kZ - C\Omega - V\Phi$ , where  $X$  specifies the set,  $Y$  the number of vertices,  $Z$  the number of vehicles in the original CVRP instance,  $\Omega$  the number of clusters, and  $\Phi$  the number of vehicles in the GVRP instance. The next two columns show the results obtained by the branch-and-cut algorithm proposed by [Bektaş et al.](#)

Table 9: Results on VRPRDL instances with 40 customers.

Instance	$K$	$M$	BP		CGBH						
			best-known	time/s	Obj	time/s	GAP/%	nbRoute	minL	maxL	averL
41_v1		11	3203	1249.35	3203	2.34	0	10	2	9	4.0
42_v1		10	2799	3.00	2799	1.35	0	9	1	8	4.4
43_v1		9	2607	7200.00	2607	3.86	0	8	2	12	5.0
44_v1		8	2261	98.52	2261	2.39	0	7	3	7	5.7
45_v1		11	3217	1.63	3217	1.44	0	10	2	7	4.0
46_v1		10	2805	3.81	2805	1.48	0	9	3	7	4.4
47_v1		12	3339	3710.35	3339	2.36	0	10	2	7	4.0
48_v1		11	3325	1.15	3325	1.39	0	10	1	8	4.0
49_v1		12	3534	104.26	3534	1.21	0	11	2	5	3.6
50_v1		10	2752	8.74	2752	4.24	0	10	1	9	4.0
41_v2	40	8	2133	854.47	2133	7.07	0	7	3	8	5.7
42_v2		8	1946	1005.36	1946	4.11	0	7	1	8	5.7
43_v2		9	1966	270.72	1966	5.29	0	8	2	9	5.0
44_v2		7	1610	41.59	1610	4.91	0	6	1	9	6.7
45_v2		9	2478	9.76	2478	5.83	0	8	2	10	5.0
46_v2		10	2469	27.37	2469	2.39	0	8	3	7	5.0
47_v2		9	1946	68.96	1946	4.55	0	7	2	8	5.7
48_v2		9	2380	477.83	2380	3.01	0	8	3	7	5.0
49_v2		10	2492	13.62	2492	2.41	0	8	2	7	5.0
50_v2		10	2443	164.37	2443	3.69	0	8	3	10	5.0
<b>Average</b>			<b>2585.25</b>	<b>765.74</b>	<b>2585.25</b>	<b>3.27</b>	<b>0</b>	<b>8.5</b>	<b>2.1</b>	<b>8.1</b>	<b>4.9</b>

(2011) running on an AMD Opteron 250 computer (2.4 GHz). Column *Opt* presents the objective value of the optimal solution. Column *time/s* is the computation time in seconds. The next two columns show the results obtained using the proposed CGBH. Column *Obj* is the objective value of the best solution obtained by the CGBH. Column *GAP/%* provides the optimality gap which is calculated as  $GAP/\% = 100\% \times (Obj - Opt)/Opt$ .

Table 10: Results on GVRP instances with  $\theta = 2$ 

Instance	Branch-and-cut		CGBH		GAP/%
	Opt	time/s	Obj	time/s	
A-n32-k5-C16-V2	519	113.20	552	2.42	6.36
A-n33-k5-C17-V3	451	1.60	451	1.63	0.00
A-n33-k6-C17-V3	465	0.70	465	1.22	0.00
A-n34-k5-C17-V3	489	0.80	501	2.10	2.45
A-n36-k5-C18-V2	505	31.50	506	4.87	0.20
A-n37-k5-C19-V3	432	0.80	432	4.11	0.00
A-n37-k6-C19-V3	584	28.20	614	2.70	5.14
A-n38-k5-C19-V3	476	3.00	476	2.93	0.00
A-n39-k5-C20-V3	557	45.60	557	5.37	0.00
A-n39-k6-C20-V3	544	4.90	544	2.72	0.00
A-n44-k6-C22-V3	608	23.20	608	4.71	0.00
A-n45-k6-C23-V4	613	6.80	613	5.11	0.00
A-n45-k7-C23-V4	674	1465.20	681	4.86	1.04
A-n46-k7-C23-V4	593	10.20	593	4.46	0.00
A-n48-k7-C24-V4	667	299.80	668	6.44	0.15
A-n53-k7-C27-V4	603	15.90	606	10.52	0.50
A-n54-k7-C27-V4	690	68.30	690	7.04	0.00
A-n55-k9-C28-V5	699	82.60	711	4.78	1.72
A-n60-k9-C30-V5	769	75.60	780	7.62	1.43
A-n61-k9-C31-V5	638	43.70	640	8.44	0.31
A-n62-k8-C31-V4	740	122.70	751	16.05	1.49
A-n63-k10-C32-V5	801	4355.20	801	9.37	0.00
A-n64-k9-C32-V5	763	1204.30	775	16.38	1.57
A-n65-k9-C33-V5	682	29.00	704	11.42	3.23
A-n69-k9-C35-V5	680	817.90	689	16.85	1.32
B-n31-k5-C16-V3	441	0.10	441	5.07	0.00
B-n34-k5-C17-V3	472	0.10	472	3.59	0.00
B-n35-k5-C18-V3	626	0.10	626	3.14	0.00
B-n38-k6-C19-V3	451	0.70	451	1.76	0.00
B-n39-k5-C20-V3	357	0.20	357	4.25	0.00
B-n41-k6-C21-V3	481	2.60	481	2.68	0.00
B-n43-k6-C22-V3	483	9.20	485	4.96	0.41
B-n44-k7-C22-V4	540	3.30	543	3.98	0.56
B-n45-k5-C23-V3	497	0.60	497	6.28	0.00
B-n45-k6-C23-V4	478	53.70	478	4.55	0.00
B-n50-k7-C25-V4	449	0.60	449	4.57	0.00
B-n50-k8-C25-V5	916	3249.20	936	5.87	2.18
B-n51-k7-C26-V4	651	0.40	670	4.56	2.92
B-n52-k7-C26-V4	450	0.10	450	8.11	0.00
B-n56-k7-C28-V4	486	3.00	486	10.57	0.00
B-n57-k7-C29-V4	751	1.80	765	11.09	1.86
B-n57-k9-C29-V5	942	22.00	942	8.76	0.00
B-n63-k10-C32-V5	816	12.20	823	14.45	0.86
B-n64-k9-C32-V5	509	0.80	509	7.42	0.00
B-n66-k9-C33-V5	808	14.40	808	14.16	0.00
B-n67-k10-C34-V5	673	35.80	681	10.05	1.19
B-n68-k9-C34-V5	704	9.20	718	10.67	1.99
B-n78-k10-C39-V5	803	248.20	805	26.08	0.25
P-n16-k8-C8-V5	239	0.00	239	0.07	0.00
P-n19-k2-C10-V2	147	0.00	147	0.78	0.00
P-n20-k2-C10-V2	154	0.00	154	1.02	0.00
P-n21-k2-C11-V2	160	0.00	160	1.26	0.00
P-n22-k2-C11-V2	162	0.10	162	1.53	0.00
P-n22-k8-C11-V5	314	0.00	314	0.17	0.00
P-n23-k8-C12-V5	312	0.80	312	0.34	0.00
P-n40-k5-C20-V3	294	2.10	294	5.97	0.00
P-n45-k5-C23-V3	337	2.20	337	8.19	0.00
P-n50-k10-C25-V5	353	1162.90	353	7.48	0.00
P-n50-k8-C25-V4	410	7200.10	410	4.22	0.00
P-n51-k10-C26-V6	427	38.80	427	1.80	0.00
P-n55-k10-C28-V5	361	1536.70	361	12.10	0.00
P-n55-k15-C28-V8	361	7200.10	361	10.20	0.00
P-n55-k7-C28-V4	415	125.20	415	3.21	0.00
P-n65-k10-C33-V5	487	1805.50	487	10.67	0.00
P-n70-k10-C35-V5	485	175.80	485	13.81	0.00
P-n76-k4-C38-V2	383	25.80	387	262.38	1.04
P-n76-k5-C38-V3	405	16.20	405	91.71	0.00
P-n101-k4-C51-V2	455	169.20	455	948.89	0.00

Table 11: Results on GVRP instances with  $\theta = 3$ 

Instance	Branch-and-cut		CGBH		GAP/%
	Opt	time/s	Obj	time/s	
A-n32-k5-C11-V2	386	0.10	386	1.31	0.00
A-n33-k5-C11-V2	315	0.50	315	0.74	0.00
A-n33-k6-C11-V2	370	1.20	370	0.58	0.00
A-n34-k5-C12-V2	419	1.70	419	1.45	0.00
A-n36-k5-C12-V2	396	1.30	396	2.39	0.00
A-n37-k5-C13-V2	347	0.70	347	2.08	0.00
A-n37-k6-C13-V2	431	19.40	431	1.05	0.00
A-n38-k5-C13-V2	367	0.70	367	1.12	0.00
A-n39-k5-C13-V2	364	4.60	364	2.93	0.00
A-n39-k6-C13-V2	403	1.20	403	1.10	0.00
A-n44-k6-C15-V2	503	323.70	548	2.23	8.95
A-n45-k6-C15-V3	474	2.90	474	3.03	0.00
A-n45-k7-C15-V3	475	7.40	475	2.66	0.00
A-n46-k7-C16-V3	462	22.70	462	2.78	0.00
A-n48-k7-C16-V3	451	19.00	459	2.99	1.77
A-n53-k7-C18-V3	440	5.90	440	7.68	0.00
A-n54-k7-C18-V3	482	57.40	482	3.86	0.00
A-n55-k9-C19-V3	473	14.10	473	2.33	0.00
A-n60-k9-C20-V3	595	885.20	596	3.34	0.17
A-n61-k9-C21-V4	473	14.50	473	3.13	0.00
A-n62-k8-C21-V3	596	859.60	596	7.60	0.00
A-n63-k9-C21-V3	593	7200.10	600	3.60	1.18
A-n64-k9-C22-V3	536	22.40	536	12.44	0.00
A-n65-k9-C22-V3	500	21.90	500	5.41	0.00
A-n69-k9-C23-V3	520	4752.40	520	11.70	0.00
B-n31-k5-C11-V2	356	0.20	356	1.71	0.00
B-n34-k5-C12-V2	369	0.00	369	2.10	0.00
B-n35-k5-C12-V2	501	0.20	501	1.68	0.00
B-n38-k6-C13-V2	370	1.30	370	1.17	0.00
B-n39-k5-C13-V2	280	0.00	280	1.68	0.00
B-n41-k6-C14-V2	407	1.00	407	1.82	0.00
B-n43-k6-C15-V2	343	0.60	343	3.79	0.00
B-n44-k7-C15-V3	395	1.50	395	2.03	0.00
B-n45-k5-C15-V2	410	0.90	410	3.01	0.00
B-n45-k6-C15-V2	336	4.80	336	1.92	0.00
B-n50-k7-C17-V3	393	0.20	393	2.43	0.00
B-n50-k8-C17-V3	598	29.40	598	2.08	0.00
B-n51-k7-C17-V3	511	0.40	511	2.22	0.00
B-n52-k7-C18-V3	359	0.00	359	3.59	0.00
B-n56-k7-C19-V3	356	23.50	356	5.25	0.00
B-n57-k7-C19-V3	558	0.90	562	5.01	0.72
B-n57-k9-C19-V3	681	471.60	681	3.76	0.00
B-n63-k10-C21-V3	599	11.30	599	6.29	0.00
B-n64-k9-C22-V4	452	2.40	452	3.10	0.00
B-n66-k9-C22-V3	609	103.50	609	6.97	0.00
B-n67-k10-C23-V4	558	7.20	558	4.62	0.00
B-n68-k9-C23-V3	523	110.00	524	6.68	0.19
B-n78-k10-C26-V4	606	8.50	606	6.90	0.00
P-n16-k8-C6-V4	170	0.00	170	0.24	0.00
P-n19-k2-C7-V1	111	0.00	111	0.10	0.00
P-n20-k2-C7-V1	117	0.20	117	0.11	0.00
P-n21-k2-C7-V1	117	0.20	117	0.11	0.00
P-n22-k2-C8-V1	111	0.10	111	0.19	0.00
P-n22-k8-C8-V4	249	0.10	249	0.18	0.00
P-n23-k8-C8-V3	174	0.10	174	0.15	0.00
P-n40-k5-C14-V2	213	1.10	213	3.33	0.00
P-n45-k5-C15-V2	238	11.10	238	2.32	0.00
P-n50-k10-C17-V4	261	5.00	261	2.75	0.00
P-n50-k7-C17-V3	262	6.40	262	1.52	0.00
P-n50-k8-C17-V3	292	7.40	292	0.79	0.00
P-n51-k10-C17-V4	309	117.60	310	0.86	0.32
P-n55-k10-C19-V4	271	18.10	271	5.93	0.00
P-n55-k15-C19-V6	274	36.00	274	5.19	0.00
P-n55-k7-C19-V3	301	78.20	301	2.30	0.00
P-n55-k8-C19-V3	378	53.60	378	0.44	0.00
P-n60-k10-C20-V4	325	282.70	325	2.40	0.00
P-n65-k10-C22-V4	372	1028.20	375	3.41	0.81
P-n70-k10-C24-V4	385	1468.30	385	4.74	0.00
P-n76-k4-C26-V2	309	122.50	309	98.30	0.00
P-n76-k5-C26-V2	309	90.10	309	54.68	0.00
P-n101-k4-C34-V2	370	6581.80	370	373.82	0.00