AN ADAPTIVE LARGE NEIGHBORHOOD SEARCH ALGORITHM FOR VEHICLE ROUTING PROBLEM WITH MULTIPLE TIME WINDOWS CONSTRAINTS

BIN FENG, LIXIN WEI* AND ZIYU HU

Engineering Research Center of the Ministry of Education for Intelligent Control System and Intelligent Equipment Yanshan University, Qinhuangdao 066004, China

and

Key Lab of Industrial Computer Control Engineering of Hebei Province Yanshan University, Qinhuangdao 066004, China

(Communicated by Kok Lay Teo)

ABSTRACT. The Vehicle Routing Problem with Multiple Time Windows (VRPMTW) is a generalization of problems in real life logistics distribution, which has a wide range of applications and research values. Several neighborhood search based methods have been used to solve this kind of problem, but it still has drawbacks of generating numbers of infeasible solutions and falling into local optimum easily. In order to solve the problem of arbitrary selection for neighborhoods, a series of neighborhoods are designed and an adaptive strategy is used to select the neighborhood, which constitute the Adaptive Large Neighborhood Search (ALNS) algorithm framework. For escaping from the local optimum effectively in the search process, a local search based on destroy and repair operators is applied to shake the solution by adjusting the number of customers. The proposed method allows infeasible solutions to participate in the iterative process to expand the search space. At the same time, an archive is set to save the high-quality feasible solutions during the search process, and the infeasible solutions are periodically replaced. Computational experimental results on VRPMTW benchmark instances show that the proposed algorithm is effective and has obtained better solutions.

1. Introduction. Physical distribution routing problem is an important part of modern logistics system optimization which is also closely related to actual production and life[26]. Nowadays, customers put forward higher requirements for the timeliness of distribution. Specifically, customers expect their logistics providers to provide services for them within a certain period of time, which is the actual background of the vehicle routing problem with time windows (VRPTW)[22]. VRPTW mainly studies the delivery service to customers with lower cost under the condition of meeting time window constraint and other basic constraints[5]. The processing of time window constraint is one of the key problems to be solved[14]. The existing

2020 Mathematics Subject Classification. Primary: 90B06, 90B40; Secondary: 90C11.

Key words and phrases. Vehicle routing problem, multiple time windows, adaptive large neighborhood search, local search.

* Corresponding author: Lixin Wei.
research has achieved abundant results\cite{24} and extensive practical applications\cite{13}, but most focused on the case of single time window, that is, each customer has only one time window. In the actual delivery, the customer may provide multiple time windows, and the logistics providers need to select one of time windows for service. To deal with multiple time windows problem, it is necessary to solve both the time window constraint and the route optimization, so the existing approaches for single time window problem cannot be simply followed. VRPMTW is of great practical value and theoretical significance, several scholars have studied this problem and made some achievements.

Favaretto\cite{8} was the first to introduce VRPMTW and the objectives were to minimize the total route duration and the total traveled distance. An ant colony systems algorithm was proposed and solved new instances that were generated by adapting the classical VRP instances. They divided the time window into several visits but no further details were provided. Belhaiza\cite{1} proposed a Hybrid Variable Neighborhood Tabu Search(HVNTS) heuristic and considered minimizing the total duration as the objective. Belhaiza\cite{2} modeled the VRP aspect from the perspective of game theory and adopted a Pareto non-dominated based method. Belhaiza\cite{3} proposed a hybridization meta-heuristic called hybrid genetic variable neighborhood search heuristic algorithm (HGVNS). A genetic crossover operator heuristic was integrated into the shaking phase of VNS. The algorithm suffered from early convergence owing to the scarcity of an effective diversification strategy. Ferreira\cite{9} modified the VNS method based on \cite{3} algorithm framework. However, the algorithm only searched in feasible search space which lead to the resulting solutions lacked diversity, and the overall improvement in benchmark instances was only 0.02\%. This indicates that effective measures must be taken to deal with and use the infeasible solutions generated during the search process in order to obtain higher-quality solutions. Hoogeboom\cite{12} improved an effective variable neighborhood evaluation framework for VRPMTW. They embedded the start interval approach in their algorithm framework to recalculate the exact route duration when neighborhood operations are evaluated during the local search. When the local optimum didn’t improve for several iterations, a new solution was obtained by a shaking procedure with removing and reinserting r\% customers for the current solution. However, their shaking procedure lacked guidance for the selection of destroy and repair operators, which increased the computational complexity of the algorithm.

The existing research mainly adopted neighborhood search based methods, but there were many types of neighborhoods, and some needed to be executed multiple times to be effective. The order of neighborhoods for execution was predetermined, and it could not be effectively adjusted when the search procedure fell into a local optimum. In addition, the treatment and repair of infeasible solutions needed to be perfected. Since VRPMTW is an NP-hard combinatorial optimization problem, it is hard to access the optimal solution in effective time by using the exact algorithm, such as branch-and-cut\cite{17}, column generation\cite{10}, and branch-and-price\cite{18}. Many meta-heuristic algorithms have been successfully applied to Vehicle Routing Problem with Time Windows(VRPTW)\cite{25}\cite{6}. As an extension of VRPTW, meta-heuristic algorithms are mainly used to solve VRPMTW. The VRPMTW has series of constraints, which leads to complex and discontinuous solution space and relatively large and diverse neighborhood structure. In order to solve the above problems, an ALNS framework\cite{19} has been modified and additional components
have been added to enhance the optimization procedure. The ALNS framework is an improvement on LNS\cite{21} and particularly suitable for strictly constrained problems. At present, ALNS algorithm has been successfully applied to solve several variants of vehicle routing problems\cite{4}\cite{11}. However, \cite{21} only provides the framework of ALNS. When it comes to specific variant problems, additional components for dealing with specific problems or improving algorithm performance should be added on the basis of the original framework\cite{15}. In this paper, ALNS was adopted as the algorithm framework and according to the characteristics of VRPMTW, the mechanisms of maintaining diversity and dealing with infeasible solutions were added to improve the optimization effect. These components attempt to repair infeasible solutions in the process of executing destroy and repair operators, and take full advantage of available information during searching process in conjunction with the ALNS framework.

The proposed algorithm bring three main methodological contributions. Firstly, in order to solve the problem of large number of neighborhoods and lack of guidance for neighborhood selection, an adaptive strategy based on destroy and repair operators was introduced in this paper, and an ALNS algorithm framework was formed. The algorithm can select neighborhood operators more specifically, so as to enhance the search intensity of the algorithm. Secondly, improvements were made to increase the diversity of solutions. The destroy and repair operators had been modified to make them more targeted to multiple time window constraints. An acceptance criteria based on simulated annealing was used to expand the search space by accepting inferior solutions with probability. A local search procedure was set up, which can adaptively adjust the number of customer removal strategies. These measures could effectively search the solution space and enhance the diversity of algorithms. Thirdly, several strategies were used to handle with infeasible solutions during the iteration. Adding penalty terms to the objective function allowed the overload and time window to temporarily violate the constraints, thus better exploring the solution space. Set up a fixed size archive to hold historical optimal solutions, and restart periodically to replace infeasible solutions that occur multiple times in a row during iteration. The implementation converts an infeasible solution to a feasible solution.

The remainder of this paper is arranged as follows. Section 2 defines the problem and provides a mathematical model. Section 3 details the proposed ALNS method for VRPMTW . Section 4 introduces the test instances and details the computational experiments and analysis the proposed method. Finally, Section 5 summarizes the full text and makes a prospect of future research.

2. Problem definition and mathematical model. The VRPMTW can be defined as a directed graph. Let $G = (V, A)$ be a complete graph ,where $V$ represents the vertex set and $A$ is the arc set. The vertex set can be divided into $V = \{I, D\}$, where $I = \{1, ..., n\}$ is a set of customers and $D$ is a set of depots. Use $K$ to represent the number of vehicles participating in the delivery. Each vehicle $k \in K$ has a maximum carrying load equal to $Q_k > 0$. The vehicle $k$ needs to provide services for customers in subset $N_k$ after leaving the depot $d_k \in D = \{1, ..., \bar{d}\}$, and finally returns to $d_k$ to form a complete circle. Each customer $i \in I$ has a demand $q_i \geq 0$ and service time $s_i \geq 0$. Each customer has a set of $\bar{p}_i$ alternative delivery time windows $W_i = \{w_{i1}, ..., w_{ip_i}\}$, where $l_{pi}, u_{pi}$ are the lower and upper bounds of time window $w_{pi}$, and $0 < l_{1i} \leq l_{pi} < u_{pi} \leq u_{\bar{p}i} < \infty$. A customer can
Table 1. Definition of the parameters and variables

| Name | Description |
|------|-------------|
| $x_{ij}^k$ | binary variable, equal to 1 if and only if arc $(i, j)$ is traversed by vehicle $k$ |
| $y_{ij}^k$ | real variable, equals to the flow carried on arc $(i, j)$ |
| $r^k$ | binary variable, equal to 1 if and only if vehicle $k$ is used |
| $v_i^k$ | binary variable, equal to 1 if and only if customer $i$ is served within its time window $p$ |
| $z_i^k$ | binary variable, equals to 1 if and only if customer $i$ is assigned to vehicle $k$ |
| $w_i^k$ | real variable, waiting time of vehicle $k$ at customer $i$ |
| $d_k$ | route duration of vehicle $k$ |
| $a_i^k$ | arrival time of vehicle $k$ at customer $i$ |
| $b_i^k$ | departure time of vehicle $k$ from customer $i$ |
| $l_{ij}$ | travel time associated with the arc $(i, j)$ |
| $s_i$ | service time at customer $i$ |
| $q_i$ | demand of customer $i$ |
| $l_i^p$ | lower bound of time window $p$ at customer $i$ |
| $u_i^p$ | upper bound of time window $p$ at customer $i$ |
| $Q_k$ | capacity of vehicle $k$ |
| $D_k$ | maximum duration of the route of vehicle $k$ |
| $F^k$ | fixed cost in time units of using vehicle $k$ |
| $M$ | an arbitrary large constant |

only be visited by one vehicle. A vehicle should wait until $l_i$, to start delivery if it reaching a customer $i$ before $l_i$.

Each route $k$ of the vehicle corresponds to the total travel distance, total travel time and total travel duration $D_k$. The total duration is equal to the sum of service time, waiting time and travel time. The objective of VRPMTW is to minimize the total cost $f_1(X)$ of meeting all customer needs, where $X$ represents a solution of VRPMTW. The total cost $f_1(X)$ consists of four parts: the total travel time, the total waiting time, the total service time and the vehicle fixed cost, which are expressed in time units. The VRPMTW can be formulated as a mixed integer linear program. Variables and parameters used in formulations are defined as follows in Table 1:

$$
\begin{align*}
    \min f_1(X) &= \sum_{k \in K} \sum_{i<j} t_{ij} x_{ij}^k + \sum_{k \in K} \sum_{i \in I} w_i^k + \sum_{i \in I} s_i + \sum_{k \in K} F^k r^k \\
    \text{subject to} & \\
    & \sum_{k \in K} z_i^k = 1, i \in V, \quad (1) \\
    & \sum_{j \in V} x_{ij}^k = \sum_{j \in V} x_{ji}^k, i \in V \text{ and } k \in K, \quad (2) \\
    & \sum_{k \in K} \sum_{j \in V} a_i^k, i \in V, \quad (3) \\
    & \sum_{k \in K} \sum_{j \in V} a_i^k, i \in V, \quad (4) \\
    & y_{ij}^k \leq Q_k x_{ij}^k, i, j \in V \text{ and } k \in K, \quad (5) \\
    & b_i^k \geq l_d - M(1 - z_i^k), d \in D \text{ and } k \in K, \quad (6) \\
    & a_i^k \leq u_d + M(1 - z_i^k), d \in D \text{ and } k \in K. \quad (7)
\end{align*}
$$
annealing, it can be used as the new current solution for the next iteration. The generated by the above methods meets the acceptance criterion based on simulated customers at the inner loop of each iteration. If the new neighborhood solution of four neighborhood operators for reducing or eliminating the number of unserved segment, the operator scores will be updated. We perform a local search consisting entire search procedure is divided into multiple segments and at the end of each operators who performed well in the past through a roulette wheel mechanism. The these customers back. The algorithm adaptively selects those removal and insertion customers from the current solution and an insertion operator is selected to re-insert \( \psi \) of iteration, the current solution generated by adapted Minimum Cost Insertion (MCI) method. At the beginning described in Algorithm 1. The algorithm starts the search from an initial solution Framework of ALNS.

3. Adaptive large neighborhood search algorithm.

3.1. Framework of ALNS. The method of this paper follows an ALNS framework described in Algorithm 1. The algorithm starts the search from an initial solution generated by adapted Minimum Cost Insertion (MCI) method. At the beginning of iteration, the current solution \( S_{curr} \) and the best solution \( S_{best} \) are considered as the initial solution simultaneously. Then, a removal operator is chosen to delete \( \psi \) customers from the current solution and an insertion operator is selected to re-insert these customers back. The algorithm adaptively selects those removal and insertion operators who performed well in the past through a roulette wheel mechanism. The entire search procedure is divided into multiple segments and at the end of each segment, the operator scores will be updated. We perform a local search consisting of four neighborhood operators for reducing or eliminating the number of unserved customers at the inner loop of each iteration. If the new neighborhood solution generated by the above methods meets the acceptance criterion based on simulated annealing, it can be used as the new current solution for the next iteration. The constraints (1) ensures that each customer can only be served by exactly one vehicle. Constraints (2) guarantee that each route of vehicle \( k \) starts and ends at the depot, and that the number of arcs leaving the customer node \( i \) is equal to the number of arcs entering its node. Constraints (3) and (4) respectively guarantee that each customer \( i \) can only have one entering arc and one leaving arc visit. Constraints (5) asserts that the cumulative capacity of the vehicle \( k \) does not exceed \( Q_k \). Constraint (6) ensures that the time for the vehicle \( k \) to depart from the depot \( d \) does not exceed \( l_d \), and constraint (7) ensures that the time for vehicle \( k \) to arrive the depot is less than or equal to \( u_d \). Constraints (8) asserts that when the customer \( i \) is allocated to the vehicle \( k \), the departure time of vehicle \( k \) at customer \( i \) is at least equal to the time that the vehicle \( k \) arrives at the customer \( i \), plus the waiting time of vehicle \( k \) and the service time of customer \( i \). Constraint (9) indicates that the arrival time of the vehicle \( k \) at customer \( j \) is equal to the departure time of the vehicle \( k \) at customer \( i \) plus the travel time \( t_{ij} \) from customer \( i \) to \( j \), if the arc \((i, j)\) has been allocated to the vehicle \( k \). Constraints (10) and (11) assert that the arrival time of vehicle \( k \) plus its waiting time at customer \( i \) is in the time window \([l_i^p, u_i^p]\), when the customer \( i \) is served by vehicle \( k \) in the time window \( p \). Constraints (12) forces to select a unique time window for each customer \( i \). Constraints (13) assert that only in the case of using the vehicle, a given customer \( i \) can be served by the vehicle \( k \). Finally, constraints (14) and (15) set the non-negativity and binary conditions.

\begin{align*}
    b_i^k &\geq a_i^k + w_i^k + s_i - M(1 - z_i^k), \ i \in I \text{ and } k \in K, \\
a_j^k &\geq t_{ij} - M(1 - x_{ij}^k), \ i, j \in V \text{ and } k \in K, \\
a_i^k + w_i^k &\geq l_i^k - M(1 - z_i^k) - M(1 - v_i^p), \ i \in I, p \in W_i \text{ and } k \in K, \\
a_i^k + w_i^k &\leq u_i^p + M(1 - z_i^k) + M(1 - v_i^p), \ i \in I, p \in W_i \text{ and } k \in K, \\
\sum_{p=1}^{p_i} v_i^p = 1, &i \in I \cap D \\
r_k &\geq z_k^k, \ i \in V \text{ and } k \in K \\
y_{ij}^k, w_i^k, d_k, a_i^k, b_i^k &\geq 0, \\
r^k, x_{ij}^k, v_i^p, z_i^k &\text{ binary.}
\end{align*}
Algorithm 1 Pseudo-code of ALNS

**Input:** Removal operators \( D_h \), insertion operators \( R_h \), initial temperature \( T_0 \), cooling rate \( c \)

**Output:** A feasible solution \( S_{best} \)

1. Generate an initial solution \( S_{init} \) by using the Minimal Cost Insertion(MCI) algorithm(Section 3.2)
2. Initialize scores and probabilities for each destroy and repair operator
3. Let \( iseg, iter \) be iteration counters with initial value of 1
4. \( S_{curr} \leftarrow S_{init}, S_{best} \leftarrow S_{init}, T \leftarrow T_0 \)
5. for \( iseg = 1 : nsegs \) do
6. select the behavior of destroy and repair operators(Section 3.4)
7. for \( iter = 1 : nters \) do
8. if \( UCL \neq \emptyset \) then
9. apply LocalSearch for \( S_{curr} \) and update \( S_{curr} \)
10. end if
11. select one destroy operator and one repair operator
12. apply selected operators to \( S_{curr} \) then get a new solution \( S_{new} \)
13. if \( f(S_{new}) < f(S_{curr}) \), or \( S_{new} \) satisfies the acceptance criterion then
14. \( S_{curr} \leftarrow S_{new} \)
15. if \( S_{new} \) is feasible then
16. sent \( S_{new} \) into \( CFS\_Pool \) and replace the worst solution in it
17. if \( f(S_{new}) < f(S_{best}) \) then
18. \( S_{best} \leftarrow S_{new} \)
19. end if
20. end if
21. update performance scores of selected operators
22. end if
23. \( T \leftarrow Tc \)
24. end for
25. update the weights of ALNS moves and reset scores(Section 3.3)
26. if \( S_{curr} \) is infeasible and \( CFS\_Pool \) not empty then
27. send the best solution \( CFS\_Pool \) to \( S_{curr} \)
28. end if
29. end for

Stop criterion is the maximum number of iterations. For the sake of enhancing search capabilities, the proposed algorithm temporarily accepts infeasible solutions.

3.2. Initial solution construction. For purpose of generating an initial solution, we applied and improved a procedure called Minimal Cost Insertion(MCI). First pick a customer at random and insert it into the current solution. This position should minimize the incremental value of the cost \( f_1(X) \) without violating capacity and time window constraints. We set an upper bound \( K_{max} = \left\lceil \gamma \sum_{i \in I} q_i \sum_{k \in K} Q_k \right\rceil, \gamma \geq 1 \) for the number of vehicles that can be used at most. If there are still free vehicle, also called empty route, available in this procedure, then the customer can be inserted into a non-empty route or directly assigned to an additional empty route. If current customer cannot be inserted into any route, it will be placed in Unserved Customers List(UCL). Then, made attempts to insert the customer into current
solution or $UCL$ repeatedly until all customers have been assigned in. Try this method ten times and take the solution with the lowest number of customers in $UCL$ as the final initial solution.

3.3. **Adaptive weight and score adjustment.** Next we will explain in detail how to select these operators in each iteration of the ALNS algorithm. More details about five removal and three insertion operators will be introduced in Sections 3.4.
The algorithm destroys and repairs the current solution by choosing a removal and an insertion operator. Each operator $i$ has a score $p_i$ and a weight $w_i$. At the beginning of each segment, all operators have equal weights and all operators scores equal to 1. The score of removal and insertion operators evolution is managed as follows: Whenever a new global optimal solution is found, the score of the removal and insertion operator will increase by the value of $\sigma_4$. When there is a new solution that is better than the current solution and has not appeared before, the score of this operator will increase by $\sigma_3$. If the new solution improves the current solution, but the solution has appeared before, the score of this method is increased by $\sigma_2$. If the new solution meets the following three conditions: worse than the current solution, but accepted by acceptance criterion, and has never been encountered before, the score of this operator increases $\sigma_1$. Then based on these operators’ weights, the roulette wheel mechanism is applied to handpick removal and insertion operators.

This article divides the entire iteration process into several segments, and uses $\text{iseg}$ to represent a segment which contains several iterations. The purpose of setting iteration segment is that the weights of all removal and insertion operators remain unchanged in each iteration segment; and update as follows at the end of each segment: $w_{i+1} = (1 - \zeta)w_i + \zeta\pi_i/\phi_i$, where $\pi_i$ denotes the scores of removal and insertion operators, $\phi_i$ represents the number of times the operator has been executed in segment $\text{iseg}$, $\zeta$ is the adjustment parameter. $\zeta \in [0, 1]$ is a reaction factor that controls how quickly the algorithm reacts to the changes in scores of each destroy/repair operator. If the value $\zeta$ is set to 1, the selection only depends on the scores obtained during the iteration segment that just ends. If the value of $\zeta$ is set to 0, the selection only depends on the weight obtained in the previous iteration segments without considering the scores obtained during the iteration segment that just ends. It should be noted that two separate roulette wheel selections are performed here to distinguish the destroy and repair operators.

3.4. Modified destroy and repair operators.

3.4.1. Destroy operators. Five removal operators are used in this destroy phase. The task of destroy phase is to remove $\psi$ customers from current solution and send these customers into Removal List (RL). Please note that if there still has unserved customers in $UCL$ before destroy phase, these customers should be included in RL and cleared the $UCL$ followed. Next we will formally introduce the removal operators used in this implementation.

Worst Removal Deleting customers in current solution $S$ can result in a change upon the savings. This operator attempts to delete those customers that can bring the greatest savings. We define the deviation between the cost when a customer $i$ is in the solution and the cost when this customer is removed as the saving value: $\Delta_i = F(S) - F_{-i}(S)$, where $F_{-i}(S)$ is the cost of the solution without customer $i$. Those customers with higher saving values are more likely to be selected. The Worst Removal operator’s pseudocode detailed in Algorithm 2.

Basic Related Removal This operator was first proposed for the VRPTW[23] and next adapted for the PDPTW[21]. The principle of Basic Related Removal is to remove similar customers and to interchange them easily by the next repair method, specifically an insertion operator, from which generating a new solution. In many studies[16][7], the Shaw removal operator has been proved to be a very effective neighborhood operator, so this paper redesigned the Shaw removal operator to tackle the VRPMTW model. Then randomly select a customer and calculate how
Algorithm 2 Worst Removal

**Input:** current solution $S_{curr}$, number of customers to remove $\psi$;  
**Output:** new current solution $S'_{curr}$, removal list $RL$;

1: $S'_{curr} \leftarrow S_{curr}$, $C \leftarrow \emptyset$
2: **while** size of $RL \leq \psi$ **do**
3:  **for** each customer $i$ in $S'_{curr}$ **do**
4:     calculate $\Delta_i$
5:     $C \leftarrow C \cup \{\Delta_i\}$
6: **end for**
7: $\Delta_{max} \leftarrow$ the maximum value of $C$, remove the maximum value from $C$
8: $i = \text{argmax} \Delta_{max}$
9: $RL \leftarrow RL \cup \{i\}$
10: $S'_{curr} \leftarrow S'_{curr} \setminus \{i\}$
11: **end while**
12: return new current solution $S'_{curr}$, new removal list ($RL$)

Algorithm 3 Basic Related Removal

**Input:** current solution $S_{curr}$, unserved customers list $UCL$, number of customers to remove $\psi$;

**Output:** new current solution $S'_{curr}$, removal list $RL$;

1: randomly select one customer $i$ from $S_{curr}$, $S'_{curr} \leftarrow S_{curr}$
2: $RL = \{i\}$, $S'_{curr} \leftarrow S'_{curr} \setminus \{i\}$
3: **while** size of $RL \leq \psi$ **do**
4:     $m =$ number of customers in $S'_{curr}$
5:     randomly select a customer $i$ from $S'_{curr}$
6:     sort all customers (except $i$) of $S'_{curr}$ according to the similarity to $i$ based on (16)
7:     $z =$ random number $\in [0,1)$
8:     select a customer $j$ at position $\lfloor m \cdot z \rfloor$ in $S_{curr}$
9:     $RL = RL \cup \{j\}$
10: $S'_{curr} \leftarrow S'_{curr} \setminus \{j\}$
11: **end while**
12: return new current solution $S'_{curr}$, new removal list $RL$

similar it is to all other customers. Those customers with high similarity are selected and sent to $RL$. The similarity can be named as relatedness $\Upsilon_{ij}$ which is computed according to the formula below:

$$
\Upsilon_{ij} = \alpha_s \Upsilon^s_{ij} + \alpha_q \Upsilon^q_{ij} + \alpha_t \Upsilon^t_{ij}
$$

$$
= \alpha_s \cdot \frac{d_{ij}}{d_{max}} + \alpha_q \cdot \frac{|q_i - q_j|}{q_{max} - q_{min}} + \alpha_t \cdot \frac{|(\bigcup_{w \in W_i} w) \cap (\bigcup_{w \in W_j} w)|}{\max\{\big|\bigcup_{w \in W_i} w\big|, \big|\bigcup_{w \in W_j} w\big|\}} 
$$

The distance relatedness $\Upsilon^s_{ij}$ results straightforwardly from the distance $d_{ij}$ and the maximum distance between two vertices ($d_{max}$), while the demand relatedness $\Upsilon^q_{ij}$ results from demand of $i$ and $j$ as well as the maximum and minimum demand over all customers ($q_{max}$ and $q_{min}$). $\Upsilon^t_{ij}$ denotes the relative time window overlap between customer $i$ and $j$. Intuitively speaking, the more similar customers are
more likely to exchange without violating constraints. See Algorithm 3 for the pseudocode of Basic Related Removal.

**Improved Related Removal** This heuristic is an improvement of the Basic Related Removal operator introduced above and the pseudocode is shown in Algorithm 4. It selects customers from UCL instead of randomly selecting from all customers. In the case of ensuring that the total number is not greater than \( \psi \), we try to evenly distribute the number of high similarity customers for each customer in UCL, and then add them to RL. The purpose of designing this operator is to insert the customer from UCL back into the solution, thereby reducing the probability of becoming an infeasible solution. In order to avoid removing some customers repeatedly, we count the number of times for each customer in the operator has been selected before. Those customers who have been selected less often will have a greater chance of being selected to enter the RL in next iteration.

**Algorithm 4 Improved Related Removal**

\begin{algorithm}
\textbf{Input:} current solution \( S_{\text{curr}} \), unserved customers list UCL, number of customers to remove \( \psi \).
\textbf{Output:} new current solution \( S_{\text{curr}}' \), removal list RL.
1: set temporary customer set \( N_c \) whose customer comes from \( S_{\text{curr}}' \).
2: set \( \psi_i = \left\lceil \frac{\psi \text{ number of customers in UCL}}{UCL} \right\rceil \).
3: for each customer \( i \) in set UCL do
4: sort \( N_c \) according to the similarity to customer \( i \), marked as \( N_c' \).
5: select \( \psi_i \) numbers of customers from \( N_c' \), and send to set \( C_i \).
6: RL = RL \( \cup \) C_i, remove the customers \( C_i \) from \( N_c \).
7: end for
8: return new current solution \( S_{\text{curr}}' \), new removal list RL.
\end{algorithm}

**Route Removal** As the name of this operation indicates, the route removal operator randomly selects a route from the current solution and deletes all its customers. If the number of customers in the current route is less than \( \psi \), another route will be randomly selected and deleted. Repeat the above operation until the total number of deleted customers reaches or exceeds \( \psi \). The pseudocode for this operator is described in Algorithm 5.

**Random Removal** This operator deletes \( \psi \) customers randomly from the current solution which helps to expand the search space. The pseudocode for this operator is listed in Algorithm 6.

3.4.2. **Repair operators.** The insertion operators insert the customers in RL into the solution processed by the removal operator according to certain rules. If there are still customers that cannot be inserted after the insertion operation, these unserved customers will remain in UCL. Three insertion operators were used in this paper.

**Greedy Insertion** Based on the idea of greedy strategy, the operator inserts the selected customer into its best position and best route. If a customer \( i \) can be inserted into the route \( r \) at position \( k \), the insert cost can be calculated as: \( \Delta_i = \Delta H(i, k, r) = d_{ik} + d_{(i+1)k} - d_{(i+1)i} \). To determine the best insert position, we calculate \( \Delta_i^* = (i^*, k^*, r^*) = \text{argmin}(H(i, k, r)) \). It inserts customer \( i^* \) in route \( r^* \) at its minimum cost position \( k^* \). This greedy heuristic strategy (Algorithm 7) will insert those unserved customers one by one until all customers have been inserted or no more customers can be inserted.
Algorithm 5 Route Removal

Input: current solution $S_{curr}$, number of customers to remove $\psi$;

Output: new current solution $S'_{curr}$, removal list $RL$;

1: $S'_{curr} \leftarrow S_{curr}$, $RL \leftarrow \emptyset$, $n \leftarrow 0$, $m \leftarrow 0$
2: while $n \leq \psi$ do
3: randomly select a route $r$ (not selected before) in $S_{curr}$, $m =$ number of customers in route $r$, $n \leftarrow n + m$
4: if $m \geq \psi - n$ then
5: acquire $(m - \psi + n)$ consecutive customers from $r$ and send them to $RL$
6: else
7: acquire all customers from $r$ and send them to $RL$
8: end if
9: end while
10: $S'_{curr} \leftarrow S'_{curr} \{RL\}$
11: return new current solution $S'_{curr}$, new removal list $RL$

Algorithm 6 Random Removal

Input: current solution $S_{curr}$, number of customers to remove $\psi$;

Output: new current solution $S'_{curr}$, removal list $RL$;

1: randomly select one customer $i$ from $S_{curr}$, $S'_{curr} \leftarrow S_{curr}$
2: $RL = \{i\}$, $S'_{curr} \leftarrow S'_{curr} \{i\}$
3: while size of $RL \leq \psi$ do
4: randomly select a route $r$ in $S'_{curr}$, $m =$ number of customers in route $r$
5: $z =$ random number $\in [0, 1]$
6: select a customer $j$ at position $\lceil m \cdot z \rceil$ in $r$
7: remove $j$ from $r$, and add to $RL$
8: $S'_{curr} \leftarrow S'_{curr} \{j\}$
9: end while
10: return new current solution $S'_{curr}$, new removal list $RL$

Regret Insertion  The regret insertion operator is designed to select those customers who will regret the most if they were not inserted in the current iteration. The basic regret insertion operator sorts the removed customers according to the regret values. For a certain customer, its regret value can be expressed as the difference between the customer’s best insertion position and its second best insertion cost. In other words, the basic regret insertion operator selects the insertion position that will produce the greatest regret value if not inserted currently. The regret-$k$ method extended the basic method by selecting the customer $i$ such that $i = \arg\max_{i \in RL} \{\sum_{j=2}^{k} (\Delta f_j^i - \Delta f_1^i)\}$, where $RL$ is the unscheduled customers, and $\Delta f_j^i$ denotes the insertion cost of customer $i$ in the $j$th cheapest insertion position. The regret-2 operator was used in this paper, the regret value of regret-2 calculated as: $\Delta_i = \max_{i \in RL} \{\Delta f_2^i - \Delta f_1^i\}$. The Regret Insertion operator’s pseudocode detailed in Algorithm 8.

Random Insertion  The random insertion operator inserts the customers from $RL$ into current solution in a random order. Time window constraint cannot be violated during inserting process and this operator is also to maintain the diversity of solutions.
Algorithm 7 Greedy Insertion

Input: current solution $S'_{curr}$, removal list $RL$, unserved customers list $UCL$,
Output: new current solution $S_{curr}$;
1: $RL \leftarrow RL \cup UCL$, $UCL \leftarrow \emptyset$, $C \leftarrow \emptyset$
2: for each customer $i$ in $RL$ do
3: calculate $\Delta_i$ for $i$ to insert into each position of $S'_{curr}$
4: $C \leftarrow C \cup \{\Delta_i\}$
5: end for
6: while $RL \neq \emptyset$ do
7: $\Delta_{min} \leftarrow$ the minimum value of $C$, remove the minimum value from $C$
8: if insert $i$ to route $r$ at position $k$ violates the constraints then
9: $UCL \leftarrow UCL \cup \{i\}$
else
11: insert $i$ to route $r$ at position $k$ in $S'_{curr}$
end if
13: $RL \leftarrow RL \setminus \{i\}$
14: update $C$
15: end while
16: $S_{curr} \leftarrow S'_{curr}$
17: return new current solution $S_{curr}$, updated $UCL$

Algorithm 8 Regret Insertion

Input: current solution $S'_{curr}$, removal list $RL$, unserved customers list $UCL$,
Output: new current solution $S_{curr}$;
1: $RL \leftarrow RL \cup UCL$, $UCL \leftarrow \emptyset$, $C \leftarrow \emptyset$
2: for each customer $i$ in $RL$ do
3: calculate $\Delta_i$ for $i$ to insert into $S'_{curr}$
4: $C \leftarrow C \cup \{\Delta_i\}$
5: end for
6: while $RL \neq \emptyset$ do
7: $\Delta_{max} \leftarrow$ the maximum value of $C$, remove the maximum value from $C$
8: if insert $i$ to $S'_{curr}$ violates the constraints then
9: $UCL \leftarrow UCL \cup \{i\}$
else
11: insert $i$ to $S'_{curr}$
end if
13: $RL \leftarrow RL \setminus \{i\}$
14: update $C$
15: end while
16: $S_{curr} \leftarrow S'_{curr}$
17: return new current solution $S_{curr}$, updated $UCL$

3.5. Strategies for maintaining diversity.

3.5.1. Local search. As mentioned before, insertion operators were used in ALNS to insert customers from $RL$ into the current solution. This process may cause a few customers failed to insert. For this reason, a $UCL$ pool was set up to store these customers. If the $UCL$ is not empty at each iteration, which means there are
Algorithm 9 Local Search Procedure

Input: current solution $S_{curr}$, number of customers to remove $\psi^{LS}$, unserved customers list $UCL$;

Output: new current solution $S_{curr}$, the updated $UCL$;

1: $d_i = \{\text{ImprovedRelatedRemoval, RandomRemoval}\}, i \in \{1, 2\}$,
2: $r_j = \{\text{RegretInsertion, RandomInsertion}\}, j \in \{1, 2\}$
3: $\psi^{LS} \leftarrow \psi$(executes only after every $noi_{max}$ iterations), $noi \leftarrow 0$
4: while $noi \leq noi_{max}$ do
5: for each $i \in \{1, 2\}$ do
6: $S'_{curr} \leftarrow d_i(S_{curr})$
7: for each $j \in \{1, 2\}$ do
8: $S''_{curr} \leftarrow r_j(S'_{curr})$
9: if $UCL = \emptyset$ then
10: $S_{curr} \leftarrow S''_{curr}, \psi^{LS} \leftarrow \psi$
11: jump out of all the loops
12: end if
13: end for
14: end for
15: if $UCL \neq \emptyset$ then
16: $S_{curr} \leftarrow S''_{curr}$ with less remaining customers in $UCL$
17: $\psi^{LS} \leftarrow \psi^{LS} + 1$ (the upper bound of $\psi^{LS}$ is $\psi_{max}^{LS}$)
18: end if
19: end while
20: update $UCL$
21: return new current solution $S_{curr}$, the updated $UCL$

still several customers not be served, a $LocalSearch$ consisting of four operators will be called for the current solution. Then choose two destroy operators: $ImprovedRelatedRemoval$, $RandomRemoval$ and two repair operators: $RegretInsertion$ and $RandomInsertion$ for the local search stage. The proposed local search performs four times destroy-repair operations to the current solution (one destroy operation and one repair operation for each time), and adopts the first improvement acceptance principle. Once the number of customers in $UCL$ is 0, this searching progress is broke off and the current solution is updated. After four time operations, there are still customers remaining in $UCL$, then the solution corresponding to $UCL$ with the least number of customers was chosen as the current solution. The $LocalSearch$ procedure detailed in Algorithm 9.

The purpose of setting up the $LocalSearch$ procedure is to reduce the number of customers in $UCL$ to zero as well as repairing the infeasible solution to become a feasible solution. The size of $RL \psi^{LS}$ represents the number of customers to remove and insert, which has different effects on the current solution. The strategy adopted here is to set the size of $RL$ in $LocalSearch$ procedure to $\psi$ at the beginning of iteration, which is the same value as in the main loop; if the number of customers in $UCL$ does not reduce to zero for $noi_{max}$ iterations, the size of $RL$ increases by 1 up to a maximum of $\psi_{max}^{LS}$. The purpose of increasing the size of $RL$ is to perturb the current solution, so as to escape from local optimum. The setting of related parameters will be detailed in section 4.3.
3.5.2. Acceptance criterion. To make the algorithm escape from the local optimum, a Simulated Annealing (SA) acceptance criterion is embedded in proposed algorithm framework [20]. It is always been accepted if the neighborhood solution improved the current solution generated by removal and insertion methods. Otherwise this neighborhood solution is accepted with a probability of $e^{-(f(S_{curr})-f(S_{new}))/T}$, where $T > 0$ represents temperature, $S_{curr}$ is the current solution and $S_{new}$ is neighborhood solutions. The initial temperature is set as a constant $T_0$. During the iteration of the algorithm, the current temperature is proportionally reduced to $c \cdot T$. This proportional coefficient $c$ is between 0 and 1 and usually close to 1 to achieve slow cooling. The main intention for adopting SA as the acceptance criterion is to increase the diversification of the search space by accepting worse solutions at the beginning.

3.6. Strategies for dealing with infeasible solutions.

3.6.1. Route feasibility check. Purposefully accepting infeasible solutions during the iteration is an important way to increase the diversification of the algorithm. Since the ALNS uses destroy and repair operators, this procedure will generate infeasible solutions especially violate the multiple time windows constraint. This paper draws on the Forward Start Interval Algorithm (FSIA) proposed by Hoogeboom [12] to find the best time window selection scheme corresponding to each route, which method can also determine whether the time window constraint is satisfied.

3.6.2. Penalty terms for objective function. To evaluate infeasible solutions, the objective function is appropriately modified in this paper by adding some penalty terms. The newly constructed objective function adds weighted penalties for vehicle overload $\nu$ and time window violation $\mu$. The specific expression is shown below:

$$f(X) = f_1(X) + \beta_1 \sum_{i \in I} \min_{p \in W_i} \mu_i + \beta_2 \sum_{k \in K} \nu_k$$

$$\mu_i = \max\{0, \min_{p_i = 1, \ldots, p_i}{|a_i - l_{p_i}|, |a_i - u_{p_i}|}\}$$

$$\nu_k = \max\{0, \sum_{k \in K} q_i - Q_k\}$$

The extent to which each customer $i$ violates the time window constraint can be measured as (18), where $a_i$ is the time when vehicle starts to serve customer $i$. For each vehicle $k$, the overload of capacity is calculated as (19). The two violations $\mu_i$ and $\nu_k$ at each customer or vehicle are corresponding with penalties $\beta_1 \mu_i$ and $\beta_2 \nu_k$, where $\beta_1$ and $\beta_2$ are the weights. $f_1(X)$ represents the original objective function and the new objective function $f(X)$ will be adopted in the algorithm.

3.6.3. Current feasible solutions pool. At each iteration, the feasible solution(s) will send into an archive called Current Feasible Solutions Pool (CFS_Pool) with size $NP$. When the number of solutions exceeds $NP$, the worst solution will be removed automatically to keep the size of the pool. If the generated solution is infeasible at the end of each segment, the best solution will be popped out of the CFS_Pool as the current solution. The purpose of this mechanism is to not only make full use of the infeasible solutions generated during the iteration, especially the high-quality solution information contained therein, but also to repair and correct the infeasible solutions as feasible solutions, which allowing the algorithm to converge to the optimal target gradually.
4. **Computational result.** This section will show the relevant experiments that conducted. Section 4.1 presents the benchmark instances that used in this paper. Then, an explanation on how ALNS parameters have been tuned is presented in Section 4.2. Section 4.3 details the setting of RemovalList size. In Section 4.4, we evaluate the performance of the proposed algorithm in this paper on solution quality. Proposed algorithm was implemented in Java and tested on an Intel CORE i5 2.5 GHz computer with 16GB of memory.

4.1. **The VRPMTW benchmark instances.** Belhaiza\[1\] generated a benchmark for VRPMTW from Solomon VRPTW instances. The 48 instances includes six classes and all of which contain 100 customers. According to the width of customer’s time window, each of the six classes can be subdivided into two types: Type1 for narrow and Type2 for wide. There are also three types of distribution for each class: RM for randomly distributed, CM for clustered distributed and RCM for randomly clustered distributed. Each instance has from one to ten non-overlapping time windows and the optimal solution for each instance is unknown. The vehicle cost, in time units, includes the waiting, travel and service times and fixed cost. The fixed cost is set equal to the vehicle capacity, i.e., 200, 700, 200, 1000, 200 and 1000 for instance sets CM1, CM2, RCM1, RCM2, RM1 and RM2, respectively.

4.2. **Parameter tuning.** In order to obtain better results, various parameters of the proposed algorithm needed to be set. The principle of parameter adjustment is gradual and the specific operation process is shown below. As the most important component in ALNS, those parameters of the destroy and repair operators should be determined first. Here we first tried to adjust the parameters of Related Removal operator. Then using Greedy Insertion as the repair operator to execute the ALNS algorithm. Please note that the algorithm didn’t use SA acceptance criterion for this step. We first focused on adjusting one of the parameters and test multiple times under the same conditions of other parameters. Specifically, the optimization example was solved five times for each value, and the value that provides the best average solution quality was retained. Next step used the above method to adjust Worst Removal operator, mainly to adjust the weight value of each component. When adjusting the subsequent parameters, we adopted the adjusted and complete destroy and repair operator instead of using a single operator to delete and insert.

After completing the parameter adjustment for the destroy and repair operator, we continued to adjust some relevant parameters in simulated annealing acceptance criteria, which was also one of the important measures to enable the method to find high quality solutions. In order to adjust the \( \sigma_1, \sigma_2, \sigma_3, \sigma_4, \) and \( \zeta \) for the roulette wheel mechanism introduced in Section 3.3, numerical tests on all examples were performed adjustment. The cooling rate parameter \( c \) controls the decrease in temperature during simulated annealing acceptance criteria phase in Section 3.5. Based on the experience of existing research and actual tests, we chose to set this value to 0.9 for subsequent calculation experiments. A higher temperature value means that the temperature decreases more slowly, leading to a higher probability of accepting a poorer existing solution in the iteration. Therefore, several sets of values were tried as the initial temperature \( T_0 \), and finally chosen 0.4.

Table 2 lists all the parameters adopted.

4.3. **Setting of removalList.** In order to tune parameters for RL in LocalSearch procedure, including \( v^{LS}, v_{max}^{LS} \), and \( noi_{max} \), several groups of different values were set according to the experience of ALNS algorithm. By using the parameters set in
Table 2. Parameters of ALNS

| Parameter | nsegs | nters | σ4 | σ3 | σ2 | σ1 | ζ | T0 | c | β1 | β2 |
|-----------|-------|-------|----|----|----|----|---|----|---|----|----|
| Value     | 1000  | 100   | 10 | 5  | 3  | 1  | 0.7| 0.4 | 0.9| 100| 100|

Table 3. Parameters of RemovalList in LocalSearch procedure

| Parameters (ψ_{LS}, ψ_{LS}^{max}, noi_{max}) | Average in 3 groups |
|---------------------------------------------|---------------------|
|                                             | CM | RCM | RM |
| (12,20,200)                                 | 13574.6 | 4201.8 | 4109.6 |
| (12,20,500)                                 | 13341.7 | 4197.2 | 4042.6 |
| (12,20,800)                                 | 12947.1 | 4141.0 | 4023.6 |
| (15,22,200)                                 | 12927.3 | 4094.3 | 3815.3 |
| (15,22,500)                                 | 12724.2 | 4055.7 | 3737.7 |
| (15,22,800)                                 | 12729.0 | 4056.1 | 3736.4 |
| (18,25,200)                                 | 12845.1 | 4092.3 | 3746.5 |
| (18,25,500)                                 | 12813.0 | 4084.5 | 3741.0 |
| (18,25,800)                                 | 12748.9 | 4088.3 | 3739.2 |

section 4.2, the complete algorithm was run on all benchmark instances. There are nine combinations considered in this test: (12,20,200), (12,20,500), (12,20,800), (15,22,200), (15,22,500), (15,22,800), (18,25,200), (18,25,500) and (18,25,800). Each instance was run five times and the average value was taken. The result presented in Table 3 shows that the combination (15,22,500) performs the best in terms of the average value. Therefore, based on this analysis, we choose to use (ψ_{LS}, ψ_{LS}^{max}, noi_{max})=(15,22,500) in the rest of the numerical experiments.

4.4. Computational results. In the final experiment, we compare the performance of proposed ALNS with state-of-the-art results published in Hoogeboom[12], Belhaiza[1] and Belhaiza[3]. As Belhaiza[3] and Hoogeboom[12] performed a single run of 100,000 iterations, so the proposed ALNS runs one time for 100,000 iterations. The computational results are presented per instance in Table4-Table6. Run the proposed algorithm 30 times continuously and list the best solution values obtained in Table4-Table6. In Table4-Table6, each column is represented as follows: m denotes the number of vehicles; HVNTS, HGVNS and EAVNS represent the best upper bound obtained by Belhaiza[1], Belhaiza[3] and Hoogeboom[12] separately; ALNS denote the best solution obtained by proposed method; Let gap1, gap2 and gap3 represent the relative improvement percentages of ALNS corresponding to HVNTS, HGVNS and EAVNS. It can be seen from Table4-Table6 that the overall average improvement of the proposed algorithm in this paper compared to HVNTS, HGVNS and EAVNS is 1.19%, 0.79% and 0.41%, respectively. Table4-Table6 also show that, except for the RM2 and RCM2 classes, the two algorithm obtain nearly the same average best solution value, and the performance of ALNS is slightly better than EAVNS on all class instances. Class RM2 occupies a small number but widely distributed customer time windows, the average number of time windows is the smallest and the distance between each pair of consecutive time windows is large. Due to the characteristics of time windows, the RM2 class is easier to solve than the other classes, which makes the best solution found by the two algorithms very close(especially RM204-RM208). We also found 31 of new best known solutions from 48 instances in the entire benchmark and increased the quality of the solutions of all instances by 0.80% on average.
Table 4. ALNS results on VRPMTW instances (Group CM)

| Instance | m | HVNTS | HGVNS | EAVNS | ALNS | gap1 | gap2 | gap3 |
|----------|---|-------|-------|-------|------|------|------|------|
| CM101    | 10| 12320 | 12319.1| 12345.4| 12151.0| 1.37 | 1.36 | 1.57 |
| CM102    | 11| 12492.1| 12482.3| 12467.8| 12151.0| 1.51 | 1.44 | 1.13 |
| CM103    | 11| 12641.2| 12632.4| 12592.2| 12450.5| 1.34 | 1.42 | 0.12 |
| CM104    | 13| 12083.4| 12027.0| 12066.3| 12083.4| 0.00 | -0.47 | -0.14 |
| CM105    | 10| 12073.9| 12059.0| 12066.4| 11995.4| 0.65 | 0.53 | 0.59 |
| CM106    | 10| 12324.2| 12318.0| 12108.4| 12092.2| 1.88 | 1.83 | 0.13 |
| CM107    | 10| 11990.4| 11986.0| 11985.9| 11970.3| 0.17 | 0.13 | 0.13 |
| Average  | 10.6| 12376.6| 12356.3| 12321.8| 12265.4| 0.89 | 0.72 | 0.46 |

| Instance | m | HVNTS | HGVNS | EAVNS | ALNS | gap1 | gap2 | gap3 |
|----------|---|-------|-------|-------|------|------|------|------|
| RCM101   | 5 | 13520.1| 13498.8| 13468.4| 13418.3| 0.75 | 0.60 | 0.37 |
| RCM102   | 6 | 14027.3| 14025.1| 14020.2| 14010.4| 0.12 | 0.10 | 0.07 |
| RCM103   | 5 | 13497.2| 13465.8| 13486.5| 13464.6| 0.24 | 0.01 | 0.16 |
| RCM104   | 5 | 13359.8| 13344.0| 13356.9| 13344.0| 0.12 | 0.00 | 0.10 |
| RCM105   | 4 | 12884.1| 12827.8| 12896.8| 12829.5| 0.42 | -0.01 | 0.23 |
| RCM106   | 4 | 13499.7| 13444.0| 13456.9| 13444.0| 0.12 | 0.00 | 0.10 |
| RCM107   | 4 | 12881.1| 12749.7| 12756.8| 12746.7| 0.32 | 0.02 | 0.08 |
| Average  | 4.6| 13231.8| 13198.5| 13210.3| 13181.4| 0.38 | 0.13 | 0.22 |

Table 5. ALNS results on VRPMTW instances (Group RCM)

| Instance | m | HVNTS | HGVNS | EAVNS | ALNS | gap1 | gap2 | gap3 |
|----------|---|-------|-------|-------|------|------|------|------|
| RCM101   | 10| 4098.9| 4081.2| 4080.6| 4076.1| 0.55 | 0.12 | 0.11 |
| RCM102   | 10| 4222.6| 4188.3| 4184.3| 4122.7| 2.73 | 1.57 | 1.47 |
| RCM103   | 10| 4174.3| 4150.4| 4148.3| 4140.5| 0.81 | 0.24 | 0.19 |
| RCM104   | 10| 4156.3| 4144.0| 4141.2| 4128.3| 0.67 | 0.38 | 0.31 |
| RCM105   | 10| 4216.7| 4207.0| 4208.2| 4190.4| 0.62 | 0.40 | 0.42 |
| RCM106   | 10| 4219.9| 4187.7| 4191.8| 4173.1| 1.11 | 0.35 | 0.45 |
| RCM107   | 10| 4542.4| 4521.5| 4516.5| 4511.4| 0.68 | 0.22 | 0.11 |
| RCM108   | 10| 4614.5| 4565.2| 4566.2| 4532.5| 1.78 | 0.72 | 0.74 |
| Average  | 10.3| 4280.7| 4254.9| 4254.6| 4234.4| 1.07 | 0.50 | 0.48 |

4.5. Relative performances of the destroy and repair operators. In each iteration of the ALNS, a new solution generated by applying one destroy operator and one repair operator. The selection of these operators is performed by using the adaptive selection mechanism proposed in Section 3.3. In the first experiment, we checked whether all operators were used to produce good solutions. We ran the adjusted algorithm on all instances, and each time a new optimal solution was generated, we recorded the operator used. Since the best solution can be improved relatively easily from the early stage of iteration, we skipped the first 10% of iterations from the count. This experiment proved that all the operators were used, but the proportions were different. The statistics was shown in FIGURE 2. We can see that the number of uses of destroy operators has been better balanced,
Table 6. ALNS results on VRPMTW instances (Group RM)

| Instance | m | HVNTS  | HGVSNS | EAVNS  | ALNS  | gap1  | gap2  | gap3  |
|----------|---|---------|--------|--------|-------|-------|-------|-------|
| RM101    | 10| 4041.9  | 4027.1 | 4026.1 | 4009.4| 0.81  | 0.44  | 0.42  |
| RM102    | 9 | 3765.1  | 3751.2 | 3774.8 | 3730.0| 0.93  | 0.56  | 1.19  |
| RM103    | 9 | 3708.5  | 3703.0 | 3700.6 | 3704.0| 0.12  | -0.33 | -0.09 |
| RM104    | 9 | 3718.0  | 3701.2 | 3707.1 | 3697.6| 0.55  | 0.10  | 0.26  |
| RM105    | 9 | 3688.8  | 3687.2 | 3690.5 | 3689.7| -0.02 | -0.07 | 0.02  |
| RM106    | 9 | 3692.9  | 3708.4 | 3714.8 | 3713.5| -0.56 | -0.14 | 0.03  |
| RM107    | 9 | 3701.4  | 3692.8 | 3700.4 | 3686.0| 0.42  | 0.18  | 0.39  |
| RM108    | 9 | 3792.1  | 3722.6 | 3738.1 | 3727.6| 1.70  | -0.14 | 0.28  |
| Average  | 9.1| 3755.7  | 3749.2 | 3756.6 | 3744.7| 0.49  | 0.11  | 0.31  |
| RM201    | 2 | 4808.2  | 4805.4 | 3888.9 | 3804.4| 20.88 | 20.83 | 2.17  |
| RM202    | 2 | 3739.0  | 3706.8 | 3721.9 | 3706.8| 0.86  | 0.00  | 0.41  |
| RM203    | 2 | 3710.3  | 3696.9 | 3693.2 | 3691.7| 0.50  | 0.14  | 0.04  |
| RM204    | 2 | 3691.9  | 3674.5 | 3671.7 | 3674.5| 0.47  | 0.00  | -0.08 |
| RM205    | 2 | 3689.9  | 3681.1 | 3668.4 | 3671.0| 0.51  | -0.08 | -0.07 |
| RM206    | 2 | 3703.4  | 3684.9 | 3672.6 | 3673.5| 0.81  | 0.31  | -0.02 |
| RM207    | 2 | 3701.7  | 3664.3 | 3662.4 | 3664.3| 1.01  | 0.00  | -0.05 |
| RM208    | 2 | 3682.8  | 3664.3 | 3663.6 | 3664.3| 0.50  | 0.00  | -0.02 |
| Average  | 2 | 3840.9  | 3820.7 | 3705.3 | 3693.8| 3.19  | 2.65  | 0.30  |

Figure 2. Number of times each destroy and repair method was used to produce a new best solution

while for repair operators, the regret operator alone has achieved nearly 50% of the hits.

From the performance of repair operators, the computational complexity of the regret operator is slightly higher, but it can bring better solution improvements. In order to verify the performance of each operator, as the second experiment, we ran each instance ten times while excluding one operator and keeping other operators. Table 7 provides statistics on the operators. The comparison was done based on the average percentage of solution degradation and also the maximum percentage of solution degradation. The maximum degradation shows the maximum
Table 7. Evaluation of contribution of each operator

| Operator                | Average degradation(%) | Maximum degradation(%) |
|-------------------------|-------------------------|------------------------|
| Worst Removal           | 0.26                    | 0.86                   |
| Basic Related Removal   | 0.20                    | 0.91                   |
| Improved Related Removal| 0.25                    | 0.99                   |
| Route Removal           | 0.22                    | 1.15                   |
| Random Removal          | 0.17                    | 0.84                   |
| Greedy Insertion        | 0.27                    | 1.82                   |
| Regret Insertion        | 0.35                    | 2.31                   |
| Random Insertion        | 0.17                    | 0.73                   |

Table 8. Execution counts of the operators leading to the discovery of a new solution

| Operator                | Best solution | Current solution | Simulated annealing |
|-------------------------|---------------|------------------|---------------------|
| Worst Removal           | 922           | 73627            | 1007                |
| Basic Related Removal   | 2341          | 88112            | 756                 |
| Improved Related Removal| 4267          | 93581            | 543                 |
| Route Removal           | 45            | 45990            | 4352                |
| Random Removal          | 64            | 41686            | 6235                |
| Greedy Insertion        | 2511          | 38656            | 457                 |
| Regret Insertion        | 7539          | 95684            | 365                 |
| Random Insertion        | 582           | 9083             | 2120                |

The results showed that the exclusion of any destroy or repair operator resulted in a deterioration in the quality of the solution (relative to the optimal solution). These results indicated the usefulness of all of our destroy and repair operators in the case of this problem setting. So we proposed to keep all destroy and repair operators in the following experiments since this combination has the best results.

In order to further explore the effectiveness of each operator in the process of improving the solution, we evaluated the effectiveness of these operators in finding (i) a new best solution, (ii) a better current solution, and (iii) a new solution accepted with simulated annealing, as presented in Table 8. These values were accumulated over 5 runs for all 48 instances. In this test, one destroy(repair) operator and all repair(destroy) operators were kept at a time. Columns 2 to 4 in Table 8 correspond to cases (i) to (iii), respectively. Compared to other operators, “Improved Related Remove” is seen to be more effective in finding a new best solution and also a better current solution. The moves “Regret Insertion” also seems more successful in updating both the best solution and the current solution. As stated before, the diverse characteristics of its operators are one of the strongest features of the ALNS algorithm. Even though some operators show more success, it is not possible to obtain as good solutions without the synergy of the other operators. Our analysis suggests that all operators included in the design of the ALNS algorithm make a contribution to its performance either by intensifying or by diversifying the search as a result of their structural characteristics.

5. Conclusion. In order to solve the Vehicle Routing Problem with Multiple Time Windows (VRPMTW), this paper improves the existing Adaptive Large Neighborhood Search (ALNS) heuristic. We first established a mixed integer programming
model for this problem. Because the multiple time windows constraints interconnect various routes of the vehicles, solving the proposed problem is more complicated than solving standard vehicle routing problems with (single) time window, especially when using neighborhood search-based methods. For this reason, this paper used the destroy and repair operators based on large neighborhood search as the main neighborhood.

The proposed ALNS algorithm uses five removal and three insertion operators for neighborhood search, matching with simulated annealing as the acceptance criteria to accept inferior solutions or even infeasible solutions with a certain probability, which effectively expanding the search space. A local search process is used to repair the infeasible solutions generated in the iterative process. With the employ of external archives, periodic replacement of infeasible solutions helps to search for more potential areas. Proposed algorithm was tested on 48 benchmark instances designed for VRPMTW which adopted by several recent researches. The experimental results show that the method in this paper has obtained better solutions overall, and found some optimal solutions. Further experimental analysis showed that all the operators designed for ALNS in this paper improved the solution quality by enhancing or diversifying search, thus contributing to the algorithm performance. A future work may consider more destroy and repair operators for multiple time windows, including more efficient mechanism for choosing operators and handling infeasible solutions.

Acknowledgments. This work was supported by Project supported by National Natural Science Foundation of China (No.62003296), National Key Research and Development Program of China (2018YFB1702300), the Natural Science Foundation of Hebei (No.F2020203031), Science and Technology Project of Hebei Education Department (No.QN20202225).

REFERENCES

[1] S. Belhaiza, P. Hansen and G. Laporte, A hybrid variable neighborhood tabu search heuristic for the vehicle routing problem with multiple time windows, Comput. Oper. Res., 52 (2014), 269–281.
[2] S. Belhaiza and R. M’Hallah, A Pareto non-dominated solution approach for the vehicle routing problem with multiple time windows, in 2016 IEEE Congress on Evolutionary Computation (CEC), (2016), 3515–3524.
[3] S. Belhaiza, R. M’Hallah and G. B. Brahim, A new hybrid genetic variable neighborhood search heuristic for the vehicle routing problem with multiple time windows, in 2017 IEEE Congress on Evolutionary Computation (CEC), (2017), 1319–1326.
[4] H. Ben Ticha, N. Absi, D. Feillet and A. Quilliot, Multigraph modeling and adaptive large neighborhood search for the vehicle routing problem with time windows, Comput. Oper. Res., 104 (2019), 113–126.
[5] K. Braekers, K. Ramaekers and I. V. Nieuwenhuyse, The vehicle routing problem: State of the art classification and review, Computers & Industrial Engineering, 99 (2016), 300–313.
[6] S. Chen, R. Chen, G.-G. Wang, J. Gao and A. K. Sangaiah, An adaptive large neighborhood search heuristic for dynamic vehicle routing problems, Computers & Electrical Engineering, 67 (2018), 596–607.
[7] H. Chentli, R. Ouafi and W. Ramdane Cherif-Khettaf, A selective adaptive large neighborhood search heuristic for the profitable tour problem with simultaneous pickup and delivery services, RAIRO Oper. Res., 52 (2018), 1295–1328.
[8] D. Favaretto, E. Moretti and P. Pellegrini, Ant colony system for a VRP with multiple time windows and multiple visits, J. Interdiscip. Math., 10 (2007), 263–284.
[9] H. S. Ferreira, E. T. Bogue, T. F. Noronha, S. Belhaiza and C. Prins, Variable neighborhood search for vehicle routing problem with multiple time windows, Electronic Notes in Discrete Mathematics, 66 (2018), 207–214.

[10] E. Glize, N. Jozefowiez and S. U. Ngueveu, An exact column generation-based algorithm for bi-objective vehicle routing problems, Combinatorial Optimization, 208–218, Lecture Notes in Comput. Sci., 10856, Springer, Cham, 2018.

[11] F. Hammami, M. Rekik and L. C. Coelho, A hybrid adaptive large neighborhood search heuristic for the team orienteering problem, Comput. Oper. Res., 123 (2020), 105034, 18 pp.

[12] M. Hoogeboom, W. Dullaert, D. Lai and D. Vigo, Efficient neighborhood evaluations for the vehicle routing problem with multiple time windows, Transportation Science, 54 (2020), 400–416.

[13] Z. Hu, Z. Wei, H. Sun et al, Optimization of metal rolling control using soft computing approaches: A review, Arch. Computat. Methods Eng., 28 (2021), 405–421.

[14] M. N. Kritikos and P. Z. Lappas, Computational Intelligence and Combinatorial Optimization Problems in Transportation Science, In: Tsihrintzis G., Virvou M. (eds) Advances in Core Computer Science-Based Technologies. Learning and Analytics in Intelligent Systems, vol 14. Springer, Cham.

[15] Y. Li, H. Chen and C. Prins, Adaptive large neighborhood search for the pickup and delivery problem with time windows, profits, and reserved requests, European J. Oper. Res., 252 (2016), 27–38.

[16] R. Liu, Y. Tao and X. Xie, An adaptive large neighborhood search heuristic for the vehicle routing problem with time windows and synchronized visits, Comput. Oper. Res., 101 (2019), 250–262.

[17] T. Liu, Z. Luo, H. Qin and A. Lim, A branch-and-cut algorithm for the two-echelon capacitated vehicle routing problem with grouping constraints, European J. Oper. Res., 266 (2018), 487–497.

[18] S. Mirzaei and S. Whlk, A branch-and-price algorithm for two multi-compartment vehicle routing problems, Euro Journal on Transportation & Logistics, 8 (2019), 1–33.

[19] D. Pisinger and S. Ropke, A general heuristic for vehicle routing problems, Comput. Oper. Res., 34 (2007), 2403–2435.

[20] P. H. Richard, S. Allyson, J. R. Kees and C. C. Leandro, The vehicle routing problem with simultaneous pickup and delivery and handling costs, Computers & Operations Research, 115 (2020), 104858.

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

[22] V. Schmid, K. F. Doerner and G. Laporte, Rich routing problems arising in supply chain management, European J. Oper. Res., 224 (2013), 435–448.

[23] P. Shaw, Using constraint programming and local search methods to solve vehicle routing problems, Principles and Practice of Constraint Programming — CP98. CP, 286 (1998), 417–431.

[24] T. Vidal, G. Laporte and P. Matl, A concise guide to existing and emerging vehicle routing problem variants, European J. Oper. Res., 286 (2020), 401–416.

[25] D. Vigo and P. Toth, Vehicle Routing: Problems, Methods, and Applications, 2nd edition, SIAM, 2014.

[26] X. Yan, B. Xiao and Z. Zhao, Multi-objective vehicle routing problem with simultaneous pick-up and delivery considering customer satisfaction, in 2019 IEEE International Conference on Smart Manufacturing, Industrial & Logistics Engineering (SMILE), Volume (2019), 93–97.

Received March 2021; revised July 2021; early access November 2021.

E-mail address: fengbinstu@foxmail.com
E-mail address: wlx2000@ysu.edu.cn
E-mail address: hzy@ysu.edu.cn