EURO Journal on Transportation and Logistics 10 (2021) 100039

Contents lists available at ScienceDirect

EURO Journal on Transportation and Logistics

journal homepage: www.journals.elsevier.com/euro-journal-on-transportation-and-logistics

The vehicle routing problem with relaxed priority rules

Thanh Tan Doan [a], Nathalie Bostel [a], Minh Hoàng Hà [b,*]

[a] Université de Nantes, CNRS, LS2N, F-44000, Nantes, France
[b] ORLab, Faculty of Computer Science, Phenikaa University, Hanoi, Viet Nam

ARTICLE INFO

Keywords:
Vehicle Routing Problem
d-relaxed priority rule
Mixed integer linear programming
Adaptive large neighborhood search

ABSTRACT

The Vehicle Routing Problem (VRP) is one of the most studied topics in Operations Research. Among the numerous variants of the VRP, this research addresses the VRP with relaxed priority rules (VRP-RPR) in which customers are assigned to several priority groups and customers with the highest priorities typically need to be served before lower priority ones. Additional rules are used to control the trade-off between priority and cost efficiency. We propose a Mixed Integer Linear Programming (MILP) model to formulate the problem and to solve small-sized instances. A metaheuristic based on the Adaptive Large Neighborhood Search (ALNS) algorithm with problem-tailored components is then designed to handle the problem at larger scales. The experimental results demonstrate the performance of our proposed algorithm. Remarkably, it outperforms a metaheuristic recently proposed to solve the Clustered Traveling Saleman Problem with *d*-relaxed priority rule (CTSP-*d*), a special case of VRP-RPR, in both solution quality and computational time.

1. Introduction

Transportation is one of the key sectors of the logistics industry. It plays an increasing crucial role in globalization. According to Rodrigue et al. (2013), the transportation costs account for about 10% of the total cost of a product. Saving transportation costs becomes a competitive factor in socio-economic activities. Therefore, since being introduced by Dantzig and Ramser (1959), the VRP has quickly become one of the most important optimization problems in transportation. Today, the VRP is a well-known problem with many variants and applications in both commercial and non-commercial activities. In this study, we focus on a VRP variant in which customers are classified into different priority groups and rules are integrated to control service plan. We name this problem the Vehicle Routing Problem with Relaxed Priority Rules (VRP-RPR). It is formally defined as follows. Given an undirected graph $G = (N, E)$ where $N = \{0, \ldots, n\}$ is a set of nodes and $E = \{(i,j) : i,j \in N, i \neq j\}$ is a set of edges, node 0 represents the depot where a fleet of k_{max} vehicles is based. There are several types of vehicle and we denote $\tau(k)$ the type of vehicle k. Nodes indexed from 1 to n represent customers. Each customer i is associated with a demand q_i and a priority p_i in a set P of priorities sorted from the highest to the lowest values, $P = \{1, \ldots, p_{max}\}$. Each edge $(i,j) \in E$ represents a possible trip between a node i and a node j with transportation cost l_{ij}. The goal is to find up to k_{max} vehicle routes starting and ending at the depot, such that:

- Each customer is served at most one time,
- The total demand quantity of any route k does not exceed the vehicle capacity $C_{\tau(k)}$,
- The length of any route k is never greater than a given value $L_{\tau(k)}$,
- Two rules, "*d*-relaxed priority" and "Order of Demand Fulfillment (*ODF*)", defined below, are satisfied,
- A weighted summation of the total demand lost and the total travel cost is minimized.

We now present two important rules introduced in the study of Panchamgam (2011) in order to manage the priorities. The first one is the *d*-relaxed priority rule, represented by a value d that allows flexible service control. A route k is considered to satisfy the *d*-relaxed priority rule if at any point i of k, the vehicle visits the next location with priority no greater than $p + d$ where p is the highest priority group among all locations visited after i in this route. By setting different values for d ($d \in N$), we can achieve different solutions in which the orders of customers respect the priorities strictly, partially or not at all. Choosing an appropriate value for d mainly depends on the emergency level. That is to say, if the situation is very urgent, a solution should be generated with a strict priority rule, i.e., d is set to zero, which is generally costly. Otherwise, if the situation is less urgent, we can increase this value to save cost and accept a solution in which high priority locations may be served after lower priority ones. The second rule, called *ODF*, ensures

* Corresponding author.
E-mail address: hoang.haminh@phenikaa-uni.edu.vn (M.H. Hà).

https://doi.org/10.1016/j.ejtl.2021.100039
Received 18 October 2020; Received in revised form 16 April 2021; Accepted 22 April 2021

that at the end of the day (or time horizon), all higher priority locations need to be satisfied before the vehicles can satisfy any lower priority one.

We illustrate how these rules work with a graph including 10 customers and one depot. The customers are divided into 3 priority groups, each including three customers except the third which includes four. Fig. 1 depicts three feasible solutions corresponding to three values of d. The numbers in the circles represent priorities of customers and letter **D** in the square represents the depot. It should be mentioned that Fig. 1 depicts a typical solution produced by our model that Panchamgam (2011) called "Local Timing". In this model, the d-relaxed priority rule needs to be satisfied in each route but not necessarily across routes. For example, with $d = 0$, the rule ensures that the highest priority groups are served before the lowest ones in routes A and B. However, priority 2 in route A can be served before priority 1 in route B and so on. In other words, we do not consider service times between priority groups located in different routes. Contrary to the "Local Timing" model, Panchamgam (2011) presented another model called "Global Timing" where the d-relaxed priority rule holds not only in each route but also across routes. The author claimed that this model helps high priority customers to be served faster than the "Local Timing" model but solutions are generally much more costly. Therefore, we leave the "Global Timing" model for future study.

Back to the example in Fig. 1, when $d = 0$, the relaxation is not allowed, then in both routes A and B, the higher priority customers are visited before the lower priority ones. For $d = 1$, the solution is partially relaxed with one level of relaxation for each priority group. As a result, customers with priority 1 can be served before or after customers with priority 2 and customers with priority 2 can be served before or after customers with priority 3 but all customers with priority 1 must be served before customers with priority 3 (see route B). For $d = 2$, each priority group now has two levels of relaxation then priorities from 1 to 3 can be served in any order. Thus, the solution is completely relaxed and considering the cases where $d \geq p_{max}$ is not necessary.

Due to the *ODF* rule definition, if in a solution, at least one customer with $p = 3$ is served, then all customers with $p = \{1, 2\}$ must be served. In Fig. 1, all solutions satisfy this rule. For the case $d = 2$, all customers are served. But in cases $d = \{0, 1\}$, the priority constraint is less relaxed then we cannot serve a customer of the lowest priority.

The problem has some applications in practice, especially in situations where classifying customers into different priority groups is needed. Panchamgam (2011) addressed this in the context of last-mile distribution in the humanitarian relief operations. Priorities of locations are assumed to be assigned in advance by the authorities and relief fleet needs to deliver essential goods to vulnerable locations with respect to several rules, including the rules mentioned above. The VRP-RPR can also model the situation where unmanned aerial vehicles (UAVs) are used to monitor targets with different priorities, as in a study by Shetty et al. (2008). In addition, it could be applicable to the distribution of commercial products if the priority of client delivery is determined by its storage level. Out-of-stock locations should be considered to be more important than others and clustered into the highest priority group. More recently, the routing problems with the d-relaxed priority rule can model technician scheduling problems faced by internet service providers as introduced in Hà et al. (2020).

There are several VRP variants in which the customers are also divided into clusters. The Generalized Vehicle Routing Problem (GVRP) studied in Ghiani and Improta (2000); Hà et al. (2014) considers disjoint clusters and requires that only one customer in each cluster is visited. The Selective Vehicle Routing Problem (SVRP) in Sabo et al. (2020) generalizes the GVRP in the sense that the customers are divided into clusters, but they may belong to one or more clusters. For the Clustered Vehicle Routing Problem (CluVRP) Battarra et al. (2014), and Pop et al. (2018), a vehicle visiting one customer in a cluster must visit all the remaining customers therein before leaving it. These problems are different from our VRP-RPR because they do not consider priorities associated to the clusters. Therefore, clusters can be visited in any order.

When demand is replaced by profit, the VRP-RPR belongs to the wide class of the vehicle routing problems with profits. More specifically, the team orienteering problem (TOP), the profitable VRP (VRPP), the VRP with private fleet and common carrier (VRPPFCC), and the capacitated profitable tour problem (CPTP). All correspond to the VRP-RPR instances with a single group. The standard CVRP corresponds to instances with one group and identical vehicles with unlimited maximum route length. The VRP-RPR can also be reduced to the Clustered Traveling Salesman Problem with d-relaxed priority rule (CTSP-d) defined by Hà et al. (2020) if we use a single vehicle without capacity and maximum route length constraints. As a result, it also includes the Clustered Traveling Salesman Problem with a Prespecified Order on the Clusters (CTSP-PO) in Potvin and Guertin (1998) because the CTSP-PO is a special case of the CTSP-d when $d = 0$. The VRP-RPR is a variant of the Humanitarian Vehicle Routing Problem (HVRP) of Panchamgam (2011), which is mentioned in Chapter 14 of the book by Toth and Vigo (2014). It extends the HVRP in the sense that we allow flexible dispatch, i.e., unnecessary vehicles may not be mobilized, while in the HVRP, all available vehicles must be used in the solution. Moreover, we also consider heterogeneous fleet, instead of homogeneous fleet as in the HVRP. However, we do not consider the "global timing" constraint as mentioned above.

Our contributions are as follows. First, we generalize the problem proposed in Panchamgam (2011) by allowing heterogeneous fleets with flexible dispatch. The generalization is motivated by the fact that in practical situations, fleets may include different types and sizes. If demand is greater than supply, all vehicles should be dispatched if possible. Otherwise, unnecessary dispatches should be rejected to save resources. We propose a MILP model with enhanced constraints that can solve several small instances to optimality with MILP solvers. We also introduce a meta-heuristic based on the ALNS framework with problem-tailored components to solve larger instances. In particular, we propose two new removal operators, a local search, and three acceleration techniques, as well as a procedure to verify feasible insertions with respect to the d-relaxed priority rule in $\mathcal{O}(1)$ to speed up the process. The experiments carried out on existing CTSP-d instances as well as on new VRP-RPR instances show the effectiveness of our approaches. Compared with the work of Hà et al. (2020), we consider a more general problem with multiple vehicles and optional visits, i.e., several customers can be unvisited in the solution. The metaheuristic proposed in Hà et al. (2020) to solve the CTSP-d cannot handle the VRP-RPR without re-designing all of its main components. In addition, our new algorithm with removal and repair operators based on the ALNS framework is totally different from the method used in Hà et al. (2020), which relies mainly on local searches. When tested on CTSP-d instances, our ALNS outperfoms the heuristic method in Hà et al. (2020) in terms of both solution quality and running time. Specially, 23 best known CTSP-d solutions have been improved in this study.

The rest of the article is organized as follows. Section 2 provides a brief review of relevant problems. Section 3 presents a MILP model for our VRP-RPR. Section 4 describes a metaheuristic based on the ALNS framework to solve the problem. Section 5 presents and discusses experimental results. Finally, Section 6 provides our conclusions and

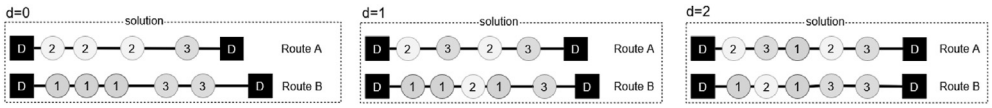

Fig. 1. Feasible solutions in different values of d.

suggestions for future studies.

2. Literature review

As discussed in Section 1, the VRP-RPR can be reduced to four important problems: TOP, VRPP, VRPPFCC, and CPTP. The state-of-the-art methods for these problems can be found in Bulhões et al. (2018) and Goeke et al. (2018). The readers can refer to the surveys by Archetti et al. (2014), Vansteenwegen et al. (2010), and Vidal et al. (2013) for more information on the wide class of the vehicle routing problems with profits.

The CTSP-PO problem defined in Potvin and Guertin (1998) is similar to the VRP-RPR in the sense that priorities are imposed on customer clusters. However, the priority rule defined in the CTSP-PO requires that the higher priority clusters must be always visited before the lower priority ones, which is stricter and less flexible than the *d*-relaxed priority rule. This rule can lead to inefficient routes in terms of transportation cost. As mentioned above, the CTSP-PO can be seen as the VRP-RPR with $d = 0$. Also in Potvin and Guertin (1998), the CTSP-PO was solved by a genetic algorithm. In Ahmed (2014), a hybrid genetic algorithm was developed using sequential constructive crossover (SCX), 2-opt search, and a local search.

Several articles in the literature have studied VRPs while taking into account the priority of each customer. Yang et al. (2015) introduced a dynamic VRP with time windows and multiple priorities. Each customer has a priority level represented by a weight, which is multiplied by the delayed service time at customer position to produce a penalty value. A weighted sum function aims to minimize both the total travel distance and the total penalty. With this method, higher priorities tend to be served before lower ones. Song and Ko (2016) published a real-life problem in the perishable food transportation sector. In this study, priorities are not assigned to customers beforehand. Instead, during the search, each unrouted customer i is set a priority that equals to the number of unrouted customers within a radius R around i divided by their average demand. The customer with the highest priority will be selected to be served next. This heuristic method makes it possible to serve as many customers as possible without conflict with the objective of maximum demand satisfaction.

In the context of emergency routing, many problems use the priority assignment as a method to tag the urgency level of the affected locations. Oran et al. (2012) presented a location-routing problem with a set of potential sites for emergency facilities and a set of emergency points. The problem considers multiple types of emergency where each type of emergency requires a type of vehicle. The authors proposed a facility location model and a routing model. First, the facility location model allocates vehicles to the facilities in such a way that as many prioritized emergency points as possible are covered. Second, after achieving solution from the first model, the routing model with time windows and priority consideration is taken into consideration. In this study, each emergency point i is assigned a priority p_k^i corresponding to vehicle of type k. This priority is the basis to calculate weight α_k^i of i in the objective functions of both models. Sheu (2007) developed a three-layer logistic distribution model that allows fast delivery of relief goods to disaster-affected areas. These areas are grouped into several clusters by a fuzzy-based clustering technique where areas in the same cluster have the same level of urgency. After that, the authors proposed a method to assess cluster priority (it is also the priority of the areas inside). A network distribution model is then formulated to solve the problem of distributing multiple relief goods from distribution centers to prioritized areas in clusters. Gralla and Goentzel (2018) presented a transport planning problem for relief supplies. In this study, authorities preassign priorities to both relief supplies and locations, based on their emergency levels. The objective is a utility function that considers the total amount of supplies delivered, the proportion of high priority goods delivered, the speed of delivery, and the operation cost.

Although the priority of customers is an important aspect that should be taken into account when making delivery schedules, the *d*-relaxed priority rule has not been well studied in the literature. To the best of our knowledge, Panchamgam (2011) initially introduced the HVRP with this rule. Several MILP models used for different fleet sizes and service types were proposed and solved to optimality with CPLEX on several small instances with up to 30 nodes and homogeneous fleets with up to 2 vehicles. Panchamgam et al. (2013) studied a special case of the HVRP called Hierarchical Traveling Salesman Problem (HTSP). In this research, the authors derived worst-case bounds with respect to the classical TSP and were able to show that the bounds are tight. This problem was later mentioned in Chapter 14 of Toth and Vigo (2014). Kuang (2012) solved the HTSP with a 2-Opt heuristics on four data sets (up to 64 nodes). Recently, Hà et al. (2020) presented a new MILP model for the HTSP and renamed this problem as CTSP-*d*. The authors also proposed the first metaheuristic based on the GILS-RVND algorithm of Silva et al. (2012) to solve this problem. The GILS-RVND combines the features of the Greedy Randomized Adaptive Search (GRASP), Iterated Local Search (ILS), and Random Variable Neighborhood Descent (VND). The algorithm was tested on randomly generated instances with up to 200 nodes.

3. Mathematical formulation

In this section, based on the formulation introduced in *Appendix D.3* of Panchamgam (2011), we propose a MILP model for the VRP-RPR. Our model differs from the original one in three respects. First, as mentioned in Section 1, we allow heterogeneous fleets. Second, the number of vehicles in our model is also a decision variable in order to possibly save resources in some cases. Third, we do not use variables representing starting service time at each customer location. Instead, we introduce variables that represent the visiting order on a service route. As shown in Hà et al. (2020), using these variables may improve the performance of the MILP-based exact method. Additional sets and parameters are denoted as follows:

$\overline{N_i}$	Set of nodes excluding node i, $\overline{N_i} = N \setminus \{i\}$
G_p	Group of nodes with priority $p \in P$
K	Set of k_{max} vehicles available, $K = \{1,\ldots,k_{max}\}$
A	Predefined parameter controlling the trade-off between the total demand lost and the total cost ($0 \leq \alpha \leq 1$)

There are four types of variables in the formulation. For each customer $i \in \overline{N_0}$, we define a binary variable v_i^k, set to 1 if and only if vehicle k visits node i, as well as a non-negative variable u_i^k representing the visiting order of node i on the route of vehicle k. We also use a binary variable x_{ij}^k equal to 1 if vehicle k travels from i to j. Finally, z_p is a binary variable, defined for each group $p \in P$, and set to 1 if the total demand of nodes in this group is satisfied. The formulation for the VRP-RPR is stated as:

Objective function:

$$\text{Minimize} \quad \alpha\left(\sum_{i\in\overline{N_0}} q_i - \sum_{k\in K}\sum_{i\in\overline{N_0}} q_i v_i^k\right) + (1-\alpha)\sum_{k\in K}\sum_{(i,j)\in E} l_{ij}x_{ij}^k \tag{1}$$

Subject to:

$$\sum_{j\in\overline{N_0}} x_{0j}^k \leq 1, \quad \sum_{i\in\overline{N_0}} x_{i0}^k \leq 1 \quad \forall k \in K \tag{2}$$

$$v_i^k = \sum_{j\in N} x_{ji}^k \quad \forall i \in \overline{N_0}, k \in K \tag{3}$$

$$\sum_{k\in K} v_i^k \leq 1 \quad \forall i \in \overline{N_0} \tag{4}$$

$$\sum_{j\in\overline{N_i}} x_{ji}^k = \sum_{j\in\overline{N_i}} x_{ij}^k, \quad \forall i\in\overline{N_0}, k\in K \tag{5}$$

$$u_0^k = 0 \quad \forall k\in K \tag{6}$$

$$u_i^k + 1 - B\left(1 - x_{ij}^k\right) \le u_j^k \quad \forall i\in N, j\in\overline{N_0}, k\in K \tag{7}$$

$$u_i^k + 1 \le u_j^k \quad \forall p,q\in P, i\in G_p, j\in G_q, q\ge p+d+1, k\in K \tag{8}$$

$$\sum_{i\in\overline{N_0}} q_i v_i^k \le C_{\tau(k)} \quad \forall k\in K \tag{9}$$

$$\sum_{(i,j)\in E} l_{ij} x_{ij}^k \le L_{\tau(k)} \quad \forall k\in K \tag{10}$$

$$\sum_{k\in K}\sum_{i\in G_p} q_i v_i^k \ge z_p \sum_{i\in G_p} q_i \quad \forall p\in P \tag{11}$$

$$\sum_{k\in K}\sum_{i\in G_p} q_i v_i^k \le z_{p-1} \sum_{i\in G_p} q_i \quad \forall p\in P-\{1\} \tag{12}$$

$$x_{ij}^k \in \{0,1\} \quad \forall i,j\in N, k\in K \tag{13}$$

$$v_i^k \in \{0,1\} \quad \forall i\in\overline{N_0}, k\in K \tag{14}$$

$$z_p \in \{0,1\} \quad \forall p\in P \tag{15}$$

$$u_i^k \in \mathbb{R}^+ \quad \forall i\in N, k\in K \tag{16}$$

The objective function (1) aims to minimize the weighted combination of the total demand lost and the total travel cost. Constraints (2) state that if a vehicle leaves the depot, it has to return. Constraints (3) and (4) guarantee that a customer is served by at most one vehicle. Constraints (5) ensure the flow conservation for each vehicle. Constraints (6) initialize variables u at the depot. Constraints (8) restrict u variables at nodes i and j if vehicle k travels from i to j and also take the role of subtour elimination. Here, B is a sufficiently large number and can be estimated by $|N|$. Constraints (8) enforce the d-relaxed priority rule on each route. Constraints (9) require that the total demand delivered does not exceed the vehicle capacity. Constraints (10) enforce the maximum route length for each vehicle. Constraints (11) and (12) enforce the *ODF* rule for each vehicle. Constraints (13) through (16) define the domain of the decision variables. The combination of the capacity constraint and the *ODF* rule leads to the following constraints, which can be used to strengthen the model:

$$\sum_{k\in K}\sum_{i\in G_1} q_i v_i^k \le \min\left(\sum_{i\in G_1} q_i, \sum_{k\in K} C_{\tau(k)}\right) \tag{17}$$

$$\sum_{k\in K}\sum_{i\in G_p} q_i v_i^k \le \max\left(0, \sum_{k\in K} C_{\tau(k)} - \sum_{r=1}\sum_{i\in G_r} q_i\right) \quad \forall p\in P-\{1\} \tag{18}$$

Constraints (17) express the upper limit of the vehicle capacity for the first priority group, while constraints (18) imply the upper limit of the vehicle capacity for groups with priority p. We adapt constraints from C.9 to C.12 in Panchamgam (2011) to speed up the model with the set of constraints below:

$$\sum_{k\in K}\sum_{i\in G_p}\sum_{j\in G_q} x_{ji}^k = 0 \quad \forall q\ge p+d+1 \tag{19}$$

$$\sum_{k\in K}\sum_{i\in G_p}\sum_{j\in G_q} x_{ij}^k \le 1 \quad \forall q\ge p+d+1 \tag{20}$$

Constraints (19) state that a vehicle never moves from a node with priority $q\ge p+d+1$ to a node with priority p. Constraints (20) ensure that there is at most one link from a node with priority p to a node with priority $q\ge p+d+1$.

4. An adaptive large neighborhood search algorithm with integrated local search

ALNS was first introduced by Ropke and Pisinger (2006). The entire search process is divided into w_1 segments, each segment including w_2 iterations. The process starts from an initial solution S_{init}. After each iteration, a new solution S_{new} is generated from the current solution $S_{current}$ and the best solution S_{best} during the search will be updated until the end of the search. For more detail, S_{new} is generated after destroying $S_{current}$ with an operator selected from the set of removal operators R and repairing it with another operator selected from the set of insertion operators I. The quality of S_{new} is evaluated via Simulated Annealing (SA). Operators in R and I are selected and scored by the mechanism of Roulette Wheel Selection (RWS). That is to say, after evaluating S_{new} quality, RWS adds an appropriate score to both removal and insertion operators selected in the current iteration. At the end of the current segment, for each removal or repair operator, RWS calculates its weight for the next segment by the weighted summation of accumulated scores and the current weight. The selection probability of an operator will depend on its weight.

In recent years, hybrid heuristic methods are often used to combine advantages of classical algorithms. According to our observation, applying the local search in ALNS can further improves the solution quality. Therefore, we proposed an ALNS integrated with local search as described in Algorithm 1.

4.1. Initial solution

To construct an initial solution, we use a cheapest insertion heuristic. From an empty solution with k_{max} empty routes, we insert each candidate into the solution considering methods to handle the rules described in Subsection 4.2. The candidate is selected in such a way that its insertion cost is the cheapest among the unrouted nodes. The insertion cost $\Delta_i^{(u,v)}$ of a node i in edge (u,v) is calculated as follows: $\Delta_i^{(u,v)} = l_{ui} + l_{iv} - l_{uv}$ where l_{ui}, l_{iv}, and l_{uv} are the costs of edges (u,i), (i,v), and (u,v), respectively.

Algorithm 1. The ALNS algorithm with integrated local search

4.2. Insertion operators

As mentioned above, insertion operations rebuild the partial solution by adding greedily unrouted nodes one by one to generate a new solution. Depending on the insertion criteria, different insertion strategies have been proposed. After testing different insertion operators from the literature, we decided to use four operators which give the best performance for our algorithm.

(1) **Regret insertion**: This operator is based on the method of Tillman and Cain (1972) and is used in a number of studies, e.g., Potvin and Rousseau (1993); Ropke and Pisinger (2006). The regret value of a node i can be seen as the total opportunity cost of inserting i into the first best place instead of the second best place, the third best place, and so on. Let Δ_i^k denotes the change in the objective value incurred by inserting node i into its best position in the k^{th} cheapest route. The regret value of a node i is computed by:

$$r_i = \sum_{k=2}^{k_{rg}} \left(\Delta_i^k - \Delta_i^1\right) \tag{21}$$

In this article, we set k_{rg} to 3, i.e., regret-3 heuristic is used as an

Input : Graph, set of removal operators R, set of insertion operators I, sets of local searches LS_1 and LS_2, parameters
Output: S_{best}

begin
  Construct an initial feasible solution S_{init}
  $S_{best} \leftarrow S_{current} \leftarrow S_{init}$
  Set an equal probability for operators in R and in I
  Set an initial temperature T and a cooling rate c
  for $segment = 1$ **to** w_1 **do**
    Update all probabilities for operators in R and in I
    Reset all scores to 0
    for $iteration = 1$ **to** w_2 **do**
      Select a removal operator with updated probability to destroy $S_{current}$
      Select an insertion operator with updated probability to repair $S_{current}$ to obtain S_{new}
      if S_{new} *is better than* S_{best} **then**
        Improve S_{new} with set of local search LS_1
        $S_{best} \leftarrow S_{current} \leftarrow S_{new}$
        Add score σ_1 to the selected operators
      else if S_{new} *is better than* $S_{current}$ & $segment > w_1/2$ **then**
        Improve S_{new} with set of local search LS_2
        $S_{current} \leftarrow S_{new}$
        if S_{new} *is better than* S_{best} **then**
          $S_{best} \leftarrow S_{new}$
        end
        Add score σ_2 to the selected operators
      else
        Generate a random number $y \in [0, 1)$
        if $y \geq e^{-\frac{f(S_{new})-f(S_{current})}{T}}$ **then**
          $S_{current} \leftarrow S_{new}$
          Add score σ_3 to the selected operators
        end
      end
      $T = T.c$
    end
    Update weights for operators in R and in I
  end
end

insertion operator. In each insertion iteration, the regret heuristic selects a candidate node i^* among the unrouted nodes according to this condition:

$$i^* = \arg \max_{i \in U} r_i \tag{22}$$

(2) **Regret insertion with noise**: As proposed in Ropke and Pisinger (2006), it can be necessary to use noising or randomization in repair heuristics to enhance the diversity level of the algorithm. Before selecting the candidate node to insert, a noise amount β is added to the regret value of each unrouted node to get a new value $r' = \max(0, r_i + \beta)$ where $\beta = \eta.\bar{l}$. Here $\bar{l}$ is a random number in $[-l_{max}, l_{max}]$ and η is a noise control parameter.
(3) **Cheapest insertion**: This operator uses the insertion method described in Subsection 4.1. Each candidate from the set of unrouted nodes is successively inserted into the partial solution with the smallest insertion cost. The cheapest insertion can also be seen as the regret-1 heuristic.
(4) **Cheapest insertion with noise**: Similarly to Regret insertion with noise, a noise value is also added to the insertion cost of each unrouted node before selecting the candidate node. We use the same method to calculate noise as described above.

Before inserting a node into a route, we need to check if it satisfies the *ODF* rule and the *d*-relaxed priority rule. Because this operation is repeated a number of times during the search, it must be designed carefully to speed up the algorithm. Here, we propose approaches to handle these rules efficiently.

- **Handling the *ODF* rule**: The *ODF* rule requires all higher priorities to be satisfied before any lower ones at the end of a day (time horizon). Thus, before inserting unrouted nodes into a partial solution, we sort the nodes in descending order of priorities $(p, p+1, p+2, \ldots)$ and insert them into the solution according to this order. However, the partial solution generally contains different priorities. Sometimes, we cannot satisfy this rule because the solution is full before all higher priority nodes are inserted. In this case, we continuously remove, from the solution, the most costly node of the lowest priority and insert the cheapest unrouted node with the highest priority into the solution until the *ODF* rule is satisfied.
- **Handling the *d*-relaxed priority rule**: In this rule, the nodes with priority p are visited before a node with priority $q \geq p + d + 1$. For the convenience of indexing, the depot on a route is represented by two copies. The depot where vehicles leave to serve is called the starting depot and the depot where vehicles return is called the ending depot. These depots are identical with the same location in the graph. Customer nodes are inserted between these depots with an index system considering 0 as the index of the starting depot. A node i with priority p_i satisfies the *d*-relaxed rule if it is inserted into the *Valid Insertion Range* (VIR) starting from index ι_s to index ι_e of route:
- *Starting index*: $\iota_s = \iota_{highest} + 1$, where $\iota_{highest}$ is the highest index of the nodes with priorities $p_1^* \leq p_i - d - 1$. If such a node does not exist, $\iota_s =$

1. This ensures that node i is always visited after any node with priority $p_i - d - 1$ or smaller values.

- *Ending index:* ι_e is the lowest index of the nodes with priorities $p_2^* \geq p_i + d + 1$. If this node does not exist, ι_e is set to the ending depot index. This ensures node i is always visited before any node with priority $p_i + d + 1$ or greater values.

We provide a small example to illustrate how to determine the VIRs of a node i with $p_i = 3$. Suppose that $d = 1$, in Fig. 2 we find the two VIRs of i in routes A and B of a partial solution. Number in circle represents the node priority, letter D in square represents the depot, and number above a node represents the node index. In route A, $\iota_s = 4$ because $\iota_{highest} = 3$ is the highest index of the nodes with $p_1^* \leq 1$ and $\iota_e = 8$ because it is the lowest index of the nodes with $p_2^* \geq 5$. In route B, there is no node with $p_1^* \leq 1$ and $p_2^* \geq 5$, then $i_s = 1$ and $i_e = 7$.

Finding VIRs is a time-consuming activity. To avoid computing them repeatedly, we use additional data structures to memorize the lowest and highest indices for each priority on each route. Initially, these values are set to 0. The memorized indices of priority p on a route will be updated if a node with priority p is inserted into the route and the insertion position is a new starting or/and a new ending. Moreover, we use an acceleration technique to further reduce the running time of the insertion operation as follows:

Acceleration procedure 1: Basically, each insertion operator searches for the best node to insert into the best position of a partial solution. This process repeats until no more nodes can be added. To avoid doing such a time-consuming operation whenever a node is added, we find and save the best insertion position for each unrouted node once, before the insertion takes place. After the insertion starts, each time a node is inserted into a route and two new edges are produced, we check the chances of improvement for the remaining unrouted nodes if they are inserted into these edges and update if necessary. By experiments, we observed that this method significantly speeds up the algorithm. We also note that this procedure does not work for the insertion with noise. Therefore, it is turned off in this insertion operator.

4.3. Removal operators

Removal operators play an important role in designing an efficient ALNS. The objective is to destroy a part of the current solution by removing one or several nodes/edges to create a new large neighborhood. The removal operation is performed without taking into account the rules. However, after this operation ends and a partial solution is created, we need to update the starting and ending indices of priorities on each route to ensure that the d-relaxed priority rule works exactly in the insertion phase described in Section 4.2.

In this study, after testing different removal operators from the literature, we adapt 5 operators to our problem, that are Related removal, Worst removal, Random removal, Route removal, and Graph Cluster removal. Two new operators specifically designed for the problem are Group removal and Joint Segment removal. Before removing nodes from current solution, we need to define a number of nodes to be removed n_r. For Related removal, Worst removal, and Random removal, we first select a random ratio r_1 in default range of $[0.1, 0.4]$ and define $n_r = r_1.|M|$. For the other operators, we calculate n_r in the same manner but their best ratios are determined in preliminary tests. In the operators below, U and $M = \overline{N_0} \setminus U$ denote the sets of unrouted and routed customer nodes, respectively.

1. **Related removal**: This operator was first proposed by Shaw (1998). The main principle is to successively remove a node having close relationship with another node selected from the set of unrouted nodes until achieving the predefined removal quantity. The relationship between nodes can be expressed in combinations of cost, demand, time window, etc. Traditionally, the operator follows the steps below:

- Step 1: randomly remove a node and put it into set U.
- Step 2: randomly select an unrouted node i in U and calculate the related values R_{ij} between node $i \in U$ and each routed node $j \in M$ in current solution. We calculate R_{ij} based on the normalized values of cost (l_{ij}) and the difference on the demand between i and j. The formula is expressed as follows:

$$R_{ij} = \varphi.\frac{l_{ij}}{l_{max}} + \psi.\frac{|q_i - q_j|}{q_{max}}. \tag{23}$$

where l_{max} is the maximum cost between any two nodes in the graph and $q_{max} = \max_{i \in N} q_i$ is the maximum demand of nodes.

- Step 3: sort R_{ij} in non-descending order and remove node j^* at the index of $\lfloor y^{c_1}.|M| \rfloor$, where y is a random number in $[0, 1)$ and c_1 is a randomness control parameter.
- Step 4: repeat Step 2 until n_r nodes are removed.

1. In addition, to reduce the running time of the algorithm, we perform an acceleration procedure as follows:

Acceleration procedure 2: For each node in the graph, we pre-calculate its related values with other nodes. Then for each node i, we sort the values of R_{ij} in non-descending order and save them in an additional data structure. When this operator is called, we skip calculating R_{ij} in *Step 2* and perform *Step* 3 by filtering out all R_{ij} values of nodes i and j in the current solution. Since R_{ij} values are already sorted, we only need to select node j^* as described in *Step 3*.

2. **Worst removal**: The basic principle of this operator is to remove a random number of worst nodes from the solution as described in Ropke and Pisinger (2006). We assume $f(S)$ and $f(S \setminus \{i\})$ are solution costs before and after removing node i, respectively. Denote Δ_i as the cost value of removing node i from the solution, the worst removal processes as follows:

- Step 1: calculate Δ_i for every node i in the set of routed nodes M.

$$\Delta_i = f(S) - f(S \setminus \{i\}) \tag{24}$$

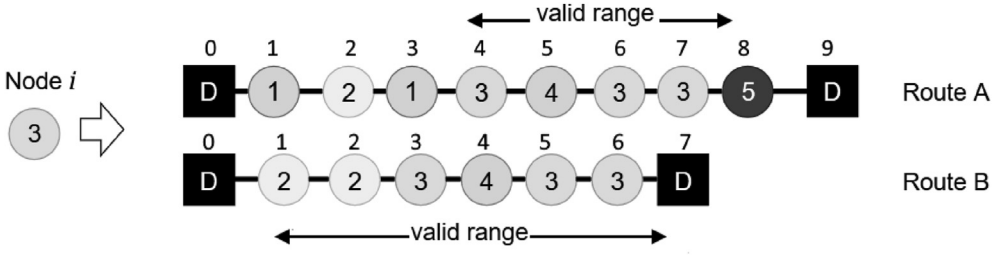

Fig. 2. Determining the valid insertion range.

- Step 2: sort Δ_i in non-increasing order, remove node i^* at index of $\lfloor y^{c_2}.|M| \rfloor$ and put i^* into U. Here, y is a random number in [0, 1), c_2 is a randomness control parameter.
- Step 3: repeat the process until n_r nodes are removed.

2. To speed up this operator, we use the following acceleration procedure:

Acceleration procedure 3: We calculate Δ_i values in *Step 1* for each node i in set M before the operator is started. After each iteration, a node is removed and we just need to update the values Δ for the nodes before and after i.

3. **Random removal**: This is the simplest operator. We randomly remove n_r nodes from current solution. This obviously helps to diversify the search.
4. **Route removal**: This operator is used in several articles such as Demir et al. (2012); Mancini (2016). Here we randomly choose a route and remove all of its nodes. The operator aims to redistribute customers between routes.
5. **Graph Cluster removal**: This operator is inspired by removal methods based on clusters of distance-proximity nodes. We first choose a random node in the solution as a seed node and remove r_2%. $|M|$ nearest routed nodes. This operator may help the proximity nodes to exchange their positions in the solution. Based on experiments, we decide to select $r_2 = 25\%$ leading to the best performance of the algorithm.
6. **Group removal**: This is our new removal operator whose purpose is to relocate nodes in a priority group among the routes of a solution. First, a group p is selected randomly, then we continuously remove a random node in this group (G_p) until $n_r = r_3\%.|G_p|$ nodes were removed. We set $r_3 = 20\%$ as the best value after tests. If $n_r < 10$, we set $n_r = |G_p|$.
7. **Joint Segment removal**: Each priority p in a route has a VIR as stated in Section 4.2. This operator aims to fix worse linking positions where VIRs connect and the positions where the depot connects with the first and the last customer nodes. For instance, reconsider route A in Subsection 1: *0-2-1-1-2-1-3-3-2-3-3-0* in case $d = 1$, the joints are *0-2, 1–3, 3–0*. After determining the joints, we remove a random number of successive nodes around both sides of a joint (including two nodes belonging to the joint). In the case that the depot belongs to a joint, we remove only the side where customer nodes are located. We choose the maximum removal ratio in a route, $r_4 = 30\%$ as the best value after tests.

4.4. Local search

Local search has been successfully integrated into the ALNS framework to solve several variants of the VRP, such as in studies of François et al. (2016); Hemmelmayr et al. (2012). Our experiments show that this integration can significantly improve the solution quality but also increase the running time, especially on large instances. Therefore, applying local search requires an appropriate strategy. In this study, we adapt three classical local search operators: 2-Opt, 2-Opt*, and CROSS presented in Chapter 5 of Toth and Vigo (2014) and apply them only in special conditions to ensure the ALNS performance (see Algorithm 1). Moreover, a move need to satisfy the constraints and the rules of this problem. For the d-relaxed priority rule, new location of a node with priority p needs to be located in a *VIR* of p. For the *ODF* rule, it is not necessary to check because if a move is considered valid, it only changes the locations of nodes in the solution. The mentioned operators are briefly described as follows:

- **2-Opt**: It is an intraroute operator, removing a pair of edges $(i, i+1)$ and $(j, j+1)$ and replacing them with a new one (i, j) and $(i+1, j+1)$. A 2-Opt move leads to a reversal in the node sequence from node $i+1$ to node j.
- **2-Opt***: This operator seeks an improvement by selecting a pair of edges $(i, i+1)$ and $(j, j+1)$ located in two different routes and replacing them by the new pair $(i, j+1)$ and $(j, i+1)$. This move does not change any sub-segment direction in the routes.
- **Cross**: The operator performs a move by exchanging two segments between two routes without changing any sub-segment direction in the routes. In the experiment, the maximum number of nodes in a segment is set to 4.

In Algorithm 1, we set LS_1 = {CROSS, 2-Opt*, 2-Opt}, LS_2 = {2-Opt*, 2-Opt}. It should be noted that CROSS is an effective but time-consuming operator. Hence, in the experiment, CROSS is used only for the new best solutions found during the search.

4.5. Adaptive selection mechanism

As described at the beginning of Section 4, the adaptive selection mechanism in ALNS uses the RWS mechanism combined with SA to assess the solution quality. The entire search process is divided into w_1 segments, with w_2 iterations in each segment. Given an operator $z \in Z$ in the current segment *seg* (Z can be the set of removal operators R or the set of insertion operators I), operator z is associated with variables representing its activity, named TS_z^{seg}, w_z^{seg}, n_z^{seg}, P_z^{seg} to record the aggregated scores, the weight, the number of times z is used, and the probability of choosing z, respectively. After each iteration, one of the scores σ_1 , σ_2, and σ_3 will be aggregated into TS_z^{seg} depending on the solution quality qualified by the SA. Here, σ_1 , σ_2, and σ_3 represent the score associated with the new best solution, the new better solution, and the new worst solution, respectively (see Algorithm 1 for more detail). All TS_z^{seg} values are reset to zero at the beginning of a segment. The *RWS* updates the weight of z in the next segment as follows:

$$w_z^{seg+1} = w_z^{seg}.(1-\rho) + \rho.\frac{TS_z^{seg}}{n_z^{seg}} \tag{25}$$

Parameter ρ ensures the balance between the weight of the current segment and the average score earned after the current segment. In the first segment, the probability of choosing z is the same for each operator. In the next segments, the probability of choosing z is based on the weight

```
1  begin
2      for i = 1 to w3 do
3          Generate a random integer y ∈ [y1, y2]
4          Perform y feasible random relocations in S_current
5          Improve S_current with set of local search LS2 to obtain S_new
6          if S_new is better than S_best then
7              S_best ←S_current ←S_new
8          else
9              S_current ←S_new
10         end
11     end
12 end
```

Table 1
Values used for the parameter tuning procedure.

Parameter	Tested values
φ	{*1*, **2**, 3, 4, 5}
ψ	{***1***, 2, 3, 4, 5}
c_1	{*1*, 3, 5, 7, ∞}
c_2	{*1*, 3, 5, 7, ∞}
η	{*0.05*, **0.1**, 0.15, 0.2, 0.25}
ρ	{*0.6*, 0.65, 0.7, **0.75**, 0.8}
σ_1	{*10*, 11, **12**, 13, 14}
σ_2	{*5*, 6, **7**, 8, 9 }
σ_3	{*1*, 2, 3, **4**, 5}

of z calculated in the current segment, according to equation (26).

$$\mathscr{P}_z^{seg+1} = \frac{w_z^{seg}}{\sum_{z=1}^{|Z|} w_z^{seg}} \tag{26}$$

4.6. Acceptance mechanism and stopping condition

The acceptance mechanism with SA is similar to Ropke and Pisinger (2006). In the search process, a new best solution replaces the best solution found so far and the current solution, a new better solution replaces the current solution, and a new worst solution is only accepted with the probability $\mathscr{P} = e^{-\frac{f(S_{new})-f(S_{current})}{T}}$. Denote by *iter* the current iteration, the temperature in the next iteration is $T_{iter+1} = c.T_{iter}$. The initial value of T is selected so that a solution with an objective value 5% worse than S_{init} is accepted with a probability 0.5. So, from the probability equation, we have the initial temperature as follows:

$$T_0 = -\frac{0.05}{\ln 0.5} f(S_{init}) \tag{27}$$

ALNS is stopped after a number of iterations, which is determined in preliminary tests.

4.7. Adaptations to solve CTSP-d

In this study, the ALNS algorithm is also adapted to solve the CTSP-*d*, a special case of the VRP-RPR. The Algorithm 1 is kept unchanged, except that we do not use the three operators that are useful only when there are multiple vehicles: route removal, regret insertion, and regret insertion with noise. In addition, we use a new procedure 2, which is activated if Algorithm 1 cannot find a new best solution after κ consecutive segments. This procedure helps diversifying the search and often provides better solutions. However, it is not applied to the VRP-RPR because from our observation, it is time-consuming while the benefit gained from the solution improvement, especially for large instances, is quite limited. Finally, we set $LS_1 = \{2\text{-Opt}\}$, $LS_2 = \{2\text{-Opt}\}$ because 2-Opt* and CROSS inter-route operators cannot be used in the CTSP-*d*.

Algorithm 2. Additional procedure applied for the experiment with CTSP-*d*.

Note. y_1 and y_2 are the lower and upper bounds on the number of relocations, respectively; w_3 is the number of iterations.

5. Experimental results

5.1. Instances and experimental settings

To evaluate the performance of the proposed approaches, we test them on two classes of instances as follows:

- CTSP-*d* instances: The first instance class is proposed by Hà et al. (2020) for the CTSP-*d*. From the TSPLIB data of Reinelt (1991), the authors in Hà et al. (2020) introduce instances of two types: R (random) and type C (clustered) indicating the distributions of priorities. The instances are labeled X-Y-p_{max}-*d* where X is the name of the original TSPLIB instances, Y represents the instance type (C or R), p_{max} is number of priority groups, and *d* is the relaxed value. Instances with prefix *berlin52* have one more letter (*a*, *b*, or *c*) at the end of their name. As showed in Hà et al. (2020), each letter denotes a random instance generated from the original instance *berlin52* in TSPLIB.
- VRP-RPR instances: Because there is no available instances for the VRP-RPR, we generate a data set based on the CTSP-*d* instances of Hà et al. (2020). The customer demands are extracted from corresponding instances in Fischetti et al. (1998). We use a heterogeneous fleet of 3 vehicle types: Type 1 ($C_1 = 500, L_1 = 5000$), Type 2 ($C_2 = 300, L_2 = 6000$), and Type 3 ($C_3 = 200, L_3 = 7000$). The vehicle fleet is created so that it cannot always serve all the customers, which would make the instances more difficult to solve. In this study, we consider that minimizing lost demand is more important than minimizing the transportation cost. Therefore, the value α is selected so that the weight of demand lost is sufficiently large compared to that of transportation cost. If the value of the summation $\sum_{k=1}^{k_{max}} L_k$ has m digits, we choose $\alpha = 1 - 1/10^{(m-3)}$. For example, considering an instance with 16 vehicles and $\sum_{k=1}^{16} L_k = 78{,}000$, the number of digits of the summation is $m = 5$, leading to $\alpha = 0.99$.

Table 3
Comparison between ALNS and GILS-RVND on the CTSP-*d* instances.

Criterion	Type R	Type C	Both types
Average cost	32/14/12	35/4/19	67/18/31
Best cost	13/3/42	10/1/47	23/4/89
Running time	57/1/0	58/0/0	115/1/0

Table 2
Impact of main operators on the performance of the ALNS algorithm.

Row	State	VRP-RPR				CTSP-*d*			
		d=0	*d=1*	*d=2*	*Avg*	*d=0*	*d=1*	*d=2*	*Avg*
1	All ON	0%	0%	0%	0%	0%	0%	0%	0%
2	Worst removal OFF	1.36%	0.12%	0.13%	**0.54%**	−0.32%	−0.64%	0.48%	−0.16%
3	Related removal OFF	−0.50%	−0.13%	0.06%	−0.19%	−0.16%	0.43%	0.69%	**0.32%**
4	Group removal OFF	0.16%	0.61%	1.72%	**0.83%**	0.36%	0.88%	0.83%	**0.69%**
5	Joint Segment removal OFF	0.60%	0.44%	0.81%	**0.62%**	0.72%	0.91%	1.25%	**0.96%**
6	Random removal OFF	0.96%	0.20%	1.17%	**0.78%**	0.92%	0.73%	0.78%	**0.81%**
7	Graph Cluster removal OFF	0.09%	0.31%	−0.04%	**0.12%**	0.21%	0.84%	0.90%	**0.65%**
8	Route removal OFF	0.49%	0.27%	−0.11%	**0.22%**				
9	Cheapest insertion OFF	−0.15%	0.13%	−0.01%	−0.01%	0.15%	0.22%	0.08%	**0.15%**
10	Cheapest insertion with noise OFF	−0.17%	−0.05%	0.09%	−0.04%	0.11%	0.19%	0.39%	**0.23%**
11	Regret insertion OFF	−0.30%	0.55%	0.05%	**0.10%**				
12	Regret insertion with noise OFF	0.53%	−0.28%	1.12%	**0.46%**				
13	Local search OFF	0.26%	1.39%	2.41%	**1.35%**	0.77%	1.93%	2.34%	**1.68%**
14	Acceleration OFF	25.20%	31.27%	33.60%	**30.02%**	35.22%	41.79%	49.98%	**42.33%**

Table 4
Results for the CTSP-*d* instances of type R.

Instance	MILP		GILS-RVND			ALNS		
	Sol	Gap	Aver	Best	Time	Aver	Best	Time
berlin52R-1-0-a	7542*	0.00	7542.0	7542*	7.7	7542.0	7542*	7.9
berlin52R-3-0-a	12765*	0.00	12765.0	12765*	6.3	12765.0	12765*	5.2
berlin52R-3-0-b	12668*	0.00	12668.0	12668*	6.5	12668.0	12668*	5.0
berlin52R-3-0-c	12483*	0.00	12483.0	12483*	6.4	12483.0	12483*	5.0
berlin52R-3-1-a	9473*	0.00	9473.0	9473*	8.3	9473.0	9473*	6.0
berlin52R-3-1-b	9442	2.39	9419.0	9419	7.0	9419.0	9419	6.2
berlin52R-3-1-c	9577	3.75	9577.0	9577	8.3	9577.0	9577	6.0
berlin52R-5-0-a	16414*	0.00	16414.0	16414*	5.7	16414.0	16414*	5.0
berlin52R-5-0-b	13759*	0.00	13759.0	13759*	5.8	13759.0	13759*	5.0
berlin52R-5-0-c	14131*	0.00	14131.0	14131*	6.5	14131.0	14131*	5.1
berlin52R-5-1-a	12417	14.53	11651.0	11651	6.9	11704.2	11651	6.3
berlin52R-5-1-b	9982	6.60	9963.5	9957	6.2	9963.7	9957	6.0
berlin52R-5-1-c	11020	9.32	10940.0	10940	6.8	10940.0	10940	6.0
berlin52R-5-3-a	9085	6.97	9020.0	9012	8.5	9028.5	9012	6.9
berlin52R-5-3-b	8036*	0.00	8036.0	8036*	7.3	8045.0	8036*	6.7
berlin52R-5-3-c	8224	2.02	8224.0	8224	7.4	8224.0	8224	6.5
kroA100-R-1-0	21282*	0.00	21365.5	21282*	95.2	21282.0	21282*	23.1
kroA100-R-3-0	38814*	0.00	38877.5	38814*	73.4	38827.9	38814*	19.9
kroA100-R-3-1	30072	16.61	29578.2	29264	92.6	29264.0	29264	23.1
kroA100-R-5-0	50192	2.23	50411.0	50192	63.0	50265.1	50192	18.5
kroA100-R-5-1	39335	21.60	36328.4	35847	73.0	35983.1	35847	22.6
kroA100-R-5-3	28548	22.34	25594.9	25370	91.0	25395.1	25370	25.7
kroB100-R-1-0	22141	1.59	22189.1	22141	83.6	22141.0	22141	23.5
kroB100-R-3-0	37706*	0.00	37770.2	37706*	69.0	37761.8	37706*	19.7
kroB100-R-3-1	31216	17.71	28652.6	28509	85.8	28454.0	**28454**	23.1
kroB100-R-5-0	50781	0.47	50863.4	50781	57.8	51290.4	50964	19.1
kroB100-R-5-1	39646	21.67	35589.3	35209	81.0	35244.9	35209	22.3
kroB100-R-5-3	28124	18.36	26205.0	26069	85.4	26316.9	26157	25.6
kroC100-R-1-0	20749	0.98	20826.1	20749	94.6	20749.0	20749	23.0
kroC100-R-3-0	37953*	0.00	38083.0	37953*	86.2	38096.8	37953*	19.8
kroC100-R-3-1	28218	12.69	28304.5	28130	86.8	28247.6	28130	23.6
kroC100-R-5-0	50085*	0.00	50099.9	50085*	69.1	50089.0	50085*	18.3
kroC100-R-5-1	39002	24.96	34365.1	33594	73.7	33759.0	33594	23.1
kroC100-R-5-3	28758	20.57	25587.6	25458	94.8	25530.3	25458	27.1
kroD100-R-1-0	21338	2.91	21336.6	21294	101.5	21294.0	21294	22.7
kroD100-R-3-0	38342	2.07	38290.3	38110	72.4	38143.2	38110	20.0
kroD100-R-3-1	28498	15.30	27886.3	27734	89.5	27755.8	27734	23.8
kroD100-R-5-0	49100*	0.00	49222.8	49100*	67.1	49100.0	49100*	20.0
kroD100-R-5-1	45094	34.19	34414.9	34246	71.6	34618.3	34246	23.2
kroD100-R-5-3	27468	18.33	25733.4	25624	86.4	26022.4	25624	30.9
kroE100-R-1-0	22068*	0.00	22129.1	22068*	100.1	22069.1	22068*	26.6
kroE100-R-3-0	37935*	0.00	37996.0	37935*	67.8	37935.0	37935*	21.1
kroE100-R-3-1	29863	10.94	29489.2	29359	90.9	29380.0	29359	26.7
kroE100-R-5-0	54197*	0.00	54285.2	54197*	66.4	54317.4	54197*	19.0
kroE100-R-5-1	52173	35.39	38700.8	38359	76.6	38916.6	38359	25.0
kroE100-R-5-3	30260	20.88	27316.3	27256	109.9	27391.6	27324	29.2
kroA200-R-1-0	30269	5.64	30176.5	29853	656.1	29369.4	**29368**	178.8
kroA200-R-3-0	52050	2.29	52237.3	51741	473.4	51754.3	**51539**	123.5
kroA200-R-3-1	43796	23.88	38471.5	38208	574.1	37983.5	**37750**	154.9
kroA200-R-5-0	67027	1.54	67757.6	67096	435.6	67123.6	**67027**	111.1
kroA200-R-5-1	–	–	48790.1	48020	537.1	48835.9	**47912**	158.2
kroA200-R-5-3	47773	37.66	34630.4	34195	495.1	34414.9	**33994**	176.1
kroB200-R-1-0	29902	3.02	30149.7	29849	697.4	29439.9	**29438**	179.3
kroB200-R-3-0	53771	2.43	54131.3	53739	525.5	53652.7	**53597**	122.8
kroB200-R-3-1	58382	42.11	39190.5	38943	606.3	38626.2	**38522**	158.6
kroB200-R-5-0	73666	5.04	73011.3	72786	469.8	72498.0	**72390**	118.3
kroB200-R-5-1	–	–	50821.8	50260	525.9	50880.5	**49944**	150.3
kroB200-R-5-3	47241	34.40	37145.9	36926	652.3	36971.0	**36664**	160.4
Average			30001.2	29829.3	158.9	29885.0	29774.4	44.4

The experiments are performed on a computer with CPU of Intel core (TM) i3-6100 3.7 GHz, 16 GB DDR3, and Microsoft Windows 10. ALNS is programmed with Java 10 and CPLEX 12.9.1 is used to solve the MILP model. The instances and the detailed results of our experiments can be found at http://orlab.com.vn/home/download.

5.2. Parameter settings

5.2.1. Cooling rate and stopping condition

We stop ALNS after 40,000 iterations ($w_1 = 200, w_2 = 200$) as a fair trade-off between solution quality and computational time. For cooling rate c, we do not use the same value for all instances as Ropke and Pisinger (2006) because preliminary tests show that with a constant number of iterations, using a fixed value of c may lead to the fact that on some instances, acceptance probability $\mathscr{P}$ in SA converges to 0 too early and therefore, the remaining iterations of ALNS are less effective. To deal with this issue, we pre-set an average acceptance probability for all instances at the last iteration of ALNS and compute the cooling rate c for each instance such that probability curves decrease towards the pre-defined value. As a result, the acceptance probability on each instance is forced to converge around the same point when ALNS ends. For more details, let denote *last* the last iteration, the acceptance

Table 5
Results for the CTSP-*d* instances of type C.

Instance	MILP		GILS-RVND			ALNS		
	Obj	Gap	Aver	Best	Time	Aver	Best	Time
berlin52C-1-0-a	7542*	0.00	7542.0	7542*	7.9	7542.0	7542*	5.1
berlin52C-3-0-a	8144*	0.00	8144.0	8144*	7.6	8144.0	8144*	5.3
berlin52C-3-0-b	8016*	0.00	8016.0	8016*	6.0	8016.0	8016*	5.0
berlin52C-3-0-c	9085*	0.00	9091.4	9085*	6.8	9085.0	9085*	5.0
berlin52C-3-1-a	7952*	0.00	7952.0	7952*	8.6	7952.0	7952*	6.0
berlin52C-3-1-b	7596*	0.00	7596.0	7596*	7.2	7596.0	7596*	6.0
berlin52C-3-1-c	7984*	0.00	7984.0	7984*	8.2	7984.0	7984*	6.0
berlin52C-5-0-a	9430*	0.00	9430.0	9430*	5.9	9430.0	9430*	5.0
berlin52C-5-0-b	8669*	0.00	8669.0	8669*	5.1	8669.0	8669*	5.1
berlin52C-5-0-c	9651	5.97	9651.0	9651	5.7	9651.0	9651	5.1
berlin52C-5-1-a	8811*	0.00	8819.8	8811*	7.2	8811.0	8811*	6.0
berlin52C-5-1-b	7948*	0.00	7948.0	7948*	6.5	7948.0	7948*	6.0
berlin52C-5-1-c	8509*	0.00	8509.0	8509*	7.0	8583.0	8583	6.0
berlin52C-5-3-a	7907	1.08	7945.0	7907	8.8	7929.1	7907	6.9
berlin52C-5-3-b	7614*	0.00	7614.0	7614*	7.1	7614.0	7614*	6.4
berlin52C-5-3-c	7631*	0.00	7631.0	7631*	8.1	7631.0	7631*	7.0
kroA100-C-1-0	21282*	0.00	21338.6	21282*	84.9	21282.0	21282*	23.1
kroA100-C-3-0	24049	1.03	24049.0	24049	67.9	24049.0	24049	20.0
kroA100-C-3-1	23392	5.89	23416.7	23069	80.1	22845.0	**22845**	22.0
kroA100-C-5-0	24745	7.67	24745.0	24745	58.5	24745.0	24745	18.0
kroA100-C-5-1	22617	4.73	22591.8	22589	84.4	22589.0	22589	21.0
kroA100-C-5-3	21443	0.25	21443.0	21443	82.5	21443.0	21443	27.7
kroB100-C-1-0	22141	1.61	22235.0	22179	83.1	22141.5	**22141**	23.2
kroB100-C-3-0	24887*	0.00	24971.3	24887*	63.5	24887.0	24887*	20.0
kroB100-C-3-1	22141*	0.00	22155.1	22141*	77.2	22208.2	22141*	22.5
kroB100-C-5-0	24793*	0.00	24794.0	24794	57.9	24794.0	24794	20.3
kroB100-C-5-1	23159*	0.00	23173.1	23159*	73.4	23159.0	23159*	23.3
kroB100-C-5-3	22179	1.85	22180.0	22141	79.7	22141.0	22141	28.1
kroC100-C-1-0	20749	0.99	20786.9	20749	94.9	20749.0	20749	23.0
kroC100-C-3-0	21340*	0.00	21440.6	21340*	58.0	21340.0	21340*	20.0
kroC100-C-3-1	20910*	0.00	20910.0	20910*	92.5	20910.0	20910*	26.1
kroC100-C-5-0	24040*	0.00	24040.0	24040*	58.3	24040.0	24040*	20.6
kroC100-C-5-1	22827	1.52	22827.0	22827	71.8	22827.0	22827	22.0
kroC100-C-5-3	21931	5.53	21344.9	21278	91.3	21292.3	21278	27.5
kroD100-C-1-0	21309	2.54	21298.5	21294	93.4	21294.0	21294	22.8
kroD100-C-3-0	23809	3.16	23833.3	23809	65.3	23809.0	23809	19.0
kroD100-C-3-1	21944*	0.00	22268.9	21944*	87.5	21944.0	21944*	22.0
kroD100-C-5-0	28297	8.33	28234.9	28228	56.3	28228.0	28228	19.0
kroD100-C-5-1	25324	6.96	25250.8	25102	72.7	25131.3	25102	21.1
kroD100-C-5-3	21759	4.18	21747.0	21744	94.9	21746.5	21744	26.9
kroE100-C-1-0	22068*	0.00	22146.9	22068*	89.8	22068.0	22068*	23.0
kroE100-C-3-0	24383*	0.00	24405.4	24383*	66.5	24416.6	24383*	19.9
kroE100-C-3-1	22121*	0.00	22125.0	22121*	76.3	22122.0	22121*	23.0
kroE100-C-5-0	26440	0.98	26443.3	26440	57.5	26440.0	26440	20.0
kroE100-C-5-1	23611*	0.00	23658.1	23611*	62.9	23611.0	23611*	23.6
kroE100-C-5-3	22455	1.61	22560.6	22455	76.5	22504.2	22455	27.0
kroA200-C-1-0	30162	5.26	30042.8	29737	641.0	29368.0	**29368**	163.0
kroA200-C-3-0	30062	2.14	30090.0	29913	607.2	29929.4	29913	135.2
kroA200-C-3-1	29481	2.13	29862.4	29435	612.4	29372.4	**29368**	148.4
kroA200-C-5-0	32382	2.46	32273.3	32224	404.4	32226.4	32224	128.8
kroA200-C-5-1	32342	6.70	31221.3	31069	467.6	31060.9	**31057**	134.6
kroA200-C-5-3	30039	2.25	30964.0	30686	565.1	30258.6	**30023**	152.8
kroB200-C-1-0	29945	3.17	30064.7	29790	628.3	29442.7	**29438**	161.3
kroB200-C-3-0	31285	2.25	31276.1	30989	565.5	31023.4	30989	120.5
kroB200-C-3-1	30585	1.81	30830.7	30457	539.3	30554.8	**30442**	131.8
kroB200-C-5-0	41009	17.19	37973.2	37909	401.5	37910.9	37909	121.1
kroB200-C-5-1	33775	10.73	33438.3	33276	540.8	33568.7	**33218**	134.0
kroB200-C-5-3	30302	2.84	30492.0	30270	566.0	30175.8	**30028**	161.8
Average			20749.8	20673.5	153.7	20659.2	20639.6	42.4

probability at the last iteration is:

$$\mathscr{P}_{last} = e^{-\frac{f\left(S_{new}^{last}\right) - f\left(S_{current}^{last}\right)}{T_{last}}}$$

From this equation, we first find the average difference $\overline{\theta}$ between the objective value of the new solution $f(S_{new}^{last})$ and that of the current solution $f(S_{current}^{last})$ then we pre-set the average acceptance probability $\mathscr{P}_{last} = 0.05$. In SA, $T_{last} = T_0.c^{(w_1.w_2-1)}$, so we can calculate c as follows:

$$c = \left(-\frac{\overline{\theta}}{T_0.\ln 0.05} \right)^{\frac{1}{w_1.w_2-1}} \tag{28}$$

To estimate $\overline{\theta}$, a preliminary experiment on all VRP-RPR instances is conducted. We perform 5 ALNS runs on each instance to find the average of $f(S_{new}^{last}) - f(S_{current}^{last})$. This average value is collected on every instance then $\overline{\theta}$ is set to the average of all the values. It should be noted that the preliminary experiment requires an initial value of c, we set $c = 0.99975$ as in Ropke and Pisinger (2006).

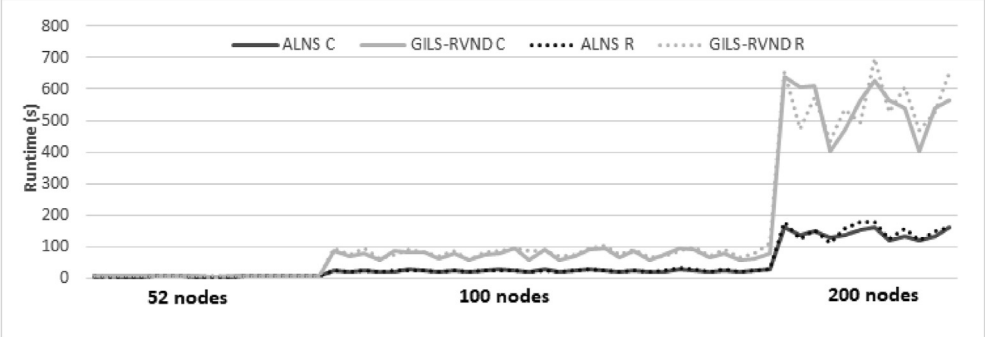

Fig. 3. Speed comparison between ALNS and GILS-RVND for the CTSP-*d*.

5.2.2. Other parameters

To tune parameters $(\varphi, \psi, c_1, c_2, \eta, \rho, \sigma_1, \sigma_2, \sigma_3)$, we use the strategy proposed in Ropke and Pisinger (2006) to find the most appropriate values. Namely, we start from an acceptable vector found by an ad hoc trial-and-error approach and improve it by allowing one parameter to take several values while keeping the rest fixed. After a parameter is tuned, we move on to the next parameter, using the values found so far and the values from the initial tuning for the parameters that have not been considered yet. This process continues until all parameters have been tuned. The procedure is performed on 6 random VRP-RPR instances (3 of type R and 3 of type C) with 100 nodes: kroA100-R-3-*d*, kroC100-R-3-*d*, kroD100-R-3-*d*, kroB100-C-3-*d*, kroD100-C-3-*d*, and kroE100-C-3-*d* ($d = 0, 1, 2$).

In Table 1, we present the parameter values used for the tuning procedure. The italic values form the initial tuning parameter vector. The final selected parameters are highlighted in bold $(\varphi, \psi, c_1, c_2, \eta, \rho, \sigma_1, \sigma_2, \sigma_3) = (2, 1, \infty, \infty, 0.1, 0.75, 12, 7, 4)$.

5.3. Selecting ALNS components

We use the instances randomly selected in Subsection 5.2.2 to analyse the impact of the ALNS components to select the appropriate operators for the experiments in next sections. The tested components include the insertion operators, the removal operators, the acceleration procedures, and the local search operators. In this experiment, we first allow to use all the components of the algorithm, then turn off each component one by one, while keeping the others to investigate its performance. Both instance classes are selected for the experiment. The idea is to compute the gap between the average result over 10 runs on each instance with a reference value, which is the objective value of the best found solution over 10 runs when all components are used. In Table 2, the first row shows the average gaps when all components are in "On" state. Each row from 2 to 13 shows the average gaps when the corresponding component is "Off". In the last row, we report the average gap in percentage of running time when the acceleration procedures are deactivated. From row 2 to row 14, a positive gap indicates that the performance of the algorithm becomes worse if the corresponding component is off and vice versa. In Table 2, the empty rows in column "CTSP-*d*" mean that the corresponding components are not applicable. Components with a positive average gap in column "*Avg*" will be selected to be used in our metaheuristic. In more details, on VRP-RPR instances, we decide to turn off the related removal, cheapest insertion, and cheapest insertion with noise, while on CTSP-*d* instances, only the worst removal is not used.

Results from Table 2 demonstrate the good performance of our

Table 6
Results of MILP and ALNS on the VRP-RPR instances.

Instance	Fleet	$d = 0$			$d = 1$			$d = 2$		
		Gap(%)	CPLEX	ALNS	Gap(%)	CPLEX	ALNS	Gap(%)	CPLEX	ALNS
kroA21-R-3	1x1000x13000	0	546.30[a]	546.30[a]	0	143.44[a]	143.44[a]	0	108.91[a]	108.91[a]
kroB21-R-3	2x400x10000	0	347.88[a]	347.88[a]	0	323.20[a]	323.20[a]	0	322.16[a]	322.16[a]
kroC21-R-3	1x400x8000; 1x300x8000; 1x200x10000	0	261.36[a]	261.36[a]	11.55	249.96	**245.96**	10.24	248.33	**245.96**
kroD21-R-3	3x300x8000; 1x200x10000	11.81	330.82	330.82	30.77	328.34	**327.30**	31.31	320.78	320.78
kroE21-R-3	1x300x8000; 2x200x7000; 2x100x6000	0	364.31[a]	364.31[a]	0	362.64[a]	362.64[a]	10.02	362.64	362.64
kroA31-R-3	1x1500x16000	0	603.90[a]	603.90[a]	0	210.09[a]	210.09[a]	0	135.02[a]	135.02[a]
kroB31-R-3	2x500x15000	0	560.41[a]	560.41[a]	8.89	546.00	**543.78**	7.99	544.82	544.82
kroC31-R-3	1x500x10000; 1x400x8000; 1x300x6000	6.88	458.51	**451.31**	11.91	432.82	**431.30**	11.91	431.30	431.30
kroD31-R-3	2x500x10000; 2x400x12000	20.72	290.15	**285.99**	48.59	268.49	**254.88**	57.65	296.00	**254.38**
kroE31-R-3	3x500x10000; 1x400x12000; 1x300x8000	20.04	274.02	**272.98**	58.80	336.97	**252.46**	51.91	273.92	**242.60**
Average		5.95	403.77	402.53	17.05	320.20	309.51	18.10	304.39	296.86
kroA21-C-3	1x1000x13000	0	108.91[a]	108.91[a]	0	108.91[a]	108.91[a]	0	108.91[a]	108.91[a]
kroB21-C-3	2x400x10000	0	319.35[a]	319.35[a]	12.82	316.62	**315.20**	11.81	315.86	**315.20**
kroC21-C-3	1x400x8000; 1x300x8000; 1x200x10000	0	248.92[a]	248.92[a]	4.77	247.60	247.60	10.89	247.60	247.60
kroD21-C-3	3x300x8000; 1x200x10000	28.92	327.01	327.01	38.26	331.22	**326.48**	40.12	331.14	**320.78**
kroE21-C-3	1x300x8000; 2x200x7000; 2x100x6000	0	352.18[a]	352.18[a]	3.63	352.18	352.18	3.81	352.18	352.18
kroA31-C-3	1x1500x16000	0	154.52[a]	154.52[a]	0	149.92[a]	*151.12*	0	135.02[a]	135.02[a]
kroB31-C-3	2x500x15000	5.68	547.52	547.52	7.91	541.55	**538.76**	8.02	539.33	**538.76**
kroC31-C-3	1x500x10000; 1x400x8000; 1x300x6000	15.72	454.01	**439.46**	15.10	444.48	**438.87**	16.83	456.73	**438.87**
kroD31-C-3	2x500x10000; 2x400x12000	39.10	259.68	**256.99**	51.67	276.63	**256.52**	52.45	269.84	**254.38**
kroE31-C-3	3x500x10000; 1x400x12000; 1x300x8000	42.94	260.02	**246.50**	49.73	259.47	**242.22**	51.48	265.29	**242.22**
Average		13.24	303.21	300.14	18.39	302.86	297.79	19.54	302.19	295.39

[a] Optimal valueColumn "Fleet" presents the characteristics of the vehicle fleet, which includes one or several types, and is showed in the form "number of vehicles × capacity × maximum route length".

problem-tailored components. The new removal operators (Group removal and Joint Segment removal) work well on both problems. In particular, among removal operators, the group removal is the best operator on the VRP-RPR instances while the joint segment removal is the best when being used to solve the CTSP-*d* instances. The local search operators considerably improve the solution quality, especially when the value of *d* is large. This is because when the problems are more relaxed, the VIRs are wider and it is easier for local searches to find better solutions. Finally, the acceleration procedures are clearly efficient, reducing up to nearly 50% of computational time on average.

5.4. Results and discussion

In this section, we report the obtained results of the ALNS with the parameter setting and the corresponding set of components selected in Subsection 5.3 on both classes of instances: CTSP-*d* and VRP-RPR. On each instance, the algorithm is run 10 times. Then, the objective value of the best solution and the average objective value on the 10 solutions are recorded.

5.4.1. Tests on the CTSP-d instances

We now investigate the performance of our metaheuristic by comparing it with the MILP model and the GILS-RVND algorithm reported by Hà et al. (2020) on the existing CTP-*d* instances. It is worth mentioning that GILS-RVND is run on a core TM i7-6700 3.40 GHz CPU, which is 83.2% faster than our CPU presented in Subsection 5.1 (the benchmark is referenced from http://www.cpuboss.com). Because the conversion of running times depends on multiple factors and is not easy to estimate exactly, we decide to present raw running times of both algorithms. For each instance, ALNS is run 10 times and three comparison criteria (objective value of the best solution, average objective value of 10 solutions found in 10 runs, and average running times in 10 runs) are recorded for the comparison. A brief comparison between ALNS and GILS-RVND over 116 test cases is reported in Table 3 while the detailed results for three approaches are shown in Tables 4 and 5.

In Table 3, the results are represented in the form x/y where x is the number of instances on which ALNS finds better solutions, y is the number of instances on which GILS-RVND is better, and z is the number of instances where both algorithms find equal solutions. The last column is the summation of the previous two columns. The obtained results show that on both instance types, ALNS clearly outperforms GILS-RVND on all comparison criteria. It is more stable and finds more better solutions in less running time. More remarkably, the ALNS finds 23 new best known solutions in overall.

In the following Tables 4 and 5, for the MILP model, we show the objective values of the solutions found (column "Sol") and gaps in percentage (column "Gap") returned by CPLEX. Because the running times of the MILP model are not reported for medium and large instances in Hà et al. (2020), we do not show CPLEX running times in the tables. In column "Sol", the optimal solutions are marked with "*", and out-of-memory cases are marked with "-". For GILS-RVND and ALNS, we report the average objective value of solutions (columns "Aver"), the objective value of the best solutions (columns "Best"), and the average running time (in seconds) over 10 runs (columns "Time"). We also highlight in bold the new best known solutions found by ALNS, and show the averaged results in the bottom row of each table to facilitate the comparison. As can be seen from the result tables, the ALNS can reach 35/37 the optimal solutions provided by the MILP model. For open instances, its best solutions are worse than MILP formulation on only two instances and better on 69 instances. Between ALNS and GILS-RVND, the ALNS can provide higher quality solutions in terms of both best and average costs. Especially, ALNS shows its advantage over GILS-RVND on the large instances of both types since it improves the best known solutions for 20 out of 24 instances with 200 nodes in much less running times. This demonstrates the ability of the ALNS algorithm to solve practical-size problems.

Fig. 3 compares more precisely the speed of the two metaheuristics. Letters R or C at the end of algorithm name indicate the instance type on which the algorithms are performed. Although ALNS operates on the slower CPU, it is still faster than GILS-RVND, especially on the large instances with 200 nodes.

We also investigate the impact of *d* on the speed of both algorithms and see that the average running times increase when the value of *d* increases. In ALNS, the main reason is that relaxing *d* leads to the increase of local search activities. For example, on the instances with 5 priority groups, compared to $d = 0$, the average number of 2-Opt moves increases by about 20.2% when $d = 1$ and 50.3% when $d = 3$. The instance type also has influence on the behavior of the ALNS. We observe that ALNS can find 25 optimal solutions on type C while on type R, it reaches optimality only on 18 cases. Moreover, on average, the gap between the average cost and the best cost of type R is also higher than that of type C. These results are consistent with the claim of Hà et al. (2020) that type C tends to be easier to solve than type R.

5.4.2. Tests on the VRP-RPR instances

In this experiment, we aim to test the performance of the MILP formulation and the ALNS on the VRP-RPR instances. Because the MILP formulation with CPLEX cannot handle medium and large instances, we create small-size instances by extracting the 20 and 30 first nodes from each CTSP-*d* instance with 100 nodes and 3 priority groups.

In Table 6, we present results for the small instances with different fleet settings and values of *d*. The results obtained by CPLEX after the time limit of 3600 s, the best solutions of ALNS found after 10 runs, and CPLEX gaps are reported in columns "CPLEX", "ALNS", and "Gap(%)", respectively for each value of *d*. $\alpha = 0.99$ is set for all instances. Because the speed of ALNS on all instances is quite fast (never exceeding 10 s), we do not report its running time. The experiment shows that with different constraints and rules imposed on vehicles, even on small-size instances, the VRP-RPR is still very hard to solve with CPLEX. Only 22 optimal values over 60 test cases are found, mostly on the instances with small fleet sizes and $d = 0$.

For the CTSP-*d*, the instances of type R are more difficult to solve to optimality than those of type C (see Hà et al. (2020)). However, the behavior is not the same for the VRP-RPR. CPLEX can solve 13 VRP-RPR instances of type R to optimality, while only 9 VRP-RPR instances of type C are successfully solved. The gaps returned by CPLEX on instances of type R are often better than those on type C. We guess additional constraints of VRP-RPR are the source of this behavior. Another observation is that the larger the values of fleet size and *d*, the more difficult the VRP-RPR instances. CPLEX gaps tend to be higher as the fleet size or *d* increases.

The results from Table 6 clearly show the performance of our ALNS. It provides solutions at least as good as those of CPLEX on all instances, except one (instance KroA31-C-3-1) in a much shorter running time. Moreover, it finds 27 better solutions (numbers highlighted in bold).

Because CPLEX cannot efficiently handle the large instances in an acceptable running time, we only provide results of the ALNS as reference for future studies in Tables A1 and A2 of the Appendix. The results show that on average, the total demand satisfied in the best solutions of type-C instances is about 1.01% higher than that in the best solutions of type-R instances while the total cost is about 2.41% lower. This phenomenon is predictable because in the instances of type C, the distances between two customers of the same group are closer, possibly leading to shorter routes which could be easier to satisfy the maximum route length constraints. Another observation is that, when the value of *d* increases, the constraints are more relaxed, and therefore we have more chance to satisfy the demand (see Columns "AvgDm" or "BSolDm"). Our algorithm can solve the large instances with up to 200 customers in acceptable running time, never bypassing 2 minutes except kroA200-R-1-0. On the tested instances, the running time averaged on all instances of type R is about 8.9% higher than that of type C. The clustered instances tend to be easier to solve for the ALNS algorithm. And finally, the results show that in several cases, not all vehicles are mobilized. Undispatched vehicles are in

the type with smallest capacity because using large-capacity vehicles is in general cheaper in terms of transportation cost. This is an advantage of our model when it allows flexible dispatch.

6. Conclusion

In this article, we study the VRP-RPR with applications in situations where the trade-off between the demand urgency and the operational cost must be taken into account. We propose a MILP model with improved constraints that allows heterogeneous fleet and flexible dispatch. Our model is general and can easily be transformed into other important VRP variants such as TOP, VRPP, VRPPFCC, and CPTP, etc. We also develop a metaheuristic based on the ALNS principle with problem-tailored components: removal operators, local search, and acceleration procedures. To further reduce the running time of the algorithm, we design repair and local search operators such that verifying the satisfaction of insertion operations w. r.t the rules (the *ODF* and the *d*-relaxed priority) is done in $\mathcal{O}(1)$. We investigate the performance of our ALNS algorithm on existing CTSP-*d* instances and the obtained results show that ALNS outperforms the GILS-RVND algorithm in the literature in terms of both criteria: running time and solution quality. For the VRP-RPR, ALNS can provide better solutions than the MILP-based exact method on the instances that CPLEX could not solve to optimaliy while the running time is more reasonable. In addition, the results of ALNS on large instances are also provided as reference for next metaheuristics. Future research directions include improving the exact method by adding efficient valid inequalities in a branch-and-cut framework to solve larger instances. Developing efficient solution approaches for the problem with "global timing" rule is also an interesting topic to follow.

APPENDIX

Tables A1 and A2 present results of ALNS on the VRP-RPR instances. The algorithm is run 10 times on each instance. In the tables, column "α" shows value of α in objective function; column "F_i" ($i = 1, 2, 3$) shows the number of vehicles of type i used in the best solution over the total number of vehicles available in the instance; columns "AvgDm" and "AvgCost" show the average demand and the average transportation cost over 10 runs; columns "BSolDm" and "BSolCost" report the total demand and the transportation cost of the best solution, respectively. Finally, column "Cust/Total" shows the total number of customers served in the best solution over the total number of customers in the instance; and column "Running time (s)" shows the average running times over 10 runs in seconds.

Table A1
Results of ALNS on the large VRP-RPR instances of type R

Instance	α	F_1	F_2	F_3	AvgDm	AvgCost	BSolDm	BSolCost	Cust/Total	Running time (s)
kroA100-R-1-0	0.99	5/5	4/4	7/7	5,038.0	65,665.4	5,038	63,972	99/99	15.9
kroB100-R-1-0	0.99	5/5	4/4	4/4	4,274.5	74,114.3	4,283	74,676	98/99	15.3
kroC100-R-1-0	0.99	6/6	5/5	4/4	5,280.8	68,802.9	5,286	68,513	94/99	17.0
kroD100-R-1-0	0.99	3/3	2/2	8/8	3,684.4	62,002.8	3,691	62,712	76/99	21.4
kroE100-R-1-0	0.99	3/3	6/6	6/6	4,159.0	86,344.0	4,159	86,231	93/99	23.1
kroA100-R-3-0	0.99	5/5	3/3	4/4	4,154.4	58,956.7	4,174	59,059	87/99	12.5
kroA100-R-3-1	0.99	5/5	3/3	4/4	4,177.9	54,098.0	4,188	55,294	86/99	13.5
kroA100-R-3-2	0.99	5/5	3/3	4/4	4,177.1	53,236.0	4,190	53,153	85/99	12.7
kroB100-R-3-0	0.99	4/4	4/4	5/5	3,783.1	74,492.8	3,792	74,011	91/99	13.0
kroB100-R-3-1	0.99	4/4	4/4	5/5	4,020.4	74,159.4	4,048	75,273	93/99	13.3
kroB100-R-3-2	0.99	4/4	4/4	5/5	4,154.1	74,325.8	4,180	75,026	96/99	12.6
kroC100-R-3-0	0.99	6/6	5/5	5/5	5,348.6	80,398.5	5,379	82,062	98/99	16.6
kroC100-R-3-1	0.99	6/6	5/5	5/5	5,412.0	74,692.6	5,412	73,404	99/99	13.9
kroC100-R-3-2	0.99	6/6	5/5	5/5	5,412.0	73,174.6	5,412	71,328	99/99	12.6
kroD100-R-3-0	0.99	4/4	5/5	6/6	4,581.2	81,020.6	4,623	82,610	90/99	15.7
kroD100-R-3-1	0.99	4/4	5/5	6/6	4,669.1	79,513.0	4,686	81,764	93/99	15.2
kroD100-R-3-2	0.99	4/4	5/5	6/6	4,675.7	77,415.0	4,686	79,120	92/99	13.9
kroE100-R-3-0	0.99	2/2	1/1	3/3	1,401.0	35,926.0	1,401	35,926	32/99	8.5
kroE100-R-3-1	0.99	2/2	1/1	3/3	1,401.0	35,926.0	1,401	35,926	32/99	8.8
kroE100-R-3-2	0.99	2/2	1/1	3/3	1,401.0	35,926.0	1,401	35,926	32/99	9.5
kroA100-R-5-0	0.999	5/5	8/8	4/6	5,038.0	74,278.1	5,038	71,687	99/99	23.6
kroA100-R-5-1	0.999	5/5	8/8	3/6	5,038.0	68,661.7	5,038	67,773	99/99	24.8
kroA100-R-5-2	0.999	5/5	8/8	2/6	5,038.0	65,331.6	5,038	64,027	99/99	25.3
kroA100-R-5-3	0.999	5/5	8/8	2/6	5,038.0	63,521.9	5,038	62,672	99/99	23.0
kroA100-R-5-4	0.999	5/5	8/8	2/6	5,038.0	62,825.6	5,038	62,084	99/99	21.6
kroB100-R-5-0	0.99	4/4	4/4	5/5	3,447.2	71,832.1	3,577	71,455	89/99	11.6
kroB100-R-5-1	0.99	4/4	4/4	5/5	3,847.8	70,996.1	3,920	72,518	94/99	14.6
kroB100-R-5-2	0.99	4/4	4/4	5/5	3,916.7	70,398.8	4,005	70,575	95/99	14.9
kroB100-R-5-3	0.99	4/4	4/4	5/5	4,057.0	72,005.4	4,133	73,949	96/99	18.2
kroB100-R-5-4	0.99	4/4	4/4	5/5	4,078.7	71,880.1	4,140	74,485	96/99	14.4
kroC100-R-5-0	0.999	7/7	4/4	7/7	5,412.0	86,435.5	5,412	84,182	99/99	16.8
kroC100-R-5-1	0.999	7/7	4/4	5/7	5,412.0	76,577.3	5,412	74,768	99/99	21.1
kroC100-R-5-2	0.999	7/7	4/4	5/7	5,412.0	72,648.6	5,412	71,473	99/99	22.6
kroC100-R-5-3	0.999	7/7	4/4	4/7	5,412.0	71,705.1	5,412	70,954	99/99	22.9
kroC100-R-5-4	0.999	7/7	4/4	5/7	5,412.0	70,739.5	5,412	70,038	99/99	23.0
kroD100-R-5-0	0.99	5/5	5/5	6/6	4,728.0	89,174.2	4,825	90,017	94/99	15.0
kroD100-R-5-1	0.99	5/5	5/5	6/6	4,990.5	86,071.2	5,081	86,229	97/99	14.8
kroD100-R-5-2	0.99	5/5	5/5	6/6	5,038.9	86,060.6	5,132	84,731	99/99	13.6
kroD100-R-5-3	0.99	5/5	5/5	6/6	5,088.7	84,361.3	5,132	82,268	99/99	13.0
kroD100-R-5-4	0.99	5/5	5/5	6/6	5,122.8	84,060.1	5,132	82,979	99/99	13.0
kroE100-R-5-0	0.99	3/3	1/1	5/5	2,050.8	54,443.8	2,124	55,014	49/99	11.9
kroE100-R-5-1	0.99	3/3	1/1	5/5	2,170.3	53,989.5	2,231	53,191	51/99	9.9

(*continued on next column*)

Table A1 (*continued*)

Instance	α	F_1	F_2	F_3	AvgDm	AvgCost	BSolDm	BSolCost	Cust/Total	Running time (s)
kroE100-R-5-2	0.99	3/3	1/1	5/5	2,268.3	53,561.1	2,296	54,165	53/99	10.8
kroE100-R-5-3	0.99	3/3	1/1	5/5	2,269.3	53,576.3	2,310	55,144	53/99	13.5
kroE100-R-5-4	0.99	3/3	1/1	5/5	2,273.8	53,930.5	2,302	54,851	53/99	9.8
kroA200-R-1-0	0.999	7/7	10/10	8/8	8,096.9	128,898.2	8,100	130,960	149/199	122.7
kroB200-R-1-0	0.999	8/8	12/12	9/9	9,268.0	153,330.0	9,268	152,500	199/199	66.8
kroA200-R-3-0	0.999	9/9	10/10	13/13	10,041.0	160,529.6	10,041	157,776	199/199	56.0
kroA200-R-3-1	0.999	9/9	10/10	13/13	10,041.0	143,058.5	10,041	141,291	199/199	63.8
kroA200-R-3-2	0.999	9/9	10/10	13/13	10,041.0	141,092.2	10,041	139,421	199/199	53.6
kroB200-R-3-0	0.999	8/8	10/10	11/11	8,960.9	165,981.5	9,045	166,494	194/199	64.3
kroB200-R-3-1	0.999	8/8	10/10	11/11	9,172.4	164,537.8	9,177	164,722	194/199	70.0
kroB200-R-3-2	0.999	8/8	10/10	11/11	9,176.5	160,227.7	9,183	161,647	195/199	63.3
kroA200-R-5-0	0.999	10/10	10/10	10/10	9,849.4	155,444.8	9,887	155,737	196/199	62.2
kroA200-R-5-1	0.999	10/10	10/10	10/10	9,959.0	150,345.2	9,980	151,164	197/199	63.9
kroA200-R-5-2	0.999	10/10	10/10	10/10	9,970.8	144,527.8	9,980	144,414	197/199	54.9
kroA200-R-5-3	0.999	10/10	10/10	10/10	9,975.8	140,581.3	9,980	137,922	197/199	60.1
kroA200-R-5-4	0.999	10/10	10/10	10/10	9,977.0	137,614.8	9,982	140,878	197/199	45.5
kroB200-R-5-0	0.999	5/5	12/12	12/12	8,330.4	167,070.1	8,422	166,616	183/199	45.8
kroB200-R-5-1	0.999	5/5	12/12	12/12	8,463.6	165,332.1	8,473	164,583	185/199	54.0
kroB200-R-5-2	0.999	5/5	12/12	12/12	8,470.5	162,877.6	8,483	163,595	180/199	53.7
kroB200-R-5-3	0.999	5/5	12/12	12/12	8,476.3	160,642.4	8,484	161,487	180/199	52.5
kroB200-R-5-4	0.999	5/5	12/12	12/12	8,471.8	157,909.1	8,481	156,510	179/199	40.9

Table A2
Results of ALNS on the large VRP-RPR instances of type C

Instance	α	F_1	F_2	F_3	AvgDm	AvgCost	BSolDm	BSolCost	Cust/Total	Running time (s)
kroA100-C-1-0	0.99	5/5	4/4	7/7	5,038.0	65,475.9	5,038	64,049	99/99	16.0
kroB100-C-1-0	0.99	5/5	4/4	4/4	4,272.8	74,058.2	4,283	74,660	98/99	14.7
kroC100-C-1-0	0.99	6/6	5/5	4/4	5,283.6	68,969.1	5,291	69,685	94/99	16.3
kroD100-C-1-0	0.99	3/3	2/2	8/8	3,685.2	61,723.5	3,689	60,932	77/99	21.9
kroE100-C-1-0	0.99	3/3	6/6	6/6	4,158.9	86,416.3	4,159	86,231	93/99	22.9
kroA100-C-3-0	0.99	5/5	3/3	4/4	4,173.1	54,843.2	4,184	55,649	80/99	14.0
kroA100-C-3-1	0.99	5/5	3/3	4/4	4,170.6	54,236.2	4,186	55,716	81/99	13.0
kroA100-C-3-2	0.99	5/5	3/3	4/4	4,172.9	54,320.9	4,185	56,074	81/99	12.3
kroB100-C-3-0	0.99	4/4	4/4	5/5	4,035.0	72,607.0	4,099	74,541	95/99	14.2
kroB100-C-3-1	0.99	4/4	4/4	5/5	4,106.7	73,975.3	4,157	74,747	96/99	14.1
kroB100-C-3-2	0.99	4/4	4/4	5/5	4,128.4	74,081.6	4,158	74,895	96/99	13.1
kroC100-C-3-0	0.99	6/6	5/5	5/5	5,412.0	73,892.4	5,412	73,415	99/99	15.0
kroC100-C-3-1	0.99	6/6	5/5	5/5	5,412.0	73,316.9	5,412	72,812	99/99	14.7
kroC100-C-3-2	0.99	6/6	5/5	5/5	5,412.0	73,677.5	5,412	73,525	99/99	14.2
kroD100-C-3-0	0.99	4/4	5/5	6/6	4,651.4	78,299.3	4,677	78,992	91/99	16.3
kroD100-C-3-1	0.99	4/4	5/5	6/6	4,603.3	79,299.1	4,629	81,478	91/99	16.0
kroD100-C-3-2	0.99	4/4	5/5	6/6	4,666.3	80,054.2	4,684	78,490	91/99	14.1
kroE100-C-3-0	0.99	2/2	1/1	3/3	1,834.2	33,368.0	1,842	32,026	61/99	12.5
kroE100-C-3-1	0.99	2/2	1/1	3/3	1,887.6	27,946.1	1,896	27,410	64/99	12.9
kroE100-C-3-2	0.99	2/2	1/1	3/3	1,886.2	26,561.5	1,898	26,817	65/99	13.5
kroA100-C-5-0	0.999	5/5	8/8	2/6	5,038.0	64,626.5	5,038	63,870	99/99	23.3
kroA100-C-5-1	0.999	5/5	8/8	2/6	5,038.0	63,458.2	5,038	62,657	99/99	22.6
kroA100-C-5-2	0.999	5/5	8/8	2/6	5,038.0	63,656.8	5,038	62,557	99/99	20.2
kroA100-C-5-3	0.999	5/5	8/8	2/6	5,038.0	63,371.9	5,038	62,666	99/99	18.4
kroA100-C-5-4	0.999	5/5	8/8	2/6	5,038.0	63,386.2	5,038	62,882	99/99	18.0
kroB100-C-5-0	0.99	4/4	4/4	5/5	3,843.0	75,177.6	3,912	74,578	84/99	12.0
kroB100-C-5-1	0.99	4/4	4/4	5/5	3,960.8	74,739.7	3,991	75,602	86/99	11.5
kroB100-C-5-2	0.99	4/4	4/4	5/5	4,038.7	75,241.2	4,055	74,959	88/99	11.3
kroB100-C-5-3	0.99	4/4	4/4	5/5	4,045.5	75,443.8	4,067	75,675	88/99	10.8
kroB100-C-5-4	0.99	4/4	4/4	5/5	4,050.0	75,333.3	4,067	75,293	88/99	10.3
kroC100-C-5-0	0.999	7/7	4/4	5/7	5,412.0	71,820.1	5,412	70,264	99/99	18.5
kroC100-C-5-1	0.999	7/7	4/4	5/7	5,412.0	70,741.4	5,412	69,763	99/99	18.0
kroC100-C-5-2	0.999	7/7	4/4	4/7	5,412.0	70,329.5	5,412	69,749	99/99	18.0
kroC100-C-5-3	0.999	7/7	4/4	5/7	5,412.0	70,196.3	5,412	69,606	99/99	17.4
kroC100-C-5-4	0.999	7/7	4/4	4/7	5,412.0	70,093.4	5,412	69,388	99/99	17.4
kroD100-C-5-0	0.99	5/5	5/5	6/6	5,128.7	86,195.2	5,132	83,268	99/99	14.1
kroD100-C-5-1	0.99	5/5	5/5	6/6	5,105.5	86,553.0	5,132	85,486	99/99	14.1
kroD100-C-5-2	0.99	5/5	5/5	6/6	5,116.7	85,684.6	5,132	83,695	99/99	13.5
kroD100-C-5-3	0.99	5/5	5/5	6/6	5,132.0	86,261.9	5,132	83,071	99/99	12.7
kroD100-C-5-4	0.99	5/5	5/5	6/6	5,113.9	84,858.9	5,132	82,823	99/99	11.7
kroE100-C-5-0	0.99	3/3	1/1	5/5	2,363.6	51,872.2	2,426	53,422	68/99	10.4
kroE100-C-5-1	0.99	3/3	1/1	5/5	2,399.6	51,954.8	2,426	52,763	68/99	9.8
kroE100-C-5-2	0.99	3/3	1/1	5/5	2,400.5	52,039.4	2,426	52,622	68/99	10.8
kroE100-C-5-3	0.99	3/3	1/1	5/5	2,408.4	52,126.6	2,426	52,622	68/99	12.3
kroE100-C-5-4	0.99	3/3	1/1	5/5	2,408.4	51,938.8	2,426	52,622	68/99	12.1
kroA200-C-1-0	0.999	7/7	10/10	8/8	8,096.3	128,878.9	8,100	134,094	153/199	104.6
kroB200-C-1-0	0.999	8/8	12/12	9/9	9,268.0	153,006.0	9,268	152,165	199/199	56.2
kroA200-C-3-0	0.999	9/9	10/10	13/13	10,041.0	141,959.9	10,041	140,937	199/199	53.1

(*continued on next column*)

Table A2 (*continued*)

Instance	α	F_1	F_2	F_3	AvgDm	AvgCost	BSolDm	BSolCost	Cust/Total	Running time (s)
kroA200-C-3-1	0.999	9/9	10/10	13/13	10,041.0	139,657.7	10,041	138,585	199/199	51.2
kroA200-C-3-2	0.999	9/9	10/10	13/13	10,041.0	139,537.8	10,041	138,517	199/199	40.0
kroB200-C-3-0	0.999	8/8	10/10	11/11	9,187.9	165,596.7	9,194	166,386	194/199	68.7
kroB200-C-3-1	0.999	8/8	10/10	11/11	9,191.4	165,361.1	9,195	166,418	194/199	56.5
kroB200-C-3-2	0.999	8/8	10/10	11/11	9,190.1	164,241.7	9,192	164,054	194/199	54.9
kroA200-C-5-0	0.999	10/10	10/10	10/10	9,945.9	144,805.4	9,947	144,812	198/199	50.4
kroA200-C-5-1	0.999	10/10	10/10	10/10	9,945.9	139,810.0	9,946	136,513	197/199	45.1
kroA200-C-5-2	0.999	10/10	10/10	10/10	9,945.0	138,802.3	9,946	136,481	197/199	47.0
kroA200-C-5-3	0.999	10/10	10/10	10/10	9,943.0	137,396.9	9,946	132,870	197/199	38.9
kroA200-C-5-4	0.999	10/10	10/10	10/10	9,943.9	135,448.1	9,946	133,141	197/199	40.6
kroB200-C-5-0	0.999	5/5	12/12	12/12	8,475.1	161,371.4	8,484	161,070	185/199	62.2
kroB200-C-5-1	0.999	5/5	12/12	12/12	8,481.4	160,859.6	8,489	163,341	184/199	60.3
kroB200-C-5-2	0.999	5/5	12/12	12/12	8,484.2	161,218.4	8,491	159,499	184/199	55.8
kroB200-C-5-3	0.999	5/5	12/12	12/12	8,482.8	160,016.5	8,492	161,139	185/199	54.9
kroB200-C-5-4	0.999	5/5	12/12	12/12	8,481.9	158,165.2	8,490	160,010	185/199	43.9

References

Ahmed, Z.H., 2014. The ordered clustered travelling salesman problem: a hybrid genetic algorithm. Sci. World J. 2014.

Archetti, C., Speranza, M., Vigo, D., 2014. Vehicle routing problems with profits. In: Toth, P., Vigo, D. (Eds.), Vehicle Routing: Problems, Methods, and Applications. SIAM, Philadelphia, PA.

Battarra, M., Erdogan, G., Vigo, D., 2014. Exact algorithms for the clustered vehicle routing problem. Oper. Res. 62 (1), 58–71, 1.

Bulhões, T., Hà, M.H., Martinelli, R., Vidal, T., 2018. The vehicle routing problem with service level constraints. Eur. J. Oper. Res. 265 (2), 544–558, 2.

Dantzig, G.B., Ramser, J.H., 1959. The truck dispatching problem. Manag. Sci. 6 (1), 80–91.

Demir, E., Bektaş, T., Laporte, G., 12, 2012. An adaptive large neighborhood search heuristic for the pollution-routing problem. Eur. J. Oper. Res. 223 (2), 346–359, 2.

Fischetti, M., Gonzalez, J.J.S., Toth, P., 1998. Solving the orienteering problem through branch-and-cut. Inf. J. Comput. 10 (2), 121–260.

François, V., Arda, Y., Crama, Y., Laporte, G., 2016. Large neighborhood search for multi-trip vehicle routing. Eur. J. Oper. Res. 255 (2), 422–441, 2.

Ghiani, G., Improta, G., 2000. An efficient transformation of the generalized vehicle routing problem. Eur. J. Oper. Res. 122 (1), 11–17.

Goeke, D., Gschwind, T., Schneider, M., 2018. Upper and lower bounds for the vehicle-routing problem with private fleet and common carrier. Discrete Appl. Math. 264, 43–61.

Hà, M.H., Bostel, N., Langevin, A., Rousseau, L.M., 2014. An exact algorithm and a metaheuristic for the generalized vehicle routing problem with flexible fleet size. Comput. Oper. Res. 43, 9–19.

Hà, M.H., Nguyen Phuong, H., Tran Ngoc Nhat, H., Langevin, A., Trépanier, M., 2020. Solving the clustered traveling salesman problem with -relaxed priority rule. Int. Trans. Oper. Res. (in press).

Gralla, E., Goentzel, J., 2018. Humanitarian transportation planning: evaluation of practice-based heuristics and recommendations for improvement. Eur. J. Oper. Res. 269 (2), 436–450, 2.

Hemmelmayr, V.C., Cordeau, J.-F., Crainic, T.G., 2012. An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. Comput. Oper. Res. 39 (12), 3215–3228.

Kuang, E., 2012. A 2-Opt-Based Heuristic for the Hierarchical Traveling Salesman Problem. Undergraduate thesis, University of Maryland.

Mancini, S., 2016. A real-life multi depot multi period vehicle routing problem with a heterogeneous fleet: formulation and adaptive large neighborhood search based matheuristic. Transport. Res. C Emerg. Technol. 70, 100–112.

Oran, A., Tan, K.C., Ooi, B.H., Sim, M., Jaillet, P., 2012. Location and routing models for emergency response plans with priorities. In: Aschenbruck, N., Martini, P., Meier, M., Tölle, J. (Eds.), Future Security. Future Security 2012. Communications in Computer and Information Science, vol. 318. Springer, Berlin, Heidelberg, pp. 129–140.

Panchamgam, K.V., 2011. Essays in Retail Operations and Humanitarian Logistics. Ph.D. thesis. Robert H.Smith School of Business, University of Maryland, College Park, Md.

Panchamgam, K.V., Xiong, Y., Golden, B., Dussault, B., Wasil, E., 2013. The hierarchical traveling salesman problem. Optimization Letters 7 (7), 1517–1524, 7.

Pop, P.C., Fuksz, L., Marc, A.H., Sabo, C., 2018. A novel two-level optimization approach for clustered vehicle routing problem. Comput. Ind. Eng. 115, 304–318.

Potvin, J.-Y., Guertin, F., 1998. A genetic algorithm for the clustered traveling salesman problem with a prespecified order on the clusters. In: Woodruff, D.L. (Ed.), Advances in Computational and Stochastic Optimization, Logic Programming, and Heuristic Search. Kluwer Academic Publishers, Norwell, MA, USA, pp. 287–299.

Potvin, J.-Y., Rousseau, J.-M., 1993. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. Eur. J. Oper. Res. 66 (3), 331–440.

Reinelt, G., 1991. TSPLIB—a traveling salesman problem library. ORSA J. Comput.

Rodrigue, J., Comtois, C., Slack, B., 2013. The Geography of Transport Systems. Taylor & Francis.

Ropke, S., Pisinger, D., 11, 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transport. Sci. 40 (4), 455–472, 4.

Sabo, C., Pop, P.C., Horvat-Marc, A., 2020. On the selective vehicle routing problem. Mathematics 8 (5).

Shaw, P., 1998. Using constraint programming and local search methods to solve vehicle routing problems. In: Principles and Practice of Constraint Programming — CP 1998. Lecture Notes in Computer Science, vol. 1520, pp. 417–431.

Shetty, V.K., Sudit, M., Nagi, R., 6, 2008. Priority-based assignment and routing of a fleet of unmanned combat aerial vehicles. Comput. Oper. Res. 35 (6), 1813–1828, 6.

Sheu, J.-B., 2007. An emergency logistics distribution approach for quick response to urgent relief demand in disasters. Transport. Res. E Logist. Transport. Rev. 43 (6), 687–709, 6.

Silva, M.M., Subramanian, A., Vidal, T., Ochi, L.S., 2012. A simple and effective metaheuristic for the minimum latency problem. Eur. J. Oper. Res. 221 (3), 513–520.

Song, B.D., Ko, Y.D., 2016. A vehicle routing problem of both refrigerated- and general-type vehicles for perishable food products delivery. J. Food Eng. 169, 61–71.

Tillman, F.A., Cain, T.M., 1972. An upperbound algorithm for the single and multiple terminal delivery problem. Manag. Sci. 18 (11), 664–682, 11.

Toth, P., Vigo, D., 2014. Vehicle Routing: Problems, Methods and Applications, 2nd Edition. Series in Optimization. SIAM, Philadelphia, PA.

Vansteenwegen, P., Souffriau, W., Oudheusden, D., 2010. The orienteering problem: a survey. Eur. J. Oper. Res. 209 (1), 1–10.

Vidal, T., Crainic, T., Gendreau, M., Prins, C., 2013. Heuristics for multi-attribute vehicle routing problems: a survey and synthesis. Eur. J. Oper. Res. 231 (1), 1–21.

Yang, Z., Emmerich, M., Back, T., 2015. Ant based solver for dynamic vehicle routing problem with time windows and multiple priorities. In: 2015 IEEE Congress on Evolutionary Computation (CEC). IEEE, pp. 2813–2819.