Adaptive large neighborhood search for the commodity constrained split delivery VRP

Wenjuan Gu, Diego Cattaruzza, Maxime Ogier, Frédéric Semet

Adaptive large neighborhood search for the commodity constrained split delivery VRP

Wenjuan Gu, Diego Cattaruzza, Maxime Ogier, Frédéric Semet

Univ. Lille, CNRS, Centrale Lille, Inria, UMR 9189 - CRIStAL, F-59000 Lille, France
Email: guwj0303@gmail.com, diego.cattaruzza@centralelille.fr, maxime.ogier@centralelille.fr, frederic.semet@centralelille.fr

Abstract: This paper addresses the commodity constrained split delivery vehicle routing problem (C-SDVRP) where customers require multiple commodities. This problem arises when customers accept to be delivered separately. All commodities can be mixed in a vehicle as long as the vehicle capacity is satisfied. Multiple visits to a customer are allowed, but a given commodity must be delivered in one delivery.

In this paper, we propose a heuristic based on the adaptive large neighborhood search (ALNS) to solve the C-SDVRP, with the objective of efficiently tackling medium and large sized instances. We take into account the distinctive features of the C-SDVRP and adapt several local search moves to improve a solution. Moreover, a mathematical programming based operator (MPO) that reassigns commodities to routes is used to improve a new global best solution.

Computational experiments have been performed on benchmark instances from the literature. The results assess the efficiency of the algorithm, which can provide a large number of new best-known solutions in short computational times.

Keywords: vehicle routing problem; multiple commodities; adaptive large neighborhood search; local search.

1 Introduction

The vehicle routing problem (VRP) and its variants have been widely studied in the literature (Toth and Vigo, 2014). In the VRP with multiple commodities, capacitated vehicles are used to deliver a set of commodities and meet the demands of customers. Four different strategies to deliver a set of commodities to customers were presented by Archetti et al. (2014): *separate routing*, *mixed routing*, *split delivery mixed routing* and *commodity constrained split delivery mixed routing*.

In the *separate routing* strategy, a specific set of vehicles is dedicated to each commodity, and any commodity has to be delivered to any customer by a single vehicle visit. In this case, considering each commodity separately is a classical capacitated VRP (CVRP). A customer is visited as many times as the number of commodities that he/she requires. In other words, if a customer needs multiple commodities, this customer needs to be served several times, even if all the commodities could be delivered by only one vehicle.

In the *mixed routing* strategy any set of commodities can be mixed in the same vehicle, and all customers must be delivered at once. If a customer requires one or more commodities, all of them are delivered by a single vehicle in a single visit. Here again, the problem that arises corresponds to a single CVRP. In this delivery strategy, when the remaining capacity of a vehicle is not sufficient

to deliver all the demand of a given customer, it is wasted with a possible increase in transportation costs.

In the *split delivery mixed routing* strategy, any set of commodities can be mixed in the same vehicle. The commodities can be split in any possible way. Moreover, a commodity can be delivered to a customer by several vehicles. The problem that arises corresponds to the split delivery VRP (SDVRP) that was introduced by Dror and Trudeau (1989) and Dror and Trudeau (1990). For an extensive review of SDVRP, the interested reader is referred to the survey by Archetti and Speranza (2012). In the split delivery mixed routing strategy, a customer can be visited several times if this is beneficial, even if this customer requires only one commodity. This strategy minimizes transportation costs but may cause inconvenience to customers (Archetti et al., 2008). For example, if one commodity is delivered by several vehicles, the handling time for the customer may increase significantly.

The *commodity constrained split delivery mixed routing* (C-SDVRP) is a strategy recently proposed by Archetti et al. (2014) to deliver multiple commodities. In the C-SDVRP, any set of commodities can be mixed in the same vehicle. The demand of a customer can be split, but only as many times as the required commodities and when a commodity is delivered to a customer, the entire required amount is handed over. As a consequence, one commodity can be delivered by only one vehicle.

The C-SDVRP shares similarities with the SDVRP. However, a significant difference is that the demand of a customer cannot be arbitrarily split since a commodity has to be delivered by the same vehicle. Thus, only splitting a request of a customer according to different commodities is allowed. Considering the convenience of customers and transportation costs, the C-SDVRP is more interesting than the other three strategies to deliver multiple commodities. This problem arises in several real-life situations, for instance in the delivery of groups of products: dairy products, fresh fruits, or vegetables to supermarkets, catering services or restaurants. Each group of products represents a commodity. These commodities can be mixed in the same vehicle. For the customer, it is acceptable to have more than one delivery, but splitting the delivery of a specific commodity (a group of products) is not practical at all. Few commodities are considered. The number of deliveries for a customer is therefore acceptable. Moreover, considering to split deliveries is beneficial for the entire logistic system since it reduces transportation costs.

Despite its practical relevance, the C-SDVRP has received very little attention. The C-SDVRP was first introduced by Archetti et al. (2014). A branch-and-cut algorithm was proposed by the authors and is able to solve to optimality 25 out of 64 small instances (15 customers) within 30 minutes. Archetti et al. (2014) also proposed a heuristic method to tackle this problem. The heuristic consists in making copies of each customer (one for each required commodity) and uses an open-access injection-ejection algorithm for the CVRP. This solving method is simple but does not seem to be very efficient. Indeed, customer replicas share the same location, so the resulting capacitated VRP has many equivalent solutions. We use an example to highlight this point.

Figure 1 shows an example where customers require multiple commodities. A square indicates the location of the depot and circles represent the locations of the three clients. The number of commodities is three. The number on each edge corresponds to the associated travel cost, and the numbers in the dotted ellipses are the demands for each commodity. To increase readability, a different colored ellipse represents each commodity. For instance, for each customer, the number in the red ellipse is the demand of commodity one required by this customer. Each vehicle has a capacity of $Q = 10$ units.

According to the heuristic used by Archetti et al. (2014), the C-SDVRP is solved as a CVRP where the set of customers contains all the customer replicas. Figure 2 shows two solutions of this example.

Solution 1 is composed of three routes to serve all of the demand. Route 1 serves the whole demand for customer 1 (6 units) and commodity 1 of customer 2 (3 units) for a total of 9 units. Route 2 delivers the remaining of the demand of customer 2 (5 units) and commodities 1 and 2 (5 units) of customer 3. The remaining demand for customer 3 (1 unit) is delivered in route 3. The three routes

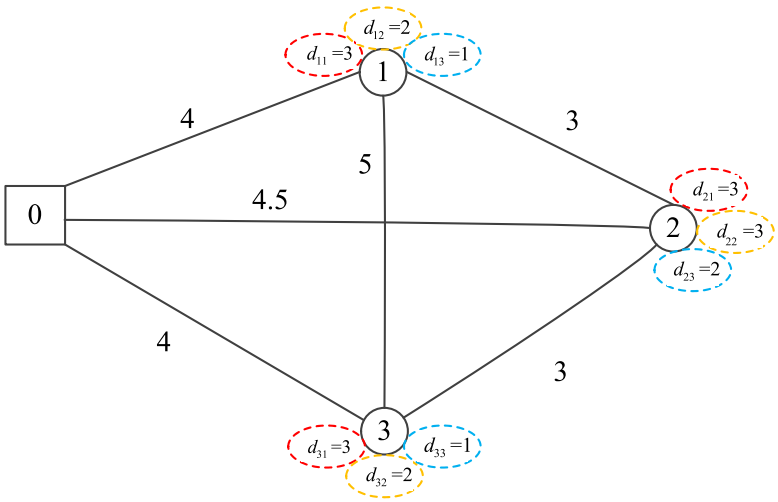
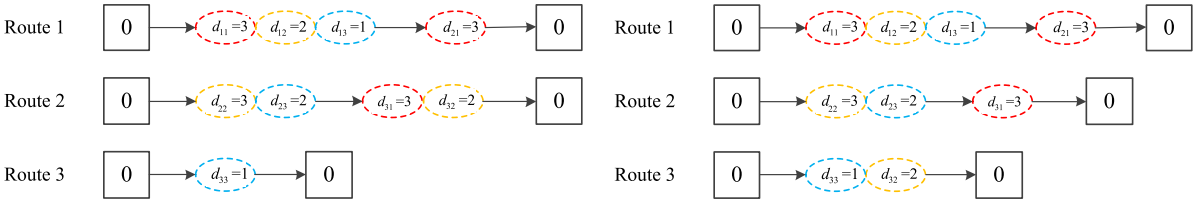


Figure 1: An example from Archetti et al. (2014).



(a) Solution 1. (b) Solution 2.

Figure 2: Two solutions of the instance in Figure 1.

cost 11.5, 11.5 and 8 respectively. The total cost is then 31.

Solution 2 is also composed of three routes. Route 1 is the same as in solution 1. Route 2 delivers the remaining of the demand of customer 2 (5 units) and commodity 1 (3 units) of customer 3. The remaining demand for customer 3 (3 units) is delivered in route 3. The three routes cost 11.5, 11.5 and 8 respectively, for a total cost of 31.

These two solutions are different. However, in essence, the two solutions are equivalent in that customers' replicas share the same location. Taking this specific feature of the problem into account to avoid exploring many equivalent solutions is a real challenge.

In a more recent work Archetti et al. (2015) proposed an extended formulation for the C-SDVRP and developed a branch-price-and-cut algorithm. This algorithm has limitations for solving large-scale instances: optimal solutions were obtained with up to 40 customers and 3 commodities per customer within 2 hours. We are not aware of any other study in the literature solving the C-SDVRP.

This paper aims at proposing an efficient heuristic to tackle medium and large sized C-SDVRP instances. To this end, we propose an adaptive large neighborhood search (ALNS) heuristic taking into account the specialty of the C-SDVRP. ALNS is an efficient metaheuristic proposed by Ropke and Pisinger (2006) and extends the large neighborhood search (LNS) heuristic (Shaw, 1997, 1998) by allowing multiple destroy and repair methods to be used within the same search. Recently, ALNS has been successfully applied to the capacitated VRP (Sze et al., 2016) and to many variants of the VRP (Azi et al., 2014, François et al., 2016, Masson et al., 2013, Sze et al., 2017).

The contributions of this paper are as follows. First, a new heuristic method for the C-SDVRP is proposed. As mentioned earlier, if a customer is replicated as many time as the required commodities, many equivalent solutions exist. Hence, to avoid this pitfall, the proposed method explicitly takes this feature into account. To improve a solution, we adapt existing local search (LS) operators to deal with a customer as a whole (i.e., with the whole demand he/she requires) or only as a part (i.e., with a single commodity he/she requires). Second, in order to further improve the quality of a new

and better encountered solution, a mixed integer programming (MIP) based operator is developed. Finally, we provide a large number of new best-known solutions for medium and large sized C-SDVRP instances up to 100 customers and 3 commodities per customer within about 10 minutes of computing time.

The rest of this paper is organized as follows. Section 2 defines the C-SDVRP. The proposed algorithm for the C-SDVRP is described in Section 3. Section 4 reports the computational results. Section 5 concludes the paper and suggests new directions for future research.

2 Problem definition

The C-SDVRP is defined on a directed graph $G = (\mathcal{V}, \mathcal{A})$ in which $\mathcal{V} = \{0\} \bigcup \mathcal{V}_C$ is the set of vertices, and $\mathcal{A}$ is the set of arcs. More precisely, $\mathcal{V}_C = \{1, ..., n\}$ represents the set of customer vertices, and 0 is the depot. A cost c_{ij} is associated with each arc $(i, j) \in \mathcal{A}$ and represents the non-negative cost of travel from i to j. Let $\mathcal{M} = \{1, ..., M\}$ be the set of available commodities. Each customer $i \in \mathcal{V}_C$ requires a quantity $d_{im} \geq 0$ of commodity $m \in \mathcal{M}$. Note that a customer $i \in \mathcal{V}_C$ may request any subset of commodities, namely d_{im} may be equal to zero for some $m \in \mathcal{M}$.

A fleet of identical vehicles with capacity Q is based at the depot and is able to deliver any subset of commodities.

The problem is to find a solution that minimizes the total transportation cost, and that involves two related decisions such as finding a set of vehicle routes that serve all customers and selecting the commodities that are delivered by a vehicle route to each customer. Moreover, each solution must be such that:

1. each route starts and ends at the depot;
2. the total quantity of commodities delivered by each vehicle does not exceed the vehicle capacity;
3. each commodity requested by each customer must be delivered by a single vehicle;
4. the demands of all customers need to be satisfied.

We use the example presented in Figure 1 to illustrate an optimal solution of a C-SDVRP instance. The solution is provided in Figure 3. In the C-SDVRP case, two vehicle routes are required to deliver all the commodities required by the customers. One route (black line) delivers all the commodities of customer 1 (6 units) and delivers commodities 1 and 3 of customer 3. The cost of this route is 13. The other route (purple line) delivers all the commodities of customer 2 (8 units) and commodity 2 of customer 3 (2 units). The cost of this route is 11.5. The total cost of the solution is 24.5. For the solutions obtained using the other three delivery strategies (separate routing, mixed routing and split delivery mixed routing), the interested reader is referred to Archetti et al. (2014).

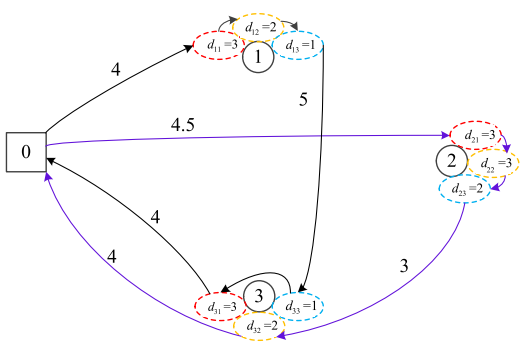


Figure 3: An optimal solution for the C-SDVRP instance proposed in Figure 1.

3 Adaptive large neighborhood search

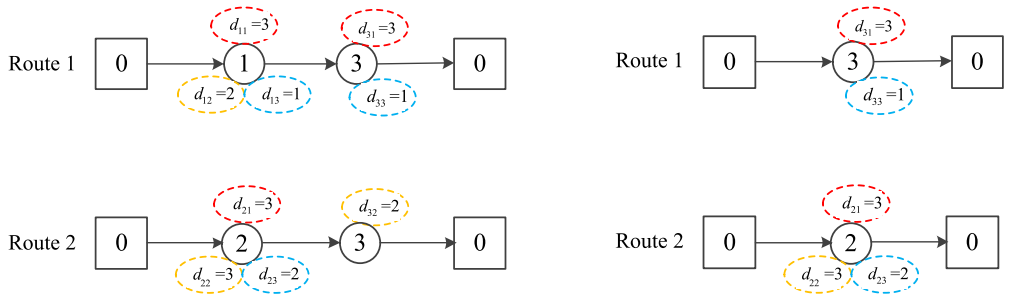
In order to tackle the C-SDVRP for medium and large instances, we propose a heuristic method based on the ALNS framework of Ropke and Pisinger (2006). We make use of local search, and we develop a mathematical programming based operator to improve the quality of solutions.

Due to the characteristics of the problem under study, one node can be duplicated as many times as the number of commodities required by the associated customer (Archetti et al., 2014). With each duplicated node, we then associate the demand of the customer for the corresponding commodity. However, the simple duplication of customers without further consideration of the customer location can produce several equivalent solutions as illustrated previously in the paper. To enhance the performance of the algorithm, we explicitly consider customer replications for each commodity *and* the customer as a single entity associated with its total demand. To this intent and for the sake of clarity, in the following, we will call the duplicated nodes *customer-commodity*, and we will use the term *customer* to refer to the customer associated with the total demand.

We represent a solution of C-SDVRP as the set of routes needed to serve all customers. In order to take into account the specific features of the C-SDVRP, a route can be represented by (1) a sequence of *customers*, or by (2) a sequence of *customer-commodities*. Note that, because a customer can be delivered by several vehicles, in the representation with customers, it is possible that a customer appears in several routes (with different commodities).

The second representation gives more flexibility but increases in complexity. To clarify this, we consider the case of removing a customer from a solution. In the first case, when removing a customer from a route, the customer associated with all the commodities delivered by the route is removed. In the second case, it is possible to remove only one commodity. We illustrate the two representations in Figures 4 and 5. We use the example presented in Figure 1. Figure 4 shows a set of routes represented as two sequences of customers. If customer 1 is removed from route 1, then customer 1 with all the three commodities is removed. If customer 3 in route 2 is removed, then customer 3 with commodity 2 is removed. Once a customer is removed, the remaining capacity of this route increases and the cost decreases. Note that even if customer 3 has been removed from route 2, he is still present in route 1.

Figure 5 shows a set of routes represented as two sequences of *customer-commodities*. In order to better understand the feature of C-SDVRP, we hide the circle which represents the customer. In fact, the two routes imply the same solution as shown in Figure 4. If commodity 2 of customer 3 in route 2 is removed, then the remaining capacity of this route increases and the cost decreases. However, if one commodity (like commodity 1) of customer 1 is removed, then the remaining capacity of this route increases but the cost of this route does not change.

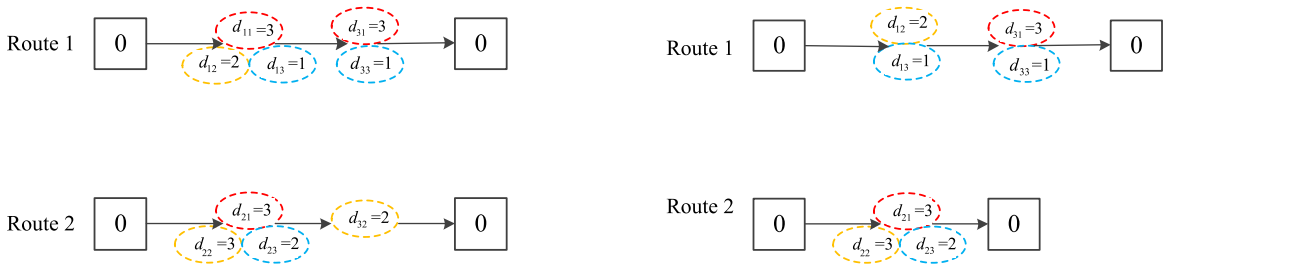


(a) Initial solution.

(b) Solution after removing customer 1 from route 1 and customer 3 from route 2.

Figure 4: Two sequences of customers.

As mentioned above, in this paper we present operators for *customers* and *customer-commodities* that work based on the specific need. It is important to note that to deal with both *customers* and *customer-commodities*, each route in the solution has two concurrent representations. In the first



(a) Initial solution.

(b) Solution after removing commodity 1 of customer 1 from route 1 and commodity 2 of customer 3 from route 2.

Figure 5: Two sequences of *customer-commodities*.

representation, each route contains a sequence of *customers*, and a set of commodities is associated with each customer (see Figure 4 for an example). In the second representation, each route contains a sequence of *customer-commodities* (see Figure 5 for an example). When using an operator, the corresponding representation (customers or *customer-commodities*) is used. The following considerations are then taken into account when translating one representation to the other:

- In the second representation with *customer-commodities*, when a route contains several commodities of the same customer, it is always optimal to group them (since there is zero cost to travel between these *customer-commodities*). Thus, in the first representation, a customer will not appear twice on the same route.
- When dealing with the second representation with *customer-commodities*, it is possible that a customer appears in different routes (with different commodities). Hence, it is possible in the first representation with customers that a customer appears in several routes (e.g., customer 3 in Figure 4(a)).
- When dealing with the first representation with customers, moving a customer means to move that customer with the associated commodities in the current route. These commodities may be a subset of the commodities required by this customer.

3.1 General framework

The basic idea of ALNS is to improve the current solution by destroying it and rebuilding it. It relies on a set of removal and insertion heuristics which iteratively destroy and repair solutions. The removal and insertion heuristics are selected using a roulette wheel mechanism. The probability of selecting a heuristic is dynamically influenced by its performance in past iterations (Pisinger and Ropke, 2010). A sketch of the method is outlined in Algorithm 1.

In Algorithm 1, s_{best} represents the best solution found during the search, while s is the current solution at the beginning of an iteration. The cost of a solution s is denoted by $f(s)$.

A removal heuristic h_{rem} and an insertion heuristic h_{ins} are applied to the current solution s. We indicate by s_{rem} and s_{ins} the intermediate solutions obtained after applying h_{rem} and h_{ins} respectively.

h_{rem} removes and h_{ins} inserts ρ customers or *customer-commodities*, where ρ is a parameter that varies between ρ^{min} and ρ^{max}. We adapt the strategy proposed by François et al. (2016) to set the value of ρ: we slightly *destroy* the current solution when a new solution has just been accepted (small values of ρ), and we increase the value of ρ proportionally to the number of iterations without improvements. The probabilities of selecting a removal or an insertion heuristic are dynamically adjusted during the search (Section 3.8).

Once the heuristics h_{rem} and h_{ins} have been applied, the solution obtained s_{ins} is possibly improved by applying a local search. The resulting solution is denoted by s' (Section 3.3).

We allow the insertion heuristics to propose solutions that violate the capacity constraint. This is done with the aim of reducing the number of routes in the solution. Infeasibility is then penalized in the objective function by adding a factor proportional to the violation. Details on the penalization of the load violation are given in Section 3.9. We then try to recover feasibility by applying the local search.

Whenever a new best solution is obtained, a mathematical programming based operator (MPO) is applied to further improve the new best solution (Section 3.7). This can be seen as an intensification phase of the algorithm.

The new solution s' is then subject to an acceptance rule. If accepted, the new solution becomes the current solution. Otherwise, the current solution does not change. This is repeated until a stopping criterion is met and the best solution found s_{best} is returned.

Algorithm 1 Adaptive large neighborhood search framework.

1: $s_{init} \leftarrow$generate an initial feasible solution using split procedure and LS
2: $\rho \leftarrow \rho^{min}$, $s \leftarrow s_{init}$, $s_{best} \leftarrow s_{init}$
3: **repeat**
4: Roulette wheel: select a removal heuristic h_{rem} and an insertion heuristic h_{ins}
5: Destroy: $s_{rem} \leftarrow$ remove ρ customers (or *customer-commodities*) from s applying h_{rem}
6: Repair: $s_{ins} \leftarrow$ insert removed *customer-commodities* into s_{rem} applying h_{ins}
7: Improve: $s' \leftarrow$ improve solution s_{ins} with local search (Algorithm 2)
8: **if** $f(s') < f(s_{best})$ **then**
9: $s_{best} \leftarrow$ improve solution s' with MPO (Section 3.7)
10: $s' \leftarrow s_{best}$
11: **end if**
12: **if** accept(s', s) **then**
13: $s \leftarrow s'$, $\rho \leftarrow \rho^{min}$
14: **else**
15: $\rho \leftarrow \rho + 1$, or ρ^{min} if $\rho = \rho^{max}$
16: **end if**
17: Update roulette wheel
18: **until** stopping criterion is met
19: return s_{best}

In the following, each component of this algorithm is explained in detail.

3.2 Initial solution

Let n_{cc} be the number of *customer-commodities*. A feasible initial solution is constructed as follows. First, we randomly determine a sequence of *customer-commodities* and we construct a giant tour $S = (S_0, S_1, \ldots, S_{n_{cc}})$, where S_0 represents the depot and S_i is the i^{th} *customer-commodity* in the sequence. Then, we apply a split procedure to get a feasible solution. This procedure is inspired by the works of Beasley (1983) and Prins (2004). It works on an auxiliary graph $\mathcal{H} = (\mathcal{X}, \mathcal{A}_{cc}, \mathcal{Z})$, where $\mathcal{X}$ contains $n_{cc}+1$ nodes indexed from 0 to n_{cc}, where 0 is a dummy node and node $i, i > 0$, represents *customer-commodity* S_i. $\mathcal{A}_{cc}$ contains one arc (i, j), $i < j$, if a route serving *customer-commodities* S_{i+1} to S_j is feasible with respect to the capacity Q of the vehicle. The weight z_{ij} of arc (i, j) is equal to the cost of the trip that serves $(S_{i+1}, S_{i+2}, \ldots, S_j)$ in this same order. The optimal splitting of the giant tour S corresponds to a minimum cost path from 0 to n_{cc} in $\mathcal{H}$. Finally, this feasible solution is improved by applying local search (Algorithm 2).

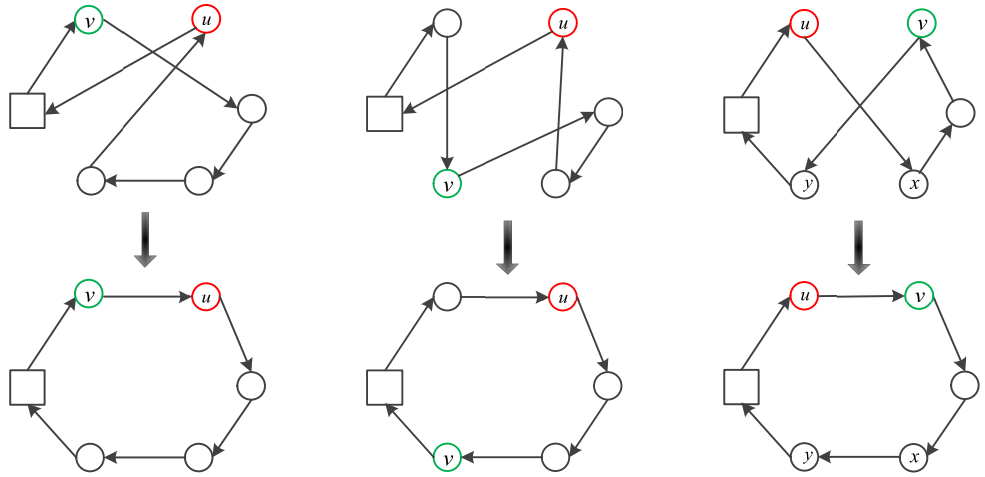
3.3 Local search

In order to improve a solution, a local search procedure (LS) is applied. LS is based on a set of classical operators that work on customers and on *customer-commodities*.

Let us first introduce some notation that will be useful in the remaining part of the section. Let u and v be two different nodes. They are associated with a customer or a *customer-commodity*, or one of them may be the depot depending on the operator. These nodes may belong to the same route or different routes. Let x and y be the successors of u and v in their respective routes. $R(u)$ denotes the route that visits node u.

Operators on customers

Here we present the operators that are defined for customers. These operators consider a customer together with all the commodities delivered to this customer in a given route. The different operators are illustrated in Figure 6 (where we only represent the intra-route cases).



(a) Insert customer. (b) Swap customers. (c) 2-opt on customers.

Figure 6: Local search operators with customers.

Insert customer: this operator removes a customer u and inserts it after customer v.
Swap customers: this operator swaps the positions of customer u and customer v.
2-opt on customers: if $R(u) = R(v)$, this operator replaces (u, x) and (v, y) by (u, v) and (x, y).

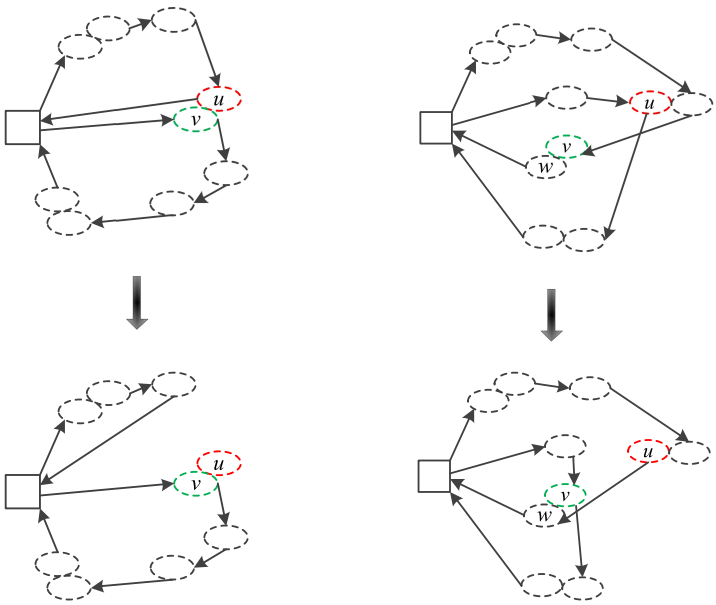
Operators on *customer-commodities*

Here, we present the operators that are defined for *customer-commodities*. The different operators are illustrated in Figure 7.

Insert *customer-commodity*: this operator removes a *customer-commodity* u then inserts it after *customer-commodity* v.

Swap *customer-commodities*: this operator swaps *customer-commodity* u and *customer-commodity* v.

We can notice that the insert and swap operators for *customer-commodities* allow some moves that are not feasible when we consider only customers. In most cases, the insertion of a customer or of only one of the commodities to be delivered has the same cost. However, during the search, we allow the violation of the vehicle capacity, and the infeasibility is penalized (see Section 3.9). In an infeasible solution with some overcapacity in a route, it may be possible that inserting customers in other routes also leads to an infeasibility; while inserting only a *customer-commodity* decreases the infeasibility or leads to a feasible solution. Similarly, a swap of customers may not be feasible (or



(a) Insert *customer-commodity*. (b) Swap *customer-commodities*.

Figure 7: Local search operators with *customer-commodities*.

lead to an increase in the cost) due to the vehicle capacity restriction. At the opposite, when the commodities of some customers are delivered in several routes, swapping *customer-commodities* may be feasible with respect to the capacity and decreases the solution cost. As an example, in Figure 7, case (b), v and w are two commodities for the same customer, while u is a commodity for another customer who requires two commodities. Swapping u with both v and w may not be possible because of vehicle capacity while swapping *customer-commodities* u and v decreases the cost of the solution.

Note that we do not consider a 2-opt operator based on *customer-commodities*. Since this operator works on elements of the same route, it is never beneficial to split apart *customer-commodities* associated to the same customer. It follows that its behavior would be the same as the operator 2-opt on customers.

LS is applied when the split procedure has generated an initial solution or when the removal and insertion heuristics have modified the solution. In the first case, we do not allow the solution to be infeasible with respect to the vehicle capacity. In the second case, the insertion heuristic can produce infeasible solutions. As a consequence, LS (and thus the operators) may have to address the infeasibility of the current solution. LS is depicted in Algorithm 2. Given a solution, first, four local search operators are applied: *insert customer*, *insert customer-commodity*, *swap customers* and *swap customer-commodities*. They are invoked iteratively until there is no further improvement. Then the operator *2-opt of customers* is applied. If it improves the solution, we reiterate with the first four operators. After exploring the neighbor defined by each operator, the move that improves the most is implemented. When all the local search operators fail, routes of the current solution are concatenated and split algorithm is applied again. This strategy is inspired by Prins (2009). If this provides a better solution, the whole procedure is repeated.

3.4 Removal heuristics

This section describes the set of removal heuristics we propose to destroy the current solution. Heuristics *Shaw removal* and *worst removal* use the representation of a solution with customers, while *random removal* can be applied to both representations with customers and *customer-commodities*. Another operator that randomly removes one route is also considered.

Shaw removal: this heuristic aims to remove a set of customers which are similar based on a

Algorithm 2 Local search procedure.

```
Let s be a (feasible or infeasible) solution
repeat
    repeat
        repeat
            Obtain s' by applying the insert customer operator; s = s'
            Obtain s' by applying the insert customer-commodity operator; s = s'
            Obtain s' by applying the swap customers operator; s = s'
            Obtain s' by applying the swap customer-commodities operator; s = s'
        until No improvement has been obtained
        Obtain s' by applying the 2-opt of customers operator; s = s'
    until No improvement has been obtained
    Obtain a giant tour by concatenation of trips
    Apply the split procedure
until No improvement has been obtained
```

specified criterion (e.g., location or demand). When customers with really different characteristics are removed, it is likely that each customer is then reinserted at the same position in the solution. Hence, by removing similar customers, Shaw removal aims to provide a different solution once an insertion heuristic has been applied.

Here, we define similarity between two customers as the distance between these two customers. The heuristic works as follows: a first customer i is randomly selected and removed. We then compute the similarity between customer i and the other customers (here, it is the distance), and sort the customers in a list L, according to the similarity with customer i. A determinism parameter p_d ($p_d \geq 1$) is used to have some randomness in the customer selection to be removed in L, with a higher probability for the firsts customers. The removed customer then plays the role of customer i and the procedure is repeated until ρ customers have been removed. The interested reader can find a detailed Shaw removal pseudocode in Ropke and Pisinger (2006).

Worst removal: this heuristic aims at removing the customers who induce a high cost in the solution. More precisely, at each iteration, we first calculate for each customer the cost decrease if it is removed from the solution. Then customers are sorted in decreasing order according to these values. As in Shaw removal, a determinism parameter p_d controls the randomization in the choice of the worst customer to remove.

Random removal: this removal heuristic randomly chooses ρ customers and removes them from the current solution. It can also be applied with *customer-commodities*, by randomly removing ρ *customer-commodities*.

Route removal: in this removal heuristic, an entire route from the current solution is randomly selected, and all the *customer-commodities* on this route are removed.

3.5 Insertion heuristics

In this section, we describe the insertion heuristics implemented in the proposed ALNS algorithm. In this work, all insertion heuristics consider *customer-commodities*.

Greedy insertion: in this insertion heuristic, at each iteration, for each removed *customer-commodity* i, we first compute the *best insertion cost* Δf_i^1: cost of inserting the *customer-commodity* at its best position into the solution (i.e., the insertion that minimizes the increase of the cost of the solution). Then, the *customer-commodity* with the minimum insertion cost is selected to be inserted at its best position. After each iteration, the insertion costs of the remaining removed *customer-commodities* are recomputed. This process stops when all *customer-commodities* have been inserted.

Regret insertion: this insertion heuristic chooses, at each iteration, the removed *customer-*

commodity which produces the biggest regret if it is not inserted at its best position at the current iteration. In regret-k heuristic, at each iteration, we first calculate, for each removed *customer-commodity* i, Δf_i^1 the cost of inserting i at its best position, and Δf_i^η ($\eta \in \{2; ...; k\}$ the cost of inserting i at its η^{th} best position. Then, for each *customer-commodity* i the regret value is computed as: $reg_i = \sum_{\eta=2}^{k}(\Delta f_i^\eta - \Delta f_i^1)$. This represents the extra cost if i is not inserted at the current iteration in its best position. Finally, the *customer-commodity* with highest regret value reg_i is inserted at its best position into the solution. The heuristic continues until all *customer-commodities* have been inserted. For regret-k insertion heuristics in this work, we have considered the values of k to be two and three.

Random insertion: this insertion heuristic randomly chooses a removed *customer-commodity*, and randomly chooses the insertion position in the solution.

Note that when inserting *customer-commodities*, violations of vehicle capacity are allowed and penalized in the cost function (see Section 3.9). However, we also impose a maximum capacity violation on each route. Hence, it is possible that a *customer-commodity* cannot be inserted in any route of the current solution. In this case, we create one additional route which includes this *customer-commodity*.

3.6 Acceptance and stopping criterion

When the removal, insertion and LS steps have been applied, we use a simulated annealing criterion to determine if the new solution obtained s' is accepted. However, a deterministic decision rule is applied in two cases. At each iteration of the algorithm, if s' has a lower cost than the current solution s ($f(s') < f(s)$), then s' is accepted. The solution s' is rejected if the costs $f(s')$ and $f(s)$ are equal. We reject solutions with the same cost in order to avoid working with equivalent solutions where some *customer-commodities* belonging to the same customer have been exchanged.

When $f(s') > f(s)$, then s' is accepted with probability $e^{-(f(s')-f(s))/T}$, where $T > 0$ is the temperature. As proposed in Ropke and Pisinger (2006), the initial temperature is set such that a solution which is $w\%$ worst than the initial solution s_{init} is accepted with a probability p_{accept}. More formally, T is chosen such that $e^{-(w \cdot f(s_{init}))/T} = p_{accept}$. Then, at each iteration of the ALNS, the temperature T is decreased using the formula $T = T \cdot \gamma$, where $\gamma \in [0, 1]$ is the cooling factor.

The stopping criterion for the whole procedure is a fixed number of ALNS iterations.

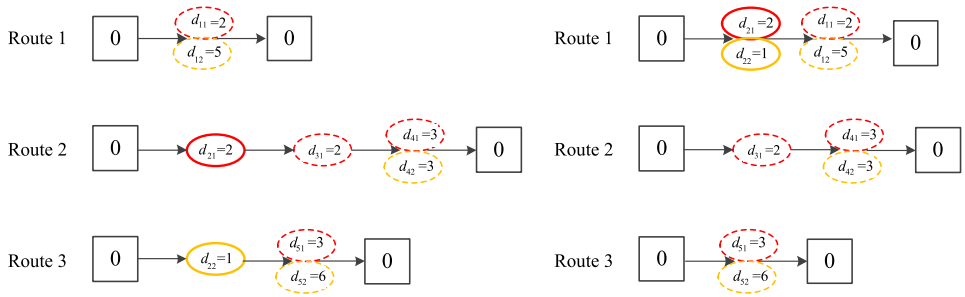
3.7 Mathematical programming based operator to reassign commodities

When a new best solution is identified, we intensify the search by applying a mathematical programming based operator (MPO). The main purpose is to assign the visits to a specific customer among the solution routes in a different way by solving a capacitated facility location problem.

We use Figures 8 (a) and (b) to explain the idea behind MPO. In the example, we assume the vehicle capacity is 10 units. The number d_{im} in the ellipse reflects the demand of commodity m required by customer i. We focus on the commodities of customer 2: the ellipses with a solid line in Figure 8. We assume that Figure 8 (a) is a solution obtained after LS. Customer 2 has two commodities: the first is delivered on route 2, and the second is delivered on route 3. Inserting or swapping one of these two *customer-commodities* does not provide a better solution. However, we can notice that the deliveries to customer 2 can be reassigned to the first route and the total cost decreases, as shown in Figure 8 (b). The reader can notice that the operators implemented in LS do not consider this kind of movements.

Let us introduce the notation that we need to formally present MPO. First, we assume that customer i is considered. We indicate by:

- $\mathcal{M}_i$ the set of commodities required by customer i;
- s_i the solution obtained from the current solution by removing all the visits to customer i;



(a) Solution before using MPO. (b) Solution after using MPO.

Figure 8: MPO for reassigning commodities.

- $\mathcal{R}_i$ the set of routes in s_i;
- c_r^i the cost to insert (best insertion) customer i in route $r \in \mathcal{R}_i$;
- Q_r^i the remaining capacity in route $r \in \mathcal{R}_i$.

Then, we introduce the following binary decision variables:

$$x_{mr}^i = \begin{cases} 1 & \text{if the delivery of commodity } m \text{ of customer } i \text{ is assigned to route } r \in \mathcal{R}_i; \\ 0 & \text{otherwise.} \end{cases}$$

$$x_r^i = \begin{cases} 1 & \text{if at least the delivery of one commodity required by customer } i \text{ is assigned to route } r \in \mathcal{R}_i; \\ 0 & \text{otherwise.} \end{cases}$$

The mathematical program that we solve in order to apply MPO is the following:

$$(IP_{MPO}) \; min \sum_{r \in \mathcal{R}_i} c_r^i x_r^i \tag{1}$$

$$\text{s.t.} \sum_{r \in \mathcal{R}_i} x_{mr}^i = 1, \qquad \forall\, m \in \mathcal{M}_i \tag{2}$$

$$\sum_{m \in \mathcal{M}_i} d_{im} x_{mr}^i \leq Q_r^i x_r^i, \qquad \forall\, r \in \mathcal{R}_i \tag{3}$$

$$x_{mr}^i \in \{0,1\}, \qquad \forall\, m \in \mathcal{M}_i, r \in \mathcal{R}_i \tag{4}$$

$$x_r^i \in \{0,1\}, \qquad \forall\, r \in \mathcal{R}_i \tag{5}$$

This mathematical program corresponds to a capacitated facility location problem, where only the costs related to the inclusion of a new route in the solution (the fixed cost) are taken into account. The objective function (1) aims to minimize the total insertion cost. Constraints (2) require that the delivery of each commodity (i.e., the delivery of each *customer-commodity*) must be assigned to one route. Constraints (3) impose that the total quantity of commodities assigned to a selected vehicle does not exceed its remaining capacity. Constraints (4)-(5) define the decision variables.

(IP_{MPO}) is solved for each $i \in \mathcal{V}_C$, but only the reassignment of visits associated with the highest cost reduction is effectively implemented.

3.8 Adaptive weight adjustment

A roulette wheel is used to give more or less importance to the removal and insertion heuristics to be used. The procedure implemented is based on the principles described in Ropke and Pisinger (2006) and François et al. (2016). This last variant includes a normalization process for the score. The main difference in our approach is that removal and insertion heuristics are not selected independently. A pair of removal and insertion heuristics $h_p = \{h_{rem}; h_{ins}\}$ is chosen in each iteration. We denote by $\mathcal{H}_p$ the set of pair of heuristics to be used. Each pair h_p is associated with a weight ω_{h_p}, a score π_{h_p}, and θ_{h_p} the number of times that pair h_p has been used. Initially, all pairs of heuristics have the same weight and $\sum_{p \in \mathcal{H}_p} \omega_{h_p} = 1$.

We define a segment as a fixed number of ALNS iterations in the proposed algorithm. During a segment, the weights of all pairs are kept constant. Before starting a new segment, for each pair $h_p \in \mathcal{H}_p$, the score π_{h_p} and the number of times the pair is used θ_{h_p} are reset to 0. During a segment, each time a pair h_p is used, θ_{h_p} is increased by 1, and if the new solution s' is accepted after using pair h_p, the score π_{h_p} is updated according to: $\pi_{h_p} \leftarrow \pi_{h_p} + \sigma_\mu$, where σ_μ ($\mu \in \{1; 2; 3\}$) reflects different cases regarding the score change π_{h_p}. That is, the score π_{h_p} is increased by σ_1 when s' is a new best solution, or σ_2 when s' is a new improved solution ($f(s') < f(s)$) but not a new best solution, or σ_3 when s' is not an improved solution but has been accepted according to the simulated annealing criterion. The values σ_μ ($\sigma_\mu \in [0, 1]$) are normalized to satisfy $\sigma_1 + \sigma_2 + \sigma_3 = 1$.

At the end of each segment, we update all the weights of the pairs of heuristics based on the recorded scores. First, the score π_{h_p} is updated as: $\pi_{h_p} \leftarrow \frac{\pi_{h_p}}{\theta_{h_p}}$, where θ_{h_p} is the number of times that pair h_p was used in this segment. If $\theta_{h_p} = 0$, we set π_{h_p} to the same value as in the previous segment. Then, the scores of all pairs of heuristics are normalized:

$$\bar{\pi}_{h_p} = \frac{\pi_{h_p}}{\sum_{h \in \mathcal{H}_p} \pi_h}. \tag{6}$$

Let $\omega_{h_p,q}$ be the weight of pair h_p used in segment q, and $\lambda \in [0, 1]$ a factor which determines the change rate in the performance of the pair of heuristics. At the end of segment q, the weight of all pairs of heuristics h_p to be applied in segment $q + 1$ is updated as:

$$\omega_{h_p,q+1} = (1 - \lambda)\omega_{h_p,q} + \lambda \bar{\pi}_{h_p}. \tag{7}$$

3.9 Infeasibility penalization scheme

In our implementation of the ALNS, we allow some violations of the vehicle capacity in order to reduce the number of routes. Let K_{init} be the number of vehicles used in the initial solution s_{init}. Then, the vehicle capacity Q can be extended by an amount of $Q_{extra} = \frac{Q}{K_{init}}$. If a vehicle delivers more than Q units of products, we penalize the infeasibility by adding to the solution cost a term proportional to the load violation which is $\beta l(s)$, where $l(s)$ is the total load violation of the solution s and β is the penalty rate.

The penalty rate β is related to capacity violations. Initially, β is set equal to a minimum value β_{min} computed as

$$\beta_{min} = 10 \cdot \frac{f(s_{init})}{\sum_{i \in \mathcal{V}_C, m \in \mathcal{M}} d_{im}},$$

where $f(s_{init})$ is the cost of the initial solution obtained after applying the split procedure and LS. This ratio approximates an average transportation cost per unit of demand.

The penalty rate β is dynamically modified during the search since a high rate may eliminate infeasibility quickly, but may also prevent searching other promising regions. We keep track of the number of infeasible solutions obtained during the last consecutive I iterations of the ALNS algorithm. If I_{inf} infeasible solutions are obtained consecutively, the value of β is increased to 2β. Similarly, if

I_{feas} feasible solutions are generated consecutively, the value of β is decreased to $\max\{\beta_{min}; \beta/2\}$.

4 Computational experiments

In this section, we present the results obtained by the proposed ALNS heuristic. The algorithm was implemented in C++ and ran on an Intel (R) Core(TM) i7-4600U, 2.10GHz, and 16GB of RAM.

We first describe in Section 4.1 the test instances on which we evaluate our algorithm. In Section 4.2 we report the values of the parameters that we set for the ALNS. Then, in Section 4.3 we perform a sensitivity analysis to determine which insertion and removal heuristics lead to the best performance of the proposed algorithm. A set of settings is tested on a subset of instances, and the best is used to perform all other computational tests. In Section 4.4, we analyze the effect of the number of iterations on the computational time and the solution quality. Then, Section 4.5 reports the results on the testbed. The effectiveness of the MPO is studied in Section 4.6, while in Section 4.7 we validate the effectiveness of integrating a local search in the large neighborhood search. We examine how the computational time of our method varies according to the instance size in Section 4.8. Finally, Section 4.9 provides managerial insights about the split customers that are delivered by more than one vehicle.

4.1 Instances

To assess the efficiency of our algorithm, we perform computational experiments on the benchmark instances proposed by Archetti et al. (2014). These instances are built from the R101 and C101 Solomon instances for the VRP with Time Windows (Solomon (1987)), from which the customer locations are kept. From instances R101 and C101, several instances for the C-SDVRP were generated by considering the n first customers, with $n \in \{15, 20, 40, 60, 80, 100\}$.

In the testbed, up to 3 commodities are taken into account. The number of commodities is indicated by M ($M \in \{2; 3\}$). Each commodity is required by a customer with a probability p of 0.6 (on average each customer needs 2 out of 3 commodities) or with probability p equal to 1 (each customer requires all commodities). The quantity of each commodity required by a customer varies within the intervals $\Delta = [1, 100]$ or $\Delta = [40, 60]$. A last parameter $\alpha \in \{1.1, 1.5, 2, 2.5\}$ is used to determine the vehicle capacity from the original one in Solomon instances. We indicate by $\mathcal{P} = (\mathcal{I}, M, p, \Delta, \alpha)$ (where $\mathcal{I} \in \{R101, C101\}$) the set of parameters used by Archetti et al. (2014) to generate instances.

When $n = 15$, 64 instances are available, one for each combination of parameters in set $\mathcal{P}$. These are referred as small instances. When $n \in \{20, 40, 60, 80\}$, 80 mid-size instances are available: 20 instances for each value of n. For the mid-size instances, the combination of parameters in $\mathcal{P}$ is restricted to $M = 3$, $\alpha = 1.5$ and $\Delta = [1, 100]$. Hence $\mathcal{I}$ and p can take two values each, leading four combinations of parameters. For each combination, five instances have been generated, leading to 20 instances in total. When $n = 100$, 320 large instances are available, that means five instances for each combination of parameters in set $\mathcal{P}$.

In the following sections, we present the parameter setting for our algorithm and the results obtained. We provide detailed results on each of the 464 instances in the Appendix A. Due to the high number of instances, we only report average results. In particular, we present results for each *group* of instances, where a group is defined by a quartet $(n, \mathcal{I}, M, p)$. Results for a group are averaged on the values of α and Δ.

We summarize all the notations used to present the results in Table 1.

4.2 ALNS parameters

In this section, we present the values of the parameters set in our ALNS algorithm. The probabilities to select the pairs of removal and insertion heuristics are updated after a number of iterations called a segment. In our implementation, we define a segment as 100 iterations.

Table 1: Notations for computational results

Symbol	Meaning
C101/R101	Name of the original Solomon instance
n	Number of customers
M	Number of commodities
p	Probability that a customer requires a commodity
id	Instance id (5 mid and large sized instances are generated for each combination of parameters in $\mathcal{P}$)
Δ	Demand range of each customer for each commodity
α	Value for generating the vehicle capacity (see Archetti et al. (2015))
CC_m	Total number of customers that require commodity m
n_{cc}	Total number of *customer commodities*
$avg.n_{cc}$	Average total number of *customer commodities* in each group of instances
$nbIns$	Number of instances in each group
OPT	Optimal solution value from literature (when available)
BKS	Best known solution value from literature (when available)
$Cost$	Best solution cost found by the proposed algorithm
Δ_O	Percentage of improvement between Cost and OPT ($\Delta_O = 100 * (Cost - OPT)/OPT$)
Δ_B	Percentage of improvement between Cost and BKS ($\Delta_B = 100 * (Cost - BKS)/BKS$)
$\Delta_{O/B}$	To indicate that the value is Δ_O if an optimal value if available, and Δ_B otherwise
$avg.\Delta_O$	Average percentage of improvement between Cost and OPT ($\Delta_O = 100 * (Cost - OPT)/OPT$)
$avg.\Delta_B$	Average percentage of improvement between Cost and BKS ($\Delta_B = 100 * (Cost - BKS)/BKS$)
$min.\Delta_O$	Minimum percentage of improvement between Cost and OPT ($\Delta_O = 100 * (Cost - OPT)/OPT$)
$max.\Delta_B$	Maximum percentage of improvement between Cost and BKS ($\Delta_B = 100 * (Cost - BKS)/BKS$)
$t(s)$	CPU time in second
$avg.t(s)$	Average CPU time in seconds
$nbOPT$	Number of optimal solutions obtained
nbE	Number of solution values obtained equal to the best known
$nbNBK$	Number of new best known solutions obtained
nbR	Number of routes
$nbMPO$	Number of times that MPO is called for a instance
$nbMPOimp$	Number of times that MPO improves the new best solution during the search
$avg.nbMPO$	Average number of times that the MPO is called for each group of instances
$avg.nbMPOimp$	Average number of times that the MPO improves the current best solution for each group of instances
*	Indicates the solution value is optimal
$nbSplit$	Number of split customers (that are delivered by more than one vehicle)
$nb2\text{-}split$	Number of split customers delivered by exactly two vehicles
$nb3\text{-}split$	Number of split customers delivered by exactly three vehicles
$nbNearDepot$	Number of split customers that are close to the depot
$nbLargeDemand$	Number of split customers with a large demand
$nbCluster$	Number of split customers located inside a cluster of customers

To set the values of σ_1, σ_2, σ_3, preliminary experiments were carried out on instances with $n = 80$ customers for some combinations. The best results were obtained when $\sigma_1 = 0.7$, $\sigma_2 = 0.1$, $\sigma_3 = 0.2$. We set the reaction factor λ that appears in Equation (7) to 0.5 as proposed in Masson et al. (2013) (instead of $\lambda = 0.1$ as in Ropke and Pisinger (2006)) to ensure higher reactivity when performing fewer iterations. The determinism parameter p_d in Shaw removal and worst removal heuristics is equal to 6 as in Ropke and Pisinger (2006).

The acceptance of a new current solution is based on a simulated annealing criterion. The initial value of the temperature T is set so that a solution that is w% worse than the current solution is accepted with probability p_{accept} with $w = 0.35$ and $p_{accept} = 0.7$. In addition, we set the cooling factor γ to 0.999.

In order to destroy a solution, we need to determine the number ρ of customers (or *customer-commodities*) to be removed. We follow the scheme proposed by François et al. (2016): small moves are applied when a new solution has just been accepted, while large moves are applied when no new solution has been accepted in the most recent iterations. The value of ρ evolves in the interval $[\rho^{min}, \rho^{max}]$. For small instances ($n = 15$), we set $\rho^{min} = N/2$, $\rho^{max} = N$, while for the other instances, we set $\rho^{min} = N/10$ and $\rho^{max} = N/4$. When the removal heuristic is customer-based, then N represents the number of customers ($N = n$). When the removal heuristics are defined for *customer-commodities*, N represents the number of *customer-commodities* ($N = n_{cc}$).

To update the penalization rate for capacity violations, we consider the number of infeasible and feasible solutions that are obtained consecutively. These values are respectively set to $I_{inf} = 50$ and

$I_{feas} = 5$ after performing preliminary experiments.

4.3 Efficiency assessment for the removal and insertion heuristics

In Sections 3.4 and 3.5, we propose several removal and insertion heuristics to be used inside the ALNS framework. Before testing the proposed algorithm on the set of instances, we use a subset of instances in order to determine the removal and insertion heuristics that are used inside the ALNS framework. We also consider configurations with only one removal and one insertion heuristic, which corresponds to designing a large neighborhood search (LNS). This permits to emphasize the efficiency of individual insertion and removal heuristics, and also to point out the benefit of the adaptive approach included in the design of the ALNS when choosing the removal and insertion heuristics.

This analysis is performed to tune the proposed algorithm. We only use the 20 instances with $n = 80$ customers. A summary of these experiments is reported in Table 2. The first column shows which configuration is considered (LNS or ALNS). The second column enumerates the configurations that we tested. Columns 3 to 7 indicate which removal heuristics are used in a specific configuration. Among them, 'RandC' and 'RandCC' represent the random removal heuristic considering customers and *customer-commodities*, respectively. Columns 8 to 11 indicate which insertion heuristics are used. In columns 12 to 14 we report the following average statistics for each configuration: the gap with the best-known solutions, the computational time in seconds, the number of new best-known solutions (see Table 1).

Table 2: LNS configurations compared to ALNS configurations.

		Removal heuristics					Insertion heuristics				Results		
	conf.	Shaw	RandC	RandCC	Worst	Route	Greedy	Regret-2	Regret-3	Rand	$avg.\Delta_B$	$avg.t(s)$	$nbNBK$
LNS	1	×						×			**-0.77**	521.18	**20**
	2		×					×			-0.72	438.34	19
	3			×				×			-0.54	276.76	17
	4				×			×			-0.23	714.33	13
	5					×		×			0.01	333.60	10
	6	×					×				-0.61	533.30	20
	7	×							×		**-0.78**	541.81	**19**
	8	×								×	-0.07	1106.18	11
	9		×							×	-0.10	1146.84	13
ALNS	10	×	×	×	×	×	×	×	×	×	**-0.71**	562.29	**20**
	11	×	×				×	×			-0.72	513.49	19
	12	×	×					×	×		-0.77	500.97	19
	13	×	×				×	×	×		**-0.79**	511.00	**20**
	14	×	×	×			×	×			-0.68	467.99	20
	15	×	×		×		×	×			-0.63	555.34	20
	16	×	×		×		×	×	×		-0.66	530.26	19
	17	×	×	×			×	×	×		-0.79	477.49	19

In the first five configurations, we study the influence of the removal heuristic. To this intent, the insertion heuristic is kept fixed. Using the Shaw removal heuristic, the LNS is able to improve all the best-known solutions.

Configurations 6 to 8 point out the impact of the insertion heuristic. We fixed the removal heuristic to the Shaw removal since it provided the best results in the previous experiments. These tests show that all insertion heuristics perform well except the random insertion. To further evaluate the behavior of the random insertion heuristic we consider configuration 9 where we modify the removal heuristic. It can be observed that the results do not improve. Among the LNS configurations, combining the Shaw removal with the regret-3 insertion provides the best results.

In the second part of Table 2, we consider different configurations for the ALNS framework, with several removal and insertion heuristics. Configuration 10 shows the performance of the ALNS algorithm using all proposed heuristics. In other configurations, only a subset of them is invoked in the ALNS. As we can see, using all the proposed removal and insertion heuristics does not provide

the best results. The best configuration is configuration number 13: an ALNS with two removal heuristics: Shaw and random removal of customers, and three insertion heuristics: greedy, regret-2 and regret-3. In the following sections, all reported computational experiments have been conducted using this configuration.

4.4 Analysis with respect to the number of iterations

We examine the impact of the number of iterations for different instance sizes. Preliminary computational experiments showed that the algorithm converges after a certain number of iterations. Thus, we aim to determine the number of iterations that would be a good compromise between the solution quality and the computational time.

For small instances and medium instances with 20 customers (mid-20), we run our algorithm with the number of iteration *iter* limited to 100, 1000, 3000 and 5000. We compare the results with those reported in Archetti et al. (2014) and Archetti et al. (2015). In Table 3 and Table 4 we report average statistics for different values of *iter*: the gap with respect to the optimal values, the computational time in seconds, the number of optimal solutions obtained (see Table 1).

Results over the testbed are indicated in bold. Detailed results are provided in Appendix A.

According to Table 3, the proposed algorithm solves to optimality 57 out of 64 small instances when 100 iterations are allowed, within an average computational time of 2 seconds. When we increase the number of iterations to 3000, the number of optimal solutions reaches 63 with an average computational time of 12 seconds. The average gap with optimal solutions obtained by Archetti et al. (2015) varies from 0.10% to 0.03%. Increasing the number of iterations to 5000 does not improve the quality of the results.

When $n = 20$, results reported in Table 4 indicate that our method identifies an optimal solution in 5000 iterations for 18 out of the 19 instances for which Archetti et al. (2015) provided optimal values. The average gap with the best-known values is less than 0.01%. Over the whole set, the average CPU time is less than 1 minute.

Since large instances usually require more iterations to obtain high-quality results, we compare the ALNS algorithm behavior with 5000 and 10000 iterations on mid-80 instances. In Table 5, we report average results and the gap with respect to the best-known values. Detailed results are provided in Appendix A. ALNS can provide best-known solutions for the 20 mid-80 instances in the benchmark. Within 5000 iterations, the best-known solutions can be improved by about 0.79% on average and the CPU times is less than 9 minutes. When 10000 iterations are executed, the solutions are averagely 0.92% better than the best-known values. On the other side, the average computational times are almost doubled.

4.5 Computational experiments on the whole testbed

In this section, we consider the results obtained on the whole set of instances for the C-SDVRP with the designed ALNS algorithm. As determined in the previous sections, the removal and insertion heuristics are the ones of configuration 13, and the number of iterations is equal to 3000 (resp. 5000) on small (resp. medium and large) instances. The algorithm is run once on each instance.

The comparison is made with the results reported in Archetti et al. (2014) and Archetti et al. (2015). Archetti et al. (2015) propose a branch-and-price algorithm that can solve to optimality instances with up to 40 customers but that can systematically close instances with up to 20 customers. When the optimal value is not available we compare to the best known solution (BKS) given either by Archetti et al. (2014) or Archetti et al. (2015).

For the 64 small instances, an optimal solution is known. Results reported in Table 3 indicate that our algorithm finds an optimal solution for 63 out of 64 instances. The average optimality gap is 0.03%.

Table 3: Impact of the number of iterations on small instances.

instances						ALNS (100 iterations)			ALNS (1000 iterations)			ALNS (3000 iterations)			ALNS (5000 iterations)		
	n	M	p	$avg.n_{cc}$	$nbIns$	$avg.\Delta_O$	$avg.t(s)$	$nbOPT$	$avg.\Delta_O$	$avg.t(s)$	$nbOPT$	$avg.\Delta_O$	$avg.t(s)$	$nbOPT$	$avg.\Delta_O$	$avg.t(s)$	$nbOPT$
C101	15	2	0.6	22	8	0.00	0.93	8	0.00	2.82	8	0.00	7.24	8	0.00	11.65	8
C101	15	2	1	30	8	0.01	1.64	7	0.00	4.45	8	0.00	10.63	8	0.00	16.86	8
C101	15	3	0.6	28	8	0.00	1.13	8	0.00	3.92	8	0.00	10.40	8	0.00	17.12	8
C101	15	3	1	45	8	0.47	3.72	4	0.33	9.81	7	0.21	22.26	7	0.21	34.62	7
R101	15	2	0.6	22	8	0.00	1.07	8	0.00	2.89	8	0.00	7.02	8	0.00	11.05	8
R101	15	2	1	30	8	0.00	2.10	8	0.00	4.75	8	0.00	10.76	8	0.00	16.99	8
R101	15	3	0.6	28	8	0.00	1.26	8	0.00	3.88	8	0.00	9.65	8	0.00	15.59	8
R101	15	3	1	45	8	0.35	3.36	6	0.00	8.63	8	0.00	20.21	8	0.00	31.57	8
total					64	**0.10**	**1.90**	**57**	**0.04**	**5.14**	**63**	**0.03**	**12.27**	**63**	**0.03**	**19.43**	**63**

Table 4: Impact of the number of iterations on mid-20 instances.

instances						ALNS (100 iterations)			ALNS (1000 iterations)			ALNS (3000 iterations)			ALNS (5000 iterations)		
	n	M	p	$avg.n_{cc}$	$nbIns$	$avg.\Delta_{O/B}$	$avg.t(s)$	$nbOPT$	$avg.\Delta_{O/B}$	$avg.t(s)$	$nbOPT$	$avg.\Delta_{O/B}$	$avg.t(s)$	$nbOPT$	$avg.\Delta_{O/B}$	$avg.t(s)$	$nbOPT$
C101	20	3	0.6	37.4	5	0.26	2.41	3	0.10	8.11	4	0.10	20.66	4	0.00	33.38	5
C101	20	3	1	60	5	0.66	5.75	1	0.11	21.13	2	0.04	49.66	4	0.00	77.14	5
R101	20	3	0.6	37.4	5	0.01	3.87	4	0.00	9.60	5	0.00	22.09	5	0.00	34.87	5
R101	20	3	1	60	5	0.82	10.68	1	0.16	25.66	2	0.01	54.17	3	0.01	81.72	3
		total			20	**0.44**	**5.68**	**9**	**0.09**	**16.12**	**13**	**0.04**	**36.65**	**16**	**0.00**	**56.78**	**18**

Table 5: Impact of the number of iterations on mid-80 instances.

instances						ALNS (5000 iterations)			ALNS (10000 iterations)		
	n	M	p	$avg.n_{cc}$	$nbIns$	$avg.\Delta_B$	$avg.t(s)$	$nbNBK$	$avg.\Delta_B$	$avg.t(s)$	$nbNBK$
C101	80	3	0.6	150.4	5	-0.77	373.36	5	-0.88	690.87	5
C101	80	3	1	240	5	-0.75	704.20	5	-0.83	1339.55	5
R101	80	3	0.6	150.4	5	-0.84	319.83	5	-1.06	605.74	5
R101	80	3	1	240	5	-0.81	646.61	5	-0.93	1196.69	5
	total				20	**-0.79**	**511.00**	**20**	**-0.92**	**958.21**	**20**

For the mid-20 instances ($n = 20$), 19 out of 20 instances have a known optimal value. The proposed algorithm can find 18 out of 19 optimal values. On average, the gap with respect to BKS (either optimal or feasible) is lower than 0.01%.

For the mid-40 instances ($n = 40$), average results are reported in Table 6. Only 5 optimal values are known. On average, the proposed ALNS finds solutions that are 0.26% better than the best-known solutions (either optimal or feasible). Our algorithm finds 4 out of the 5 known optimal solution while 9 new best-known solutions have been identified. The algorithm runs in less than 2 minutes on average.

Table 6: Summary of results on mid-40 sized instances.

instances						ALNS results						
	n	M	p	$avg.n_{cc}$	$nbIns$	$avg.\Delta$	$min\Delta$	$max\Delta$	$avg.t(s)$	$nbOPT$	nbE	$nbNBK$
C101	40	3	0.6	76.2	5	-0.12	-0.42	0.15	78.14	2	-	2
C101	40	3	1	120	5	-0.76	-1.91	0.00	161.39	-	-	4
R101	40	3	0.6	76.2	5	0.15	0.00	0.29	80.70	1	-	-
R101	40	3	1	120	5	-0.32	-0.78	0.00	148.33	1	1	3
		total			20	**-0.26**	-1.91	0.29	**117.14**	**4**	**1**	**9**

For instances with $n \geq 60$, no optimal solution is available. For the mid-60 instances ($n = 60$) (see Table 7), the ALNS finds solutions that are on average 0.49% better than the BKS. Among them, 15 new best-known values are obtained. The CPU times are less than 5 minutes on average.

Table 7: Summary of results on mid-60 sized instances.

instances						ALNS results					
	n	M	p	$avg.n_{cc}$	$nbIns$	$avg.\Delta_B$	$min\Delta_B$	$max\Delta_B$	$avg.t(s)$	nbE	$nbNBK$
C101	60	3	0.6	110	5	-0.56	-1.40	0.00	193.12	1	4
C101	60	3	1	180	5	-0.38	-1.13	0.25	403.78	-	3
R101	60	3	0.6	110	5	-0.41	-0.83	0.00	177.20	1	3
R101	60	3	1	180	5	-0.61	-1.20	-0.14	353.24	-	5
	total				20	**-0.49**	-1.40	0.25	**281.83**	**2**	**15**

As presented in Table 5 on mid-80 instances ($n = 80$), the ALNS identifies solutions that are 0.79% better than the BKS, and new best-known values are found for all instances.

Table 8 reports the results on the 320 large instances ($n = 100$). The ALNS finds 300 new best-known values with an average improvement of 0.70%. The computational time is around 10 minutes which is very reasonable when considering the instance size.

4.6 Effectiveness of MPO operator in the ALNS algorithm

As described in Section 3.7 we developed a mathematical programming based operator (MPO) to re-assign commodities for one customer to the routes of a solution. Due to the increase observed in computational time consumption, we decided to use it only to improve a new global best solution further. In this section, we analyze the results on medium and large instances to prove the effectiveness

Table 8: Summary of results on large sized instances.

instances						ALNS results				
	n	M	p	$avg.n_{cc}$	$nbIns$	$avg.\Delta_B$	$min\Delta_B$	$max\Delta_B$	$avg.t(s)$	$nbNBK$
C101	100	2	0.6	136.4	40	-0.66	-1.69	0.24	330.02	38
C101	100	2	1	200	40	-0.77	-1.64	0.03	569.11	39
C101	100	3	0.6	188.4	40	-0.77	-2.04	0.73	557.46	39
C101	100	3	1	300	40	-1.17	-2.88	0.32	1106.23	37
R101	100	2	0.6	136.4	40	-0.53	-1.50	1.05	323.48	36
R101	100	2	1	200	40	-0.55	-1.28	0.29	557.74	36
R101	100	3	0.6	188.4	40	-0.53	-1.23	0.02	548.02	39
R101	100	3	1	300	40	-0.62	-1.72	0.25	1078.70	36
		total			320	**-0.70**	-2.88	1.05	**633.85**	**300**

of MPO in our algorithm. In Table 9, we indicate the average number of times that MPO is called for each group of instances as well as the average number of times that MPO improves the new best solution for each group of instances. Detailed results are reported in Appendix A (Table 13 and Table 14).

Table 9: Effectiveness of MPO in the ALNS algorithm.

instances				results			
	n	M	p	$avg.n_{cc}$	$nbIns$	$avg.nbMPO$	$avg.nbMPOimp$
C101	20	3	0.6	37.4	5	2.60	0.00
C101	20	3	1	60	5	7.00	0.20
R101	20	3	0.6	37.4	5	4.00	0.20
R101	20	3	1	60	5	9.80	0.20
C101	40	3	0.6	76.2	5	8.00	0.20
C101	40	3	1	120	5	16.40	2.00
R101	40	3	0.6	76.2	5	12.40	0.00
R101	40	3	1	120	5	18.60	1.80
C101	60	3	0.6	110	5	20.80	0.20
C101	60	3	1	180	5	30.80	1.60
R101	60	3	0.6	110	5	18.00	0.80
R101	60	3	1	180	5	24.60	1.60
C101	80	3	0.6	150.4	5	29.00	1.60
C101	80	3	1	240	5	32.00	6.40
R101	80	3	0.6	150.4	5	23.80	1.00
R101	80	3	1	240	5	29.60	2.40
C101	100	2	0.6	136.4	40	27.05	0.85
C101	100	2	1	200	40	29.65	2.00
C101	100	3	0.6	188.4	40	31.60	1.33
C101	100	3	1	300	40	37.28	3.33
R101	100	2	0.6	136.4	40	27.15	0.53
R101	100	2	1	200	40	30.15	2.10
R101	100	3	0.6	188.4	40	32.40	2.23
R101	100	3	1	300	40	34.48	3.58

For medium instances (namely for $n = 20, 40, 60, 80$), the impact of MPO on the quality of the solution increases as the instance size increases. Especially, when $p = 1$, MPO has a greater impact than when $p = 0.6$. The main reason is that, when $p = 1$, all customers require all the commodities while a smaller number of commodities are required when $p = 0.6$. As a consequence, for the same number of customers in the instance, the number of *customer-commodities* is larger. Therefore, when $p = 1$, the difficulty of solving the problem increases as well as the possibilities to reassign commodities with MPO. For large instances ($n = 100$), the same conclusions on the performance of MPO can be drawn.

4.7 Evaluation of the LS in the ALNS algorithm

In this section, we evaluate the importance of the local search in our ALNS algorithm. We run, on mid-80 instances, the ALNS algorithm without LS and MPO with a limit of 400000 iterations. We then compare the results obtained by the ALNS with LS and MPO on 5000 iterations. This choice is made to have a comparison fair: we remove two optimization components, but we allow a large number of iterations.

We compare the results obtained by the proposed algorithm, indicated by *ALNS+LS+MPO*, with the same algorithm when LS and MPO are deactivated. This last version is indicated as *ALNS-LS-MPO*. The performance comparison of both variants is reported in Table 10 for mid-80 and large instances.

It is clear from Table 10 that after 400000 iterations the results obtained by the ALNS without LS and MPO are of lower quality than those obtained by the original ALNS. We only obtain 15 new best-known values for the 20 mid-80 instances. Moreover, the average improvement (0.39%) does not compete with the improvement obtained with the proposed algorithm (ALNS+LS+MPO) while the computational times are similar. The same observations can be made from the results for large instances. We then conclude on the importance of the LS and MPO in the proposed ALNS algorithm.

4.8 Trend between instance size and computational time

Last, we examine how the CPU time required by the ALNS varies according to the instance size. We consider the results obtained with 5000 iterations (even for the small instances) to perform the analysis. We sort the 464 (small, medium and large) instances according to the number of *customer-commodities* instead of the number of customers. The variation of the computational time $avg.t(s)$ according to the number of *customer-commodities* $avg.n_{cc}$ is depicted on Figure 9.

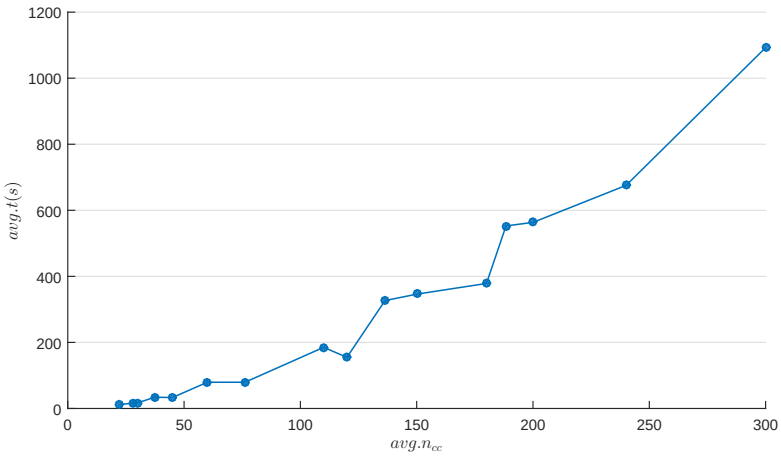


Figure 9: The $avg.t(s)$ of the ALNS with respect to $avg.n_{cc}$

When the size of the instances increases, the average computational time significantly increases (not in a linear fashion). This behavior is not surprising since when the size of the instances increases, it takes more time to operate the LS operators, which computational complexity is $O(n_{cc}^2)$. Nevertheless, the average computational time of our ALNS algorithm for the large instances ($n_{cc} = 300$) remains reasonable with less than 19 minutes.

4.9 Characteristics of split customers

As mentioned above, considering *customer-commodity* makes the problem more complex with equivalent solutions since several commodities are related to the same customer and then have the same

Table 10: Comparison between two ALNS variants on mid-80 and large instances.

instances						ALNS+LS+MPO (5000 iterations)					ALNS-LS-MPO (400000 iterations)				
	n	M	p	$avg.n_{cc}$	$nbIns$	$avg.\Delta_B$	$min\Delta_B$	$max\Delta_B$	$avg.t(s)$	$nbNBK$	$avg.\Delta_B$	$min\Delta_B$	$max\Delta_B$	$avg.t(s)$	$nbNBK$
C101	80	3	0.6	150.4	5	-0.77	-1.48	-0.01	373.36	5	-0.70	-1.50	0.11	330.77	4
C101	80	3	1	240	5	-0.75	-1.36	-0.24	704.20	5	-0.49	-1.08	0.02	686.68	4
R101	80	3	0.6	150.4	5	-0.84	-1.24	-0.05	319.83	5	-0.49	-1.03	0.57	348.14	4
R101	80	3	1	240	5	-0.81	-1.39	-0.50	646.61	5	0.11	-0.83	1.71	712.20	3
		total			20	**-0.79**	-1.48	-0.01	**511.00**	**20**	**-0.39**	-1.50	1.71	**519.45**	**15**
C101	100	2	0.6	136.4	40	-0.66	-1.69	0.24	330.02	38	-0.23	-1.52	1.21	360.30	28
C101	100	2	1	200	40	-0.77	-1.64	0.03	569.11	39	-0.30	-1.58	1.30	606.03	26
C101	100	3	0.6	188.4	40	-0.77	-2.04	0.73	557.46	39	-0.28	-1.66	1.66	576.18	25
C101	100	3	1	300	40	-1.17	-2.88	0.32	1106.23	37	-0.57	-2.47	1.20	1256.61	26
R101	100	2	0.6	136.4	40	-0.53	-1.50	1.05	323.48	36	0.00	-0.75	1.12	366.70	20
R101	100	2	1	200	40	-0.55	-1.28	0.29	557.74	36	-0.08	-1.24	1.83	614.35	25
R101	100	3	0.6	188.4	40	-0.53	-1.23	0.02	548.02	39	0.21	-0.94	1.61	576.27	16
R101	100	3	1	300	40	-0.62	-1.72	0.25	1078.70	36	0.10	-1.78	2.67	1211.77	21
		total			320	**-0.70**	-2.88	1.05	**633.85**	**300**	**-0.14**	-2.47	2.67	**696.03**	**187**

location. Identifying the customers who are good candidates for being split into *customer-commodities* can be beneficial for decision-makers. In this section, we propose to study the characteristics of customers who are delivered by more than one vehicle. We name these customers *split customers*. We provide detailed results on the 20 mid-80 instances with 5000 iterations for the ALNS algorithm.

Table 11 reports these detailed results for several characteristics about the split customers in the best solution obtained on each instance. The first three columns report the characteristics of the instance, and the fourth column indicates the identification number of the instance. Column *nbSplit* reports the number of split customers (out of 80), and columns *nb*2-*split* and *nb*3-*split* indicate the number of customers delivered by respectively 2 and 3 vehicles. Columns *nbNearDepot* and *nbLargeDemand* report the number of split customers located near the depot and the number of split customers with a large demand respectively. As Nagy et al. (2015), we consider that a customer is located near the depot if he or she is one of the 25% of customers closest to the depot; and we consider that a customer has a large demand if he or she is one of the 25% of customers with the largest demand. Note that the demand of a customer is the sum of the demands for the commodities they need. Column *nbCluster* reports the number of split customers located inside a cluster of customers. As Nagy et al. (2015), we consider a customer to be in a cluster if at least five other customers are inside its neighborhood. Two customers are neighbors if the distance between these two customers is less than 30% of the average distance between the depot and all the customers in the instance. Using this definition, in clustered instances (C101) more than 70% of customers are in a cluster, while in random instance (R101) less than 30% of customers are in a cluster.

Table 11: Characteristics of split customers in the best solutions of mid-80 instances (5000 iterations).

instances									
	n	*p*	*id*	*nbSplit*	*nb2-split*	*nb3-split*	*nbNearDepot*	*nbLargeDemand*	*nbCluster*
C101	80	0.6	1	7	7	0	2	3	7
			2	5	5	0	3	1	5
			3	7	7	0	4	2	5
			4	9	8	1	3	5	7
			5	10	10	0	1	7	6
		1	1	20	19	1	5	5	18
			2	14	14	0	4	4	14
			3	18	18	0	6	4	13
			4	14	14	0	4	4	14
			5	18	18	0	6	2	16
R101	80	0.6	1	6	6	0	4	6	4
			2	5	5	0	3	2	2
			3	7	7	0	2	3	4
			4	5	5	0	3	1	3
			5	6	6	0	6	3	3
		1	1	13	13	0	3	3	7
			2	14	13	1	7	1	4
			3	13	13	0	4	2	6
			4	14	14	0	4	4	6
			5	17	17	0	6	5	7

From the results in Table 11, it is clear that there are more split customers when customers require more commodities. p denotes the probability that a customer requires a commodity. When $p = 0.6$, the average number of split customers is 6.7, while when $p = 1$, there are 15.5 split customers on average. Moreover, very few split customers are delivered by three vehicles, i.e., one vehicle for each of the required commodities. In the 20 instances, there are only three split customers in this case. In addition, one important feature of split customers is to be inside a cluster. Indeed, for C101 instances, 86% of split customers are inside a cluster. For R101 instances, 46% of split customers are inside a cluster, while less than 30% of customers are in a cluster in these instances. Proximity to the depot is also an important feature for split customers since 36% of split customers are near the depot. It is less

obvious to link large demands to split customers since 30% of split customers have a large demand.

5 Conclusions

In this paper, we presented a dedicated heuristic algorithm for the C-SDVRP. The proposed algorithm is based on an adaptive large neighborhood search framework introduced by Ropke and Pisinger (2006). This is the first heuristic specifically designed with the aim to provide high-quality solutions for the medium and large size instances. According to the main feature of the C-SDVRP, i.e. the requirement of different commodities, we adapt some classical local search moves to consider either with a customer (i.e., a customer and all its commodities) or a *customer-commodity* (namely, a single commodity required by a customer). We developed a mathematical programming based operator to intensify the search and further improve the best solutions. The results show that our ALNS algorithm is very effective in finding high-quality solutions on large size instances. In particular, our method outperforms the algorithms proposed in Archetti et al. (2014) and in Archetti et al. (2015).

The proposed ALNS algorithm could then be adapted to tackle other variants of routing problems with split deliveries. One of these variants is the case with multiple depots and available quantities at each depot. In this case, because of the limited quantities available for each commodity at each depot, it is worth considering splitting deliveries to find feasible solutions. Another interesting variant of the problem is the VRP with divisible deliveries and pickup (Gribkovskaia et al. (2007), Nagy et al. (2015)): in this case, delivery and pickup naturally represent two different commodities, but a pickup operation increases the use of the vehicle capacity, and it could then be optimal to visit twice the same customer in one route.

Acknowledgments

This work is partially supported by the CSC (China Scholarship Council) and by the ELSAT 2020 project. The authors thank C. Archetti and N. Bianchessi for providing the benchmark instances. Thanks are also due to the referees for their valuable comments.

A Detailed results on the benchmark instances

Table 12, Table 13 and Table 14 report the detailed results for the small, medium and large instances respectively. We report values in bold whenever we improve (or optimize or equal to) the respective instances.

Table 12: Detailed computational results for the small sized instances ($n = 15$).

instances										ALNS results			
	M	p	n_{cc}	CC_1	CC_2	CC_3	Δ	α	OPT	$Cost$	Δ_O	$t(s)$	nbR
C101	2	0.6	22	10	12		[1,100]	1.1	283.3404	283.3404	**0.00**	6.01	6
		0.6	22	10	12		[40,60]	1.1	480.4342	480.4342	**0.00**	5.07	11
		1	30	15	15		[1,100]	1.1	422.4965	422.4965	**0.00**	10.64	9
		1	30	15	15		[40,60]	1.1	685.1662	685.1662	**0.00**	5.19	15
		0.6	22	10	12		[1,100]	1.5	241.0386	241.0386	**0.00**	8.67	5
		0.6	22	10	12		[40,60]	1.5	348.2731	348.2731	**0.00**	5.63	7
		1	30	15	15		[1,100]	1.5	340.5474	340.5474	**0.00**	14.70	7
		1	30	15	15		[40,60]	1.5	490.2973	490.2973	**0.00**	8.27	10
		0.6	22	10	12		[1,100]	2	200.9092	200.9092	**0.00**	8.66	4
		0.6	22	10	12		[40,60]	2	239.6829	239.6829	**0.00**	5.80	5
		1	30	15	15		[1,100]	2	239.9424	239.9424	**0.00**	13.57	5
		1	30	15	15		[40,60]	2	357.5929	357.5929	**0.00**	9.14	7
		0.6	22	10	12		[1,100]	2.5	170.0453	170.0453	**0.00**	10.31	3
		0.6	22	10	12		[40,60]	2.5	207.8259	207.8259	**0.00**	7.74	4
		1	30	15	15		[1,100]	2.5	205.7830	205.7830	**0.00**	12.29	4
		1	30	15	15		[40,60]	2.5	302.1813	302.1813	**0.00**	11.26	6
R101	2	0.6	22	10	12		[1,100]	1.1	408.8971	408.8971	**0.00**	7.43	7
		0.6	22	10	12		[40,60]	1.1	565.3383	565.3383	**0.00**	4.12	11
		1	30	15	15		[1,100]	1.1	537.2029	537.2029	**0.00**	9.75	9
		1	30	15	15		[40,60]	1.1	679.0232	679.0232	**0.00**	5.26	15
		0.6	22	10	12		[1,100]	1.5	353.0119	353.0119	**0.00**	6.77	5
		0.6	22	10	12		[40,60]	1.5	455.9724	455.9724	**0.00**	6.88	8
		1	30	15	15		[1,100]	1.5	443.0829	443.0829	**0.00**	12.31	7
		1	30	15	15		[40,60]	1.5	558.9565	558.9565	**0.00**	8.03	10
		0.6	22	10	12		[1,100]	2	301.4316	301.4316	**0.00**	8.50	4
		0.6	22	10	12		[40,60]	2	379.2848	379.2848	**0.00**	7.69	6
		1	30	15	15		[1,100]	2	370.5561	370.5561	**0.00**	14.73	5
		1	30	15	15		[40,60]	2	444.1868	444.1868	**0.00**	9.27	8
		0.6	22	10	12		[1,100]	2.5	280.7646	280.7646	**0.00**	7.50	3
		0.6	22	10	12		[40,60]	2.5	342.6854	342.6854	**0.00**	7.27	5
		1	30	15	15		[1,100]	2.5	323.5761	323.5761	**0.00**	13.68	4
		1	30	15	15		[40,60]	2.5	401.6087	401.6087	**0.00**	13.03	6
C101	3	0.6	28	12	7	9	[1,100]	1.1	333.4709	333.4709	**0.00**	11.29	7
		0.6	28	12	7	9	[40,60]	1.1	440.2241	440.2241	**0.00**	11.11	9
		1	45	15	15	15	[1,100]	1.1	428.5164	435.7543	1.69	25.05	8
		1	45	15	15	15	[40,60]	1.1	638.0919	638.0919	**0.00**	20.69	13
		0.6	28	12	7	9	[1,100]	1.5	262.8069	262.8069	**0.00**	10.47	5
		0.6	28	12	7	9	[40,60]	1.5	306.6363	306.6363	**0.00**	9.86	6
		1	45	15	15	15	[1,100]	1.5	315.9600	315.96	**0.00**	25.38	6
		1	45	15	15	15	[40,60]	1.5	457.9430	457.943	**0.00**	16.15	9
		0.6	28	12	7	9	[1,100]	2	204.9380	204.938	**0.00**	10.83	4
		0.6	28	12	7	9	[40,60]	2	263.2896	263.2896	**0.00**	9.34	5
		1	45	15	15	15	[1,100]	2	265.0623	265.0623	**0.00**	23.37	5
		1	45	15	15	15	[40,60]	2	347.3580	347.358	**0.00**	22.65	7
		0.6	28	12	7	9	[1,100]	2.5	168.2958	168.2958	**0.00**	10.09	3
		0.6	28	12	7	9	[40,60]	2.5	202.9044	202.9044	**0.00**	10.25	4
		1	45	15	15	15	[1,100]	2.5	206.6970	206.697	**0.00**	20.36	4
		1	45	15	15	15	[40,60]	2.5	310.7978	310.7978	**0.00**	24.45	6
R101	3	0.6	28	12	7	9	[1,100]	1.1	401.7502	401.7502	**0.00**	10.06	7
		0.6	28	12	7	9	[40,60]	1.1	497.1385	497.1385	**0.00**	8.78	10
		1	45	15	15	15	[1,100]	1.1	491.0411	491.0411	**0.00**	22.81	9
		1	45	15	15	15	[40,60]	1.1	679.0232	679.0232	**0.00**	17.69	15
		0.6	28	12	7	9	[1,100]	1.5	347.3693	347.3693	**0.00**	9.94	5
		0.6	28	12	7	9	[40,60]	1.5	410.813	410.813	**0.00**	8.19	7
		1	45	15	15	15	[1,100]	1.5	409.2905	409.2905	**0.00**	22.31	6
		1	45	15	15	15	[40,60]	1.5	541.0336	541.0336	**0.00**	15.16	9
		0.6	28	12	7	9	[1,100]	2	303.1439	303.1439	**0.00**	10.47	4
		0.6	28	12	7	9	[40,60]	2	343.7159	343.7159	**0.00**	11.04	5
		1	45	15	15	15	[1,100]	2	345.8351	345.8351	**0.00**	22.07	5
		1	45	15	15	15	[40,60]	2	444.1868	444.1868	**0.00**	17.91	8
		0.6	28	12	7	9	[1,100]	2.5	278.2234	278.2234	**0.00**	8.02	3
		0.6	28	12	7	9	[40,60]	2.5	312.3100	312.31	**0.00**	10.69	4
		1	45	15	15	15	[1,100]	2.5	320.3490	320.349	**0.00**	23.38	4
		1	45	15	15	15	[40,60]	2.5	393.8357	393.8357	**0.00**	20.34	6

Table 13: Detailed computational results for the mid sized instances ($n \in \{20; 40; 60; 80\}$).

instances									ALNS results					
	n	p	n_{cc}	CC_1	CC_2	CC_3	id	OPT/BKS	$Cost$	$\Delta_{O/B}$	$t(s)$	$nbMPO$	$nbMPOimp$	nbR
C101	20	0.6	37	15	11	11	1	573.8617*	573.8617	**0.00**	31.67	2	0	6
			39	13	10	16	2	592.0651*	592.0651	**0.00**	29.17	0	0	7
			38	13	13	12	3	595.5289*	595.5289	**0.00**	36.29	4	0	7
			40	9	14	17	4	617.8838*	617.8838	**0.00**	40.64	3	0	8
			33	9	10	14	5	628.2804*	628.2804	**0.00**	29.14	4	0	8
		1	60	20	20	20	1	750.6251*	750.6251	**0.00**	82.73	7	0	9
			60	20	20	20	2	714.6453*	714.6453	**0.00**	78.86	9	1	9
			60	20	20	20	3	626.1553*	626.1553	**0.00**	67.16	6	0	9
			60	20	20	20	4	747.7008*	747.7008	**0.00**	84.32	9	0	10
			60	20	20	20	5	768.5189*	768.5189	**0.00**	72.62	4	0	10
R101	20	0.6	37	15	11	11	1	457.8598*	457.8598	**0.00**	38.69	7	0	6
			39	13	10	16	2	667.0131*	667.0131	**0.00**	38.14	2	0	7
			38	13	13	12	3	455.0534*	455.0534	**0.00**	34.87	3	1	7
			40	9	14	17	4	589.9082*	589.9082	**0.00**	37.92	6	0	8
			33	9	10	14	5	663.2159*	663.2159	**0.00**	24.73	2	0	8
		1	60	20	20	20	1	599.8380*	599.8380	**0.00**	70.50	7	0	9
			60	20	20	20	2	863.8829*	864.1552	0.03	94.33	15	0	9
			60	20	20	20	3	617.9120	617.9662	0.01	78.97	10	0	9
			60	20	20	20	4	712.0175*	712.0175	**0.00**	83.18	7	1	10
			60	20	20	20	5	794.4068*	794.4068	**0.00**	81.62	10	0	10
C101	40	0.6	72	19	24	29	1	844.5669	841.0159	**-0.42**	70.98	4	0	9
			77	29	23	25	2	1005.8069	1002.4435	**-0.33**	80.82	17	0	12
			78	24	26	28	3	879.2568*	879.2568	**0.00**	83.96	8	0	11
			81	28	28	25	4	921.0554	922.4288	0.15	74.44	5	0	12
			73	25	19	29	5	868.7426*	868.7426	**0.00**	80.50	6	1	11
		1	120	40	40	40	1	1330.0617	1304.7180	**-1.91**	155.89	16	2	18
			120	40	40	40	2	1357.7864	1357.7890	0.00	163.58	18	3	19
			120	40	40	40	3	1309.3540	1299.4327	**-0.76**	149.76	12	0	16
			120	40	40	40	4	1238.8599	1238.3923	**-0.04**	158.73	17	1	17
			120	40	40	40	5	1287.4121	1273.2181	**-1.10**	178.96	19	4	16
R101	40	0.6	72	19	24	29	1	761.7828	764.0135	0.29	94.07	17	0	9
			77	29	23	25	2	896.0147	896.8111	0.09	57.28	8	0	12
			78	24	26	28	3	851.0276*	851.0276	**0.00**	98.89	20	0	11
			81	28	28	25	4	973.9919	976.5911	0.27	78.62	6	0	12
			73	25	19	29	5	854.3462*	855.1124	0.09	74.66	11	0	11
		1	120	40	40	40	1	1246.4958	1242.8772	**-0.29**	157.76	21	2	18
			120	40	40	40	2	1244.9154	1238.4378	**-0.52**	148.03	19	2	19
			120	40	40	40	3	1056.1320*	1056.1320	**0.00**	131.46	15	2	16
			120	40	40	40	4	1255.4495	1245.6884	**-0.78**	160.65	24	1	17
			120	40	40	40	5	1101.1214	1101.1214	**0.00**	143.75	14	2	16

Table 13 (continued).

instances									ALNS results					
	n	p	n_{cc}	CC_1	CC_2	CC_3	id	OPT/BKS	$Cost$	$\Delta_{O/B}$	$t(s)$	$nbMPO$	$nbMPOimp$	nbR
C101	60	0.6	107	34	36	37	1	1241.1029	1228.7010	**-1.00**	182.54	20	0	15
			112	40	33	39	2	1331.7718	1331.7718	**0.00**	204.25	33	1	17
			110	39	34	37	3	1184.7776	1180.6147	**-0.35**	189.09	14	0	14
			113	41	34	38	4	1303.1604	1284.9398	**-1.40**	195.14	19	0	16
			108	32	30	46	5	1303.7277	1303.3246	**-0.03**	194.56	18	0	16
		1	180	60	60	60	1	2003.0431	1996.6060	**-0.32**	332.53	26	0	27
			180	60	60	60	2	1667.6461	1671.8163	0.25	402.08	19	0	24
			180	60	60	60	3	1816.1236	1795.5325	**-1.13**	445.02	43	3	23
			180	60	60	60	4	1906.5617	1909.0054	0.13	361.95	22	2	26
			180	60	60	60	5	1650.5799	1637.1273	**-0.82**	477.33	44	3	22
R101	60	0.6	107	34	36	37	1	1293.2293	1286.0364	**-0.56**	181.69	19	1	15
			112	40	33	39	2	1326.5733	1317.7203	**-0.67**	136.28	13	1	17
			110	39	34	37	3	1028.5158	1028.5158	**0.00**	180.90	18	1	14
			113	41	34	38	4	1235.5766	1225.3226	**-0.83**	218.23	29	1	17
			108	32	30	46	5	1149.5550	1149.5673	0.00	168.89	11	0	16
		1	180	60	60	60	1	2086.6870	2068.5355	**-0.87**	346.07	24	3	27
			180	60	60	60	2	1662.5753	1660.3220	**-0.14**	328.54	33	0	23
			180	60	60	60	3	1581.5715	1562.5867	**-1.20**	400.14	31	1	23
			180	60	60	60	4	1732.8742	1723.4548	**-0.54**	328.65	17	1	26
			180	60	60	60	5	1507.7615	1503.2634	**-0.30**	362.81	18	3	22
C101	80	0.6	142	44	51	47	1	1648.0955	1647.8619	**-0.01**	329.97	22	2	20
			157	50	55	52	2	1616.9601	1603.4580	**-0.84**	355.95	24	1	21
			148	50	47	51	3	1750.6889	1742.9882	**-0.44**	366.80	27	2	22
			158	47	51	60	4	1468.4500	1446.7739	**-1.48**	413.62	33	2	19
			147	45	42	60	5	1702.9460	1684.7620	**-1.07**	400.47	39	1	21
		1	240	80	80	80	1	2277.9557	2255.8933	**-0.97**	668.31	23	6	30
			240	80	80	80	2	2133.9879	2118.7056	**-0.72**	684.15	31	4	29
			240	80	80	80	3	2623.9684	2588.2988	**-1.36**	720.92	35	5	35
			240	80	80	80	4	2389.1831	2377.7662	**-0.48**	758.18	31	10	32
			240	80	80	80	5	2410.9786	2405.0859	**-0.24**	689.43	40	7	33
R101	80	0.6	142	44	51	47	1	1467.6251	1449.3824	**-1.24**	319.10	26	1	20
			157	50	55	52	2	1482.3639	1481.5891	**-0.05**	349.74	28	0	21
			148	50	47	51	3	1618.6780	1606.2737	**-0.77**	278.83	21	1	22
			158	47	51	60	4	1432.0982	1416.8707	**-1.06**	331.53	22	1	19
			147	45	42	60	5	1482.7288	1466.4647	**-1.10**	319.93	22	2	20
		1	240	80	80	80	1	2137.3158	2107.6678	**-1.39**	697.74	29	6	31
			240	80	80	80	2	1955.9811	1938.9429	**-0.87**	710.27	40	2	29
			240	80	80	80	3	2302.7431	2291.3231	**-0.50**	618.40	30	0	35
			240	80	80	80	4	2123.9847	2113.1821	**-0.51**	685.71	30	3	32
			240	80	80	80	5	2155.4925	2138.8196	**-0.77**	520.92	19	1	33

Table 14: Detailed computational results for the large sized instances ($n = 100$).

instances									ALNS results					
	M	p	n_{cc}	CC_1	CC_2	CC_3	Δ	α	BKS	$Cost$	Δ_B	$t(s)$	$nbMPO$	$nbMPOimp$	nbR
C101	2	0.6	134	56	78		[1,100]	1.1	2035.3013	2016.5051	**-0.92**	260.68	18	0	30
			140	60	80				2180.0203	2170.9690	**-0.42**	270.33	22	1	35
			135	62	73				2082.1981	2076.9958	**-0.25**	236.59	21	1	32
			140	68	72				2148.4188	2133.8412	**-0.68**	268.38	29	0	32
			133	55	78				2041.9553	2016.1057	**-1.27**	270.64	31	2	31
C101	2	0.6	134	56	78		[1,100]	1.5	1573.6610	1571.8390	**-0.12**	325.56	22	9	23
			140	60	80				1713.4453	1690.1594	**-1.36**	322.81	34	2	25
			135	62	73				1594.5043	1579.4423	**-0.94**	326.38	42	3	23
			140	68	72				1620.3658	1593.0295	**-1.69**	295.26	26	1	23
			133	55	78				1544.2592	1542.7069	**-0.10**	327.96	32	0	23
C101	2	0.6	134	56	78		[1,100]	2	1484.4280	1484.8897	0.03	443.90	35	0	17
			140	60	80				1601.5050	1594.3828	**-0.44**	390.17	25	3	19
			135	62	73				1571.8295	1551.0808	**-1.32**	358.75	25	1	17
			140	68	72				1539.4321	1532.8325	**-0.43**	409.17	32	1	18
			133	55	78				1545.9487	1537.2621	**-0.56**	345.92	25	1	17
C101	2	0.6	134	56	78		[1,100]	2.5	1293.4366	1290.4392	**-0.23**	509.62	34	3	14
			140	60	80				1386.2707	1378.6220	**-0.55**	407.99	25	1	15
			135	62	73				1323.7604	1318.8051	**-0.37**	429.45	20	0	14
			140	68	72				1337.2558	1319.5941	**-1.32**	475.72	36	0	14
			133	55	78				1316.6055	1307.1230	**-0.72**	408.30	26	3	14
C101	2	0.6	134	56	78		[40,60]	1.1	3717.5992	3686.3063	**-0.84**	212.18	18	1	60
			140	60	80				3943.6197	3923.7703	**-0.50**	211.63	13	0	65
			135	62	73				3755.8823	3720.0138	**-0.95**	247.43	17	0	60
			140	68	72				3840.6868	3815.5816	**-0.65**	271.71	37	0	63
			133	55	78				3673.3203	3645.8466	**-0.75**	230.77	34	0	60
C101	2	0.6	134	56	78		[40,60]	1.5	2685.9916	2670.5453	**-0.58**	259.21	22	0	42
			140	60	80				2824.0916	2780.6093	**-1.54**	269.90	31	0	44
			135	62	73				2633.8403	2606.9999	**-1.02**	278.21	35	0	41
			140	68	72				2738.3272	2703.7315	**-1.26**	306.65	35	0	43
			133	55	78				2628.9645	2606.6952	**-0.85**	249.35	23	0	42
C101	2	0.6	134	56	78		[40,60]	2	2371.4171	2364.7218	**-0.28**	331.61	32	0	30
			140	60	80				2509.9521	2489.1959	**-0.83**	358.79	26	0	32
			135	62	73				2408.7012	2401.3033	**-0.31**	343.90	24	0	30
			140	68	72				2443.4540	2437.5545	**-0.24**	353.23	30	0	31
			133	55	78				2464.7661	2440.5193	**-0.98**	339.58	30	0	31
C101	2	0.6	134	56	78		[40,60]	2.5	1968.1149	1963.9711	**-0.21**	368.04	12	0	24
			140	60	80				2031.8053	2027.4671	**-0.21**	367.94	22	0	25
			135	62	73				1954.6604	1950.7756	**-0.20**	406.74	31	0	23
			140	68	72				2002.5383	1987.3101	**-0.76**	343.06	30	1	24
			133	55	78				1973.9611	1978.6316	0.24	367.40	20	0	24

Table 14 (continued).

instances									ALNS results					
	M	p	n_{cc}	CC_1	CC_2	CC_3	Δ	α	BKS	$Cost$	Δ_B	$t(s)$	$nbMPO$	$nbMPOimp$	nbR
C101	2	1	200	100	100		[1,100]	1.1	3146.6597	3096.0220	**-1.61**	493.02	34	3	48
			200	100	100				3081.7941	3047.0941	**-1.13**	459.72	24	2	50
			200	100	100				3376.5593	3341.0044	**-1.05**	530.95	39	6	53
			200	100	100				3336.4515	3293.8945	**-1.28**	508.36	35	8	52
			200	100	100				3063.1397	3025.7796	**-1.22**	459.09	28	1	49
C101	2	1	200	100	100		[1,100]	1.5	2356.8893	2334.4763	**-0.95**	526.05	30	2	35
			200	100	100				2326.5291	2301.5070	**-1.08**	566.56	41	2	36
			200	100	100				2526.2982	2518.5285	**-0.31**	554.75	34	6	39
			200	100	100				2500.0207	2479.4547	**-0.82**	534.61	34	0	38
			200	100	100				2303.9564	2297.1414	**-0.30**	527.30	25	3	36
C101	2	1	200	100	100		[1,100]	2	2129.0342	2124.8932	**-0.19**	603.94	25	2	27
			200	100	100				2169.8922	2170.6119	0.03	629.32	35	4	27
			200	100	100				2256.4305	2245.4843	**-0.49**	617.32	31	2	29
			200	100	100				2269.9257	2262.6622	**-0.32**	554.71	19	1	29
			200	100	100				2160.1093	2156.8393	**-0.15**	523.47	19	0	27
C101	2	1	200	100	100		[1,100]	2.5	1760.6323	1753.7203	**-0.39**	693.48	36	3	21
			200	100	100				1836.8031	1811.6854	**-1.37**	756.15	33	6	22
			200	100	100				1882.7674	1871.8435	**-0.58**	637.95	29	4	23
			200	100	100				1910.0314	1895.2543	**-0.77**	716.49	32	3	23
			200	100	100				1810.3104	1808.1843	**-0.12**	738.02	48	5	21
C101	2	1	200	100	100		[40,60]	1.1	5603.0836	5536.0546	**-1.20**	433.77	19	0	93
			200	100	100				5547.3519	5504.4821	**-0.77**	454.40	26	1	95
			200	100	100				5749.2419	5679.6959	**-1.21**	424.29	17	0	96
			200	100	100				5711.3460	5645.4455	**-1.15**	463.91	18	0	95
			200	100	100				5535.6135	5479.9888	**-1.00**	469.36	23	0	92
C101	2	1	200	100	100		[40,60]	1.5	3913.4546	3849.0963	**-1.64**	507.87	29	1	62
			200	100	100				3909.5761	3867.0017	**-1.09**	482.07	28	2	63
			200	100	100				3976.2127	3959.2347	**-0.43**	502.91	27	1	64
			200	100	100				3942.3417	3898.8312	**-1.10**	583.69	47	0	63
			200	100	100				3867.9698	3839.8436	**-0.73**	598.34	44	1	62
C101	2	1	200	100	100		[40,60]	2	3343.1458	3328.8877	**-0.43**	613.39	30	0	45
			200	100	100				3446.6946	3391.5541	**-1.60**	609.40	27	2	45
			200	100	100				3402.9488	3402.1518	**-0.02**	593.52	25	1	47
			200	100	100				3459.2005	3446.1003	**-0.38**	604.41	29	1	47
			200	100	100				3400.1668	3384.4928	**-0.46**	628.07	30	1	45
C101	2	1	200	100	100		[40,60]	2.5	2734.4477	2717.2918	**-0.63**	680.71	33	2	35
			200	100	100				2788.0824	2776.1339	**-0.43**	596.97	22	0	36
			200	100	100				2780.4184	2761.2009	**-0.69**	616.17	23	3	36
			200	100	100				2815.0422	2786.8485	**-1.00**	589.10	25	1	36
			200	100	100				2767.8571	2751.4713	**-0.59**	680.74	33	0	35

Table 14 (continued).

instances									ALNS results					
	M	p	n_{cc}	CC_1	CC_2	CC_3	Δ	α	BKS	$Cost$	Δ_B	$t(s)$	$nbMPO$	$nbMPOimp$	nbR
C101	3	0.6	179	65	50	64	[1,100]	1.1	2233.2625	2199.3967	**-1.52**	447.71	38	2	34
			194	67	63	64			2406.5116	2357.4534	**-2.04**	509.65	32	3	36
			186	62	58	66			2499.3959	2487.8220	**-0.46**	505.21	45	1	37
			193	69	57	67			2266.5132	2283.0043	0.73	448.84	27	0	36
			190	61	56	73			2529.7072	2509.7369	**-0.79**	471.85	39	4	37
C101	3	0.6	179	65	50	64	[1,100]	1.5	1724.7339	1698.9351	**-1.50**	449.14	32	7	25
			194	67	63	64			1812.1988	1807.9754	**-0.23**	579.95	28	3	27
			186	62	58	66			1908.2163	1893.4346	**-0.77**	521.23	38	2	27
			193	69	57	67			1763.0602	1745.0116	**-1.02**	568.39	47	1	26
			190	61	56	73			1938.0044	1930.7389	**-0.37**	576.69	26	4	28
C101	3	0.6	179	65	50	64	[1,100]	2	1651.6968	1626.3891	**-1.53**	633.94	38	1	19
			194	67	63	64			1654.5853	1652.1369	**-0.15**	677.73	30	2	20
			186	62	58	66			1712.4417	1684.7036	**-1.62**	648.54	40	2	20
			193	69	57	67			1702.0681	1677.4419	**-1.45**	584.07	22	0	20
			190	61	56	73			1719.8367	1714.1925	**-0.33**	600.65	25	1	21
C101	3	0.6	179	65	50	64	[1,100]	2.5	1413.2271	1411.6091	**-0.11**	681.11	34	0	15
			194	67	63	64			1442.7602	1424.9209	**-1.24**	810.87	45	0	16
			186	62	58	66			1452.9146	1428.4270	**-1.69**	712.86	42	2	16
			193	69	57	67			1458.4966	1442.4131	**-1.10**	780.48	36	0	16
			190	61	56	73			1446.4444	1444.4045	**-0.14**	756.13	31	1	17
C101	3	0.6	179	65	50	64	[40,60]	1.1	3318.3873	3257.0030	**-1.85**	401.09	32	0	53
			194	67	63	64			3509.4788	3489.2080	**-0.58**	435.08	26	1	56
			186	62	58	66			3603.3698	3582.1768	**-0.59**	399.24	25	0	55
			193	69	57	67			3518.1872	3484.3878	**-0.96**	446.43	32	0	56
			190	61	56	73			3649.5231	3615.4109	**-0.93**	496.11	48	0	57
C101	3	0.6	179	65	50	64	[40,60]	1.5	2424.2712	2407.4264	**-0.69**	419.67	21	0	37
			194	67	63	64			2525.9423	2523.9967	**-0.08**	558.51	28	0	40
			186	62	58	66			2610.6052	2600.9273	**-0.37**	452.04	26	2	39
			193	69	57	67			2535.7810	2525.7262	**-0.40**	468.16	32	0	39
			190	61	56	73			2673.1778	2640.8231	**-1.21**	479.02	25	0	40
C101	3	0.6	179	65	50	64	[40,60]	2	2270.3526	2265.3978	**-0.22**	482.40	25	1	28
			194	67	63	64			2346.2918	2326.5017	**-0.84**	643.35	26	1	30
			186	62	58	66			2316.4770	2313.2239	**-0.14**	540.93	21	0	29
			193	69	57	67			2367.4327	2352.4767	**-0.63**	574.32	27	4	30
			190	61	56	73			2339.7488	2321.6905	**-0.77**	560.01	38	3	30
C101	3	0.6	179	65	50	64	[40,60]	2.5	1875.4922	1857.8147	**-0.94**	501.48	23	1	22
			194	67	63	64			1971.8409	1948.0161	**-1.21**	643.45	28	2	24
			186	62	58	66			1927.4449	1917.8823	**-0.50**	606.26	40	1	23
			193	69	57	67			1974.0424	1972.7209	**-0.07**	618.31	30	0	24
			190	61	56	73			1929.9052	1918.9720	**-0.57**	607.40	16	1	24

Table 14 (continued).

instances									BKS	ALNS results					
	M	p	n_{cc}	CC_1	CC_2	CC_3	Δ	α		$Cost$	Δ_B	$t(s)$	$nbMPO$	$nbMPOimp$	nbR
C101	3	1	300	100	100	100	[1,100]	1.1	3458.3101	3378.7279	**-2.30**	952.16	50	5	53
			300	100	100	100			3314.9537	3245.1446	**-2.11**	914.82	43	6	51
			300	100	100	100			3294.7994	3265.0504	**-0.90**	865.51	33	8	52
			300	100	100	100			3430.0465	3370.6534	**-1.73**	825.87	34	0	55
			300	100	100	100			3214.8478	3162.0426	**-1.64**	900.59	39	7	51
C101	3	1	300	100	100	100	[1,100]	1.5	2610.2355	2561.2706	**-1.88**	972.71	30	6	39
			300	100	100	100			2514.8470	2459.3592	**-2.21**	1024.42	35	3	38
			300	100	100	100			2532.1347	2476.9681	**-2.18**	1083.84	38	6	38
			300	100	100	100			2641.7530	2565.6834	**-2.88**	923.07	37	2	40
			300	100	100	100			2455.7585	2404.8723	**-2.07**	968.00	28	5	37
C101	3	1	300	100	100	100	[1,100]	2	2300.4077	2280.2733	**-0.88**	1211.44	42	5	29
			300	100	100	100			2192.9203	2193.6158	0.03	1153.59	43	3	28
			300	100	100	100			2260.6841	2253.2807	**-0.33**	1142.33	28	3	28
			300	100	100	100			2410.0967	2390.4263	**-0.82**	1099.96	28	5	30
			300	100	100	100			2247.5337	2231.8800	**-0.70**	1240.23	26	4	28
C101	3	1	300	100	100	100	[1,100]	2.5	1929.7279	1914.9262	**-0.77**	1348.23	17	4	23
			300	100	100	100			1871.0897	1856.8834	**-0.76**	1606.09	20	0	23
			300	100	100	100			1867.4470	1873.4338	0.32	1519.55	26	3	23
			300	100	100	100			2013.6222	1985.1207	**-1.42**	1396.51	41	0	24
			300	100	100	100			1910.9060	1875.5780	**-1.85**	1625.37	56	8	22
C101	3	1	300	100	100	100	[40,60]	1.1	5239.6555	5196.4247	**-0.83**	906.42	34	1	86
			300	100	100	100			5143.4560	5059.3108	**-1.64**	1016.59	51	4	84
			300	100	100	100			5200.9560	5103.2213	**-1.88**	1054.92	36	3	83
			300	100	100	100			5291.7632	5243.8349	**-0.91**	987.19	50	1	89
			300	100	100	100			5174.4761	5114.0431	**-1.17**	978.28	41	0	85
C101	3	1	300	100	100	100	[40,60]	1.5	3789.0183	3755.7431	**-0.88**	829.41	32	1	60
			300	100	100	100			3789.1122	3733.9771	**-1.46**	944.05	48	3	60
			300	100	100	100			3752.1660	3732.6484	**-0.52**	971.90	53	2	60
			300	100	100	100			3839.3184	3770.5935	**-1.79**	897.98	35	6	61
			300	100	100	100			3825.3398	3740.0157	**-2.23**	1020.43	52	4	60
C101	3	1	300	100	100	100	[40,60]	2	3343.5307	3331.3655	**-0.36**	1145.10	38	1	45
			300	100	100	100			3322.9033	3307.6401	**-0.46**	1081.25	25	1	44
			300	100	100	100			3318.3125	3297.9859	**-0.61**	1044.56	27	2	44
			300	100	100	100			3424.6490	3382.5992	**-1.23**	1254.62	39	0	45
			300	100	100	100			3379.4211	3330.9490	**-1.43**	1240.97	50	4	45
C101	3	1	300	100	100	100	[40,60]	2.5	2807.4471	2785.0606	**-0.80**	1190.80	39	3	36
			300	100	100	100			2746.6361	2735.8088	**-0.39**	1259.03	43	4	35
			300	100	100	100			2754.3082	2760.9908	0.24	1255.07	24	7	35
			300	100	100	100			2803.2445	2789.5742	**-0.49**	1251.01	56	3	36
			300	100	100	100			2767.2392	2746.4348	**-0.75**	1145.49	24	0	36

Table 14 (continued).

instances									ALNS results					
	M	p	n_{cc}	CC_1	CC_2	CC_3	Δ	α	BKS	$Cost$	Δ_B	$t(s)$	$nbMPO$	$nbMPOimp$	nbR
R101	2	0.6	134	56	78		[1,100]	1.1	1916.2385	1887.4950	**-1.50**	279.09	27	1	31
			140	60	80				2153.6485	2152.5710	**-0.05**	300.47	31	1	35
			135	62	73				1927.2419	1920.3454	**-0.36**	265.37	21	0	33
			140	68	72				1971.0569	1956.0302	**-0.76**	302.81	31	3	32
			133	55	78				1833.7028	1828.4456	**-0.29**	306.71	25	0	32
R101	2	0.6	134	56	78		[1,100]	1.5	1496.8719	1482.2547	**-0.98**	319.35	29	0	22
			140	60	80				1685.8891	1670.7013	**-0.90**	307.37	39	1	25
			135	62	73				1530.9401	1517.8384	**-0.86**	296.24	34	0	23
			140	68	72				1549.2746	1541.2523	**-0.52**	307.34	17	0	24
			133	55	78				1476.0508	1464.6001	**-0.78**	312.00	23	0	23
R101	2	0.6	134	56	78		[1,100]	2	1226.8612	1213.6448	**-1.08**	355.57	37	0	17
			140	60	80				1363.4234	1351.5177	**-0.87**	378.88	35	0	19
			135	62	73				1254.4877	1249.8766	**-0.37**	370.13	31	0	18
			140	68	72				1268.1761	1260.8517	**-0.58**	418.11	33	2	17
			133	55	78				1203.2001	1200.0221	**-0.26**	325.71	31	2	17
R101	2	0.6	134	56	78		[1,100]	2.5	1067.2709	1059.0853	**-0.77**	385.07	28	2	13
			140	60	80				1178.0644	1169.7717	**-0.70**	453.09	43	0	15
			135	62	73				1087.0068	1084.9342	**-0.19**	414.91	33	0	14
			140	68	72				1102.6288	1104.8424	0.20	409.88	28	4	14
			133	55	78				1050.7626	1061.7723	1.05	430.69	30	0	14
R101	2	0.6	134	56	78		[40,60]	1.1	3358.5111	3333.0578	**-0.76**	263.31	25	0	60
			140	60	80				3575.0576	3553.3423	**-0.61**	250.22	19	0	65
			135	62	73				3344.4207	3327.4991	**-0.51**	223.60	20	0	61
			140	68	72				3406.2463	3386.6904	**-0.57**	275.68	25	0	63
			133	55	78				3269.4765	3244.8496	**-0.75**	236.62	30	0	60
R101	2	0.6	134	56	78		[40,60]	1.5	2438.2268	2420.4671	**-0.73**	261.82	22	0	41
			140	60	80				2610.9667	2598.2538	**-0.49**	293.50	24	0	44
			135	62	73				2427.5931	2414.9036	**-0.52**	274.40	29	0	41
			140	68	72				2503.2952	2481.8594	**-0.86**	274.12	17	1	43
			133	55	78				2440.5711	2414.1367	**-1.08**	313.44	36	0	42
R101	2	0.6	134	56	78		[40,60]	2	1895.5429	1895.7308	0.01	326.88	26	0	30
			140	60	80				1995.3201	1992.3865	**-0.15**	330.00	22	0	32
			135	62	73				1896.6321	1881.4183	**-0.80**	326.29	22	0	30
			140	68	72				1915.4443	1906.1707	**-0.48**	353.72	34	0	31
			133	55	78				1852.4127	1861.0882	0.47	324.98	20	0	31
R101	2	0.6	134	56	78		[40,60]	2.5	1587.2595	1577.0403	**-0.64**	344.39	31	0	24
			140	60	80				1668.3025	1659.9154	**-0.50**	330.66	21	1	25
			135	62	73				1578.8911	1570.4920	**-0.53**	343.04	21	0	24
			140	68	72				1608.9971	1591.3215	**-1.10**	348.60	24	2	24
			133	55	78				1561.0713	1558.7095	**-0.15**	305.21	12	1	24

Table 14 (continued).

instances									ALNS results					
	M	p	n_{cc}	CC_1	CC_2	CC_3	Δ	α	BKS	$Cost$	Δ_B	$t(s)$	$nbMPO$	$nbMPOimp$	nbR
R101	2	1	200	100	100		[1,100]	1.1	2771.6431	2756.7887	**-0.54**	531.18	38	6	49
			200	100	100				2834.4928	2827.3512	**-0.25**	426.29	19	2	49
			200	100	100				3016.8985	2985.2441	**-1.05**	507.52	35	3	53
			200	100	100				3048.9983	3031.0145	**-0.59**	533.16	35	3	53
			200	100	100				2800.6961	2784.0475	**-0.59**	496.69	29	4	49
R101	2	1	200	100	100		[1,100]	1.5	2128.3373	2122.9233	**-0.25**	530.35	29	2	35
			200	100	100				2194.4059	2183.9167	**-0.48**	578.51	42	3	36
			200	100	100				2299.5101	2295.0886	**-0.19**	501.07	25	3	39
			200	100	100				2307.0220	2297.4593	**-0.41**	582.93	39	3	38
			200	100	100				2149.6086	2137.8987	**-0.54**	507.42	24	1	36
R101	2	1	200	100	100		[1,100]	2	1715.9710	1698.3738	**-1.03**	551.73	25	1	27
			200	100	100				1730.2647	1722.1094	**-0.47**	589.96	33	3	27
			200	100	100				1813.5162	1816.3983	0.16	547.78	31	4	29
			200	100	100				1827.3575	1832.7354	0.29	582.61	32	7	29
			200	100	100				1717.4555	1704.9316	**-0.73**	512.40	16	1	27
R101	2	1	200	100	100		[1,100]	2.5	1461.4911	1452.8434	**-0.59**	682.70	32	2	21
			200	100	100				1466.1825	1460.9983	**-0.35**	705.62	43	9	22
			200	100	100				1533.0524	1534.9745	0.13	633.60	33	4	23
			200	100	100				1545.9473	1541.2999	**-0.30**	702.58	34	4	23
			200	100	100				1431.3942	1432.8548	0.10	631.85	38	0	21
R101	2	1	200	100	100		[40,60]	1.1	4878.3554	4815.8420	**-1.28**	477.78	28	1	93
			200	100	100				4885.1949	4854.3103	**-0.63**	533.46	46	0	95
			200	100	100				4954.1386	4928.0506	**-0.53**	396.24	18	0	97
			200	100	100				4906.3430	4886.1829	**-0.41**	357.43	15	0	95
			200	100	100				4812.6715	4778.5328	**-0.71**	390.36	16	0	93
R101	2	1	200	100	100		[40,60]	1.5	3515.9354	3489.7450	**-0.74**	602.15	39	3	62
			200	100	100				3584.7753	3544.2844	**-1.13**	554.11	32	0	64
			200	100	100				3592.7577	3551.5178	**-1.15**	609.51	45	0	64
			200	100	100				3583.5791	3546.3662	**-1.04**	512.74	25	3	63
			200	100	100				3495.4743	3472.3544	**-0.66**	586.66	41	3	62
R101	2	1	200	100	100		[40,60]	2	2640.2882	2622.6570	**-0.67**	636.79	37	2	45
			200	100	100				2669.1453	2657.6846	**-0.43**	485.80	15	1	46
			200	100	100				2690.6244	2665.3118	**-0.94**	578.84	29	1	46
			200	100	100				2704.5457	2695.0898	**-0.35**	620.18	28	0	47
			200	100	100				2655.6363	2641.4628	**-0.53**	550.42	24	1	46
R101	2	1	200	100	100		[40,60]	2.5	2153.4354	2141.4426	**-0.56**	644.84	25	0	35
			200	100	100				2213.9592	2189.4736	**-1.11**	571.24	23	0	36
			200	100	100				2222.5353	2202.7464	**-0.89**	675.78	38	1	36
			200	100	100				2204.7443	2203.0546	**-0.08**	650.99	34	3	36
			200	100	100				2159.9779	2149.3260	**-0.49**	538.39	16	0	35

Table 14 (continued).

instances									ALNS results					
	M	p	n_{cc}	CC_1	CC_2	CC_3	Δ	α	BKS	$Cost$	Δ_B	$t(s)$	$nbMPO$	$nbMPOimp$	nbR
R101	3	0.6	179	65	50	64	[1,100]	1.1	2091.9773	2082.4706	**-0.45**	448.50	37	2	34
			194	67	63	64			2248.9343	2221.3245	**-1.23**	534.88	43	2	36
			186	62	58	66			2137.6592	2134.4081	**-0.15**	447.11	27	2	37
			193	69	57	67			2131.7891	2131.5677	**-0.01**	458.80	24	1	36
			190	61	56	73			2205.0653	2183.8628	**-0.96**	478.30	31	0	38
R101	3	0.6	179	65	50	64	[1,100]	1.5	1638.1808	1628.8003	**-0.57**	507.92	39	8	25
			194	67	63	64			1735.8727	1726.5884	**-0.53**	564.75	38	5	26
			186	62	58	66			1688.8930	1678.2108	**-0.63**	563.18	48	7	27
			193	69	57	67			1695.4234	1683.4501	**-0.71**	502.74	27	3	26
			190	61	56	73			1708.9017	1698.7518	**-0.59**	508.70	24	0	27
R101	3	0.6	179	65	50	64	[1,100]	2	1344.1715	1333.3018	**-0.81**	579.85	38	4	19
			194	67	63	64			1417.4800	1409.0886	**-0.59**	638.09	33	4	20
			186	62	58	66			1382.2839	1366.0804	**-1.17**	554.83	24	2	20
			193	69	57	67			1385.4181	1380.4631	**-0.36**	628.49	23	3	20
			190	61	56	73			1377.7779	1372.9908	**-0.35**	612.43	32	3	21
R101	3	0.6	179	65	50	64	[1,100]	2.5	1146.9121	1144.9686	**-0.17**	703.87	39	4	15
			194	67	63	64			1214.4350	1211.4598	**-0.24**	739.74	37	7	16
			186	62	58	66			1188.0089	1183.2168	**-0.40**	707.54	47	2	16
			193	69	57	67			1190.1513	1188.4567	**-0.14**	750.52	30	0	16
			190	61	56	73			1203.4458	1199.0058	**-0.37**	676.18	27	1	17
R101	3	0.6	179	65	50	64	[40,60]	1.1	3048.1305	3027.5465	**-0.68**	455.96	37	2	54
			194	67	63	64			3278.5334	3244.0645	**-1.05**	444.34	25	2	57
			186	62	58	66			3035.4818	3030.9153	**-0.15**	451.00	25	0	55
			193	69	57	67			3126.1265	3115.1466	**-0.35**	432.35	24	1	56
			190	61	56	73			3144.6563	3112.9713	**-1.01**	469.27	37	1	57
R101	3	0.6	179	65	50	64	[40,60]	1.5	2234.7360	2224.7829	**-0.45**	447.71	29	2	37
			194	67	63	64			2376.7317	2368.8940	**-0.33**	531.41	31	1	39
			186	62	58	66			2255.2386	2254.6679	**-0.03**	528.31	38	3	39
			193	69	57	67			2315.3242	2313.7430	**-0.07**	459.05	18	1	39
			190	61	56	73			2371.8751	2356.9995	**-0.63**	463.99	19	0	40
R101	3	0.6	179	65	50	64	[40,60]	2	1806.0300	1787.6283	**-1.02**	468.94	27	1	28
			194	67	63	64			1915.8784	1898.5402	**-0.90**	573.18	25	1	30
			186	62	58	66			1780.7996	1776.2130	**-0.26**	560.20	34	0	29
			193	69	57	67			1844.3127	1832.6458	**-0.63**	615.95	33	2	30
			190	61	56	73			1857.0664	1850.7579	**-0.34**	539.93	42	1	30
R101	3	0.6	179	65	50	64	[40,60]	2.5	1507.1235	1489.3193	**-1.18**	475.82	37	2	22
			194	67	63	64			1603.4522	1587.9639	**-0.97**	639.35	39	0	24
			186	62	58	66			1505.6898	1505.9298	0.02	489.36	26	3	23
			193	69	57	67			1559.1172	1555.8633	**-0.21**	651.61	35	2	24
			190	61	56	73			1563.5929	1554.1869	**-0.60**	616.64	47	4	24

Table 14 (continued).

instances									ALNS results					
	M	p	n_{cc}	CC_1	CC_2	CC_3	Δ	α	BKS	$Cost$	Δ_B	$t(s)$	$nbMPO$	$nbMPOimp$	nbR
R101	3	1	300	100	100	100	[1,100]	1.1	3071.7439	3041.7396	**-0.98**	893.51	45	4	53
			300	100	100	100			2946.0070	2934.6564	**-0.39**	837.63	29	6	52
			300	100	100	100			2985.6872	2964.3072	**-0.72**	934.43	41	7	52
			300	100	100	100			3105.7428	3079.8490	**-0.83**	813.04	31	5	55
			300	100	100	100			2852.7976	2858.3302	0.19	944.08	32	4	51
R101	3	1	300	100	100	100	[1,100]	1.5	2345.1176	2325.9298	**-0.82**	930.92	31	7	39
			300	100	100	100			2265.2202	2250.6250	**-0.64**	945.72	35	4	38
			300	100	100	100			2296.8218	2300.1563	0.15	1068.96	37	5	38
			300	100	100	100			2371.0709	2365.1241	**-0.25**	1020.65	37	3	40
			300	100	100	100			2190.7829	2185.2957	**-0.25**	956.33	20	5	37
R101	3	1	300	100	100	100	[1,100]	2	1859.7774	1847.8667	**-0.64**	1070.10	31	3	29
			300	100	100	100			1816.4798	1800.5916	**-0.87**	1023.86	34	1	28
			300	100	100	100			1825.7908	1813.6036	**-0.67**	1117.82	40	5	28
			300	100	100	100			1876.5282	1871.2735	**-0.28**	1052.49	26	3	30
			300	100	100	100			1765.6787	1763.3915	**-0.13**	1277.70	27	7	28
R101	3	1	300	100	100	100	[1,100]	2.5	1578.8174	1561.1608	**-1.12**	1208.35	36	4	23
			300	100	100	100			1539.1188	1522.9296	**-1.05**	1498.51	37	8	23
			300	100	100	100			1550.6387	1527.6379	**-1.48**	1474.97	33	3	23
			300	100	100	100			1597.1673	1584.9913	**-0.76**	1285.72	28	1	24
			300	100	100	100			1510.7153	1494.2903	**-1.09**	1395.11	27	6	22
R101	3	1	300	100	100	100	[40,60]	1.1	4607.7839	4605.8439	**-0.04**	887.44	25	1	87
			300	100	100	100			4540.1249	4510.2998	**-0.66**	820.83	23	1	85
			300	100	100	100			4560.4271	4509.0150	**-1.13**	848.02	25	0	85
			300	100	100	100			4678.0437	4643.9582	**-0.73**	821.14	23	1	89
			300	100	100	100			4541.4038	4518.3114	**-0.51**	900.92	29	1	85
R101	3	1	300	100	100	100	[40,60]	1.5	3411.5788	3379.7642	**-0.93**	1001.82	47	2	60
			300	100	100	100			3373.6096	3362.6067	**-0.33**	1095.28	56	3	60
			300	100	100	100			3409.7802	3385.7239	**-0.71**	1056.57	54	3	60
			300	100	100	100			3432.4292	3408.6962	**-0.69**	1021.90	43	5	61
			300	100	100	100			3433.4163	3374.5175	**-1.72**	1022.52	49	3	60
R101	3	1	300	100	100	100	[40,60]	2	2639.2852	2626.6402	**-0.48**	1149.13	35	3	45
			300	100	100	100			2609.9934	2604.937	**-0.19**	1033.69	20	1	44
			300	100	100	100			2600.4690	2582.7011	**-0.68**	1123.78	40	4	44
			300	100	100	100			2656.4737	2636.4516	**-0.75**	1344.94	41	4	46
			300	100	100	100			2620.8533	2627.2844	0.25	1138.19	25	2	45
R101	3	1	300	100	100	100	[40,60]	2.5	2195.1808	2196.4906	0.06	1226.16	41	2	36
			300	100	100	100			2186.6508	2170.1547	**-0.75**	1264.51	34	6	35
			300	100	100	100			2192.0684	2179.0868	**-0.59**	1255.90	46	5	35
			300	100	100	100			2241.4465	2220.6163	**-0.93**	1227.96	32	0	36
			300	100	100	100			2179.4536	2167.7968	**-0.53**	1157.51	34	5	35

References

Archetti, C., Bianchessi, N., and Speranza, M. G. (2015). A branch-price-and-cut algorithm for the commodity constrained split delivery vehicle routing problem. *Computers & Operations Research*, 64:1–10.

Archetti, C., Campbell, A. M., and Speranza, M. G. (2014). Multicommodity vs. single-commodity routing. *Transportation Science*, 50(2):461–472.

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.

Archetti, C. and Speranza, M. G. (2012). Vehicle routing problems with split deliveries. *International transactions in operational research*, 19(1-2):3–22.

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

Beasley, J. (1983). Route-first cluster-second methods for vehicle routing. *Omega*, 11:403–408.

Dror, M. and Trudeau, P. (1989). Savings by split delivery routing. *Transportation Science*, 23(2):141–145.

Dror, M. and Trudeau, P. (1990). Split delivery routing. *Naval Research Logistics (NRL)*, 37(3):383–402.

François, V., Arda, Y., Crama, Y., and Laporte, G. (2016). Large neighborhood search for multi-trip vehicle routing. *European Journal of Operational Research*, 255(2):422–441.

Gribkovskaia, I., Halskau, Ø., Laporte, G., and Vlček, M. (2007). General solutions to the single vehicle routing problem with pickups and deliveries. *European Journal of Operational Research*, 180(2):568–584.

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

Nagy, G., Wassan, N. A., Speranza, M. G., and Archetti, C. (2015). The Vehicle Routing Problem with Divisible Deliveries and Pickups. *Transportation Science*, 49(2):271–294.

Pisinger, D. and Ropke, S. (2010). Large neighborhood search. In *Handbook of metaheuristics*, pages 399–419. Springer.

Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12):1985–2002.

Prins, C. (2009). A GRASP $\times$ evolutionary local search hybrid for the vehicle routing problem. *Bio-inspired algorithms for the vehicle routing problem*, pages 35–53.

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.

Shaw, P. (1997). A new local search algorithm providing high quality solutions to vehicle routing problems. *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK*.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 417–431. Springer.

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

Sze, J. F., Salhi, S., and Wassan, N. (2016). A hybridisation of adaptive variable neighbourhood search and large neighbourhood search: Application to the vehicle routing problem. *Expert Systems with Applications*, 65:383–397.

Sze, J. F., Salhi, S., and Wassan, N. (2017). The cumulative capacitated vehicle routing problem with min-sum and min-max objectives: An effective hybridisation of adaptive variable neighbourhood search and large neighbourhood search. *Transportation Research Part B: Methodological*, 101:162–184.

Toth, P. and Vigo, D. (2014). Vehicle Routing: Problems, Methods, and Applications. *MOS-SIAM Series on Optimization. Philadelphia: SIAM. 2nd ed.*