A new constraint programming model and a linear programming-based adaptive large neighborhood search for the vehicle routing problem with synchronization constraints

Minh Hoàng Hà*
ORLab, Faculty of Computer Science, Phenikaa University, Hanoi, Vietnam

Tat Dat Nguyen
ORLab, Faculty of Information Technology, VNU University of Engineering and Technology, Hanoi, Vietnam

Thinh Nguyen Duy, Hoang Giang Pham, Thuy Do
Department of Computer Science, FPT University, Hanoi, Vietnam

Louis-Martin Rousseau
École Polytechnique de Montréal and CIRRELT, Montréal, Canada

Abstract. We consider a vehicle routing problem which seeks to minimize cost subject to time window and synchronization constraints. In this problem, the fleet of vehicles is categorized into regular and special vehicles. Some customers require both vehicles' services, whose service start times at the customer are synchronized. Despite its important real-world application, this problem has rarely been studied in the literature. To solve the problem, we propose a Constraint Programming (CP) model and an Adaptive Large Neighborhood Search (ALNS) in which the design of insertion operators is based on solving linear programming (LP) models to check the insertion feasibility. A number of acceleration techniques is also proposed to significantly reduce the computational time. The computational experiments show that our new CP model finds better solutions than an existing CP-based ALNS, when used on small instances with 25 customers and with a much shorter running time. Our LP-based ALNS dominates the CP-based ALNS, in terms of solution quality, when it provides solutions with better objective values, on average, for all instance classes. This demonstrates the advantage of using linear programming instead of constraint programming when dealing with a variant of vehicle routing problems with relatively tight constraints, which is often considered to be more favorable for CP-based methods. We also adapt our algorithm to solve a well-studied variant of the problem, and the obtained results show that the algorithm provides good solutions as state-of-the-art approaches and improves four best known solutions.

Keywords. Vehicle routing problem, time window, synchronization constraint, constraint programming, adaptive large neighborhood search.

1. Introduction

Most of the requirements related to our daily routines are made by a service provider coming to our premises. These types of services can be home care delivery, maintenance operations, public utilities, etc. In such services, efficient delivery and timely service play important roles. This is why the class of the Vehicle Routing Problem, coupled with the Scheduling Problem, comprises a large class of problems with many variations and applications. The main focus of this research is to study the Vehicle Routing Problem

with Synchronization Constraints (VRPSC), where both time window and synchronization constraints are present. In the latter constraints, some customers may require the service of two vehicles whose service start times at the customer must be synchronized.

In a recent industrial project with an Internet Service Provider (ISP) in Vietnam, the authors witnessed several contexts where synchronization constraints arose. The ISP company has to perform installation services for new subscribers and maintenance services for subscribed clients. Both services are performed by technicians who mainly use motorbikes for travelling. In many cases, a customer asks for services by two technicians belonging to two different teams: one being the "physical" team, which takes charge of the hardware (cable wire, modem, etc.); while the other team manages the signal. To further illustrate the problem, when a customer requires a service from the physical team, two technicians must be mobilized as one helps the other with equipment set-up, such as installing a ladder and other protective gear. In addition, an intern technician, who is in a probationary period, needs to be coupled with an experienced technician at customer locations. When servicing a customer, the company requires that the service start time of both technicians be as close as possible in order to reduce their waiting time and to limit the disruption to the customer. In the case of our ISP partner, a delay of no more than 15 minutes is permitted. The problem was initially introduced in [8] and can be used to model other real world applications such as home care delivery, aircraft fleet assignment, ground handling, and forest operations (see [20] for more information).

Hojabri et al. [8] proposed a constraint programming-based Adaptive Large Neighborhood Search (cp-ALNS) with insertion operators exploiting constraint propagation capabilities to guarantee the feasibility of a new generated solution. Different from the popular ALNS proposed in [18] to solve VRPs, the cp-ALNS does not try to add all unserved requests one by one but, rather, adds all of them at once to create a new complete solution. Several removal operators were specifically designed for the problem. Numerical results are reported on instances derived from Vehicle Routing Problem with Time Window (VRPTW) benchmark instances, with up to 200 customers and 100 synchronizations.

A dynamic version of the problem was introduced in [20] where customer requests were not foreseen, but arrived one-by-one in real time. The problem was modeled as a CP program from which a metaheuristic was designed. We note that the methods proposed in [20, 8] were based on the same CP model which uses variables representing the successor of a node, $AllDifferent$ and subtour elimination constraints introduced in [16]. However, no result of the pure CP model has been reported.

Dohn et al. [3] consider synchronization constraints defined in the same way as in the VRPSC. Their problem is called the vehicle routing problem with time windows and temporal dependencies (VRPTWTD). Unlike the VRPSC, there is only one type of vehicle in the VRPTWTD. Two compact formulations of the problem are proposed and the Dantzig-Wolfe decomposition of these formulations are used to develop a column generation-based solution approach. Four different master problem formulations are proposed and a tailored time window branching is used to force feasibility of the relaxed master problems. A computational experiment is performed to quantitatively access strengths and weaknesses of the proposed formulations.

Several VRPSC variants in which a customer can require services from more than two vehicles are also studied in the literature. Labadie et al. [11] study a VRPSC variant with simultaneous synchronization constraints where a service center (the depot) offers several types of services and customers may demand more than one service to be provided simultaneously. To solve this problem, a mixed integer programming model and a local

search-based metaheuristic are developed. In the local search procedure, the service start time at the customers requiring synchronized visits is fixed to the values in the input solution.

Several articles [2, 19, 1, 13, 15] study a different variant of the VRPSC in which the synchronized customers must be serviced at exactly the same time and the fleet of vehicles is homogeneous (only one type of vehicles). In the following, we call this variant as the VRP with time windows and synchronized visits (VRPTWSyn). Bredström and Rönnqvist [2] consider the problem arising in the context of home care crew scheduling. The problem is first formulated as a mixed integer programming (MIP) model that sets pairwise synchronization and pairwise temporal precedence between customer visits. It is then solved through a MIP-based heuristic. Rasmussen et al. [19] solve a similar problem with an exact branch-and-price algorithm. Due to the application context, there is no capacity constraint and specific issues about home care crews are taken into account, in addition to the synchronization requirements, like care giver preferences, customer priority, and the ability of a particular care giver to serve a given customer. Also, not all customers must be serviced because visits can be rescheduled or canceled.

Based on the special requirement that synchronized customers must be serviced at exactly the same time, several authors design metaheuristics with efficient components to tackle the problem. To handle this special version of the synchronization constraint, they prohibit cross-synchronization, e.g. visiting u then v by a vehicle, visiting i then j by another vehicle, and finally realizing that u and j are the same customer as well as v and i. Afifi et al. [1] propose a simulated annealing algorithm with dedicated local searches (SA-ILS). They use a reduced solution that contains only the synchronization visits to filter out cross synchronizations from the set of possible positions for insertions during solution construction and local search phases. Based on a similar idea, Liu et al. [13] propose an ALNS and use a square matrix to record the visit sequence among each pair of visited special customers. Applying the cross-synchronization idea to the VRPSC to handle its more general synchronization constraint can remove feasible good neighborhoods, decreasing the performance of our metaheuristics. Thus, the adaptation of the metaheuristics proposed in [1, 13] for the VRPSC problem would not be efficient. Recently, Parragh et al. [15] evaluate several different ways to deal with pairwise synchronization constraints in the context of two problems: the VRPTW with pairwise synchronization, and the service technician routing and scheduling problem. They propose three ways to address the synchronization requirement: individual synchronized timing optimization; global synchronized timing optimization; and adaptive time window. The idea of the first two approaches is to keep the ALNS untouched, while the last one makes some modifications to the insertion scheme in order to identify good service start times for synchronized visits.

In [5], a new variant arising in the context of airport ground handling, named abstract vehicle routing problem with worker and vehicle synchronization (AVRPWVS), is studied. The AVRPWVS deals with routing workers to ground handling jobs such as unloading baggage or refuelling an aircraft. Each job has a time window and can be performed by a variable number of workers. A worker can use vehicles to travel between locations or can be moved with other workers by a driver. Two mathematical multi-commodity flow formulations based on time-space networks are proposed to model synchronization constraints including movement and load synchronization. Moreover, the authors develop a branch-and-price heuristic that employs both conventional variable branching and a novel variable fixing strategy. More recently, Sarasola and Doerner [21] consider a variant

of the VRP with synchronization constraints in which multiple depots are considered and a customer can require more than two deliveries. The problem requires that the allowed amount of non-service time between the first and last delivery to a customer does not exceed a given value. This synchronization constraint can be seen to be more general than the one considered in the VRPSC because a customer can request the service of more than two vehicles. However, there is no time window associated with either customers or deliveries.

Pillac et al. [17] introduce a Dynamic Technician Routing and Scheduling Problem (D-TRSP) which deals with a limited crew of technicians serving a set of requests. In the D-TRSP, each technician has a set of skills, tools and spare parts required by each request. In addition to designing a route at the beginning of each day, two types of decisions must be managed in real time. First, whenever a new request appears, we must decide whether it is accepted or not. And second, whenever a technician finishes serving a request, we need to find the next request to serve. For a survey on further synchronization issues in the context of vehicle routing problems, we refer the readers to [4].

Our main contributions are as follows: we propose a new CP model to address the VRP with time windows and synchronization constraints, as well as a linear programming-based ALNS. Different from the CP model in the literature, our CP uses *sequence* and *interval* variables of IBM ILOG CP Optimizer [23] to formulate the problem. The most notable feature of our metaheuristic is that we use linear programming and a number of acceleration techniques to quickly check the feasibility of insertion operations integrated in the popular ALNS [18]. This is the first time LP models are used instead of CP to design the ALNS algorithm for the VRP with synchronization constraints. The computational experiments carried on the benchmark data show the good performance of our methods. More precisely, our CP model can provide better solutions on small-size instances than the cp-ALNS of [8] in a much shorter running time. Our lp-ALNS dominates the cp-ALNS in terms of solution quality, as it improves 620 out of 681 best known solutions. When being adapted to solve the VRPTWSyn, it is competitive to state-of-the-art metaheuristics in terms of solution quality and find four new best known solutions.

The remainder of the paper is organized as follows: Section 2 introduces the problem definition and our new CP model; the detailed description of lp-ALNS is provided in Section 4; experimental results are reported in Section 5; and finally, we conclude our work in Section 6.

2. Problem definition and a mixed integer programming model

The problem may be formally defined as follows. We have a directed graph $G = (V, A)$, where $V = \{v_0\} \cup V_s \cup \overline{V}$ is the set of vertices representing customer locations and A is the set of arcs. The vertex v_0 is the depot where a set of vehicles K is located. Vehicles in set K are again divided into two sub-sets: regular vehicles (set K_r) and special vehicles (set K_s). All regular vehicles have capacity Q. $\overline{V}$ is the set of regular customers which are visited by regular vehicles only while V_s is the set of special customers, where each requires the visits of both types of vehicle. Let V_s^c be the set of vertices that are copies of special customers V_s. Let V_r be defined as $V_r = \overline{V} \cup V_s^c$, which is the set of customer vertices that must be visited by regular vehicles. Each vertex i in V_s is associated with a demand q_i. Let v_0^b and v_0^e be the vertices where the vehicle starts and ends its route. Note that these two vertices have the same location as v_0. Additionally, we define $V_r^+ = V_r \cup \{v_0^b\} \cup \{v_0^e\}$ the set of vertices appearing on routes of regular vehicles; and $V_s^+ = V_s \cup \{v_0^b\} \cup \{v_0^e\}$

the set of vertices visited by special vehicles. A service time h_i^l is associated with vertex $i \in V_l^+$, $l \in \{r, s\}$. Note that the service times at the depot and its copies are set to 0. A time window $[o_i, u_i]$ is imposed on each vertex $i \in V_r^+ \setminus \{v_0^b\}$. Finally, each arc $(i, j) \in A$ is associated with non-negative values c_{ij}^l and t_{ij}^l representing the travel cost, which can be estimated by distance or all moving expenses, and travel time from vertex i to vertex j for a vehicle of type l.

The problem then consists in constructing routes for the fleet of vehicles such that the total travel cost incurred by the fleet of vehicles is minimized and the following constraints are satisfied:

- Each vehicle must begin its route at the depot, deliver services to customers and finally return to the depot.
- Each regular customer is served by exactly one regular vehicle.
- Each special customer is served by exactly one regular vehicle and one special vehicle.
- The total demand serviced by a regular vehicle must not exceed its capacity Q.
- A regular vehicle must start its service at a vertex $i \in V_r^+ \setminus \{v_0^b\}$ within the time window $[o_i, u_i]$.
- The service start time at a special customer $i \in V_s$ visited by a special vehicle must be within a time window $[t_{m_i} - \alpha_i, t_{m_i} + \beta_i]$. Here, α_i and β_i are given parameters representing a possible delay between regular and special services at customer i, while t_{m_i} is the service start time at vertex m_i (mirror of i in the regular vehicle route).

Several mixed integer programs have been proposed for variants of the VRPSC in [2, 1, 15]. However, we have not seen any MIP formulation for the VRPSC itself. In the following, we introduce a MIP that can be used to solve small-size VRPSC instances to optimality. There are two types of variables in our formulation as follows:

- x_{ij}^k: binary variables equal to 1 if vehicle $k \in K_l$ travels from vertex $i \in V_l^+$ to vertex $j \in V_l^+$, $l \in \{r, s\}$; equal to 0 otherwise.
- y_i: service start time at which vertex $i \in V_r^+ \cup V_s^+$ is serviced.

The VRPSC can be stated as:

$$\text{Minimize} \quad \sum_{l\in\{r,s\}} \sum_{k\in K_l} \sum_{i\in V_l^+} \sum_{j\in V_l^+:j\neq i} c_{ij}^l x_{ij}^k \tag{1}$$

$$\text{Subject to} \sum_{k\in K_l} \sum_{j\in V_l^+} x_{ij}^k = 1 \qquad \forall l \in \{r,s\}, i \in V_l^+ : i \neq j \tag{2}$$

$$\sum_{j\in V_l^+:j\neq v_0^b} x_{v_0^b j}^k = 1 \qquad \forall l \in \{r,s\}, k \in K_l \tag{3}$$

$$\sum_{i\in V_l^+:i\neq v_0^e} x_{i v_0^e}^k = 1 \qquad \forall l \in \{r,s\}, k \in K_l \tag{4}$$

$$\sum_{j\in V_l^+:j\neq i} x_{ij}^k = \sum_{j\in V_l^+:j\neq i} x_{ji}^k \qquad \forall l \in \{r,s\}, k \in K_l, i \in V_l^+ \tag{5}$$

$$\sum_{i\in V_r} \sum_{j\in V_r^+:j\neq i} q_i x_{ij}^k \leq Q \qquad \forall k \in K_r \tag{6}$$

$$y_i + (h_i^l + t_{ij}^l)x_{ij}^k - M(1 - x_{ij}^k) \leq y_j \qquad \forall l \in \{r,s\}, k \in K_l, i \in V_l^+, j \in V_l^+ : i \neq j \tag{7}$$

$$o_i \leq y_i \leq u_i \qquad \forall i \in V_r^+ \setminus \{v_0^b\} \tag{8}$$

$$y_{m_i} - \alpha_i \leq y_i \qquad \forall i \in V_s \tag{9}$$

$$y_i \leq y_{m_i} + \beta_i \qquad \forall i \in V_s \tag{10}$$

$$x_{ij}^k \in \{0,1\} \qquad \forall k \in K_l, l \in \{r,s\}, i \in V_l^+, j \in V_l^+ : i \neq j \tag{11}$$

$$y_i \geq 0 \qquad \forall i \in V_r^+ \cup V_s^+ \tag{12}$$

The objective function (1) minimizes the total travel cost. Constraints (2) ensure that each vertex is visited by exactly one vehicle. Constraints (3) and (4) force vehicles to begin and end their route at the depot. The flow conservation is satisfied by Constraints (5) while the capacity requirement of each regular vehicle is found in (6). Constraints (7) represent the relationship between variables y and x. More precisely, they compute the service start time at vertex j based on vertex i if the vehicle travels from i to j. Here, M is a very large number, and can set to $M = u_{v_0^e}$. Constraints (8) are time window constraints. Synchronization requirements are respected by Constraints (9) and (10). Finally, Constraints (11) and (12) define the variables' domains.

3. A new constraint programming model

We now present the new CP model for the problem. First, it is worth mentioning that unlike MIP formulations, there is no standard in CP formulation because it strongly depends on each CP package. In this study, we formulate the model using generic keywords and syntaxes of IBM ILOG CP Optimizer [23] adapted from the CP formulations proposed in [12, 6, 7, 14]. Our model uses the following variables:

- Itv_i^l — interval variable that represents the time interval of size h_i^l for the visit of vertex $i \in V_l^+$, $l \in \{r,s\}$;
- $ItvAlt_{ik}^l$ — optional interval variable that represents the time interval for the visit of vehicle $k \in K_l$ at vertex $i \in V_l^+$, $l \in \{r,s\}$;
- Seq_k^l — sequence variable that represents all working time intervals of vehicle $k \in K_l$, $l \in \{r,s\}$;

An interval variable represents the interval of time during which a task can occur. It contains a starting point, an end point, a size, and it can be optional. A decision variable is used to represent whether or not an interval is present. If an interval is marked as optional, it may be absent in the solution. Sequence is a type of variable in IBM ILOG OPL which can be empty or can contain a subset of variables. A sequence represents all intervals that are present in the solution. The constraints in our model are as follows:

1. Function to link each interval to a location.

$$\textbf{\textit{type}}\ \text{function}\ \theta(Seq_k^l, ItvAlt_{ik}^l) = i \qquad \forall i \in V_l^+, k \in K_l, l \in \{r, s\} \tag{13}$$

   A non-negative integer value θ is defined for each pair (Seq_k^l, $ItvAlt_{ik}^l$), $l \in \{r, s\}$, to indicate the type of the interval variable in a sequence. This value will be used in ***noOverlap*** constraints as shown below.
2. Each customer must be served by one vehicle of the corresponding type.

$$\textbf{\textit{alternative}}\ (Itv_i^l, ItvAlt_{ik}^l : k \in K_l) \qquad \forall i \in V_l^+, l \in \{r, s\} \tag{14}$$

   The ***alternative*** function ensures that exactly one set of intervals $ItvAlt_{ik}^l$ is present in the solution; and interval variable starts and ends together with the interval variable Itv_i^l.
3. Travel time between two customers must be taken into account. In the following, $T_l \in \{t_{ij}^l\}$ is the matrix representing travel times between two vertices i and j in the set V_l^+.

$$\textbf{\textit{noOverlap}}\ (Seq_k^l, T_l) \qquad \forall k \in K_l, l \in \{r, s\} \tag{15}$$

   The ***noOverlap*** constraint on sequence variable Seq_k^l states that the sequence defines a chain of non-overlapping intervals, and any interval in the chain is constrained to end before the start of the next interval in the chain. For each ***noOverlap*** constraint, a transition matrix T_l is used to define the minimal non-negative distance separating consecutive intervals in a sequence. For example, if interval $ItvAlt_{ik}^l$ appears before interval $ItvAlt_{jk}^l$ in the sequence Seq_k^l, a minimal distance $t_{\theta_i\theta_j}^l$ must be respected between the end of $ItvAlt_{ik}^l$ and the start of $ItvAlt_{jk}^l$, where θ_i and θ_j denote the types of $ItvAlt_{ik}^l$ and $ItvAlt_{jk}^l$ in the sequence Seq_k^l.
4. All vehicles start their route at the starting depot.

$$\textbf{\textit{first}}\ (Seq_k^l, ItvAlt_{v_0^b k}^l) \qquad \forall k \in K_l, l \in \{r, s\} \tag{16}$$

   first(p, j) function states that if interval j is present, it will be the first interval in the sequence p. These constraints force each vehicle to start it route at vertex v_0^b.
5. All vehicles finish their routes at the corresponding ending depot.

$$\textbf{\textit{last}}\ (Seq_k^l, ItvAlt_{v_0^e k}^l) \qquad \forall k \in K_l, l \in \{r, s\} \tag{17}$$

   Similar to ***first***(p, j) function, ***last***(p, j) function states that if interval j is present, it will be the last interval in the sequence p. These constraints are to force each vehicle to finish its route at ending depot v_0^e.

6. Time window constraints.

$$o_i \leq \textbf{\textit{startOf}}(Itv_i^r) \leq u_i \qquad \forall i \in V_r^+ \setminus \{v_0^b\} \tag{18}$$

$\textbf{\textit{startOf}}(j)$ represents the start of interval j whenever the interval variable j is present.

7. Capacity constraints.

$$\sum_{i \in V_r} q_i . \textbf{\textit{presenceOf}}\,(ItvAlt_{ik}^r) \leq Q \qquad \forall k \in K_r \tag{19}$$

$\textbf{\textit{presenceOf}}(j)$ is equal to 1 if interval variable j is present in the solution, 0 otherwise.

8. Synchronization constraints.

$$\textbf{\textit{startOf}}\,(Itv_{m_i}^r) - \alpha_i \leq \textbf{\textit{startOf}}\,(Itv_i^s) \qquad \forall i \in V_s \tag{20}$$

$$\textbf{\textit{startOf}}\,(Itv_i^s) \leq \textbf{\textit{startOf}}\,(Itv_{m_i}^r) + \beta_i \qquad \forall i \in V_s \tag{21}$$

We compute the total cost traveled by a vehicle of type $l \in \{r, s\}$ as follows:

$$Cost_k^l = \textstyle\sum_{i \in V_l^+} c_{ij}^l \; \forall k \in K_l \text{ where}$$
$$j = \textbf{\textit{typeOfNext}}\;(Seq_k^l, ItvAlt_{ik}^l, i, i).$$

Then the objective function can be written as:

$$\textbf{Minimize} \sum_{l=\{r,s\}} \sum_{k \in K_l} Cost_k^l \tag{22}$$

4. Linear programming-based adaptive large neighborhood search algorithm

To tackle the VRPSC problem, we designed an Adaptive Large Neighborhood Search (ALNS) heuristic, which is based on the Large Neighborhood Search (LNS) introduced by [22]. At each iteration, LNS explores a large neighborhood, which can rearrange a large part of the current solution, therefore allowing the search to move to other promising regions of the search space.

More precisely, LNS decomposes the original problem by unfixing some decision variables, leading to a partial solution. The unfixed decision variables define a neighborhood of solutions that can be explored by a specific procedure via, possibly, a heuristic or a Mixed Integer Programming (MIP) solver. If the procedure finds an improved solution, it becomes the new current solution and a new large neighborhood is defined around it. This process is repeated until a stopping criterion is reached.

A first key point is the selection of fixed variables to create a partial solution. In fact, the number of fixed variables impacts the size of the neighborhood (the more fixed variables, the narrower the neighborhood). A common strategy is to dynamically vary the number of removed variables. A second key point lies in the selection of fixed/removed variables which can be based on a random choice or a more sophisticated strategy to guide the search. Finally, the procedure that explores the neighborhood should provide good quality solutions in a short amount of time. Adaptive Large Neighborhood Search is an extension of LNS with a number of different insertion and removal operators. In comparison with LNS, a component that adaptively chooses among a set of removal and insertion

operators is added to the algorithm. The pseudo-code of a ALNS to solve problems with minimizing objective function is shown in Algorithm 1. At each iteration, a randomly selected pair of operators (with procedures SelectDestruction and SelectRepair, lines 5 and 6) is applied to the current solution (line 7), with probabilities p^{remove}, p^{insert} for the set of removal and insertion operator, respectively. These probabilities are updated by a learning process (line 12). The more an operator i has contributed to the solution quality, the larger its probability p_i of being chosen.

Algorithm 1: General ALNS

```
1  Create an initial solution sol
2  s_best := sol
3  Initialize p^remove and p^insert
4  while the stop-criterion is not met do
5      re := SelectRemoval(p^remove)
6      in := SelectInsertion(p^insert)
7      solution sol' = GenerateNewSolution(sol, re, in)
8      if cost(sol') < cost(s_best) then
9          s_best := sol'
10     if accept(sol', sol) then
11         sol := sol'
12     update p^remove and p^insert
13 return s_best
```

4.1. Insertion operators

4.1.1. Cheapest insertion heuristic

The purpose of the insertion operation is to reinsert unserviced requests into the solution. For this task, one can use the cheapest insertion heuristic which inserts the customer into a route at a feasible position making the objective value increase the least. The process is repeated until all customers are serviced or no more customers can be inserted. Note that the customers are considered in the order determined by their cheapest insertion position. The insertion cost of a regular customer j into a regular route positioned between two consecutive vertices i and $i+1$ (denoted by IC_k^r) is computed as:

$$IC_j^r = c_{ij}^r + c_{j(i+1)}^r - c_{i(i+1)}^r \tag{23}$$

Whenever a special customer is considered for insertion, it will be added to two positions: one on a regular route and another on a special route. This must be incorporated when computing the insertion cost of special customers. The average value is used to compute the insertion cost of a special customer j at positions between vertices i and $i+1$ on a regular route and between vertices i' and $i'+1$ on a special route as follows:

$$IC_j^s = \frac{(c_{im_j}^r + c_{m_j(i+1)}^r - c_{i(i+1)}^r) + (c_{i'j}^s + c_{j(i'+1)}^s - c_{i'(i'+1)}^s)}{2} \tag{24}$$

It is worth noting that, using the pure summation to compute IC_j^s could make the insertion cost of the special customers higher than that of the regular customers. Therefore, we use the average value to avoid the situation where the regular customers are always added before the special ones.

4.1.2. Regret heuristics

As in [18], we also use regret-k heuristics as repair operators. Instead of selecting the customer with the least insertion cost in each construction step, the regret heuristics select the customer with the highest regret-k value, computed as follows: we denote $f_{i,j}$ the insertion cost when inserting customer i at its best position in route j. If this insertion is infeasible w.r.t time window and synchronization constraints, the insertion cost is set to infinity, i.e. $f_{i,j} = \infty$. Let r_{ik} be the route where vertex i has the k-th lowest insertion cost. The regret-k value RV_i of customer i is then calculated as:

$$RV_i = \sum_{j=1}^{k} (f_{(i,r_{ij})} - f_{(i,r_{i1})}) \tag{25}$$

The regret-k heuristics chooses the unvisited customer i with the highest regret-k value RV_i and insert it into the feasible position leading the least insertion cost. Ties are broken by selecting customers with the lowest insertion cost $f_{(i,r_{i1})}$. Informally speaking, we choose the customer that leads to the largest regret, if it is not inserted right now. If it happens that some vertices can be feasibly inserted in less than k routes in the current solution, then the vertex with the fewest number of feasible routes is selected. This ensures that the vertex which does not have many insertion options in the current solution will be considered first.

It can be observed that the cheapest insertion heuristic, which we mentioned earlier, is a special case of the regret heuristic with $k = 1$ due to the tie-breaking rule. For any $k > 1$, the regret heuristic looks further into future solutions to decide the choice of insertion. In this research, we use the regret heuristics with $k \in \{2, 3\}$ to design insertion operators for our ALNS.

4.1.3. Checking insertion feasibility of regular customers

When inserting a vertex into a position of the current partial solution, it is required to verify if the insertion satisfies the capacity, time window and synchronization constraints. As the insertion operation is repeated multiple times during the search, designing a quick verification procedure is critical to speed up the overall algorithm. As the capacity constraint is easily checked in $\mathcal{O}(1)$, we focus on the time window and synchronization constraints only. Verifying the feasibility of an insertion operation w.r.t these constraints are more complex because they delay subsequent visits leading to other violations. As proposed in [10], the time window constraint can be checked in $\mathcal{O}(1)$ by pre-computing the maximum delay (push forward) that is allowed at each arc of the current solution, without violating time windows. In this research, we also reuse this idea to handle both time window and synchronization constraints when inserting regular vertices.

Given a partial solution, in order to consider all possible positions to insert an unserved regular customer i into the routes of a partial solution, we calculate the maximum duration of time (also called maximum delay) that can be spared after a vehicle finishes serving vertex $j-1$ and before it starts to serve next vertex j without violating any constraints at the following vertices. This value is denoted δ_a, where $a = (j-1, j)$ represents the arc from $j-1$ to j. We calculate the maximum delays at all the arcs of the current solution using the following linear programming model:

Let $\overline{V_r}$ and $\overline{V_s}$ be the set of all vertices visited by regular and special vehicles in current solution *sol*, respectively. Denote $\overline{A} = A_r \cup A_s$ the set of arcs forming the routes in *sol*; A_r and A_s are the sets of arcs on regular and special routes, respectively. We use two types

of variables: τ_i is the service start time at vertex i and δ_a is the maximum delay on arc $a = (j-1, j)$.

$$\text{(F1)} \quad \textbf{Maximize} \quad \delta_a \tag{26}$$

$$\textbf{Subject to} \quad o_i \leq \tau_i \leq u_i \qquad \forall i \in \overline{V_r} \tag{27}$$

$$-\alpha_i \leq \tau_i - \tau_{m_i} \leq \beta_i \qquad \forall i \in \overline{V_s} \tag{28}$$

$$\tau_{i-1} + h^l_{i-1} + t^l_{(i-1,i)} \leq \tau_i \qquad \forall (i-1, i) \in A_l \setminus \{a\}, l \in \{r, s\} \tag{29}$$

$$\tau_{j-1} + h^r_{j-1} + t^r_{(j-1,j)} + \delta_{(j-1,j)} \leq \tau_j \tag{30}$$

$$\tau_i \geq 0 \qquad \forall i \in \overline{V_r} \cup \overline{V_s} \tag{31}$$

$$\delta_a \geq 0 \tag{32}$$

Objective (26) is to maximize the maximum delay on arc a. Constraints (27) and (28) respectively ensure time window and synchronization constraints at all vertices of the current solution. Constraints (29) represent the relationship between the starting times at vertices $(i-1)$ and i when a vehicle travels from $(i-1)$ to i. Constraint (30) has the same meaning as constraints (29) but is written for arc a. Finally, constraints (31) and (32) define the domain of variables.

After all the maximum delays are available, we check if an unserved regular vertex i can be inserted at the position between vertex $j-1$ and vertex j by computing the earliest arrival time ($arrivalTime$) and the waiting time ($waitTime$) at i as follows:

$$arrivalTime_i = \tau_{j-1} + h^r_{j-1} + t^r_{(j-1)i}$$
$$waitTime_i = \max(o_i - arrivalTime_i, 0)$$

Finally, we check if feasibility of the insertion satisfies the following constraints:

$$t^r_{(j-1)i} + t^r_{ij} - t^r_{(j-1)j} + waitTime_i + h^r_i \leq \delta_{(j-1,j)} \tag{33}$$

$$arrivalTime_i \leq u_i \tag{34}$$

Constraint (33) ensures that the insertion does not lead to a violation of time window and synchronization constraints at all the following customers in the current solution. Constraint (34) verifies the time window constraint of vertex i.

Although it is fast to solve a LP model of type (F1), the running time of the insertion operators is still expensive due to the large quantity of LP models solved during the search. Through observation, we note that constructing the model to find the maximum delay on each arc takes more computational time than solving it. Thus, we propose the following model to reduce the running time of the insertion operators:

$$\text{(F2)} \quad \textbf{Maximize} \quad \sum_{a \in \overline{A}} \varpi_a \delta_a \tag{35}$$

$$\textbf{Subject to} \quad (27), (28), (31)$$

$$\tau_{i-1} + h^l_{i-1} + t^l_{(i-1,i)} + \delta_{(i-1,i)} \leq \tau_i \qquad \forall (i-1, i) \in A_l, l \in \{r, s\} \tag{36}$$

$$\delta_a \geq 0 \qquad \forall a \in \overline{A} \tag{37}$$

Objective (35) is to maximize the weighted maximum delay at all arcs of the solution. Here, ϖ_a is a given binary coefficient representing the weight of arc $a \in \overline{A}$. Constraints (36) indicate the relationship between variables δ and τ. To find the maximum delay on an arc a, we just need to set its weight ϖ_a to 1 and weights of other arcs to 0. The proposed model (F2) allows us to save a lot of time by constructing a model once and then creating a new one by changing two coefficients in the objective function only. As a result, we can avoid constructing multiple models from scratch. A preliminary experiment shows that using this method helps reduce the running time of the overall algorithm by at least 30%. After each model calculates the maximum delay of arc $(j-1,j)$, the value of variables τ_{j-1}, τ_j, and $\delta_{(j-1,j)}$ are also saved for checking insertion feasibility of special customers, as described in the following section.

4.1.4. Checking insertion feasibility of special customers

Whenever a special customer is selected to be added into the current solution, it will be inserted in two positions: one in regular route and the other in a special route. Multiple insertion operations make it impossible to use the maximum delay for feasibility verification purposes. However, we can still utilize the computed maximum delays to quickly filter out infeasible insertions as follows.

As mentioned above, after solving each model to obtain the maximum delay on an arc $(j-1,j)$, the service start times of vertices $j-1$ and j or, in other words, the values of τ_{j-1} and τ_j are saved. τ_{j-1} can be seen as the earliest time vertex $j-1$ can be serviced while τ_j can be seen as the latest time vertex j can be serviced. To avoid misunderstanding these notations, we denote τ_{j-1} as et_{j-1} and τ_j as lt_j. Using these saved values, we can quickly check if special vertex i and its mirror m_i cannot be inserted on arc $a_s = (j_s - 1, j_s)$ of a special route and arc $a_r = (j_r - 1, j_r)$ of a regular route, respectively. First, the lower bound (denoted by lb) and upper bound (denoted by ub) of arrival times at m_i when being inserted on arc a_r and i when being inserted on arc a_s are computed as follows:

$lb_{m_i} = et_{j_r-1} + h^r_{j_r-1} + t^r_{j_r-1,m_i}$
$ub_{m_i} = lt_{j_r} - h^r_{m_i} - t^r_{m_i,j_r}$
$lb_i = et_i + h^s_{j_s-1} + t^s_{j_s-1,i}$
$ub_i = lt_{j_s} - h^s_i - t^s_{i,j_s}$.

It can be seen that the synchronization constraints will be violated in the following two cases:

$$lb_i - ub_{m_i} > \beta_i \qquad \text{or} \qquad ub_i - lb_{m_i} < -\alpha_i \tag{38}$$

The insertions of i and m_i also need to satisfy the time window constraint at vertex m_i and the maximum delays computed from the model (F2). Thus, we can utilize this property to rapidly verify the feasibility of insertions. Unlike regular vertices, possible waiting times at i and m_i are created not only by time window constraints, but also by synchronization constraints. The lower bounds of waiting times created by time window ($waitTime^{tw}_j$) and synchronization ($waitTime^{sync}_j$) at a vertex j can be computed as follows:

$waitTime^{tw}_{m_i} = \max(0,\, o_{m_i} - lb_{m_i})$
$waitTime^{tw}_i = 0$
$waitTime^{sync}_{m_i} = \max(0,\, lb_i - \beta_i - lb_{m_i})$
$waitTime^{sync}_i = \max(0,\, lb_{m_i} - \alpha_i - lb_i)$

Figure 1 illustrates the computation of waiting times when $\alpha = 0$ and $\beta = 10$. In Figure 1a, the vehicle arrives at the customer location before the time window and has to wait 30 minutes before starting delivery. In Figure 1b, the special vehicle arrives before the regular vehicle, but it has to wait until the regular vehicle starts to service the customer because the value of α is zero. Thus, the waiting time due to the synchronization constraint, in this case, is 45 minutes.

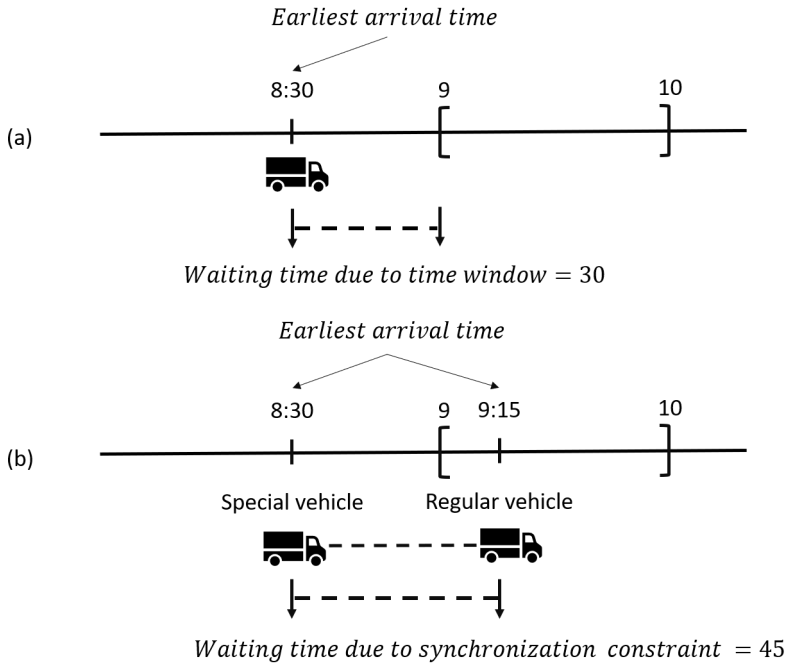

Figure 1: Computing waiting times due to time window and synchronization constraints

Hence, we can calculate the lower bounds of waiting times at vertex j by taking the maximum value between $waitTime_j^{tw}$ and $waitTime_j^{sync}$:

$waitTime_{m_i} = \max(waitTime_{m_i}^{tw}, waitTime_{m_i}^{sync})$
$waitTime_i = \max(waitTime_i^{tw}, waitTime_i^{sync})$

After all the values above are calculated, we can skip the insertions which violate one of the following constraints:

$$t^r_{(j_r-1)m_i} + t^r_{m_i j_r} - t^r_{(j_r-1)j_r} + waitTime_{m_i} + h^r_{m_i} \leq \delta_{(j_r-1,j_r)} \tag{39}$$

$$t^s_{(j_s-1)i} + t^s_{ij_s} - t^s_{(j_s-1)j_s} + waitTime_i + h^s_i \leq \delta_{(j_s-1,j_s)} \tag{40}$$

$$lb_{m_i} \leq u_{m_i} \tag{41}$$

Constraints (39) and (40) verify if the insertions satisfy the maximum delays while constraint (41) checks the time window at the regular vertex m_i. In addition, based on the characteristic of the k-regret heuristics, we can only consider the insertions if their cost is smaller than the current k-th best insertion cost.

It is worth mentioning that validation procedures (38)-(41), which run in $\mathcal{O}(1)$ if the values of the variables of each program (F2) are available, can detect plenty of infeasible insertions, thus saving a lot of run time of the algorithm. A preliminary experiment

on instances with 50 customers and 25 synchronizations shows that our fast validation procedures help to increase the algorithm's speed by up to 20 times. This ratio increases on larger instances with more synchronizations. However, if the insertion passes all the validations, we cannot ensure that the insertion is indeed feasible w.r.t the time windows and synchronization constraints. As a consequence, whenever the insertion of a special request passes the checks above, we must still examine if it does not make the solution infeasible. The following LP model (F3), which does not have any objective function, is used to check if a special vertex i can be added in arc $a_r = (j_r - 1, j_r)$ on a regular route and arc $a_s = (j_s - 1, j_s)$ on a special route:

$$\text{(F3)} \quad \textbf{Subject to} \quad o_p \le \tau_p \le u_p \qquad \forall p \in \overline{V_r} \cup \{m_i\} \tag{42}$$

$$-\alpha_p \le \tau_p - \tau_{m_p} \le \beta_p \qquad \forall p \in \overline{V_s} \cup \{i\} \tag{43}$$

$$\tau_{p-1} + h^l_{p-1} + t^l_{(p-1,p)} \le \tau_p \qquad \forall (p-1,p) \in A_l \setminus \{a_l\}, l \in \{r,s\} \tag{44}$$

$$\tau_{j_r-1} + h^r_{j_r-1} + t^r_{(j_r-1,m_i)} \le \tau_{m_i} \tag{45}$$

$$\tau_{m_i} + h^r_{m_i} + t^r_{(m_i,j_r)} \le \tau_{j_r} \tag{46}$$

$$\tau_{j_s-1} + h^s_{j_s-1} + t^s_{(j_s-1,i)} \le \tau_i \tag{47}$$

$$\tau_i + h^s_i + t^s_{(i,j_s)} \le \tau_{j_s} \tag{48}$$

$$\tau_p \ge 0 \qquad \forall p \in \overline{V_r} \cup \overline{V_s} \cup \{i, m_i\} \tag{49}$$

The meaning of variables τ, other parameters, and constraints (42)-(44) can be derived from (F1) and (F2). Constraints (45)-(48) are similar to (44), but written for 4 new arcs that are added by the insertion of vertices i and its mirror m_i: $(j_r - 1, m_i)$, (m_i, j_r), $(j_s - 1, i)$, and (i, j_s).

4.2. Removal operators

The destroy operators remove a fraction of vertices from a complete solution based on different criteria, each guiding the algorithm to another search space. The input of the operators is a complete solution *sol* and their outputs are nb_{rm} vertices that have been removed from *sol*. We use three destroy operators originally proposed by [18]: random removal, related removal and worst removal.

4.2.1. Random Removal

This is the simplest removal operator. It randomly selects nb_{rm} vertices in the solution and removes them. Other vertices remain unchanged. This obviously helps the algorithm diversify the search.

4.2.2. Related Removal

The idea of the related removal, as its name indicates, is to remove similar vertices with the expectation that they could interchange their positions to create a better solution. More specifically, to measure the similarity between two vertices i and j, we use the *relatedness* R_{ij} which is calculated as follows:

$$R_{ij} = \lambda_1 \frac{|\tau_i - \tau_j|}{maxTime} + \lambda_2 \frac{d_{ij}}{maxDis} + \lambda_3 \frac{|q_i - q_j|}{maxDem} + \lambda_4 |type_i - type_j| \tag{50}$$

To calculate the relatedness of two vertices, we take into account the differences of four characteristics: their service start times (τ); the distance between them (d_{ij}); their

demand size; and their type. Note that the value of τ is obtained from solving a program of type (F3) whenever a new complete solution is found; and $type_i$ is set to 1 if vertex i is special, and to 0 if it is regular. The difference is normalized such that it only takes values from the interval [0, 1]. In this formula, $maxTime$, $maxDis$, and $maxDem$ indicate the largest service start time, the largest distance, and the largest demand of all vertices in the solution, respectively. In addition, each characteristic i is associated with a weight λ_i to measure its importance.

4.2.3. Worst Removal

The worst removal operator removes the vertices that are very expensive, with the expectation that these vertices might be located in wrong places. Given a request i served by some vehicle in a solution sol, we define the cost of the vertex Δ_i as the difference between the cost of sol and the cost of the new solution when vertex i is removed from sol. The worst removal heuristic repeatedly chooses a vertex i with the largest cost Δ_i until nb_{rm} vertices have been removed.

In order to add more diversification to our algorithm, the related and worst removal operators will not always remove the most related (or expensive) request. Instead, they are randomized by removing the $\lfloor y^{p_r}|R|\rfloor$-th most related (or expensive) request where R is the set of vertices in the solution and y is a random number in [0, 1], and parameter p_r is used to control the randomization. If p_r is small, the most related (in the case of related removal) or expensive (in the case of worst removal) vertex is selected, while less related (or expensive) vertices may be chosen for larger values of p_r with a probability that decreases with the cost Δ_i. The values of p_r are taken from [18].

Finally, our lp-ALNS also uses acceptance criteria embedded in a simulated annealing framework, adaptive score adjustment to select operators in a dynamic fashion, and adding noise to insertion cost to increase diversification. All these components and their parameter settings are taken from [18] without any change.

5. Computational results

This section aims to 1) compare the effectiveness of the approaches: MIP model, CP model, lp-ALNS, and cp-ALNS on small-instances, 2) compare the performance of lp-ALNS and cp-ALNS on all available instances, and 3) assess the quality of lp-ALNS when it is adapted to solve the VRPTWSyn. For the first and second experiments, we test the methods on the instances proposed in [8] with the number of customer $|V| = 25$, 50, 100, and 200. These instances were generated from the VRPTW instances of [24, 9] and are of three types, depending on the customers' distribution. The customers are randomly located in the instances of type R, clustered in type C, and both randomly located and clustered in type RC. The instances are also categorized into two classes based on the capacity of vehicles. The first class (which includes C1, R1, RC1) consists of instances with a relatively small capacity Q compared to the total customer demand, while in the second class (C2, R2, RC2), the capacity is relatively large. Note that in the instances of types R and RC, the vertices are identically distributed in classes 1 and 2, while this is not true for type C. The travel time and travel cost between two vertices are set to the Euclidean distance. The original VRPTW instances are transformed into VRPSC instances as follows: the number of special customers $|V_s|$ is set to $\lceil n_s.|V|\rceil$ where n_s is the percentage of special customers. There are three values for n_s: 5%, 25%, and 50%. More precisely, the first customer in the VRPTW instances is considered a special customer

and the next special customers are selected using a constant interval defined by $\frac{1}{n_s}$. In the case of the synchronization constraint, the values of α_i and β_i are set to 0 and 10 for every special customer $i \in V_s$, respectively. Finally, we found out from private contact that inexact results were reported in [8] for three instances C101, C105 and C106 with 25 customers and 2 synchronizations, so we removed these instances from our experiments.

For the third experiment, we use the standard instances in [2]. The benchmark which is generated to simulate the scheduling problem in home care services, includes 10 data sets. The number of customers is selected from three values: 20, 50, and 80. There are five types of time windows of increasing sizes: fixed to a single value (F), small (S), medium (M), large (L) and no (arbitrarily large) time windows (A) where a larger time window covers a smaller one. In each instance, about 10% of the customers need to be pairwise synchronized.

The MIP and CP models are coded in IBM OPL 12.8.0 while the lp-ALNS is implemented in C++ using CPLEX 12.8.0 for the solution of the linear programs. All the methods were run on a 2.10GHz Intel E5-2683. Note that the cp-ALNS in [8] which is used to provide comparison with our algorithm was run on a 3.07GHz Xeon(R) X5675, which is similar to our processor. Since different CPU speed conversion techniques can provide very different results, we decided to present the raw running times, letting the readers choose their preferred approach. The parameter setting of lp-ALNS is chosen empirically. We have tested many settings and the following setting gives the best performance, in terms of both quality and computational time for our algorithm. The number of removed vertices in the removal operators nb_{rm} is a random integer between 4 and $\min(40, \lfloor 0.4 * |V| \rfloor)$. In related removal operator, the values of λ_1, λ_2, λ_3, and λ_4 are set to 4, 2, 1, and 4, respectively. The lp-ALNS stops after 25 000 iterations. All the detailed results can be found in http://www.orlab.com.vn/home/download.

5.1. Comparison of CP model, MIP model, lp-ALNS, and cp-ALNS on the small-size instances

In the first experiment, we compare the results obtained by our MIP model, CP model and lp-ALNS with those of cp-ALNS, as reported in [8]. Because our CP and MIP models cannot handle the large instances, we only consider instances with 25 customers in this case. The limited running time of the CP approach for each instance is set to 5 minutes and 3 hours. The first value aims to verify the capability of finding good solutions in a short running time while the second value aims to investigate if the method can solve these instances to optimality. Table 1 shows the number of times each one of our methods finds a better solution (Columns "Better"), an equal solution (Columns "Equal"), or a worse solution (Columns "Worse") compared to cp-ALNS. The columns "Gap" report the average gaps (in percentage) between the solution costs of our methods and those of cp-ALNS. The negative values in these columns indicate that our methods provide better solutions in terms of objective function values. The results obtained show that although our CP model-based algorithms cannot solve any instance to optimality in 3 hours, they do provide quite good solutions. Remarkably, the CP model performs better than cp-ALNS on 56 instances in a much shorter running time (5 minutes vs a couple of hours of cp-ALNS). It can be observed that CP models work better on the instances of the first class (C1, R1, and RC1) and worse on the instances of second class. The instances with shorter routes tend to be easier for our CP model. The results clearly show that our lp-ALNS is the most efficient. It is the most efficient method in terms of solution quality, as it provides 128 better solutions compared to cp-ALNS and is worse on only 7 out of

Data	Sync	lp-ALNS				CP (5 min)				CP (3h)			
		Better	Equal	Worse	Gap	Better	Equal	Worse	Gap	Better	Equal	Worse	Gap
R1	2	12	0	0	-3.22	9	0	3	-0.82	11	0	1	-2.30
R2		11	0	0	-2.49	1	0	10	10.18	1	0	10	9.69
C1		6	0	0	-2.16	6	0	0	-1.45	5	0	1	-1.33
C2		2	6	0	-0.83	1	3	4	2.72	1	3	4	2.72
RC1		7	1	0	-0.79	3	0	5	5.24	5	0	3	1.05
RC2		4	3	1	0.12	0	0	8	9.77	0	0	8	10.92
R1	7	12	0	0	-4.59	7	0	5	6.36	11	0	1	-2.37
R2		8	0	3	-1.98	0	0	11	13.41	0	0	11	9.57
C1		6	3	0	-2.25	5	0	4	-1.68	5	0	4	-1.95
C2		3	5	0	-0.49	2	3	3	4.00	2	3	3	4.00
RC1		8	0	0	-2.80	5	0	3	3.53	6	0	2	-0.95
RC2		7	0	1	-2.93	1	0	7	9.93	0	0	8	8.95
R1	13	12	0	0	-3.82	6	0	6	7.10	8	0	4	-1.02
R2		9	1	1	-2.74	0	0	11	12.14	0	0	11	8.60
C1		5	3	1	-1.78	3	0	6	1.42	4	0	5	-0.48
C2		2	6	0	-1.84	4	0	4	3.23	4	0	4	2.84
RC1		8	0	0	-1.72	3	0	5	4.58	4	0	4	3.35
RC2		6	2	0	-2.25	0	0	8	15.03	0	0	8	14.60

Table 1: CP and lp-ALNS vs cp-ALNS on 25-customer instances

165 instances. Moreover, the gaps, on average, are negative on all instance classes (except RC2). Our lp-ALNS can improve the objective values on average by up to 4.59% (class R1).

We now compare the solutions of lp-ALNS with those of the exact MIP-based method. The running time of the exact method for each instance is set to 3 hours and the default settings of CPLEX are used. Table 2 shows, for each class of instances, the number of instances successfully solved to optimality (Column "#opt"), the average running time in seconds (Column "Time"), and the average gap returned by CPLEX (Column "Gap") of the MIP method. The next columns report the number of times lp-ALNS finds a better solution (Column "Better"), an equal solution (Column "Equal"), or a worse solution (Column "Worse") compared to cp-ALNS. The last column "Gap" shows the average gaps (in percentage) between the solution costs of lp-ALNS and those of MIP. The negative values in these columns indicate that lp-ALNS provides better solutions than MIP in terms of objective function values.

We observe that the MIP model can solve 63 out of 168 instances to optimality. The gap values in the last column of Table 2 are never positive, showing the good performance with regard to solution quality of lp-ALNS compared with the MIP method. More precisely, lp-ALNS provides 67 better solutions and 93 equal solutions while the MIP model performs better on only 8 instances.

5.2. Comparison of lp-ALNS and cp-ALNS on all the instances of [8]

In the second experiment, we investigate the performance of lp-ALNS on all instances. The computational results are summarized in Figures 2 and 3, and Table 4 in the Appendix. Figure 2 shows that our algorithm clearly dominates cp-ALNS in terms of solution quality. It provides better average results on all instance classes. Figure 3 reports the num-

Data	Sync	MIP			lp-ALNS			
		#opt	Time (s)	Gap	Better	Equal	Worse	Gap
R1	2	3	4,877	13.38	8	4	0	-2.17
R2		3	8,333	8.54	4	6	1	-0.94
C1		6	5,074	5.09	0	9	0	0.00
C2		6	3,585	3.24	0	8	0	0.00
RC1		3	7,823	21.47	1	7	0	-0.03
RC2		3	7,517	22.32	2	5	1	-0.57
R1	7	3	7,105	11.30	9	3	0	-2.33
R2		2	8,072	9.52	5	3	3	-0.75
C1		5	4,347	7.81	2	7	0	-1.33
C2		5	4,303	4.78	2	6	0	-1.70
RC1		3	7,999	20.53	2	6	0	-0.33
RC2		2	7,607	22.62	3	4	1	-0.65
R1	13	3	6,832	15.05	8	3	1	-1.81
R2		2	9,099	18.01	7	3	1	-3.43
C1		5	4,736	13.52	4	5	0	-2.58
C2		6	4,503	6.15	1	7	0	-1.25
RC1		2	8,412	33.62	3	5	0	-0.56
RC2		1	9,314	35.09	6	2	0	-3.98

Table 2: MIP vs lp-ALNS on 25-customer instances

ber of new best-known solutions found by lp-ALNS for each class of instances. A total of 620 best-known solutions have been found with our lp-ALNS.

Moreover, Table 4 in the Appendix shows that the improvement on the objective value obtained by lp-ALNS when comparing the initial and final solutions is significant, especially on large instances (up to 16.75 %). The relatively high gap between the final and initial solutions shows the efficiency of construction and deconstruction operators. However, similar to cp-ALNS, the computation time of our algorithm is still high. It depends heavily on the number of customers $|V|$ and the number of special customers $|V_s|$. More specifically, in these cases, the number of variables and constraints in the LP models increase rapidly, leading to larger programs which are harder to solve.

5.3. Comparison of lp-ALNS with the state-of-the-art approaches for the VRPTWSyn

Finally, we have also adapted our lp-ALNS to available benchmark instances of the VRPTWSyn from the literature [2] with the number of customers ranging from 20 to 80. Because LP models are flexible to modify or incorporate constraints, the adaptation is simple and does not take much effort in implementation. Also, the algorithm is executed without any modification in settings and parameters. We compare our heuristic method to the current best existing heuristics proposed in Bredström and Rönnqvist [2] (BR08), Afifi etl al. [1] (ADM16), Parragh and Doerner [15] (PD18), and Liu et al. [13] (LTX19). These results are reported in Table 3. It contains the following information for each instance: the number of customers (n), the gaps in percentage between the best solutions produced by each method and those of lp-ALNS as well as the gaps in percentage between the best known solutions over all methods (column BKS) and those of lp-ALNS. A positive gap means that our method provides a better solution than the other, and vice versa. The

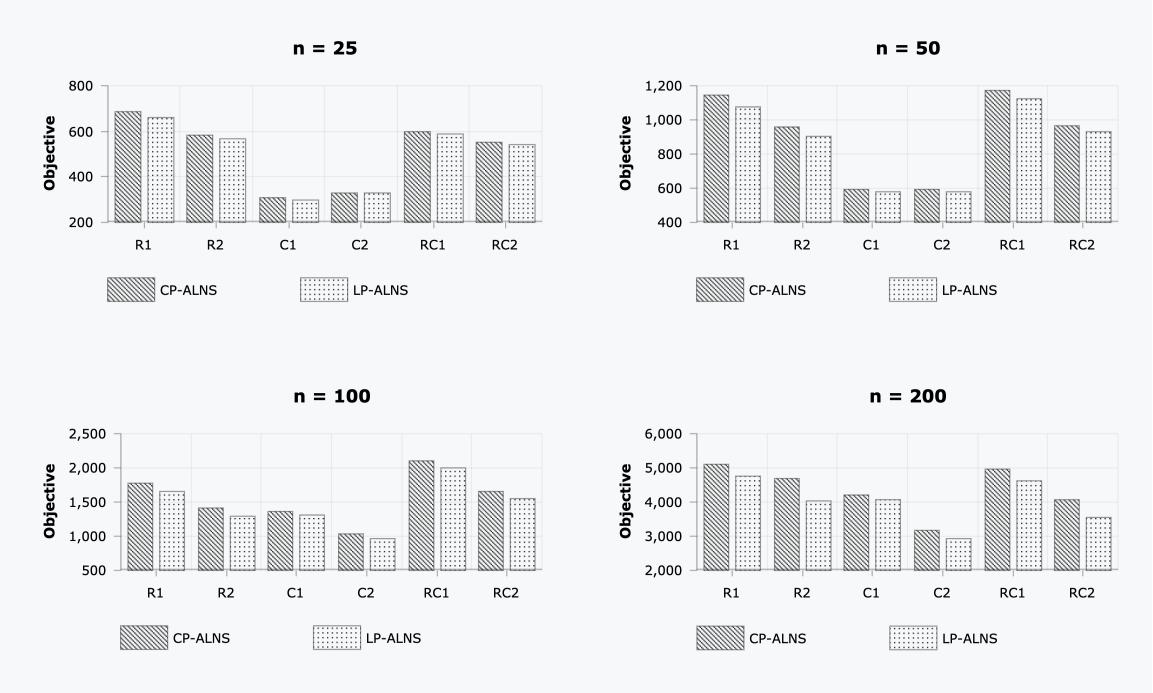

Figure 2: Comparison between lp-ALNS and cp-ALNS in terms of objective values on average

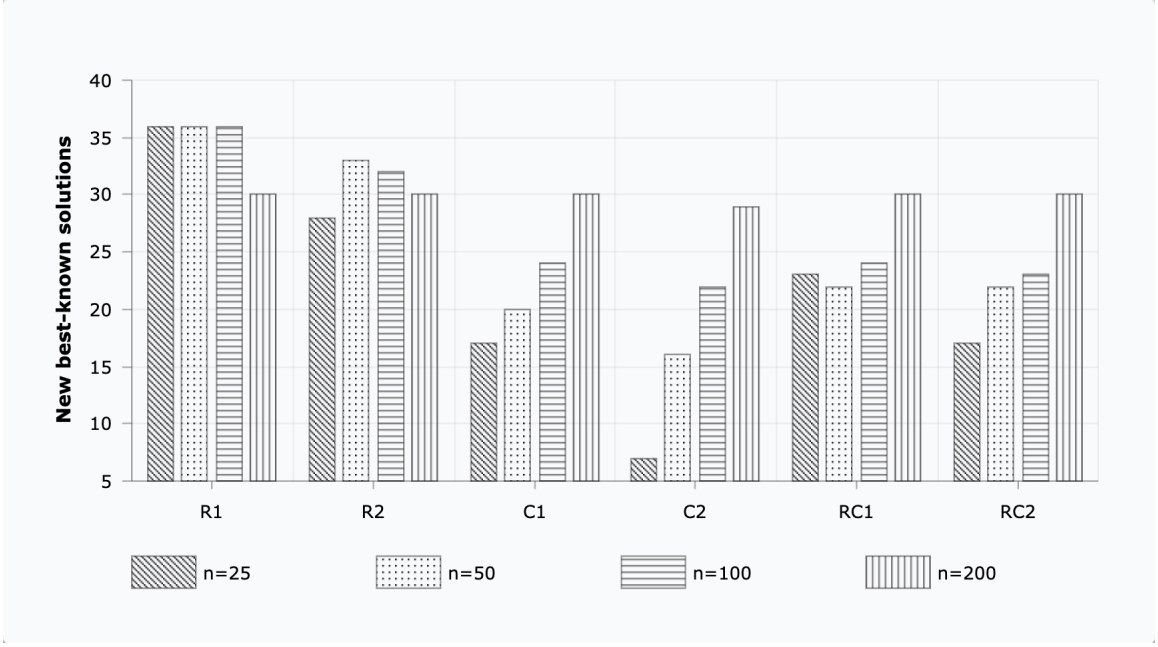

Figure 3: Number of new best-known solutions found by lp-ALNS

results as well as the running time in seconds of our method are reported in columns "Obj." and "Time (s), respectively. In several cases, no results are reported in the literature for some methods on some instances. These cases are indicated by a "-". New best known solutions found by lp-ALNS are marked in bold.

Despite the fact that lp-ALNS is designed mainly to solve the VRPSC and not the VRPTWSyn, our method still provides good solutions for the VRPTWSyn. It dominates three approaches: BR08, ADM16, and PD18 with regard to the average gap. It provides slightly worse solutions than LTX19 and BKS. However, the average gaps are quite small (-0.13 and -0.01, respectively). In particular, our method provides four new best results for instances 6A, 8A, 9A, and 10A. On the other hand, our method is quite slow as it takes about 10 hours to solve the instances with 80 customers.

6. Conclusion

In this research, we study an important variant of the vehicle routing problem with synchronization constraints, which has numerous real-world applications. We propose a new CP model and an ALNS algorithm. The most remarkable feature of our ALNS is that we use linear programming to check the feasibility of insertions. A number of acceleration techniques have been proposed to significantly reduce the computation time of the algorithm. The obtained results on benchmark instances from the literature show the good performance of our method. Our CP model can even provide better solutions than a CP-based ALNS for small instances in much shorter running times. Our lp-ALNS dominates cp-ALNS in terms of solution quality, by producing 620 new best-known solutions out of 681 instances, with improvement gaps that are relatively high. We also apply the method to solve the VRPTWSyn, a problem that is well studied in the literature. The tests on benchmark instances show that our metaheuristic provides good solutions as state-of-the-art algorithms. We also improve four best known solutions.

The research perspectives are numerous. First, although our CP model provides very good solutions on small instances, it cannot prove any of them optimal. This observation, combined with the fact that there is no efficient exact method so far to solve the VRP with synchronization constraint, indicates that this class of problems is hard to solve. The development of an efficient exact method is still an open question. Second, we believe that our lp-ALNS can be a used as a general framework, since it is easy to incorporate other constraints into the LP models. Thus, applying our method to solve other hard variants of VRPs could be an interesting research direction. Finally, our lp-ALNS is still quite time-consuming. Other acceleration techniques exploiting the special structures of the LP programs that validate insertion feasibility are required to obtain an efficient general solver for VRPs with rich attributes.

Acknowledgment

We thank Ngan Ha Duong (VNU University of Engineering and Technology) for performing a part of experiments. We also appreciate the two anonymous reviewers for their insightful comments and suggestions.

Instance	n	BR08	ADM16	PD18	LTX19	BKS	lp-ALNS	
							Obj.	Time (s)
1F	20	0.00	-	0.00	-	0.00	5.13	1,337
1S	20	0.00	0.00	0.00	0.00	0.00	3.55	1,366
1M	20	0.00	0.00	0.00	0.00	0.00	3.55	1,295
1L	20	0.00	0.00	0.00	0.00	0.00	3.39	1,332
1A	20	7.12	-	0.00	-	0.00	2.95	1,290
2F	20	0.00	-	0.00	-	0.00	4.98	1,575
2S	20	0.00	0.00	0.00	0.00	0.00	4.27	1,632
2M	20	0.00	0.00	0.00	0.00	0.00	3.58	1,365
2L	20	0.00	0.00	0.00	0.00	0.00	3.42	1,627
2A	20	15.97	-	0.00	-	0.00	2.88	1,443
3F	20	0.00	-	0.00	-	0.00	5.19	1,503
3S	20	0.00	0.00	0.00	0.00	0.00	3.63	1,371
3M	20	0.00	0.00	0.00	0.00	0.00	3.33	1,401
3L	20	0.00	0.00	0.00	0.00	0.00	3.29	1,407
3A	20	13.14	-	0.00	-	0.00	2.74	1,540
4F	20	0.00	-	0.00	-	0.00	7.21	1,357
4S	20	0.00	0.00	0.00	0.00	0.00	6.14	1,455
4M	20	1.41	0.00	0.00	0.00	0.00	5.67	1,481
4L	20	3.31	0.00	0.00	0.00	0.00	5.13	1,422
4A	20	14.45	-	0.00	-	0.00	4.29	1,311
5F	20	0.00	-	0.00	-	0.00	5.37	1,433
5S	20	0.00	0.00	0.00	0.00	0.00	3.93	1,586
5M	20	0.00	0.00	0.00	0.00	0.00	3.53	1,696
5L	20	0.00	0.00	0.00	0.00	0.00	3.34	1,494
5A	20	16.01	-	0.00	-	0.00	2.81	1,583
6F	50	-	-	0.00	-	-0.07	14.46	9,168
6S	50	68.18	0.00	0.00	0.00	0.00	8.14	9,909
6M	50	66.23	0.00	0.13	0.00	0.00	7.70	9,637
6L	50	66.25	0.00	0.00	0.00	0.00	7.14	9,149
6A	50	104.83	-	2.59	-	2.59	**5.80**	9,313
7F	50	-	-	0.00	-	0.00	13.02	9,213
7S	50	79.50	0.00	0.00	0.00	0.00	8.39	11,110
7M	50	79.81	0.00	0.00	0.00	0.00	7.48	10,723
7L	50	67.44	0.00	0.00	0.00	0.00	6.88	9,594
7A	50	117.34	-	0.00	-	0.00	5.71	9,186
8F	50	-	-	-	-	-	-	-
8S	50	-	-0.42	-0.42	-0.42	-0.42	9.58	9,889
8M	50	-	0.00	0.00	0.00	0.00	8.54	9,841
8L	50	89.03	-0.25	0.12	0.00	-0.25	8.02	9,958
8A	50	99.85	-	0.15	-	0.15	**6.51**	6,413
9F	80	-	-	-	-	-	-	-
9S	80	-	-0.17	1.00	0.00	-0.17	11.95	38,787
9M	80	-	-0.36	0.00	-0.36	-0.36	10.96	32,461
9L	80	95.09	-1.04	-0.47	-1.60	-1.60	10.60	36,283
9A	80	172.50	-	1.31	-	1.31	**8.40**	33,710
10F	80	-	-	0.50	-	0.00	12.08	30,825
10S	80	88.62	-0.12	-0.70	-0.81	-0.81	8.61	30,296
10M	80	99.87	-0.65	0.00	-0.52	-0.65	7.67	35,538
10L	80	138.62	5.01	0.00	-0.27	-0.27	7.38	32,791
10A	80	179.21	-	0.16	-	0.16	**6.30**	33,667
Average		41.07	0.07	0.09	-0.13	-0.01		

Table 3: Comparison of lp-ALNS with other metaheuristics proposed for the VRPTWSyn

References

[1] Sohaib Afifi, Duc-Cuong Dang, and Aziz Moukrim. Heuristic solutions for the vehicle routing problem with time windows and synchronized visits. *Optimization Letters*, 10(3):511–525, 2016.

[2] David Bredström and Mikael Rönnqvist. Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. *European Journal of Operational Research*, 191(1):19–31, 2008.

[3] Anders Dohn, Matias Sevel Rasmussen, and Jesper Larsen. The vehicle routing problem with time windows and temporal dependencies. *Networks*, 58(4):273–289, 2011.

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

[5] Martin Fink, Guy Desaulniers, Markus M. Frey, Ferdinand Kiermaier, Rainer Kolisch, and François Soumis. Column generation for vehicle routing problems with multiple synchronization constraints. *European Journal of Operational Research*, 272(2):699–711, 2019.

[6] Khaled Ghedira. *Constraint Satisfaction Problems: CSP Formalisms and Techniques.* John Wiley & Sons, 2013.

[7] Vikas Goel, M. Slusky, W.-J. van Hoeve, Kevin C. Furman, and Yufen Shao. Constraint programming for LNG ship scheduling and inventory management. *European Journal of Operational Research*, 241(3):662–673, 2015.

[8] Hossein Hojabri, Michel Gendreau, Jean-Yves Potvin, and Louis-Martin Rousseau. Large neighborhood search with constraint programming for a vehicle routing problem with synchronization constraints. *Computers & Operations Research*, 92:87–97, 2018.

[9] J. Homberger and H. Gehring. Two evolutionary metaheuristics for the vehicle routing problem with time windows. *INFOR*, 37:297–318, 1999.

[10] G.A.P. Kindervater and Savelsbergh M.W.P. *Vehicle routing: handling edge exchanges*, pages 337–360. John Wiley & Sons, 1997.

[11] Nacima Labadie, Christian Prins, and Yanyan Yang. Iterated local search for a vehicle routing problem with synchronization constraints. In Begoña Vitoriano, Eric Pinson, and Fernando Valente, editors, *ICORES 2014 - Proceedings of the 3rd International Conference on Operations Research and Enterprise Systems, Angers, Loire Valley, France, March 6-8, 2014*, pages 257–263. SciTePress, 2014.

[12] Philippe Laborie. IBM ILOG CP optimizer for detailed scheduling illustrated on three problems. In Willem Jan van Hoeve and John N. Hooker, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009, Pittsburgh, PA, USA, May 27-31, 2009, Proceedings*, volume 5547 of *Lecture Notes in Computer Science*, pages 148–162. Springer, 2009.

[13] Ran Liu, Yangyi Tao, and Xiaolei Xie. An adaptive large neighborhood search heuristic for the vehicle routing problem with time windows and synchronized visits. *Computer & Operations Research*, 101:250–262, 2019.

[14] Andy M.Ham. Integrated scheduling of m-truck, m-drone, and m-depot constrained by time-window, drop-pickup, and m-visit using constraint programming. *Transportation Research Part C: Emerging Technologies*, 91:1–14, 2018.

[15] Sophie N. Parragh and Karl F. Doerner. Solving routing problems with pairwise synchronization constraints. *Central European Journal of Operations Research*, 26(2):443–464, 2018.

[16] Gilles Pesant, Michel Gendreau, Jean-Yves Potvin, and Jean-Marc Rousseau. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. *Transportation Science*, 32(1):12–29, 1998.

[17] V. Pillac, C. Guéret, and A. L. Medaglia. *A Fast Reoptimization Approach for the Dynamic Technician Routing and Scheduling Problem*, pages 347–367. Springer International Publishing, Cham, 2018.

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

[19] Matias Sevel Rasmussen, Tor Justesen, Anders Dohn, and Jesper Larsen. The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies. *European Journal of Operational Research*, 219(3):598–610, 2012.

[20] Louis-Martin Rousseau, Michel Gendreau, and Gilles Pesant. The synchronized dynamic vehicle dispatching problem. *INFOR Inf. Syst. Oper. Res.*, 51(2):76–83, 2013.

[21] Briseida Sarasola and Karl F. Doerner. Adaptive large neighborhood search for the vehicle routing problem with synchronization constraints at the delivery location. *Networks*, 75(1):64–85, 2020.

[22] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In Michael J. Maher and Jean-Francois Puget, editors, *Principles and Practice of Constraint Programming - CP98, 4th International Conference, Pisa, Italy, October 26-30, 1998, Proceedings*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Springer, 1998.

[23] IBM Software. Ibm ilog cplex optimization studio v12.6.3, 2015.

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

Appendix

Table 4 reports the comparison between lp-ALNS and cp-ALNS in terms of objective values on average. The three first columns represent the size, class name, and the number of special customers of each instance class. Columns 4 and 5 report the objective of solutions (Column "FinalObj") and running time in seconds (Column "RunTime") on average of the cp-ALNS. Next columns show the results on average for each instance class of our lp-ALNS. More precisely, "InitialObj" is the objective value of the initial solution constructed by regret-2 heuristic. "FinalObj" is the objective value of the final solution after the search is stopped. "RunTime" reports the computational time in seconds. Column "Imp%" shows the improvement of final solutions compared with initial ones. The final column "gap%" presents the gap between lp-ALNS and cp-ALNS solutions. A negative value means lp-ALNS provides a better solution.

Data			cp-ALNS		lp-ALNS				gap (%)
Size	Class	Sync	FinalObj	RunTime	InitialObj	FinalObj	RunTime	Imp%	
25	R1	2	544.2487	8406.7	667.7745	526.9739	1347.8	20.86%	-3.22%
	R2		458.2109	9978.2	594.1048	446.6154	1319.3	24.42%	-2.49%
	C1		246.9088	9255.3	371.2795	241.5390	1292.7	32.26%	-2.16%
	C2		286.4098	7643.6	342.2520	283.9314	1415.9	16.63%	-0.83%
	RC1		520.1329	6805.4	584.4385	515.7725	1480.5	11.23%	-0.79%
	RC2		485.1764	7672.3	624.6635	485.7430	1650.5	21.82%	0.12%
25	R1	7	692.9658	7846.3	888.4291	661.7204	2697.3	25.68%	-4.59%
	R2		587.3948	10289.6	753.2626	575.4996	3011.7	22.83%	-1.98%
	C1		318.3507	8304.2	485.0793	311.0783	2489.7	34.81%	-2.25%
	C2		325.0070	7308.5	411.1374	323.3936	2090.1	20.95%	-0.49%
	RC1		623.5873	6388.2	900.0518	605.3236	2707.9	31.69%	-2.80%
	RC2		573.4323	8828.2	796.9595	556.5850	2987.9	29.59%	-2.93%
25	R1	13	821.1631	7683.6	1113.333	790.7901	3989.3	28.71%	-3.82%
	R2		701.3905	10194.9	930.8132	681.8161	4523.5	26.21%	-2.74%
	C1		356.2599	6594.2	585.0626	349.7486	3481.7	39.33%	-1.78%
	C2		384.0774	6977.4	461.5924	376.7076	3511.1	17.86%	-1.84%
	RC1		657.5279	5892.3	1009.331	644.6960	3844.1	35.93%	-1.72%
	RC2		600.0163	8504.3	905.7041	586.7275	4460.4	33.84%	-2.25%
50	R1	3	909.5803	19518.2	1143.763	860.4969	4508.1	24.75%	-5.69%
	R2		747.6849	23888.9	999.7746	710.4967	4496.9	28.77%	-5.03%
	C1		460.1213	39268.2	554.8499	445.4759	4532.7	19.14%	-3.13%
	C2		476.6964	35246.2	633.3015	466.1748	4804.5	25.20%	-2.02%
	RC1		959.1396	17810.0	1165.249	932.1801	4539.8	19.44%	-2.98%
	RC2		788.3569	20046.5	1201.248	775.9619	4857.4	35.07%	-1.65%
50	R1	13	1178.034	19893.2	1508.424	1103.480	10834.1	26.81%	-6.57%
	R2		991.7101	23972.1	1352.777	931.9242	13253.3	30.88%	-6.03%
	C1		625.0501	18575.3	951.2536	607.6679	9646.9	35.13%	-2.70%
	C2		610.4964	22449.2	907.3918	603.0134	13477.1	31.94%	-1.20%
	RC1		1231.836	17850.4	1664.564	1167.640	11213.3	29.66%	-5.15%
	RC2		1014.296	22600.4	1629.658	975.5894	12489.1	39.79%	-3.96%
50	R1	25	1354.380	19456.9	1786.241	1259.770	18880.4	29.63%	-7.15%
	R2		1134.506	24311.6	1580.263	1057.765	26470.6	32.82%	-6.96%
	C1		689.7339	16392.7	1100.512	674.8498	18270.7	38.23%	-2.09%
	C2		697.0899	20694.2	1069.437	668.8730	23789.4	35.90%	-3.85%
	RC1		1331.455	17536.3	2064.870	1270.645	18259.1	38.31%	-4.30%
	RC2		1087.594	23088.4	1858.651	1041.364	21808.5	43.37%	-4.50%
100	R1	5	1429.048	82716.3	1847.463	1349.893	18128.8	26.94%	-5.87%
	R2		1114.927	107910.0	1587.720	1038.766	19082.2	34.41%	-6.98%
	C1		1017.524	41401.6	1493.110	1005.144	21153.7	32.45%	-1.20%
	C2		841.6708	75868.3	1156.707	792.8305	20527.3	30.57%	-5.74%
	RC1		1637.621	73411.3	2187.973	1575.960	19985.8	27.97%	-3.86%
	RC2		1286.366	97137.4	1983.771	1219.061	20101.6	38.63%	-5.47%
100	R1	25	1789.477	80930.0	2353.730	1667.065	39167.0	29.28%	-7.01%
	R2		1407.660	106302.7	2108.117	1303.016	55645.6	37.91%	-7.66%
	C1		1441.884	74134.6	2335.434	1396.531	38251.0	39.86%	-3.13%
	C2		1087.373	70059.3	1670.794	1004.834	39562.4	38.45%	-7.24%
	RC1		2146.339	71074.1	2985.869	2045.495	40795.9	31.44%	-4.73%
	RC2		1709.736	99529.7	2594.230	1610.915	51314.6	37.95%	-6.06%
100	R1	50	2092.569	72804.5	2831.534	1931.541	59256.5	32.07%	-7.91%
	R2		1693.735	104309.4	2526.176	1513.136	89195.5	40.16%	-10.91%
	C1		1609.269	72829.5	2911.811	1545.038	57106.8	46.57%	-3.91%
	C2		1180.336	54500.8	1927.523	1078.205	69467.0	41.82%	-8.42%
	RC1		2522.189	67991.7	3650.564	2361.475	60008.8	35.39%	-6.45%
	RC2		1945.313	95459.4	3173.071	1817.931	63274.9	42.44%	-6.83%
200	R1	10	4144.381	126805.8	5674.585	3893.875	37843.3	31.74%	-6.55%
	R2		3713.917	161933.9	5317.305	3320.119	37547.4	37.72%	-11.05%
	C1		3391.796	100174.6	5108.380	3299.623	39677.3	34.62%	-2.67%
	C2		2578.896	145855.5	3861.907	2364.314	36008.3	38.13%	-8.26%
	RC1		4088.110	138911.3	5642.754	3884.939	38358.8	31.03%	-4.95%
	RC2		3303.763	149639.7	4898.392	2917.336	39697.6	40.55%	-11.91%
200	R1	50	5074.646	136440.0	7240.237	4724.183	74670.6	35.08%	-7.46%
	R2		4769.591	163572.0	6608.872	4074.128	93781.0	38.47%	-14.99%
	C1		4238.077	120709.5	7290.512	4126.756	74785.5	43.11%	-2.65%
	C2		3254.876	161222.8	5851.064	3008.252	81829.4	48.25%	-7.49%
	RC1		4944.824	144934.9	7017.894	4573.909	78424.0	34.65%	-7.50%
	RC2		4039.977	155597.6	6108.987	3539.513	92402.6	41.75%	-12.49%
200	R1	100	6139.267	137336.0	8718.394	5704.425	116804.2	34.72%	-7.55%
	R2		5616.638	157111.4	7993.287	4699.330	154303.7	41.40%	-16.75%
	C1		4964.759	111991.6	9117.560	4757.915	112905.5	47.57%	-4.16%
	C2		3646.121	132923.9	6969.890	3374.519	146172.7	50.71%	-7.31%
	RC1		5836.127	148733.9	8265.386	5395.134	122680.9	34.72%	-7.58%
	RC2		4882.749	159016.6	7138.363	4204.200	153580.6	41.12%	-13.99%

Table 4: Comparison between lp-ALNS and cp-ALNS