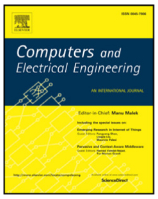
Computers and Electrical Engineering 000 (2018) 1–12

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng



An adaptive large neighborhood search heuristic for dynamic vehicle routing problems[☆]

Shifeng Chen[a], Rong Chen[a,*], Gai-Ge Wang[b], Jian Gao[a], Arun Kumar Sangaiah[c]

[a] Department of Information Science and Technology, Dalian Maritime University, Dalian 116026, Liaoning, China
[b] College of Information Science and Engineering, Ocean University of China, Qingdao 266100, Shandong, China
[c] School of Computing Science and Engineering, VIT University, Vellore 632014, Tamil Nadu, India

ARTICLE INFO

Article history:
Received 4 March 2017
Revised 28 February 2018
Accepted 28 February 2018
Available online xxx

Keywords:
Dynamic vehicle routing
Time windows
Adaptive large neighborhood search
Limited vehicles

ABSTRACT

The vehicle routing in real-life transportation, distribution and logistics may change with time, especially when there is existing technology that can produce real-time routing data. In this paper, a metaheuristic procedure based on an Adaptive Large Neighborhood Search (ALNS) algorithm is proposed to solve the Dynamic Vehicle Routing Problem (DVRP) with limited vehicles and hard-time windows. The ALNS involves ad hoc destroy/repair heuristics and a periodic perturbation procedure. In addition, an efficient feasibility check has been designed for inserting customer. By conducting several computational experiments with Lackner's benchmark, we show that the present approach can solve real-time problems within a very short time while improving the quality of the solution. The average number of vehicles is smaller than that of existing algorithms, the maximum average error of the vehicle traveling distance is reduced, and the average computation time remains the same.

© 2018 Elsevier Ltd. All rights reserved.

1. Introduction

The Vehicle Routing Problem (VRP) [1] consists of determining round vehicle trip(s) so that the total cost (e.g. travel time and travel distance) is minimized while ensuring the delivery of different products to customers according to their particular orders. With recent advances in telecommunications and computer hardware, the Dynamic VRP has emerged to address the large amount of real-time monitoring data, such as current vehicle locations, dynamic customer locations and requests. The model can be used to decide on a new route plan to improve service quality and reduce distribution costs. More recently, a number of variants have been investigated and reported. Among them are the dynamic traveling salesman problem [2], the dynamic traveling repairman problem [3], the dynamic dial-a-ride problem [4], and the dynamic pickup and delivery problem [5].

The DVRP is more complex than the classic VRP in that dynamic requests necessitate the decision-making within a tight timeframe based on uncertain information. A good example is the DVRP with Time Windows (DVRPTW), which is used to serve customers within a pre-defined time period. A tabu search algorithm [6] was the first algorithm used on the DVRPTW. Pankratz et al. solved the dynamic pickup and delivery problem with a genetic algorithm [7]. Hong presented

☆ Reviews processed and recommended for publication to the Editor-in-Chief by Guest Editor Dr. R.C. Poonia.
* Corresponding author.
E-mail addresses: chens@dlmu.edu.cn (S. Chen), rchen@dlmu.edu.cn (R. Chen), gaigewang@gmail.com (G.-G. Wang), gaojian@dlmu.edu.cn (J. Gao), arunkumarsangaiah@gmail.com (A.K. Sangaiah).

https://doi.org/10.1016/j.compeleceng.2018.02.049
0045-7906/© 2018 Elsevier Ltd. All rights reserved.

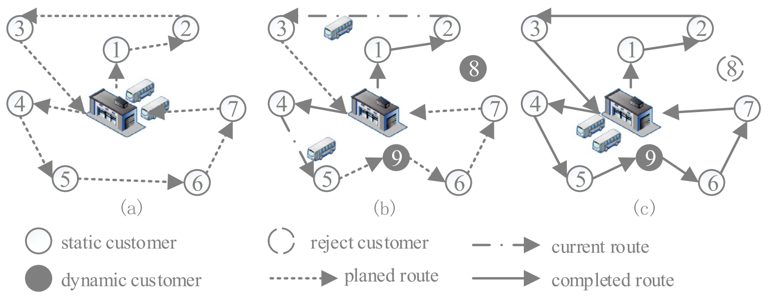


Fig. 1. Illustration of a typical dynamic vehicle routing problem with time windows.

an Improved Large Neighborhood Search (ILNS) algorithm that decomposed the DVRPTW into several static VRPTWs [8]. de Armas and Melián-Batista [9] developed a metaheuristic algorithm for tackling two variants of DVRPTW with several real-world constraints: multiple time windows, customers' priorities and vehicle customer constraints. This study continued in [10], where a General Variable Neighborhood Search (GVNS) algorithm was proposed to solve the DVRPTW as a real-world application, while Saint-Guillain et al. introduced a multi scenario approach in [11]. In recent surveys on DVRPs, Pillac et al. [12] and Ritzinger et al. [13] introduced different problem variations such as deterministic and stochastic DVRPs and commonly used approaches for solving DVRPs. We refer the reader to Psaraftis et al. [14] that provide an overview and a classification of the numerous versions of real-time routing and dispatching problems.

Despite the promising results in solving the DVRPTW, current algorithms often assume that there is no limit on the number of vehicles. They fail to fully capture the reality that there are limited resources, such as vehicles, and manpower. Thus, in real world applications, the DVRPTW is over-constrained, which may lead to conflicting objectives. Still less is known about how to construct a good initial solution and find an optimal solution.

This paper focuses on a more constrained DVRPTW that features heterogeneous vehicles and a soft constraint on their number. A heterogeneous fleet of vehicles for serving customer requests is stationed at a single depot at the beginning of the planning horizon. We present an improved ALNS heuristic for the resolution of the DVRPTW. The problem is decomposed into a series of static VRPTWs. The approach attempts to solve a static problem at each time slice that contains all the known customer requests that have not been visited so far. Therefore, we propose several operators that take advantage of the structure and adapt existing operators for the static VRP. In addition, we apply a 2-opt* local search to perturb the current solution. We report computational results on Lackner's benchmark that are derived from the well-known VRPTW instances of Solomon [15] and compare their performance with existing algorithms such as ILNS [8] and GVNS [10]. In comparison, the average computation time is basically the same, but the average number of vehicles is at least 4.93% better than the existing two methods. The maximum average error of the vehicle traveling distance is 4.63% less than that obtained. The results indicate that our method can solve the DVRPTW with a limited number of vehicles, and return effective solutions in a very short time.

The rest of this paper is organized as follows. In Section 2, we present a mathematical model for the DVRPTW. In Section 3, we describe some heuristic methods from the ALNS algorithm for solving the DVRPTW. The computational results are shown in Section 4. Finally, important conclusions from the results are discussed.

2. Problem description

To better understand the DVRPTW, Fig. 1 illustrates an instance of the DVRPTW, with one central depot, two vehicles, a set of static customers, and a set of customers dynamically requesting during work hours. The house located in the center of the graph is the depot, which is the origin and destination of the vehicles. The white nodes denote the static customers, while black ones denote dynamic customers. Fig. 1(a) presents a static routing scenario with seven customers whose orders were known in advance. To complete their orders, two vehicles are allotted to deliver goods. Fig. 1(b) shows how the old plan is influenced by new orders from customers 8 and 9. In point, the first vehicle is too far from customer 8 to serve her/him in the specified time window, so customer 8 is rejected (denoted by dashed node 8 in Fig. 1(c)). In contrast, the order by customer 9 is acceptable, and the second vehicle can visit her/him just in time. Finally, two vehicles return to the depot after completing all orders.

Formally, a DVRPTW is defined by a complete graph $G=(V, E)$, where V represents a set of vertices consisting of a depot node v_0, and customer nodes, and $E=\{(i, j): i, j\in V, i\neq j\}$ denotes a set of arcs, each of which represents the known travel cost t_{ij} between the node i and j. In formalizing the DVRPTW, customers who placed their orders before planning (i.e. their locations, demands, and time windows are known beforehand) are called static customers and are denoted as V_s. On the other hand, customers whose orders are placed dynamically are dynamic customers, denoted as V_d. Thus, the set $V' ,= V_s \cup V_s = \{v_1, v_2, \ldots, v_n\}$ represents all customers, and a solution of the DVRPTW is a path (e.g. v_0, v_1, v_2, v_3, v_0) in G that starts from v_0, sequentially visits certain customers (e.g. v_1, v_2, v_3,) and returns to v_0.

In the following formulation, each customer $v_i \in V'$ can be represented as a vector $v_i = (x_i, y_i, q_i, s_i, e_i, l_i, T_i, b_i)$, where (x_i, y_i) represents the location of customer v_i, q_i denotes his/her demand, s_i is the service time, e_i is the earliest start time, l_i is the latest start time, T_i is the request service time (for a static customer, for which $T_i = 0$), and b_i is the begin service time. $[e_i, l_i]$ defines the time window for this customer service. Thus, a vehicle must wait if it arrives at customer v_i before e_i. Each arc $(i, j) \in E$ associated with a cost (i.e. a travel distance d_{ij} or a travel time t_{ij}), represents the optimal route that utilizes vehicles to satisfy customer services while ensuring economic goals (i.e. minimal distance d_{ij} in the present study). To do so, each customer must be served once by a vehicle, and each vehicle k must takes only one route, which is associated with a nonnegative capacity Q_k greater than or equal to the summation of all demands on that route. Two binary decision variables ξ_{ijk} and χ_k are defined as follows:

$\xi_{ijk} = 1$ if arc (i, j) is travelled by vehicle k, and 0 otherwise.

$\chi_k = 1$ if vehicle k is used, and 0 otherwise.

The DVRPTW model can be read as follows:

$$\text{minimize} \sum_{(i,j)\in E} \sum_{k\in K} d_{ij} \cdot \xi_{ijk} + \gamma \cdot \sum_{k\in K} \chi_k \cdot g_k \tag{1}$$

where g_k corresponds the fixed cost of vehicle k. Subject to:

$$\sum_{i\in V} \xi_{ijk} = \sum_{i\in V} \xi_{jik} \quad j \in V', k \in K \tag{2}$$

$$\sum_{k\in K} \sum_{j\in V} \xi_{ijk} = 1 \quad i \in V' \tag{3}$$

$$\sum_{j\in V} \xi_{0jk} = \sum_{i\in V} \xi_{i0k} = 1 \quad k \in K \tag{4}$$

$$\sum_{i\in V'} \sum_{j\in V} q_i \xi_{ijk} \le Q_k \quad k \in K \tag{5}$$

$$a_i = b_{i-1} + s_i + t_{i,i-1} \quad i \in V' \tag{6}$$

$$b_i = \max\{a_i, e_i\} \tag{7}$$

$$e_i \le b_i \le l_i \tag{8}$$

$$\xi_{ijk}, \chi_k \in \{0, 1\} \tag{9}$$

The objective function (1) aims at minimizing the total travel costs and the fixed costs of used vehicles, where γ is a coefficient. Constraint (2) is a flow conservation constraint. Each customer's in-degree should equal to his/her out-degree, which is at most one. Constraint (3) ensures that each customer must be visited by exactly one vehicle. Constraint (4) exists to make sure each route starts and ends at the central depot. Constraint (5) specifies the capacities of each vehicle. Constraints (6–8) define the time windows. Finally, constraint (9) imposes restrictions on the decision variables.

3. Proposed approach

We propose an ALNS algorithm aimed at finding suitable high quality solutions for the DVRPTW within a reasonable time frame. The proposed approach is described in Algorithm 1. The algorithm starts from an initial solution containing all static customers that is constructed by *InitSol()* (see Algorithm 2). The obtained solution is then optimized by the ALNS (described in Algorithm 3). Once the optimized solution S^* is returned, vehicles begin to deliver goods for the customers while allowing a customer to submit an order dynamically. Whenever a new request i occurs (Line 3), the algorithm will check the feasibility of the request (Line 4). If the request is acceptable, then it is added to the request pool U and waits to be inserted into the current solution (Line 5). This will invoke the ALNS to improve the current optimal solution again (Line 6). Otherwise, it will be rejected (Line 8). This dynamic procedure is repeated until there is no new request that is not serviced.

In the next subsections, the procedures invoked by Algorithm 1 from the construction of the initial solution to obtaining the dynamic solutions are described in detail.

3.1. Initial solution

The initial algorithm starts with an empty solution, and the request pool U is initialized with all of the static customers. It sequentially adds routes and further extends them with customers guided by specific criteria until it obtains a feasible solution that contains all the customers. A route $r = (v_0, v_{\mathrm{seed}}, v_0)$ is initialized with a seed customer v_{seed} who is farthest from the depot or has the earliest deadline for service, After initializing the current route, the algorithm tries to insert a customer somewhere in the current route r. The customer can be inserted into the route as long as this does not violate the constraints of the service time window and capacity. Usually, there are several feasible insert positions for each customer in the request pool. A customer is elected if he/she has a feasible insertion that is better than the others. Insertions are repeatedly tested until there are no more customers with a feasible insertion. If such a customer does exist, his/her current route is inserted into the solution S, which continues to build the next route until all customers are scheduled.

The quality of feasible insertions is formally characterized via a cost function. For a customer k in the request pool, let i and j be two adjacent customers in the current route r, the cost of inserting k between i and j is defined as:

$$G(i,k,j) = \mu \times d(i,k,j) + (1-\mu) \times t(i,k,j) \ \ (0 \le \mu \le 1), \tag{10}$$

where *d(i, k, j)* is defined in Eq. (11) as the extra travel distance caused by the insertion of k.

$$d(i,k,j) = d_{ik} + d_{kj} - d_{ij}. \tag{11}$$

$t(i, k, j)$ is calculated in Eq. (12) as the delay of the service start time of the customer j after inserting customer k.

$$t(i,k,j) = b'_j - b_j, \tag{12}$$

$0 \le \mu \le 1$ is a coefficient that controls the relative importance of the travel distance and time constraint.

Each feasible insertion position is checked for each candidate customer. The customer with the least cost is then inserted at its best position. In other words, the customer k^* is inserted into route r between i^* and j^* for which

$$(i^*, k^*, j^*) = \arg\min_{i,j \in r} G(i,k,j). \tag{13}$$

3.2. Adaptive large neighborhood search

The ALNS algorithm is an extension of the Large Neighborhood Search (LNS) algorithm [16], based on destroy and repair heuristics [17]. The LNS repeatedly searches for the solution until some stopping condition is met. At each iteration, the LNS destructs some elements from a solution and rebuilds the partial solution with these elements, and a new solution is thus generated. The new solution is accepted as the incumbent if some criterion (e.g., simulated annealing) is met, and otherwise the search continues. Unlike LNS, the ALNS applies several competing destroy and repair heuristics instead of one destroy and one repair heuristic in the search process. The probability of applying a certain heuristic depends on its performance in the past. Each heuristic has a weight that characterizes its probability of be employed during the next search. In the following, we present the ALNS in more details.

Large neighborhood: The neighborhood of a solution is enlarged by a destroy heuristic, which removes some customers from the solution, and a repair method, which rebuilds the partial solution as in [18], thereby improving the overall search. At each iteration step, some customers are removed from the current solution and placed in a removal list. Then, some customers are selected from the removal list to insert into the current partial solution using the repair heuristics. If there are some customers that cannot be inserted, their number will be used to increase the penalty in the objective function accordingly. If the solution is accepted by some criterion, the current iteration is over, and the remaining customers are used to set up the removal list for the next iteration.

Adaptive search: Several heuristics are considered during iteration. As in [19], a heuristic (either the removal or the insertion heuristics) is selected by using a roulette-wheel selection principle where each of the heuristics is assigned a weight depending on its past performance. By combining the destroy and repair heuristics, we assign a weight to each pair, rather than rewarding each destroy and repair operator individually. Kovacs et al. [20] also tried this selection mechanism with slightly better results. The probability for choosing an operator pair is proportional to the weight ω_{dr} of each destroy-repair pair (*d, r*). Given destroy η_d and repair η_r heuristics, we calculate the selection probabilities as follows:

$$\rho_{dr} = \frac{\omega_{dr}}{\sum_{d=1}^{\eta_d} \sum_{r=1}^{\eta_r} \omega_{dr}}. \tag{14}$$

***Adaptive weight adjustment*:** The search is divided into several segments, each consisting of φ consecutive iterations. Each pair (*d, r*) is assigned a weight ω_{dr} and a score π_{dr}, and weights are updated according to the score obtained during the current segment. In the first segment, all weights are equal to one. At the beginning of each segment, the scores are set to zero, and throughout the segment, the observed score is increased with σ_1, σ_2, and σ_3 (where $\sigma_1 > \sigma_2 > \sigma_3$), depending on the following conditions:

σ_1, if the generated solution is a new global best solution.
σ_2, if the generated solution is better than the current one.

σ_3, if the generated solution is worse than current one but is still accepted.

At the end of each segment, the weight ω_{dr} is updated considering the scores obtained during this segment. Let o^i_{dr} be the number of the destroy-repair pair that is called in each segment i. The weight ω_{dr} is updated using the following equation

$$\omega_{dr} = \begin{cases} \rho\frac{\pi_{dr}}{o^i_{dr}} + (1-\rho)\omega_{dr} & o^i_{dr} \neq 0 \\ \omega_{dr} & o^i_{dr} = 0 \end{cases} \tag{15}$$

where $\rho \in [0, 1]$ is the reaction factor that controls the influence of the destroy-repair pair according to its weights. With $\rho = 0$, the weights do not depend on the scores and remain at their initial level. If $\rho = 1$, only the scores obtained during the segment are accounted for. While $0 < \rho < 1$, both the recent score and the past performance of each pair are considered.

***Penalized objective function*:** Like previous studies, our methods allow infeasible solutions during the search, but a penalty function is associated with infeasible solutions. The penalized objective function is defined as follows:

$$F(s) = f(s) + \lambda \cdot |\mathcal{U}| \cdot f(s_0), \tag{16}$$

where $\lambda = 0.1$ is a parameter that controls the unscheduled customers' penalty.

***Later perturbation*:** After ψ repeat times, we apply a 2-opt* operator to each vehicle route with the aim of enhancing the improvement and modifying an infeasible solution to be feasible. It is noted that the parameter ψ affects the performance of ALNS. If the perturbation is too large, the information contained within the locally optimal solution will be lost. This could lead to an infeasible solution that will require additional computation time to make the solution feasible. However, the ALNS will tend to get trapped in a local optimum if the perturbation does not work well. Hence, the effect of the size of the perturbation on the performance of the ALNS is also studied in Section 5.2.

***Acceptance and stop criteria*:** After finding a new solution we check whether it can be accepted. The following criteria are used to make the determination: accept the new solution if it is an improved solution or accept the worse solution with probability $p = \exp((F(s) - F(s'))/T)$. This method is the same as a simulated annealing method, in which the parameter T is the current temperature, and the initial temperature is:

$$T = -\frac{\tau}{\ln\theta} F(s_0). \tag{17}$$

Ropke [19] suggested $\tau = 0.05$ and $\theta = 0.5$, i.e. a solution that is 5% worse than s_0 is accepted with a probability 0.5. The ALNS stops when a specific number of consecutive unimproved iterations M have been run.

The ALNS works as follows (see Algorithm 2). It starts from an initial best solution S_{best}, with all weights and scores. Next, as it has not met the stop criterion, the following procedure is repeated. In each iteration step, we select a destroy-repair pair by employing the roulette-wheel mechanism with current weights. Then our destroy method erases a part of the currently serviced request from S_{curr}, and repair heuristics rebuilds the partial solution with these requests. After that, a new solution S_{new} is generated. Lines 8–17 check S_{new} against an acceptance criterion and update S_{best} as well as the score for the operator pair depending on S_{new}. After several iterations, it updates the weights and set the scores to 0 in Line 18. To avoid the local optimum, 2-opt* is applied to perturb the current solution. Finally, it returns the best found solution (Line 24).

3.2.1. Destroy heuristics

After a solution is constructed, another type of heuristic is required to destroy the current solution by removing several already scheduled customers. The procedure for generic removal is presented in Algorithm 3. Given a solution S and the removal number q, the algorithm elects a customer i from routes in S on the base of destroy heuristics, and the solution is updated by moving customer i into the removal list $\mathcal{L}$. The new solution is obtained by repeatedly removing q customers.

Let $\mathcal{R}$ be a set of customers served on a route in solution S. The algorithm initially sets up the removal list $\mathcal{L}$ with the request pool $\mathcal{U}$, and a ranking index $\mathcal{J}_k$ is calculated for each unvisited customer k in $\mathcal{R}$. The solution is then curtailed by repeatedly removing a chosen customer. For each selection, a custom $\mathcal{R}_i$ is chosen randomly by the degree of randomization $d \geq 1$, to avoid removing and reinserting some customers repeatedly. We repeat this process to remove q customers from their routes.

1. ***Random Removal (RR)***. There is no ranking index in this case, and the customers are randomly selected for remove operation. This heuristic tends to generate a poor set of removed customers, but it helps to diversify the search and escape local optima.
2. ***Worst Removal (WR)***. The purpose is to remove customers with high cost and then replace them to try to find a better solution. A customer is removed if deleting her/him from the current solution leads to the maximal cost saving. We compute the cost saving as the difference between the cost with the customer versus without her/him. More precisely, for a customer $k (k \in \mathcal{R})$, the index $\mathcal{J}_k$ is defined as follows:

$$\mathcal{J}_k = F(S) - F_{-k}(S), \tag{18}$$

where $F_{-k}(S)$ is the cost of the solution without customer k (the customer is removed but is not moved to $\mathcal{L}$). $\mathcal{R}$ is ranked in non-increasing order of this index.

3. ***Static-related Removal (SR)***. This heuristic attempts to remove customers that are similar in some respect. At first, we randomly select a reference customer i from $\mathcal{L}$ (if $\mathcal{L}$ is empty, we randomly choose a customer from $\mathcal{R}$ and put the customer into $\mathcal{L}$). Thereafter, we calculate the index as the relevance of the reference customer i to each customer j in $\mathcal{R}$ and sort them in non-decreasing order. This measure is defined as:

$$\mathcal{J}_k = r_{ij} = \alpha \cdot \frac{d_{ij}}{M_d} + \beta \cdot \frac{|l_i - l_j|}{M_t}, \tag{19}$$

where M_d and M_t are scaling constants, α and β defines the weight for relatedness in terms of routing costs (based on travel distance), and ending of time windows, respectively.
4. ***Time-related Removal (TR).*** This removal operator was inspired by the algorithm of [21] for the VRPTW. It removes customers that are serviced at the same time. A specific time t_s is first randomly selected as a seed, and then the customers are ordered increasingly according to an index $\mathcal{J}_k$ defined as

$$\mathcal{J}_k = \begin{cases} b_k - t_s & \text{if} \, t_s < b_k \\ 0 \quad \text{if} & b_k \le t_s \le z_k, \\ t_s - z_k & \text{if} \, t_s > z_k \end{cases} \tag{20}$$

3.2.2. Repair heuristics

In this section, we describe the repair heuristics used in Algorithm 2. A repair heuristic is utilized to iteratively insert the removed customer into the routes guided by a specific measure Δ_i. The repair heuristics consist of the following:

1. ***Greedy insertion (GI)***. This greedy insertion is a simple constructive heuristic, which repeatedly inserts the customers according to a cost measure. The insert cost is calculated as

$$\Delta_i = \Delta H(i, k, r) = d_{ik} + d_{k(i+1)} - d_{i(i+1)}. \tag{21}$$

If a customer i can be inserted into the router at position k, otherwise $\Delta H(i,\ k,\ r) = \infty$. To determine the best insert position, we calculate

$$(i^*, k^*, r^*) = \arg\min \cdot H(i, k, r). \tag{22}$$

It inserts customer i^* in route r^* at its minimum cost position k^*. This process continues until all requests have been inserted or no more requests can be inserted.
2. ***Regret insertion***. The regret heuristic is based on a regret value, or look-ahead value, which not only considers the best placement but also considers the second one, the third one and so on. A regret-k insertion locates where to put the customer with the biggest cost difference that is between the best placement and the $k-1$ alternatives. More precisely, the heuristic calculates the regret value as

$$\Delta_i = \max_{i \in L} \left\{ \sum_{h=2}^{k} (F_h(i) - F_1(i)) \right\}, \tag{23}$$

where $F_k(i)$ is the kth best place to insert customer i. At each iteration, the heuristic chooses the customer i^* to be inserted is according to:

$$i^* = \arg\max_{i \in L} \cdot \Delta_i. \tag{24}$$

In implementation, we are mainly concerned with two regret levels: *regret-2 (**R2I**)*, and *regret-3 (**R3I**)*.
3. ***Noise Insertion***. This operator extends the previous two heuristics by adding a noise term to the insertion cost. Each time, we calculate a customer's insertion cost, we then add some noise and calculate a modified insertion cost.

$$\Delta_i' = \Delta_i + \eta \cdot z \cdot d_{max}. \tag{25}$$

Parameter η controls the amount of randomization and is set equal to 0.25, $d_{\max}$ is the maximum distance between customers, and $z \in [-1,1]$.

3.3. Insertion of dynamic customer

As mentioned above, Algorithm 1 checks the feasibility of inserting a new request customer in the current solution before deciding whether to accept or reject this request. In this paper, there are three cases in which a customer cannot be inserted in the current solution: (1) there is no vehicle (including empty vehicle) that can respond to the dynamic customer service, (2) the vehicle capacities are violated, and (3) the time windows of the dynamic customer is violated. In any of the three cases, the dynamic customer will be rejected and will not be considered any longer.

We describe how to insert a dynamic request or reject it in detail. Suppose a dynamic customer v_h submits a service at time RT_h, and the vehicle k is visiting the customer v_k or is on its way in the planned route r. We check the following cases.

First, the customer v_h can be feasibly visited if:

$$b_k + s_k + t_{k,h} \le l_h \ \exists k \in K, \tag{26}$$

or

$$RT_h + t_{0,h} \le l_h \ \wedge \ RT_h + 2 * t_{0,h} + s_h \le l_0. \tag{27}$$

In other words, a feasible solution must satisfy at least one vehicle or be able to deliver a new vehicle to the customer v_h before its time window closes. Otherwise the dynamic customer is assigned to the reject pool and will not be considered any longer.

Next, if a feasible solution for such vehicle(s) is found, we then examine the vehicle capacities. If there is a vehicle to meet the following equation, we continue to check the remaining cases. Otherwise the dynamic customer is rejected and will not be added to reject pool.

$$\sum_{i \in r} q_i + q_h \le Q_k \ \exists k \in K. \tag{28}$$

After that, we check to see whether there exists a feasible insertion point. The time window is checked using the notion of forward beginning service time and backward beginning service time for each customer. For a given route $r = \langle v_0, v_1, \ldots, v_n, v_{n+1} \rangle$, the forward beginning service time, b_i, is defined in Eq. (7) as described in Section 2. Additionally, we define the backward beginning service time, z_i, as:

$$\begin{cases} z_{n+1} = l_{n+1} \\ z_i = \min\left(z_{i+1} - t_{i,i+1} - s_i \ , l_i\right) \quad 0 \le i \le n \end{cases}. \tag{29}$$

A customer v_h can be feasibly inserted into route $r = \langle v_0, \ldots, v_x, v_y, \ldots, v_0 \rangle$ between two partial paths $\langle v_0, \ldots, v_x \rangle$ and $\langle v_y, \ldots, v_0 \rangle$ if:

$$\begin{cases} b_x + s_x + t_{x,h} \le l_h \\ z_y - t_{y,h} - s_h \ge e_h \\ b_h \le z_h \end{cases}. \tag{30}$$

If an insertion point is not found, the new dynamic customer is rejected and added to the reject pool.

Finally, if the new dynamic customer does not violate the above three rules, it is inserted into the request pool, and the ALNS algorithm is applied to improve the resulting solution.

4. Experimental results

This section presents our experimental results using the algorithms described in Section 3. To validate the performance of our heuristics, we carried out many experiments using the relevant benchmarks. We first describe the benchmarks instances. We then report the results along with a comparative analysis.

All of the proposed algorithms have been implemented in the C# programming language for .NET Framework 4.0 and executed on a machine with the following configuration: Inter® Pentium® CPU G645 processor @ 2.90 GHz, 2 G RAM running Windows 7.

4.1. Benchmark

In our experiments we use Lackner's benchmark, which originates from the standard VRPTW Solomon benchmark and, which can be accessed at http://web.cba.neu.edu/~msolomon/problems.htm. Solomon's benchmark consists of 56 instances. Every instance contains 100 customer nodes and one depot node. Those nodes distribute into 100×100 Euclidean plane on which travel times equal the corresponding distances. The benchmark instances are divided into six groups, R1, R2, C1, C2, RC1, and RC2, each containing 8–12 instances.

Lackner accommodated the standard VRPTW Solomon benchmark to test the DVRPTW by requiring that a fraction of the customer's request be performed dynamically. The dynamic degree of the problem is characterized by:

$$Dod = \frac{n_d}{n} \times 100\% \tag{31}$$

where n_d denotes the number of dynamic customers, and n denotes the total number of customers. Five values of the dynamic degree (10%, 30%, 50%, 70%, and 90%) are considered for each instance in the Solomon. By combining different locations, horizontal lengths, and degrees of dynamism, the 56 base instances are extended into 280 instances. In the following experiments, we label each instance by "name-Dod", in which the name is the name of the base instance and Dod is the value of the dynamic degree. For example, instance "R101-90" means that it origins from the R101 Solomon instance and contains 90 dynamic customers.

4.2. Comparison with existing approaches

To assess the performance improvement, we run the present algorithm against Lackner's benchmark and compare with existing techniques such as ILNS [8], and GVNS [10]. We carry out comparisons of these techniques based on the ratios

Table 1
Parameters setting

Parameter	Meaning	Value
μ	Constraint coefficient	0.5
q	The number of removed customs	[10,40]
d	The degree of randomization parameter	8
α	The weight of travel distance relatedness	1
β	The weight of ending time windows relatedness	1
η	The noise parameter	0.25
σ_1	New global solution gain	1
σ_2	Better solution gain	0.4
σ_3	Worse solution but accepted gain	0.25
φ	The number of iterations in each segment	200
ρ	Reaction factor	0.9
λ	Penalty factor	0.1
τ	The reference degradation of the objective function	0.05
θ	the probability of accepting a solution with the reference degradation	0.5
c	The cooling rate	0.99975
M	The number of iterations in each unimproved search	500
ψ	The number of iterations for each perturb	200

Table 2
Comparison of the experimental results of the proposed method with other methods.

G	D	ALNS				ILNS				GVNS			
		VN	TD	IT	RR	VN	TD	IT	RR	VN	TD	IT	RR
R1	90	**13.67**	1270.20	8.61	3.92	14.25	1335.94	17.43	**2.33**	14.67	**1250.38**	14.50	3.83
	70	**13.42**	1298.86	11.31	2.83	14.33	1331.34	21.73	**1.75**	14.75	**1267.78**	10.95	3.08
	50	**13.58**	1313.35	15.22	2.17	14.08	1295.81	28.27	**0.67**	14.58	**1267.47**	11.84	1.92
	30	**13.33**	1310.23	21.99	1.50	13.92	1286.63	46.59	**0.58**	14.25	**1256.04**	15.70	1.58
	10	**13.33**	1309.10	34.19	0.58	13.50	1257.08	67.99	**0.17**	14.17	**1250.16**	15.29	0.50
C1	90	**10.44**	974.41	9.96	0.11	10.78	1039.77	6.60	0.22	10.67	**963.33**	7.81	**0.00**
	70	**10.33**	1088.38	12.45	0.11	10.78	1031.68	10.79	0.22	11.33	**1009.47**	7.67	**0.00**
	50	**10.44**	1096.58	17.37	0.11	10.89	1001.18	19.01	0.22	11.00	**992.97**	6.22	**0.00**
	30	**10.56**	1126.84	23.62	0.11	10.56	962.08	28.03	0.33	11.56	**949.95**	9.13	**0.00**
	10	**10.44**	1102.80	30.89	0.11	10.56	895.77	15.40	0.22	10.56	**898.30**	13.74	**0.00**
RC1	90	**13.63**	1501.76	5.04	**1.75**	14.00	1513.94	17.31	2.00	14.63	**1470.45**	15.39	1.88
	70	**13.13**	1510.21	6.65	**1.75**	13.88	1511.29	25.32	1.88	14.88	**1489.28**	13.43	2.13
	50	**13.63**	1520.47	8.92	**1.38**	13.63	1514.72	48.78	1.38	14.50	**1484.01**	13.72	1.75
	30	**13.00**	1484.89	12.71	**0.63**	13.88	1492.22	45.26	1.13	14.38	**1471.00**	16.51	1.00
	10	**12.88**	1473.69	16.94	**0.25**	13.38	1436.23	83.52	1.13	13.50	**1417.07**	23.01	0.50
R2	90	**3.45**	1076.77	21.82	**0.00**	3.55	**1047.82**	13.20	0.09	4.00	1086.78	16.47	**0.00**
	70	**3.36**	1088.97	29.25	**0.00**	3.64	**1032.04**	20.15	0.09	4.36	1078.03	12.74	**0.00**
	50	**3.45**	1086.09	39.30	**0.00**	3.82	**1016.52**	30.03	**0.00**	4.55	1071.83	11.96	**0.00**
	30	**3.45**	1087.52	59.93	**0.00**	4.91	**985.59**	57.07	**0.00**	4.73	1035.60	10.18	**0.00**
	10	**5.09**	1080.50	75.21	**0.00**	6.36	**950.00**	68.58	0.09	5.27	1000.00	9.48	**0.00**
C2	90	3.38	668.45	11.86	**0.00**	**3.25**	**636.79**	6.12	**0.00**	3.38	668.99	16.67	**0.00**
	70	3.25	652.63	13.32	**0.00**	**3.13**	**636.47**	10.01	**0.00**	3.38	672.95	14.03	**0.00**
	50	3.25	650.70	18.49	**0.00**	**3.13**	**604.98**	16.80	**0.00**	3.13	623.10	20.25	**0.00**
	30	**3.63**	658.09	24.22	**0.00**	**3.63**	651.42	29.87	**0.00**	3.25	**624.81**	34.82	**0.00**
	10	3.25	624.06	40.25	**0.00**	**3.00**	**594.67**	59.70	**0.00**	3.25	615.93	80.78	**0.00**
RC2	90	**3.88**	1264.94	11.31	**0.00**	4.00	**1257.19**	11.34	0.13	4.63	1275.93	28.05	**0.00**
	70	**3.88**	1261.81	14.74	**0.00**	3.88	1239.46	19.26	**0.00**	5.13	**1234.36**	16.07	**0.00**
	50	**3.88**	1260.66	20.67	**0.00**	4.25	**1190.54**	27.84	0.13	5.88	1200.26	11.46	**0.00**
	30	**3.88**	1238.82	30.76	**0.00**	5.38	**1166.04**	41.51	0.25	5.88	1172.33	11.68	**0.00**
	10	**5.75**	1253.11	42.89	**0.00**	6.75	**1103.30**	55.55	**0.00**	6.13	1153.43	13.27	**0.00**
Avg.		**8.09**	1144.50	23.00	0.58	8.50	1100.62	31.64	**0.50**	8.88	**1098.40**	16.76	0.61

of refuse service, the number of vehicles and the total distance traveled. The comparisons are presented in the following tables, in which the first column indicates the group name of instances (G), the second column shows degrees of dynamism (D) and other columns show the average vehicle number (VN), total distance (TD), insertion time (IT) and the ratio of refuse customer (RR). In collecting experimental data, ten separate executions are performed for each Lackner's benchmark instance, and we have chosen the best execution for each instance and grouped them together to calculate the average values of each group with different degrees of dynamism. The parameters are set as shown in Table 1.

As seen in the average row of Table 2, the average ratio of refuse service of ALNS is 0.58%, which is better than that of GVNS (0.61%), and slightly larger than that of ILNS (0.50%). The average vehicle number of ALNS is superior to the existing three algorithms. Their average results are 8.09 (ALNS), 8.50 (ILNS), and 8.88 (GVNS). In particular, The ALNS improved the

Table 3
Computational results for the ALNS.

G	*D*	Ratio of refuse				Avg. vehicle number				Avg. total distance			
		Best	Worst	Avg.	GVNS	Best	Worst	Avg.	GVNS	Best	Worst	Avg.	VNS
R1	90	3.92	3.58	3.65	**3.33**	13.67	15.58	**14.41**	15.34	1270.20	1387.89	1333.11	**1328.74**
	70	2.83	2.33	2.79	**2.42**	13.42	14.83	**13.95**	15.31	1298.86	1386.23	1345.85	**1340.38**
	50	2.17	1.83	1.88	**1.67**	13.58	14.83	**14.04**	15.33	1313.35	1421.24	1383.66	**1340.17**
	30	1.50	1.25	1.40	**1.17**	13.33	14.58	**13.56**	14.96	1310.23	1413.81	1366.60	**1312.73**
	10	0.58	0.58	0.54	**0.33**	13.33	14.33	**13.78**	14.73	1309.10	1418.95	1369.74	**1296.59**
C1	90	0.11	0.11	0.11	**0.00**	10.44	11.22	**10.56**	11.37	974.41	1182.01	1105.24	**1092.18**
	70	0.11	0.11	0.11	**0.00**	10.33	11.56	**10.76**	11.92	1088.38	1322.81	1207.47	**1150.27**
	50	0.11	0.11	0.11	**0.00**	10.44	11.56	**10.82**	11.81	1096.58	1390.96	1262.96	**1129.58**
	30	0.11	0.11	0.11	**0.00**	10.56	11.44	**10.99**	11.81	1126.84	1436.46	1294.45	**1081.52**
	10	0.11	0.11	0.11	**0.00**	10.44	11.67	**11.12**	11.36	1102.80	1408.86	1269.66	**986.99**
RC1	90	1.75	1.50	**1.48**	1.50	13.63	15.25	**14.38**	15.62	1501.76	1651.64	**1578.27**	1587.89
	70	1.75	1.38	1.53	**1.25**	13.13	14.63	**14.10**	15.88	1510.21	1643.41	**1585.80**	1614.43
	50	1.38	1.00	1.04	**0.88**	13.63	14.75	**13.89**	15.51	1520.47	1650.57	1586.02	**1579.34**
	30	0.63	0.75	0.78	**0.63**	13.00	14.25	**13.56**	15.22	1484.89	1620.66	1564.33	**1551.93**
	10	0.25	0.25	0.31	**0.25**	12.88	14.00	**13.28**	14.25	1473.69	1610.58	1539.29	**1474.09**
R2	90	0.00	0.00	**0.00**	**0.00**	3.45	4.64	**3.67**	3.88	1076.77	1145.14	**1133.29**	1181.31
	70	0.00	0.00	**0.00**	**0.00**	3.36	4.36	**3.55**	4.22	1088.97	1155.53	**1138.44**	1161.98
	50	0.00	0.00	**0.00**	**0.00**	3.45	4.73	**3.69**	4.49	1086.09	1201.51	**1146.93**	1153.79
	30	0.00	0.00	**0.00**	**0.00**	3.45	4.00	**3.35**	4.77	1087.52	1182.04	1142.10	**1112.92**
	10	0.00	0.00	**0.00**	**0.00**	5.09	5.50	**5.31**	5.49	1080.50	1193.41	1137.68	**1054.82**
C2	90	0.00	0.00	**0.00**	**0.00**	3.38	4.63	3.79	**3.66**	668.45	815.11	742.73	**749.33**
	70	0.00	0.00	**0.00**	**0.00**	3.25	4.50	**3.70**	3.71	652.63	837.39	745.04	**722.45**
	50	0.00	0.00	**0.00**	**0.00**	3.25	4.75	3.76	**3.53**	650.70	809.99	761.64	**670.23**
	30	0.00	0.00	**0.00**	**0.00**	3.63	4.38	3.69	**3.36**	658.09	834.73	741.69	**670.88**
	10	0.00	0.00	**0.00**	**0.00**	3.25	3.75	**3.38**	3.53	624.06	800.13	746.43	**660.93**
RC2	90	0.00	0.00	**0.00**	**0.00**	3.88	4.63	**3.96**	8.02	1264.94	1353.99	**1327.66**	2032.46
	70	0.00	0.00	**0.00**	**0.00**	3.88	5.00	**4.00**	4.94	1261.81	1400.27	**1339.28**	1359.18
	50	0.00	0.00	**0.00**	**0.00**	3.88	5.00	**4.08**	5.39	1260.66	1387.33	1347.17	**1311.97**
	30	0.00	0.00	**0.00**	**0.00**	3.88	4.63	**3.96**	5.83	1238.82	1372.79	1342.04	**1278.62**
	10	0.00	0.00	**0.00**	**0.00**	5.75	6.50	**5.86**	6.01	1253.11	1355.36	1328.66	**1240.55**
Avg.		0.58	0.50	0.53	**0.45**	8.09	9.18	**8.43**	9.38	1144.50	1293.03	1230.44	**1207.61**

Algorithm 1 Proposed approach.

```
1:  S←InitSol();//construct an initial solution containing all static customers
2:  S*←ALNS(S);
3:  While there is a request i appears do
    // check request feasibility
4:      If (i can inserted into a (or new) route in S) then
5:          Add customer i to U;
6:          S*←ALNS(S*);
7:      Else
8:          Add i to reject pool; //reject service customer i
9:      End if
10: End While
11: Return S*
```

solution quality of 26 out of the 30 groups of benchmark instances (86.67%). This means that the ALNS is better than the other algorithms in finding the minimum number of vehicles while satisfying all new requests. With regard to the average travel distance, the results of ALNS are slightly longer than those of the existing algorithms. The average distance of ALNS is 1144.50. It is worse than that of ILNS (1100.62), and of GVNS (1098.40), while the maximum average error is less than 4.63%. This means that, under the condition of a lower rate of refuse service and fewer service vehicles, ALNS is still able to find a more appropriate solution. Finally, the average computation time for each customer is 23 s. In order words, when a customer makes a requests, he/she can get his/her service order after 23 s. Specially, the longest computation time is no more than 75 s, and the fastest is 5 s. Other algorithms are estimated to be 25.00 (ILNS) and 19.94 (GVNS). It can be seen that, on average, the computation time of ALNS is basically at the same level as that of ILNS and GVNS.

To have better assessment of the performance improvement, we select the average and the worst of the 10 tests for each instance, and again, we calculate the average values of each group with different degrees of dynamism. As shown by the summarized experimental results in Table 3, the best ratio of refuse averages is 0.58 with a range of 0 to 3.92. For all instance classes, the average ratio of refuse averages to 0.53, while the worst averages to 0.50. The best average vehicle number ranges from 3.25 to 13.67 for all instance classes and averages to 8.09, while the average “average vehicle number” ranges from 3.35 to 14.41 and averages to 8.43. For every instance class, the average “average total distance” is 7.5% longer

Algorithm 2 ALNS(S).

```
1:  Initialize: set all weights equal to one and all scores equal to zero.
2:  $S_{best} \leftarrow S_{curr} \leftarrow S$, $iter \leftarrow 1$, $nonIter \leftarrow 1$;
3:  Repeat
4:      Select a destroy-repair pair $(d, r)$ using the roulette-wheel mechanism
5:      Apply the operator pair $(d, r)$ to $S_{curr}$, obtain a new solution $S_{new}$
6:      $++iter$, $++nonIter$
7:      If $F(S_{new}) < F(S_{best})$ then // Acceptance criteria
8:          $S_{best} \leftarrow S_{curr} \leftarrow S_{new}$; $nonIter \leftarrow 1$
9:          Update the score for the operator pair used with $\sigma_1$
10:     Else if $F(S_{new}) < F(S_{curr})$ then
11:         $S_{curr} \leftarrow S_{new}$
12:         Update the score for the operator pair used with $\sigma_2$
13:     Else if $S_{new}$ is accepted by the simulated annealing criterion then
14:         $S_{curr} \leftarrow S_{new}$
15:         Update the score for the operator pair used with $\sigma_3$
16:     End if
17:     If $iter$ mod $\varphi$ ==0 then //Adaptive weight adjustment
18:         Update the weight of all operator pairs and reset their scores
19:     End if
20:     If $iter$ mod $\psi$ ==0 then //Later perturbation
21:         Perform a 2-opt and 2-opt* to perturb $S_{curr}$.
22:     End if
23: Until $nonIter$ >M //Stop criteria
24: Return $S_{best}$
```

Algorithm 3 The generic structure of the removal procedure.

```
1.  Initialize the removal list with request pool, $\mathcal{L} \leftarrow \mathcal{U}$
2.  For each customer $k$ in $\mathcal{R}$ do
3.      Calculate the ranking index $\mathcal{J}_k$
4.  End for
5.  While $|\mathcal{L}| < q$ do
6.      Sort $\mathcal{R}$ according to the $\mathcal{J}_k$
7.      Choose a random number $z$ in [0, 1)
8.      $i \leftarrow z^d|\mathcal{R}|$
9.      $\mathcal{L} \leftarrow \mathcal{L} \cup \{\mathcal{R}_i\}$
10.     Remove $\mathcal{R}_i$ from $S$
11.     Update the $\mathcal{J}_k$ of all remaining customers if necessary
12. End while
13. Return $S$ and $\mathcal{L}$
```

than the best value (1144.50). We see that the average performance is not as good as the best performance data that we used before. However, the next experiment will show that the proposed algorithm still performs better than the well-known GVNS in several cases.

As in the experiments done for GVNS, we also calculated the average performance of 10 executions for each customer and compared with GVNS. As shown by the figures in bold in Table 3, ALNS is superior to GVNS in terms of average vehicle number in most cases, especially for instances R1, RC1, R2 and RC2 in which request customers are randomly located and even have tight time windows. In contrast, GVNS seems to perform better for instances C1 and C2 if request customers are in clusters and have longer time windows. On RC1-90 instances, ALNS outperforms the other algorithms.

5. Conclusions

In this paper, we have studied the DVRP with a limited number of vehicles and hard time windows – a new variant of DVRPTW with wider applications to real life transportation, distribution and logistics. To construct efficient and reliable routes, we have decomposed such a problem into a series of static VRPs to capture new customer requests occurring in a time slice. Starting from an initial feasible solution for a static VRP, we have optimized the solution through an enhanced ALNS with destroy-repair heuristics and a periodic later perturbation. We have assessed the present approach by conducting several experiments with a well-known benchmark. In terms of the ratios of refuse service, we conclude that the improved ALNS is better than others on most of the instances (Group R2, C2, and RC2). For the average vehicle number, our ALNS improved 26 solutions out of the 30 groups of benchmark instances (86.67%), while the computation time of our ALNS was basically at the same level as that of IVNS and ILNS. However, for the average travel distance, our ALNS performed slightly worse than the others while the maximum error was less than 4.63%. We can conclude that our ALNS provides a better solution for the DVRPTW with a limited number of vehicles with a reasonable computation time.

However, the proposed ALNS does not perform well in cases (e.g. Group R1) in which the customers' locations are distributed randomly, and their service time window is relatively tight. Therefore, improving the algorithm's performance in such situations is a topic of future research. Another potential future research direction is to enhance the present ALNS to tackle more general dynamic routing problems that can address customer satisfaction and customer priority for dynamic requests

Acknowledgments

This work is supported by the National Natural Science Foundation of China (nos. 61672122, 61602077, and 61402070), the Public Welfare Funds for Scientific Research of Liaoning Province of China (no. 20170005), the Natural Science Foundation of Liaoning Province of China (no. 20170540097), and the Fundamental Research Funds for the Central Universities (no. 3132016348).

References

[1] Dantzig GB, Ramser JH. The truck dispatching problem. Manage Sci 1959;6:80–91.
[2] Siemiński A. Solving dynamic traveling salesman problem with ant colony communities. In: Nguyen NT, Papadopoulos GA, Jędrzejowicz P, Trawiński B, Vossen G, editors. Computational collective intelligence: 9th international conference, ICCCI. Springer International Publishing; 2017. Nicosia, Cyprus, September 27–29, 2017, Proceedings, Part ICham, 2017, p. 277–87.
[3] Avci M, Avci MG. A GRASP with iterated local search for the traveling repairman problem with profits. Comput Ind Eng 2017;113:323–32.
[4] Lois A, Ziliaskopoulos A. Online algorithm for dynamic dial a ride problem and its metrics. Transp Res Procedia 2017;24:377–84.
[5] Ghiani G, Manni E, Romano A. A dispatching policy for the dynamic and stochastic pickup and delivery problem. In: Ntalianis K, Croitoru A, editors. Applied physics, system science and computers: proceedings of the 1st international conference on applied physics, system science and computers (APSAC2016), Dubrovnik, Croatia. September 28–30. Springer International Publishing: Cham; 2018. p. 303–9.
[6] Guertin F, Potvin J-Y, Taillard É. Parallel tabu search for real-time vehicle routing and dispatching. Transp Sci 1999:33.
[7] Pankratz G. Dynamic vehicle routing by means of a genetic algorithm. Int J Phys Distrib Logist Manage 2005;35:362–83.
[8] Hong L. An improved LNS algorithm for real-time vehicle routing problem with time windows. Comput Oper Res 2012;39:151–63.
[9] de Armas J, Melián-Batista B. Constrained dynamic vehicle routing problems with time windows. Soft Comput 2015:1–18.
[10] de Armas J, Melián-Batista B. Variable neighborhood search for a dynamic rich vehicle routing problem with time windows. Comput Ind Eng 2015;85:120–31.
[11] Saint-Guillain M, Deville Y, Solnon C. A multistage stochastic programming approach to the dynamic and stochastic VRPTW. In: Michel L, editor. Integration of AI and OR techniques in constraint programming: 12th international conference, CPAIOR 2015. Barcelona, Spain, Springer International Publishing; 2015. p. 357–74. May 18–22, 2015, ProceedingsCham.
[12] Pillac V, Gendreau M, Guéret C, Medaglia AL. A review of dynamic vehicle routing problems. Eur J Oper Res 2013;225:1–11.
[13] Ritzinger U, Puchinger J, Hartl RF. A survey on dynamic and stochastic vehicle routing problems. Int J Prod Res 2016;54:215–31.
[14] Psaraftis HN, Wen M, Kontovas CA. Dynamic vehicle routing problems: three decades and counting. Networks 2016;67:3–31.
[15] Solomon MM. Algorithms for the vehicle routing and scheduling problems with time window constraints. Oper Res 1987;35:254–65.
[16] Shaw P. Using constraint programming and local search methods to solve vehicle routing problems. In: Maher M, Puget J-F, editors. Principles and practice of constraint programming—cp98: 4th international conference, CP98 Pisa, Italy, October 26–30, 1998 Proceedings. Springer, Berlin Heidelberg; 1998. p. 417–31.
[17] Schrimpf G, Schneider J, Stamm-Wilbrandt H, Dueck G. Record breaking optimization results using the ruin and recreate principle. J Comput Phys 2000;159:139–71.
[18] Milthers NPM. Solving VRP using Voronoi diagrams and adaptive large neighborhood search. Department of Computer Science, University of Copenhagen; 2009.
[19] Ropke S, Pisinger D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transp Sci 2006;40:455–72.
[20] Kovacs A, Parragh S, Doerner K, Hartl R. Adaptive large neighborhood search for service technician routing and scheduling problems. J Scheduling 2012;15:579–600.
[21] Prescott-Gagnon E, Desaulniers G, Rousseau LM. A branch and price based large neighborhood search algorithm for the vehicle routing problem with time windows. Networks 2009;54:190–204.

Shifeng Chen received his M.S. in Computer Science and Technology from the Dalian University. He is currently pursuing his Ph.D. in the area of combinatorial optimization using heuristic and metaheuristic techniques at Dalian Maritime University.

Rong Chen is a professor in the Dalian Maritime University, China. He received the M.S. and Ph.D. degree in computer software and theory from the Jilin University, China, in 1997 and 2000. He has previously held position at Sun Yat-sen University, China. His research interests are in software diagnosis, collective intelligence, activity recognition, Internet and mobile computing.

Gai-Ge Wang is an associate professor in the Ocean University of China. He is a senior member of SAISE, SCIEI, a member of IEEE, IEEE CIS, and ISMOST. His research interests are swarm intelligence, evolutionary computation, and its applications. He just proposed three bio-inspired algorithms (monarch butterfly optimization, earthworm optimization algorithm, and elephant herding optimization).

Jian Gao is an Associate Professor in the Dalian Maritime University, China. His current research interests are in combinatorial optimization using heuristic and metaheuristic techniques.

Arun Kumar Sangaiah is an associate professor in School of Computer Science and Engineering, VIT University, India. He had received his Ph.D. degree in Computer Science and Engineering from the VIT University, Vellore, India. His area of interest includes software engineering, computational intelligence, wireless networks, bio-informatics, and embedded systems.