Egyptian Informatics Journal 25 (2024) 100421

Contents lists available at ScienceDirect

Egyptian Informatics Journal

journal homepage: www.sciencedirect.com

Full Length Article

Solving dynamic vehicle routing problem with time windows by ant colony system with bipartite graph matching

Yi Teng [a], Jinbiao Chen [b], Shiyuan Zhang [b], Jiahai Wang [b,c], Zizhen Zhang [b,c,*]

[a] School of Computer Science, Guangdong University of Education, Guangzhou 510303, China
[b] School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 51006, China
[c] Guangdong Key Laboratory of Big Data Analysis and Processing, Sun Yat-sen University, Guangzhou 51006, China

ARTICLE INFO

Keywords:
Dynamic vehicle routing problem
Ant colony system
Kuhn-Munkres algorithm
Bipartite graph matching
Metaheuristics

ABSTRACT

Dynamic Vehicle Routing Problem with Time Windows (DVRPTW) is an extension of the traditional VRPTW by considering dynamic customer characteristics. In this problem, new customers with specific time windows are received dynamically as time progresses and must be incorporated into the evolving schedule. The goal of DVRPTW is to use the minimum number of vehicles to fulfill all the customer demands with the minimum total traveling time. Little work has been done for solving this dynamic variant. In this paper, an evolutionary algorithm using Ant Colony System and Kuhn-Munkres bipartite graph matching, named ACS-KM, is proposed. It can tackle the dynamic issues during the evolutionary process. Experiments show that the proposed algorithm is very effective and has fast convergence. It achieves the best results on 40 out of 48 benchmark instances. For the DVRPTW instances with different levels of dynamicity, the algorithm can always produce better solutions than the previous works.

1. Introduction

The logistics industry plays an important role in our daily lives, such as material transportation, good collection, express delivery and so on. In the traditional logistics industry, the route planning mainly relies on expert experience. However, this way is inefficient and seriously limits the development of the logistics industry. With the development of intelligent systems in recent years, the logistics industry strives to find a good plan automatically and intelligently so as to increase the economic benefits as well as improve the operational efficiency.

Vehicle routing problem (VRP) is a classic NP-hard optimization problem [13] that can characterize most route planning requirements of the logistic industry. It targets at finding the most economic routes to serve a set of customers given the locations, demands and other features of the customers. Generally speaking, vehicles with a certain capacity start from the depot, serve a set of customers with a certain demands along the route, and finally return to the depot. In the past years, many studies on VRP and its variants, which can well model different realistic problems, have been published in the literature [47]. A well-known extension of VRP is to include the time window of each customer, known as VRPTW. Because customers may not be free all the time in practice, VRPTW requires that each customer must be served within a specified time window.

With the recent development of mobile communication and positioning technology, it is possible that new customer requests appear dynamically by communicating with the vehicle dispatching center. Then some vehicles can change their visiting order to respond to those new requests. This brings in a dynamic version of VRP (DVRP). As pointed out by Braekers et al. [4], DVRP is attracting much attention because it is closer to real-life applications. Some information such as customer requests, demands, traveling time between customers may not be determined in advance and will appear unpredictably along with the planning horizon. Here, we consider the most common case as studied in Soeffker et al. [41]: the customer requests are dynamically revealed. That is, a vehicle may receive new customer requests and then incorporate them into the schedule by rearranging those unvisited requests in a certain order. The dispatching center can decide which vehicle should respond to which new requests, or send a new vehicle to serve them.

In this paper, we focus on a more challenging version of VRP variants called the Dynamic Vehicle Routing Problem with Time Window

* Corresponding author.
E-mail addresses: tengyi@gdei.edu.cn (Y. Teng), chenjb69@mail2.sysu.edu.cn (J. Chen), 945831769@qq.com (S. Zhang), wangjiah@mail.sysu.edu.cn (J. Wang), zhangzizhen@gmail.com, zhangzzh7@mail.sysu.edu.cn (Z. Zhang).

https://doi.org/10.1016/j.eij.2023.100421
Received 19 July 2023; Received in revised form 14 November 2023; Accepted 17 November 2023
Available online 8 December 2023

(DVRPTW), which associates VRPTW with DVRP. The difficulty of DVRPTW lies in its uncertainty and irreparability. For example, some customers may appear earlier and have loose time window constraints, but they are far away from a vehicle. If the vehicle responds to the needs of these customers immediately, there may be some near-by customers with tight time window constraints that cannot be fulfilled. In this case, new vehicles have to be dispatched, thereby leading to additional cost.

In order to solve VRPTW, some exact algorithms such as Column Generation [17], Branch-and-Cut [2], Branch-and-Price [14], have been developed. They can generally solve small-scale cases of VRPs. Heuristics and meta-heuristics are more popular in scientific research since they are powerful tools to address large-scale complicated VRPs [39,45] and scheduling problems, such as flexible job shop scheduling [19], task allocation in wireless sensor networks [31], and power allocation [3]. Among various meta-heuristics, ant colony algorithm is widely applied to tackle VRPTW and has a relatively simple implementation. If the problem is extended to dynamic scenarios, the solutions need to be updated and maintained. In this case, the pheromone trails in the ant colony algorithm can help to guide the selection of subsequent customers. Although there have been extensive studies on VRPTW, research on the related DVRPTW is relatively scarce. Most studies still rely on simple evolutionary with local search methods. While these methods have broad applicability, they often converge slowly and can only provide limited optimization effects due to limited computational resources. Local search only optimizes a small number of customers and vehicles, with little consideration for large-scale customer and vehicle scheduling. In fact, with the appropriate methods, the complexity of achieving large-scale optimization can be comparable to the traditional local search methods.

Based on the above ideas, we propose an improved ant colony system (ACS) algorithm. In order to enhance the search efficiency, neighboring solutions need to be fast explored and time window constraints need to be fast checked at the same time. According to the characteristics of DVRPTW solution, we introduce a bipartite graph matching model within the neighborhood search process and employ the Kuhn-Munkres (KM) matching algorithm to solve it. It essentially performs a large neighborhood search for exploring more promising solutions in the search space.

Our contributions are summarized as follows. First, we apply ACS in DVRPTW, which is more practical and complex than static VRPTW but only reported in little literature. Next, a bipartite graph matching algorithm is devised to effectively improve the solution during the ACS evolving process. Finally, the performance of our algorithm can outperform the other existing approaches according to the numerical experiments.

The remaining of the paper is structured as follows. Section 2 reviews the related literature. Section 3 gives the problem definition of DVRPTW. Section 4 presents the proposed ACS-KM algorithm. Section 5 describes the experimental study and reports the results. Section 6 makes the conclusions.

2. Literature review

Our work mainly focuses on DVRPTW, which is a significant variant of VRP. Based on the traditional ACS methods, we propose an improved ACS algorithm to address the problem. Thus, we review the literature related to DVRPTW and the ACS methods as follows.

VRPTW and its variants have received wide attention. Numerous research has devised efficient solution methods for VRPTWs. One type of these methods is the exact optimization approach, which can obtain the optimal solution on small-to-medium size cases. For the basic VRPTW, Desrochers et al. [15] proposed an algorithm based on the set partitioning formulation and the column generation approach. For the split delivery VRPTW, Bianchessi & Irnich [2] proposed a new and tailored branch-and-cut algorithm based on a new, relaxed compact model, where some integer solutions could be infeasible. For the two-echelon VRPTW, Dellaert et al. [14] developed a branch-and-price algorithm with two path-based mathematical formulations. For a variant of the truckload open vehicle routing problem with time windows, Faiz et al. [17] presented an arc-based mixed integer linear programming model solved by a general solver as well as a path-based formulation solved by their proposed column generation framework. The other type of solution methods is the meta-heuristic approach. Bräysy & Gendreau [6,7] presented a survey of route construction heuristics, local search algorithms and meta-heuristics for VRPTW. Schneider et al. [39] provided insights about the design of effective and efficient granular solution methods for VRPTW by using granular neighborhoods and sparsification methods to improve the run-time of local-search-based metaheuristics. Ticha et al. [45] proposed a multi-graph model and an adaptive large neighborhood search heuristic algorithm for VRPTW. Some hybrid algorithms for VRPTW have also been reported. Yassen et al. [52] proposed an adaptive harmony search hybrid algorithm that embeds an adaptive selection mechanism to adaptively select a suitable local search algorithm to be applied. Pérez-Rodríguez & Hernández-Aguirre [34] proposed a hybrid estimation of distribution algorithm, which uses the generalized Mallows distribution as a probability model to describe the distribution of the solution space. Due to the generality and flexibility of metaheuristics, it works well in even some complicated variants of VRPTW, such as balanced cargo VRPTW [24], open VRPTW [5], and heterogeneous VRPTW [27].

Considering the dynamicity in practical problems, several researchers focus on DVRP. Psaraftis [36] first proposed an idea of periodic re-optimization to solve a dynamic dial-a-ride problem. Kilby et al. [23] introduced some scenarios of dynamic VRPs. Montemanni et al. [28] simulated a dynamic VRP. The concept of working day T is introduced. During the process, T is cut into n time slices with equal length. In each time slice, a static VRP is created and solved by the ant colony system. AbdAllah et al. [1] studied the same type of DVRP as that in Montemanni et al. [28]. They used an improved genetic algorithm to try to improve the quality and diversity of solutions. DVRPTW contains various dynamic characteristics. Chen & Xu [10] studied DVRPTW considering customer demands with dynamic arrival time. They proposed a column-generation based dynamic approach. Wang et al. [48] focused on DVRPTW, in which the customer locations change dynamically. They presented an ensemble learning based multi-objective evolutionary algorithm. Pan et al. [32] studied DVRPTW with time-dependent travel time and implemented an efficient hybrid adaptive large neighborhood search with a tabu search algorithm. Later, Pan et al. [33] studied multi-trip DVRPTW with time-dependent travel time. They designed a hybrid metaheuristic algorithm leveraging the adaptive large neighborhood search for guided exploration and the variable neighborhood descend for intensive exploitation. Yao et al. [51] employed an alternating direction method of multipliers (ADMM) to address a vehicle routing problem with time-dependent travel times and time windows. They developed a computationally reliable decomposition framework to iteratively improve both the primal and dual solution quality. More works on DVRP and its variants can be found in the survey papers Pillac et al. [35], Soeffker et al. [41].

Ant Colony System [16] is a specific kind of Any Colony Optimization (ACO) method, which is designed based on the behavior of ants in nature. It widely used in the study of VRPTW and its variants [8,26,12,25,44]. Stützle & Hoos [43] proposed a max-min ACS by setting upper and lower bounds on the paths that ants search, aiming at relieving iteration stagnation in the ant colony algorithm. Gambardella et al. [18] proposed a multi ant colony framework for VRPTW: MACS-VRPTW, which mainly consists of two ant colonies. One is used to find a legal solution under a given number of vehicles, while the other is used to find the shortest distance ordinarily. This method has been widely applied in dynamic problems. For example, Veen et al. [46] used the combination of cycle optimization method [36] and MACS-VRPTW [18] to solve the DVRPTW, where the customers appear dynamically over time. da Silva Junior et al. [40] also proposed a framework based

Table 1
Summarized literature for DVRPTW.

Dynamic characteristic	Method	Reference
Travel time	Large neighborhood search & Tabu search	Pan et al. [32]
Travel time & Time window	Alternating direction method of multipliers	Yao et al. [51]
Customer location	Evolutionary algorithm & Ensemble learning	Wang et al. [48]
Customer arrival	Dynamic column generation	Chen & Xu [10]
Customer arrival	Multi-Ant Colony System	Veen et al. [46]
Customer arrival	Multi-Ant Colony System & Variable neighborhood descent	da Silva Junior et al. [40]
Customer arrival	Ant Colony System & Local search	Necula et al. [30]
Customer arrival	Ant Colony System & Large neighborhood search	Our paper

on MACS with random variable neighborhood descent to address this problem. Yang et al. [50] adopted the framework of MACS, but they studied the DVRP with a variety of customer priorities. There are also other frameworks of ACS algorithms. For example, Necula et al. [30] proposed a method that uses a single ant colony based on competition of customer selection among all vehicles and compared it with the method proposed by Veen et al. [46]. Xu et al. [49] proposed a hybrid ant colony model for DVRP, which uses a traditional ant colony optimization fusing with improved K-means and crossover operation to extend the search space and avoid falling into local optimum prematurely. More recent works [9,21,37,22] have further improved ACS for VRPTW and achieved good results.

The most relevant literature for DVRPTW is summarized in Table 1. It seems that applying ACS and neighborhood search for DVRPTW is a promising direction. However, it still remains some questions to be addressed such as how to accelerate the search and overcome the shortcoming of premature stagnation.

3. Problem definition

DVRPTW [10] is defined on a complete undirected graph $G = (V, E)$, where V is the vertex set and E is the edge set. The vertex set V consists of a depot 0 and a customer set C, i.e., $V = C \cup \{0\}$. Customers may have different positions. The traveling time between two different positions i and j is t_{ij} (we consider the symmetric case, i.e., $t_{ij} = t_{ji}$). The customer set C consists of two parts: C_{static} and $C_{dynamic}$. Initially, each vehicle starts at depot 0, plans to serve each customer in C_{static} exactly once, and finally returns to the depot. During the planning horizon, some customer requirements in $C_{dynamic}$ may appear, and the original plan must be recomputed to satisfy the dynamic requirements. At the end, all the customers in C would be served.

Let K denote the set of vehicles used. The goal of DVRPTW is to optimize the size of K primarily, and the total traveling time secondly. The customer arrivals, capacity constraints, and time window constraints are considered in DVRPTW, listed as follows.

- Each customer i is associated with a demand d_i to be satisfied, a service time s_i, a time window $[e_i, l_i]$.
- Each customer i has a dynamic arrival time a_i ($a_i \le e_i < l_i$ must hold). Note that $a_i = 0,\ \forall i \in C_{static}$. Otherwise, customer i is emerged at time a_i.
- Each vehicle has an identical capacity Q.
- A vehicle must serve customer i no later than l_i. If the vehicle arrives at customer i before e_i, the service cannot be started.
- Once a vehicle has returned to the depot, it cannot be dispatched again.

For a static VRPTW, it can be formulated as the following mixed integer linear programming model [10]. The binary decision variables x_{ijk} indicate whether vehicle k traverses from vertex i to vertex j. The decision variables y_{ik} specify the starting time of the service at vertex i by vehicle k.

$$\min \sum_{k\in K}\sum_{i\in V}\sum_{j\in V} t_{ij}x_{ijk} \tag{1}$$

subject to

$$\sum_{k\in K}\sum_{j\in V} x_{ijk} = 1,\ \forall i \in C \tag{2}$$

$$\sum_{j\in C} x_{0jk} = \sum_{j\in C} x_{j0k} = 1,\ \forall k \in K \tag{3}$$

$$\sum_{i\in V} x_{ijk} = \sum_{i\in V} x_{jik},\ \forall k \in K, j \in C \tag{4}$$

$$y_{ik} + s_i + t_{ij} - y_{jk} \le (1 - x_{ijk})M,\ \forall k \in K, i, j \in V \tag{5}$$

$$e_i \sum_{j\in V} x_{ijk} \le y_{ik} \le l_i \sum_{j\in V} x_{ijk},\ \forall k \in K, i \in C \tag{6}$$

$$\sum_{i\in C} d_i \sum_{j\in V} x_{ijk} \le Q,\ \forall k \in K \tag{7}$$

$$x_{ijk} \in \{0,1\},\ \forall k \in K, i, j \in V \tag{8}$$

$$y_{ik} \ge 0,\ \forall k \in K, i \in V \tag{9}$$

The objective (1) is to minimize the total traveling time. Constraints (2) require that each customer is visited exactly once by some vehicle. Constraints (3) indicate that a vehicle starts from and ends at the depot 0. Constraints (4) denote the equality of inflow and outflow to be followed by vehicle k. Constraints (5) ensure the continuity of the vehicle visits, where M is a sufficiently large number. Constraints (6) are the time window constraints. Constraints (7) guarantee that the vehicle capacity is not exceeded.

Fig. 1 shows an example of DVRPTW, where C_{static} includes customers $\{1, 2, \ldots, 6\}$ and $C_{dynamic}$ includes customers $\{7, 8\}$. The variable b in the block indicates the actual service beginning time of the corresponding customer. It can be observed that the original plan to serve C_{static} is "$1-2-3-4-5-6$". If the plan cannot be changed, it is impossible for the vehicle to serve customers $\{7, 8\}$ after completing customer 6 due to the time window limitation. Then 2 vehicles are needed. The first vehicle takes 23 time units and the second one takes 21 time units. However, if the dynamic scenario is considered, customers $\{7, 8\}$ can be inserted into the schedule next to the service of customer 4 as they have emerged at time 7. In this case, only 1 vehicle with 26 time units is required.

4. The ACS-KM algorithm

To solve DVRPTW, evolutionary algorithms would be good choices, since the dynamic issue can be easily tackled during the evolutionary process. We hereby propose an ACS-KM algorithm, which is based on the ACS framework by further adding a new local optimization method. The reasons for choosing the ACS framework are as follows. Firstly, ACS as a typical ant colony optimization framework, has shown its success in solving various VRPs as indicated in Section 2. Secondly, ACS preserves the pheromone information in the environment, which is useful when the environment has changed a little bit. Compared with other evolutionary algorithms such as genetic algorithm and particle swarm optimization, ACS has several advantages in solving our problem [20]. For example, ACS does not require recoding and decoding of the solution vector. It directly constructs an entire solution by adding the

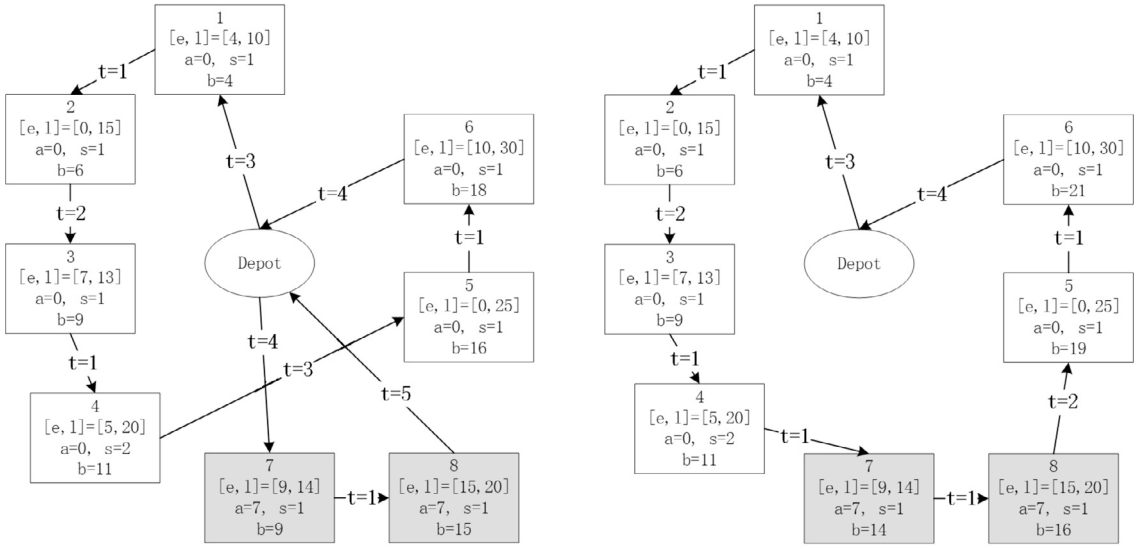


Fig. 1. An example of DVRPTW instance.

unvisited customers to the end of the current partial solution. Hence, it is much easier to handle new arrival customers with capacity and time window constraints.

As shown in Fig. 2, ACS-KM mainly involves three parts: the controller, the ACS-based algorithm and the large neighborhood search based on Kuhn-Munkres bipartite graph matching [29]. It helps to guide the search exploring in a large solution space, thereby avoiding the potential stagnation of ACS.

4.1. Controller

The framework of the optimization system is called the controller. Before the actual operation, a feasible solution that includes all known customers in C_{static} is constructed using the time-oriented nearest neighbor heuristic [42]. Specifically, a vehicle starts from the depot initially. Then the following steps are processed.

1. Search each route in the current solution and get its last visited customer i.
2. Search each customer j in the current unvisited customer list.
3. If customer j can be inserted into the route next to customer i without violating any constraints, calculate m_{ij} (see Equation (10) below), and find the smallest one with indices i^* and j^*.
4. Add customer j^* next to the customer i^*.
5. If no route is available and there are still remaining customers, a new route is open.
6. The process is terminated until there exists no unvisited customers.

As for the value of m_{ij}, it is defined as Formula (10) according to Necula et al. [30].

$$m_{ij} = 0.4 * t_{ij} + 0.4 v_{ij} + 0.2 * u_{ij} \tag{10}$$

where $v_{ij} = b_j - (b_i + s_i)$ and $u_{ij} = l_j - (b_i + s_i + t_{ij})$. Note that the service beginning time $b_j = \max(e_j, b_i + s_i + t_{ij})$ if customer j is served immediately after customer i. v_{ij} measures the waiting time of the corresponding vehicle, and u_{ij} measures the urgency of the delivery to customer j. In sum, m_{ij} is a heuristic function evaluating the appropriateness of assigning customer j next to customer i.

Once a feasible solution is made for the static customers, it is used to initialize the best solution S_{best}. After the solution initialization, the controller starts to arrange the dynamically arriving customers. To do so, the planning horizon T is split into n periods, each lasts t time units,

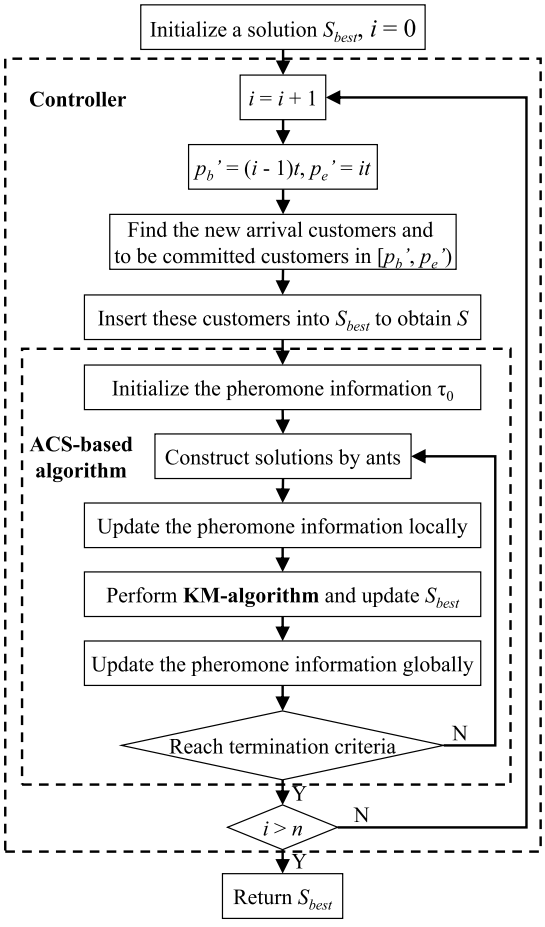


Fig. 2. Flow diagram of ACS-KM algorithm.

where n and t are preset parameters. In the beginning of each period, say p_b, the controller checks the following:

1. There are new customer(s) arrived in the last period, say $[p_b', p_e' = p_b)$.

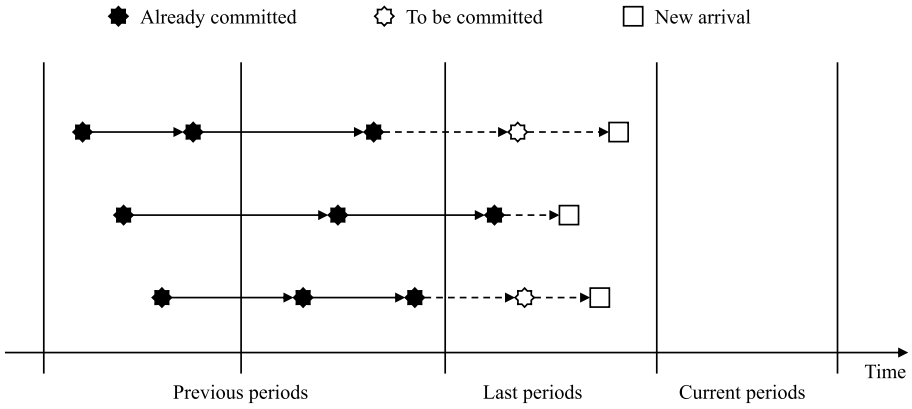


Fig. 3. An example of how controller works.

2. There are new customers to be committed after the last period. Note that a customer i is defined as committed, if $b_i \leq p_b$, i.e., the designated service beginning time is smaller than the start of the current period.

Fig. 3 gives an illustration of newly arrival customers and the customers to be committed. If one of the conditions holds, the controller inserts the corresponding customers into the current best solution S_{best}, and then restarts the ACS-based algorithm.

The insertion strategy is originated from [42]. It starts with computing the best suitable position to insert for each node, and then chooses a node which has the minimal insertion value. When inserting customer u between customer i and customer j, the value of insertion is computed as Formula (11).

$$I = 2 * t_{0u} - t(i,u,j), \tag{11}$$

where t_{0u} is the traveling time between the depot and customer u. $t(i,u,j)$ is calculated as Equation (12).

$$t(i,u,j) = 0.1 * t_1(i,u,j) + 0.9 * t_2(i,u,j), \tag{12}$$

where $t_1(i,u,j) = t_{iu} + t_{uj} - t_{ij}$ is the saving of total time [11], $t_2(i,u,j) = b_{ju} - b_j$ is the saving of beginning time.

If there is no place of existing routes that can be inserted for customer u, a new route starts from the depot is created. This process ends if there is not any remaining customers.

In summary, the controller framework is presented in Algorithm 1.

Algorithm 1 The controller framework.

```
Initialize a solution S with customers in C_static
S_best ← S
for i ← 1 to n do
    Set p'_b ← (i − 1) · t and p'_e ← i · t
    Find the new arrival customers in C_dynamic and to be committed customers in [p'_b, p'_e)
    Insert these customers into S_best to obtain a solution S
    Invoke ACS-based algorithm to improve S
    Update S_best
end for
```

4.2. ACS-based algorithm

In Section 4.1, the controller calls ACS-based algorithm at the beginning of a period when the environment is changed. The framework of ACS-based algorithm is described in Algorithm 2. It uses a standard ant colony system to generate solutions. Then the pheromone information is updated in a local manner. Next, it uses a Kuhn-Munkres bipartite graph matching to optimize the generated solutions and updates S_{best} accordingly. Finally, the pheromone information is updated again in a global manner. This process continues until a new environment is occurred or the final period is completed.

Algorithm 2 The framework of ACS-based algorithm.

```
Initialize the pheromone information
while termination criteria not reached do
    Construct solutions by ants
    Update the pheromone information locally
    Perform KM-algorithm and update S_best
    Update the pheromone information globally
end while
```

At the very beginning, the pheromone trail τ_0 is initialized as follows.

$$\tau_0 = \frac{1}{n_s \cdot L_{NN}} \tag{13}$$

where n_s is the number of static customers and L_{NN} is the total traveling time of S_{best}.

In the ant construction part of the algorithm, each ant corresponds to a particular solution. It is worth noting that those already committed customers must be copied to each ant and fixed. The remaining customers are then iteratively added into the routes of an ant. The next customer s to be added into a route v is chosen according to the following equation.

$$(v,s) = \begin{cases} \arg\max_{v \in K, s \in C} (\tau^{\alpha}_{r(v),s} \cdot \eta^{\beta}_{r(v),s}), & \text{rand}(0,1) < q_0 \\ VS, & \text{otherwise} \end{cases} \tag{14}$$

In the equation, K is the current route set, $r(v)$ is the last customer of route v, C is the neighborhood set of customer r. The neighborhood of a customer is defined as its top cl closest customers, where cl is a user-defined parameter. τ_{rs} is the pheromone trail of edge (r,s). η_{rs} is the heuristic function computed as: $\eta_{rs} = (1/m_{rs})^{\beta}$, where m_{rs} has the same meaning as Equation (10). α, β and q_0 are also user-defined parameters. VS is a (v,s) pair randomly chosen according to the following probability distribution.

$$p_{vs} = \begin{cases} \frac{\tau^{\alpha}_{r(v),s} \cdot \eta^{\beta}_{r(v),s}}{\sum_{v \in K, s' \in C} (\tau^{\alpha}_{r(v),s'} \cdot \eta^{\beta}_{r(v),s'})}, & \text{if } s \in C \\ 0, & \text{otherwise} \end{cases} \tag{15}$$

The above calculation is actually the most standard form of ACS. Note that the chosen of pair (v,s) must respect the capacity and time window constraints. If there are customers which cannot be added into the existing routes, a new vehicle route starting from the depot is added. The construction is ended when all the available customers are added into the solution.

After the ant construction, if an ant moves from r to s, it updates the pheromone trail by Equation (16).

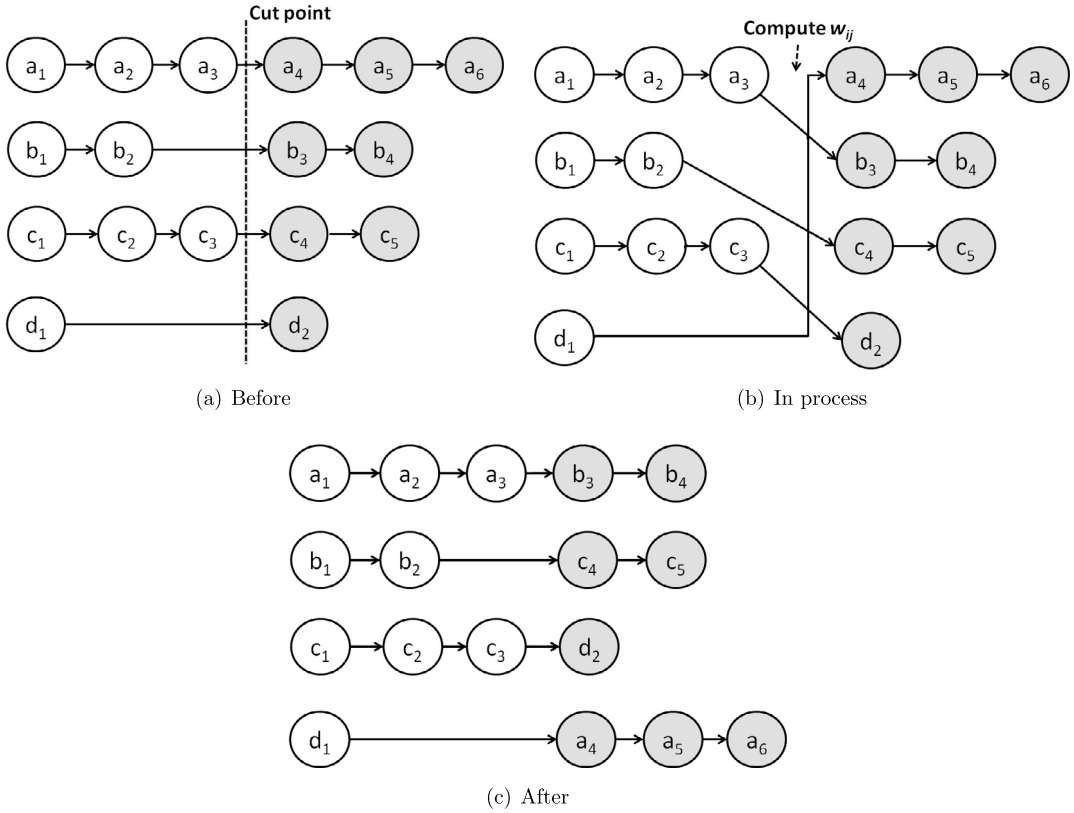


Fig. 4. An example of KM algorithm.

$$\tau_{rs} = (1 - \rho_{local}) \cdot \tau_{rs} + \rho_{local} \cdot \tau_0 \tag{16}$$

where ρ_{local} is a local pheromone decay parameter. τ_0 is referred to Equation (13) except that n_s and L_{NN} are changed to the current environment.

After all the ants complete their solution construction, the best solution (with the minimal number of vehicles first and total traveling time second) among the ants is selected. Then the Kuhn-Munkres algorithm is invoked to further optimize this solution. Finally, the pheromone trail on each edge is updated using the global best solution S_{best}.

$$\tau_{rs} = (1 - \rho_{global}) * \tau_{rs} + \begin{cases} \rho_{global}/L_{gb}, & \text{if } (r,s) \in S_{best} \\ 0, & \text{otherwise} \end{cases} \tag{17}$$

where L_{gb} is the total traveling time of S_{best}, ρ_{global} is the global decay parameter.

4.3. Large neighborhood search based on KM algorithm

In a typical ACS algorithm, it is generally difficult to modify a customer's relative position between routes during the ant construction. In literature, Savelsbergh [38] proposed relocate and exchange local search operators. The relocate operator removes one customer from one route and inserts it into another route. The exchange operator swaps the positions of two customers from two different routes. However, these two operators have some shortcoming of inefficiency. Only a few customers can alter their positions in each step. For this reason, a large neighborhood optimization is involved to further improve the solution. We therefore introduce a bipartite graph matching method. Its basic idea is to divide the routes into different parts and recombine them. It works as follows.

First, split all the routes of a solution into two parts (A, B) based on a cut point t_{cut}. The customers in a route whose service beginning time is less than or equal to t_{cut} are assigned to A, otherwise it is assigned to B (see Fig. 4(a)). Next, calculate the weight w_{ij} between the last visited customer i in A and the first visited customer j in B. w_{ij} is the total traveling time of the new route by concatenating i with j. If the concatenation leads to an infeasible route (the time window or capacity constraints are violated), set $w_{ij} = \infty$. Afterwards, apply the Kuhn-Munkres algorithm [29] to the bipartite graph (A, B, w) to find the minimum weight matching (see Fig. 4(b)). Following this matching, a possibly improved solution can be generated (see Fig. 4(c)).

Now it remains the question of how to decide a cut point. Simply enumerating all the cut point is inefficient, as finding a bipartite graph matching is time-consuming. Because the cut point t_{cut} is smaller than the service beginning time of any customers in part B, the next cut point to be selected is the smallest service beginning time of the customer in B. In this manner, t_{cut} is increased constantly. When part B is empty, the KM algorithm terminates and the best found solution is returned. In this way, by making use of the property that the relative positions of customers remain unchanged in static VRPs, we not only find neighboring solutions in the large solution space but also ensure the quality of the search.

The proposed large neighborhood search algorithm is presented in Algorithm 3. The complexity analysis of this algorithm is as follows. Let n_v be the number of vehicles currently used, n_c be the number of customers in the routes. Building the bipartite graph requires $O(n_c n_v^2)$, as there are $O(n_v^2)$ weights and each weight takes $O(n_c)$ for its calculation. Finding the minimum bipartite graph matching requires $O(n_v^3)$. The matching would be performed at most $O(n_c)$ times. Therefore, the total time complexity is $O(n_c^2 n_v^2 + n_c n_v^3)$.

5. Experiments

Our algorithms were implemented with Java. All the experiments were conducted on a machine with Intel(R) Core(TM) i5-4210U CPU clocked at 2.40 GHz and 8 GB RAM. Our source code is available at https://github.com/sysupoetry/ACS-KM-code/blob/master/ACS-KM.zip.

Algorithm 3 The large neighborhood search based on KM algorithm.

Initialize t_{cut}: the earliest service start time of customers that have not yet been served in S
Customers whose service start time is earlier than or equal to t_{cut} belong to Set A, and others belong to Set B
Each weight w_{ij} is obtained by connecting the last customer i of each vehicle in A with the first customer j of each vehicle in B
Initialize ans: total traveling time of solution S
while termination criteria not reached **do**
  Compute the optimal $w^{\star}$ of the bipartite graph (A, B, w) by KM algorithm
  $cur \leftarrow w^{\star}$
  if $cur < ans$ **then**
    $ans \leftarrow cur$
    Update the bipartite matching scheme
  end if
  Find the customer i in B whose service start time e_i is the earliest
  Move customer i to A
  if $B = \emptyset$ **then**
    Break
  end if
  $t_{cut} \leftarrow e_i$
end while
return $S^{\star}$

5.1. Datasets

To our best knowledge, there is only one dataset for DVRPTW modified from classic VRPTW benchmark instances [42], which contain 100 static customers. These instances were extended to the dynamic cases, where a dynamicity level of $X\%$ is specified. It means that there are $X\%$ requests dynamically revealed during the planning horizon. The value of X is in the interval $[0, 100]$, where $X = 0$ indicates that all the requests are static, and $X = 100$ means that they are dynamic.

The datasets are composed of six different problem data types (C1, C2, R1, R2, RC1, RC2). The types C1 and C2 have clustered nodes, R1 and R2 have nodes randomly located, RC1 and RC2 have a combination of randomly located and clustered nodes. The type with number 1 has a narrow time window constraints, while the type with number 2 has a wide time window constraints. For each type, there are four different dynamicity levels: 0%, 10%, 50% and 100%.

5.2. Parameter setup

The parameters are set as follows: $q_0 = 0.9$, $\alpha = 1.0$, $\beta = 1.0$, $\rho_{local} = \rho_{globel} = 0.9$. The number of ants is 10. In the simulation part, $T = 100$ seconds, $n = 50$, so $t = 2$ seconds for each period. The neighborhood size cl is set to 20. Most of the settings are the same with Necula et al. [30], in which they have already tuned several parameters. Then we can make a fair comparison with their work.

5.3. Results and analyses

We mainly compare our ACS-KM results with ACS results reported in Necula et al. [30]. For each instance, we conducted 30 runs and obtained the best ("Best"), average ("Avg."), standard deviation ("Std.") of two terms: the number of vehicles as NV, and the total traveling time as TT. The results are reported in Table 2. The naming of each instance consists of two parts separated by a dash: the data type and the dynamic factor.

From this table, it is easy to find that for most of the instances, the performance of ACS-KM is better than that of ACS. For the data types R1 and R2 generated by the random customer distribution, we can see that ACS-KM can improve ACS significantly on many instances due to its strong convergence. For the data types C1 and C2 generated by clustering customers, ACS-KM are competitive with ACS in terms of the best solution. This is because most of these cases have already reached the optimal solution. Compared with ACS, ACS-KM does not show an advantage with respect to "Best TT" when $X = 50\%$. This may be due to that the KM process makes the current solution trapped into the local optimum. However, considering "Avg. NV" and "Avg. TT", ACS-KM still outperforms ACS for many cases. For the data types RC1 and RC2 generated from a mixture of random and clustered customers, "Best TT" produced by ACS-KM is better than that produced by ACS except for RC101-1.0 in RC1 and RC203-1.0 in RC2. In summary, ACS-KM performs better than or equal to ACS with regard to the "Best TT" on 40 out 48 instances, and ACS-KM performs worse than ACS with regard to the "Avg. TT" on only 3 instances. Moreover, the standard deviation values are all small. This fact demonstrates the robustness of the ACS-KM algorithm.

Note that the ACS results in Table 2 are directly retrieved from Necula et al. [30]. For a fair comparison, we also execute their codes on our machine. 30 independent runs are conducted and then the average solution is calculated. We also try to fine-tune the number of ants on ACS-KM to attain better performance. Finally, we find that ACS-KM with 5 ants can lead to the excellent performance. We present the corresponding results on Type R1 and R2 instances in Table 3. The table shows that the results of ACS by our execution have very little difference with ACS reported in Necula et al. [30]. For ACS-KM with 5 ants, its average performance can improve ACS-KM with 10 ants by 0.9%, i.e., the average total time decreases from 1219.6 to 1208.9. A t-test at 5% significance level between ACS-KM and ACS shows that most of their results are significantly different, as those p-values are generally less than 0.05.

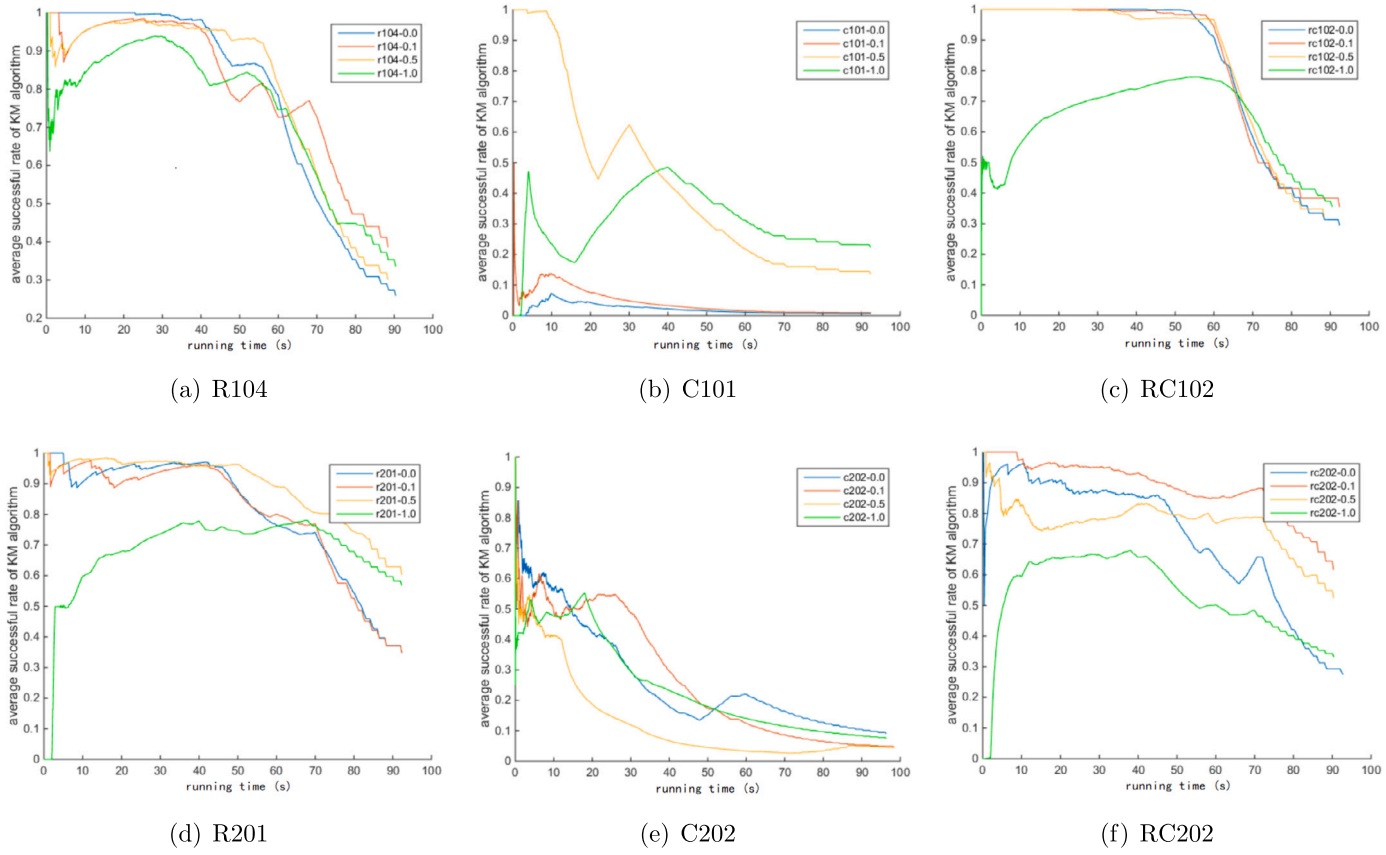
We conduct additional experiments to validate the performance of ACS-KM. We also consider the Dynamic Column Generation (DCG) approach proposed by Chen & Xu [10] for comparison. Their results for static problems can be used as a benchmark. Table 4 presents the results of ACS, ACS-KM, and DCG on 12 static VRPTW instances. It can be observed that the average solution quality of ACS-KM is 1066.1, which is 2.25% lower than that of DCG. This demonstrates the effectiveness of ACS-KM.

Fig. 5 shows the average success rate of ACS-KM algorithm on six selected instances: R104, R201, C101, C202, RC102 and RC202. The success rate is calculated by the number of times that the algorithm improves a solution over the number of iterations, so it is no greater than 1.0 along the planning horizon. From this figure, we can find that for the instance with a low dynamicity level, the curve shows a gradual downward trend. This is because there are few dynamic customers added in, and it provides insufficient room for the ACS-KM algorithm to optimize in the later stage. For the instances with a high dynamicity level, the curve shows a trend of rising first and then falling. The reason is as follows. In the early stage, the number of customers is relatively small. As time progresses, more and more dynamic customers appear, giving considerable room for the optimization. In the later stage, most customers have been committed and the solution cannot get further improved, so the success rate drops steadily. Besides, for different data sets, the success rate of the function is different. For type C with clustered customers, the success rate is low. For type R with randomly located customers or type RC with clustered and randomly located customers, the ACS-KM algorithm maintains a high success rate at the peak of the curve, and this rate is more than 0.3 and even exceeds 0.5 at the end. The above observations indicate that the ACS-KM algorithm is functioning most of the time.

Figs. 6–8 show the iteration curves of ACS and ACS-KM on three instances with tight time windows: C101, R104 and RC102. Both two algorithms took 30 independent runs, respectively. The median values of 30 runs were recorded.

For the instance C101 (see Fig. 6), most of the two curves are coincident. This is because the clustered customer dataset has a relatively simple structure, and it is easy to find the optimal solution for the algorithms. Even if dynamic customers appear midway, the algorithm can still find the optimal solution quickly, so there is no obvious difference between the two algorithms.

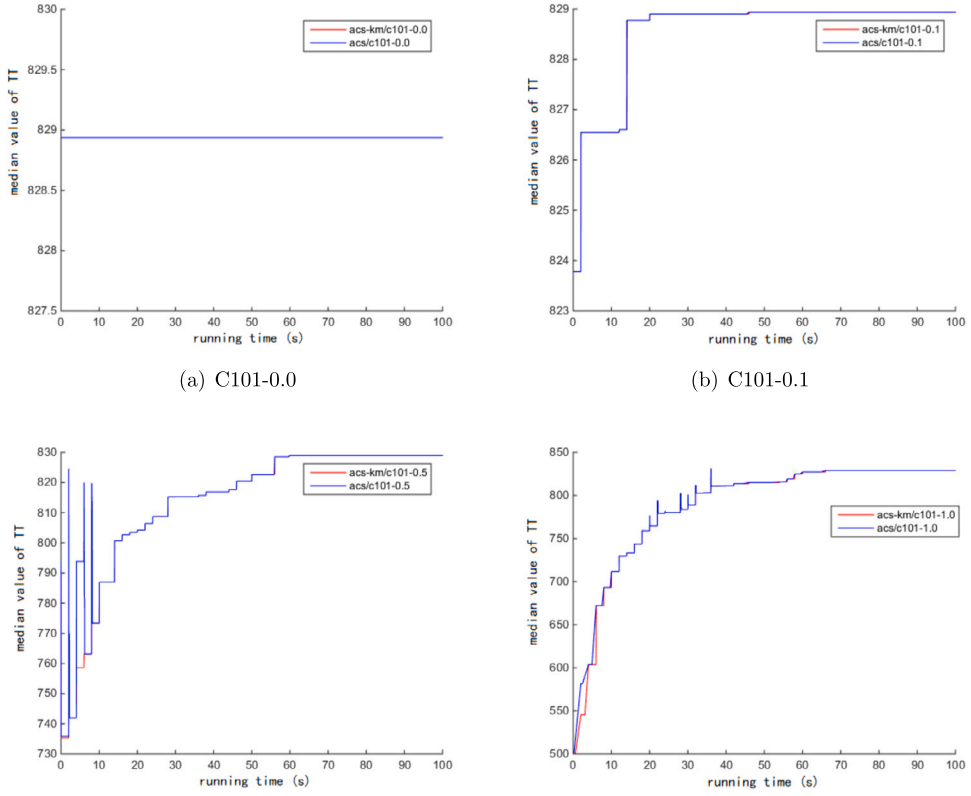
For the instances R104 and RC102 (see Figs. 7 and 8), we can find some interesting phenomena. When the dynamicity level is low,



(a) R104 (b) C101 (c) RC102

(d) R201 (e) C202 (f) RC202

Fig. 5. The average success rate of ACS-KM on six instances.



(a) C101-0.0 (b) C101-0.1

(c) C101-0.5 (d) C101-1.0

Fig. 6. The iteration curves of ACS and ACS-KM on C101.

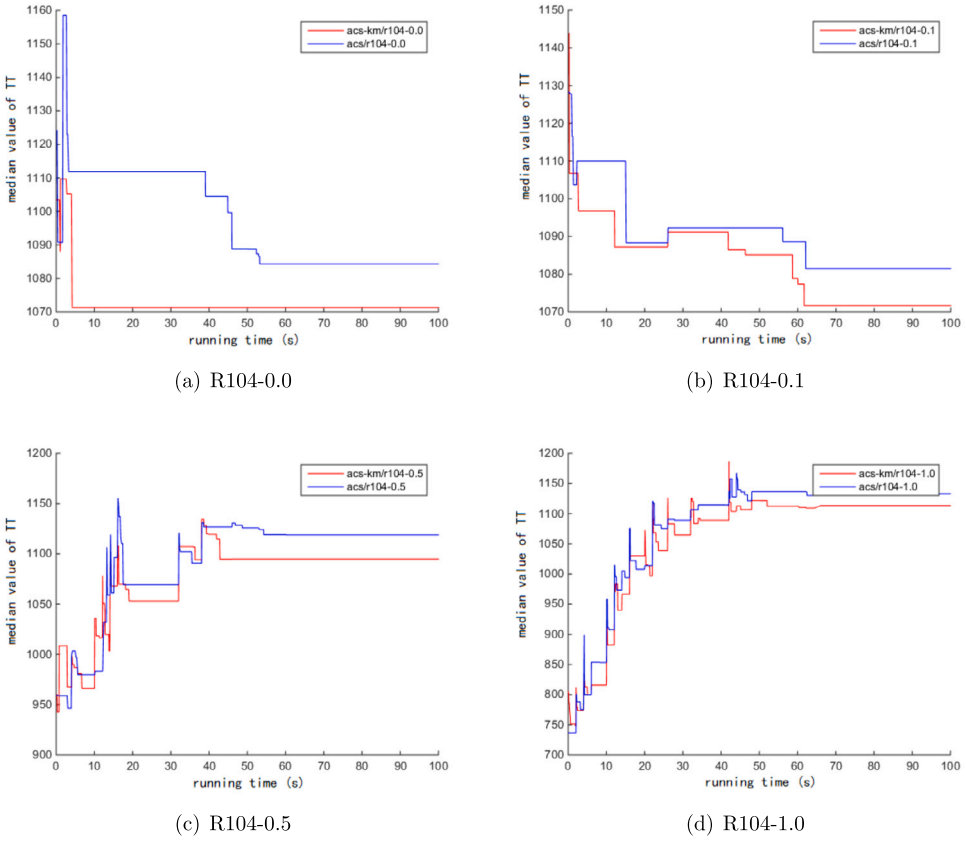


Fig. 7. The iteration curves of ACS and ACS-KM on R104.

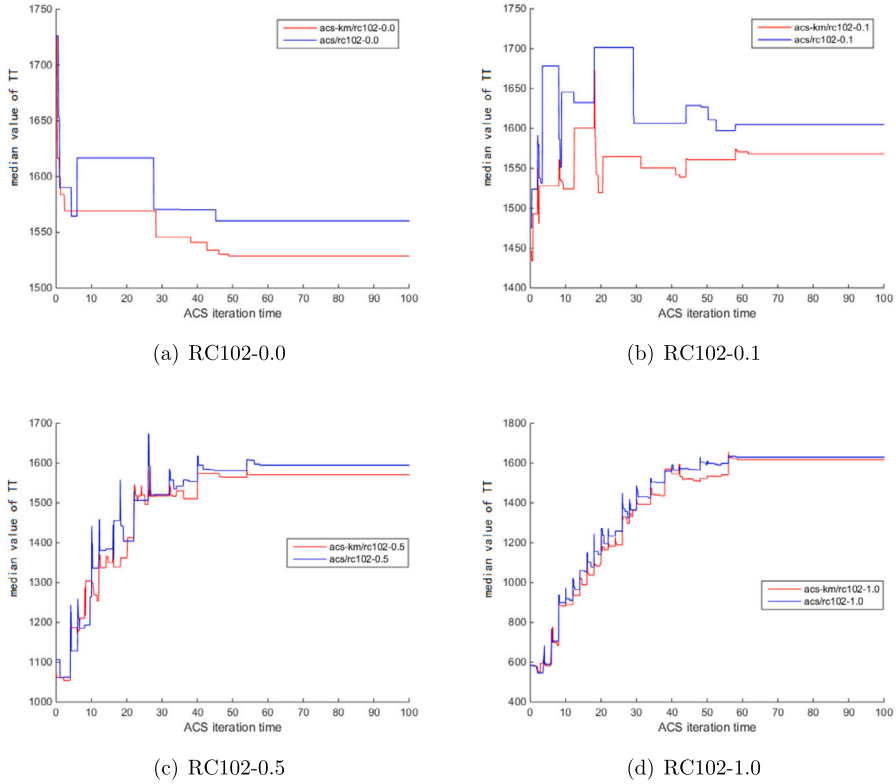


Fig. 8. The iteration curves of ACS and ACS-KM on RC102.

Table 2
Computational results of ACS and ACS-KM on six groups of DVRPTW instances.

Instance	ACS (reported in Necula et al. [30])						ACS-KM					
	Best NV	Best TT	Avg.NV	Avg.TT	Std.NV	Std.TT	Best NV	Best TT	Avg.NV	Avg.TT	Std.NV	Std.TT
R103-0.0	14	1246.5	14.00	1267.5	0.00	10.78	14	**1230.4**	14.00	1247.5	0.00	7.65
R103-0.1	14	1248.8	14.00	1272.8	0.00	14.11	14	**1234.7**	14.00	1253.2	0.00	7.65
R103-0.5	14	**1245.1**	14.00	1301.6	0.00	22.68	13	1251.1	13.97	1280.8	0.18	24.94
R103-1.0	14	1267.6	14.00	1301.5	0.00	18.17	14	**1260.4**	14.00	1284.0	0.00	13.66
R104-0.0	10	1037.7	10.80	1072.2	0.41	16.24	10	**1028.9**	10.63	1057.6	0.49	22.66
R104-0.1	10	**1023.3**	10.53	1059.8	0.51	17.04	10	1028.1	10.67	1052.4	0.48	16.76
R104-0.5	10	1054.1	10.57	1097.7	0.50	25.71	10	**1035.3**	10.37	1075.1	0.49	19.01
R104-1.0	10	1054.4	10.53	1103.9	0.51	24.57	10	**1036.8**	10.50	1095.3	0.51	22.67
Average	12	1147.2	12.30	1184.6	0.24	18.66	11.9	**1138.2**	12.27	1168.2	0.27	16.88
R201-0.0	4	1239.1	4.70	1307.2	0.47	47.92	4	**1224.1**	4.40	1319.6	0.50	52.48
R201-0.1	4	**1218.0**	4.53	1308.7	0.51	56.65	4	1240.7	4.53	1302.0	0.51	50.88
R201-0.5	4	**1323.1**	4.00	1365.7	0.00	25.78	4	1332.5	4.00	1360.5	0.00	17.01
R201-1.0	4	1357.5	4.00	1394.5	0.00	23.73	4	**1342.1**	4.00	1385.2	0.00	24.99
R202-0.0	4	1157.3	4.00	1209.1	0.00	27.18	4	**1141.6**	4.00	1176.7	0.00	14.43
R202-0.1	4	1163.8	4.00	1196.3	0.00	22.33	4	**1139.6**	4.00	1174.2	0.00	19.26
R202-0.5	4	1166.4	4.00	1229.8	0.00	34.82	4	**1157.5**	4.00	1209.5	0.00	20.08
R202-1.0	4	**1195.3**	4.00	1247.4	0.00	33.13	4	1202.2	4.00	1239.1	0.00	21.57
Average	4	1227.6	4.15	1282.3	0.12	33.94	4	**1222.5**	4.12	1270.9	0.13	27.59
C101-0.0	10	**828.9**	10.00	828.9	0.00	0.00	10	**828.9**	10.00	828.9	0.00	0.00
C101-0.1	10	**828.9**	10.00	828.9	0.00	0.00	10	**828.9**	10.00	828.9	0.00	0.00
C101-0.5	10	**828.9**	10.00	828.9	0.00	0.00	10	**828.9**	10.00	828.9	0.00	0.00
C101-1.0	10	**828.9**	10.00	828.9	0.00	0.00	10	**828.9**	10.00	828.9	0.00	0.00
C102-0.0	10	834.6	10.00	864.6	0.00	19.69	10	**828.9**	10.00	841.9	0.00	8.65
C102-0.1	10	**830.8**	10.00	845.6	0.00	12.50	10	**830.8**	10.00	839.8	0.00	9.14
C102-0.5	10	**828.9**	10.00	868.5	0.00	25.15	10	829.9	10.00	871.2	0.00	19.78
C102-1.0	10	**872.0**	10.00	881.3	0.00	5.46	10	**872.0**	10.00	877.2	0.00	3.17
Average	10	835.3	10.00	847.0	0.00	7.85	10	**834.7**	10.00	843.2	0.00	5.09
C201-0.0	3	**591.6**	3.00	591.6	0.00	0.00	3	**591.6**	3.00	591.6	0.00	0.00
C201-0.1	3	**591.6**	3.00	591.6	0.00	0.00	3	**591.6**	3.00	591.6	0.00	0.00
C201-0.5	3	**591.6**	3.00	591.6	0.00	0.00	3	**591.6**	3.00	591.6	0.00	0.00
C201-1.0	3	**591.6**	3.00	591.6	0.00	0.00	3	**591.6**	3.00	591.6	0.00	0.00
C202-0.0	3	**591.6**	3.00	600.7	0.00	15.08	3	**591.6**	3.00	598.6	0.00	14.27
C202-0.1	3	**591.6**	3.00	594.2	0.00	7.97	3	**591.6**	3.00	592.5	0.00	2.52
C202-0.5	3	**591.6**	3.00	606.9	0.00	17.39	3	**591.6**	3.00	620.5	0.00	16.80
C202-1.0	3	**591.6**	3.00	593.2	0.00	6.42	3	**591.6**	3.00	592.5	0.00	1.39
Average	3	**591.6**	3.00	595.1	0.00	5.86	3	**591.6**	3.00	596.3	0.00	4.37
RC101-0.0	15	1649.8	15.10	1669.6	0.31	14.75	15	**1643.2**	15.17	1652.1	0.38	8.55
RC101-0.1	15	1666.8	15.27	1698.4	0.45	19.91	15	**1639.9**	15.33	1671.5	0.55	14.23
RC101-0.5	14	1669.8	15.13	1733.7	0.43	28.04	15	**1645.9**	15.23	1687.4	0.43	24.29
RC101-1.0	15	**1669.9**	15.10	1720.7	0.31	26.01	15	1704.8	15.03	1713.1	0.18	8.16
RC102-0.0	13	1501.7	13.77	1538.9	0.43	21.20	13	**1487.0**	13.60	1525.4	0.50	27.08
RC102-0.1	13	1514.9	13.67	1564.0	0.48	22.93	13	**1501.2**	13.33	1550.1	0.48	28.06
RC102-0.5	13	1540.8	13.93	1580.2	0.37	28.64	13	**1510.7**	13.60	1556.3	0.50	21.84
RC102-1.0	13	1551.4	13.90	1602.6	0.48	32.53	13	**1523.4**	13.73	1573.7	0.45	30.55
Average	13.9	1595.7	14.48	1638.5	0.41	24.25	14	**1582.0**	14.38	1616.2	0.43	20.34
RC202-0.0	4	1245.4	4.00	1301.6	0.00	32.93	4	**1200.1**	4.00	1268.6	0.00	30.64
RC202-0.1	4	1234.6	4.00	1291.9	0.00	29.81	4	**1199.6**	4.00	1260.5	0.00	39.63
RC202-0.5	4	1246.4	4.00	1341.0	0.00	47.48	4	**1227.2**	4.00	1315.0	0.00	41.09
RC202-1.0	3	1277.9	3.97	1365.0	0.18	60.92	4	**1275.9**	4.00	1361.8	0.00	43.58
RC203-0.0	4	1011.6	4.00	1046.4	0.00	24.86	3	**997.2**	3.93	1045.6	0.25	55.65
RC203-0.1	3	1019.4	3.97	1076.3	0.18	41.69	3	**1008.5**	3.80	1089.1	0.41	73.29
RC203-0.5	3	1122.9	3.10	1247.4	0.31	67.22	3	**1118.7**	3.17	1213.3	0.38	67.80
RC203-1.0	3	**1097.2**	3.07	1243.4	0.25	60.90	3	1120.9	3.07	1245.7	0.25	52.47
Average	3.5	1156.9	3.76	1239.1	0.12	45.73	3.5	**1143.5**	3.75	1225.0	0.16	50.52

the best found solutions obtained by both ACS and ACS-KM get improved gradually, especially in the first few iterations. Compared with ACS, ACS-KM has an even larger optimization effect at the beginning. At the end of iterations, ACS-KM can generate significantly better solution than ACS. When the dynamicity level is high, the values of best found solutions exhibit an upward trend along the iterations. This is because there are new customers added in, leading to additional cost. With the increase of iterations, ACS converges gradually and its objective value is close to the objective value of ACS-KM. This may be due to that new customers destroy the optimal structure of the current solution. However, ACS-KM still converges fast whenever the customers dynamically appear, which indicates its suitability to quickly respond to and handle the dynamic characteristics of VRPTW.

6. Conclusions

In this paper, we propose an ACS-based algorithm with bipartite graph matching to solve DVRPTW, in which the objective is to minimize the number of vehicles and the total traveling time. We show that the KM bipartite graph matching algorithm can improve ACS significantly due to its strong convergence.

We conduct experiments on DVRPTW benchmark instances to verify the effectiveness of ACS-KM. The results show that for most cases with

Table 3
Additional computational results of ACS and ACS-KM on Type R1 and R2 instances.

Instance	ACS (our execution)		ACS-KM (10 ants)		ACS-KM (5 ants)	
	Avg. NV	Avg. TT	Avg. NV	Avg. TT (*p*-value)	Avg. NV	Avg. TT (*p*-value)
R103-0.0	14.00	1264.9	14.00	1247.5 (3.36e-08)	14.00	**1244.0** (1.19e-09)
R103-0.1	14.00	1272.5	14.00	1253.2 (1.01e-07)	14.00	**1245.6** (1.65e-11)
R103-0.5	14.00	1301.5	14.00	1280.8 (4.11e-04)	14.00	**1268.0** (1.16e-11)
R103-1.0	14.00	1302.2	14.00	1284.0 (2.85e-05)	14.00	**1279.5** (2.63e-07)
R104-0.0	10.53	1071.7	10.60	1057.6 (2.72e-02)	10.47	**1045.0** (4.85e-05)
R104-0.1	10.47	1061.6	10.47	1052.4 (2.21e-02)	10.57	**1044.2** (7.60e-05)
R104-0.5	10.47	1091.0	10.37	1075.1 (3.25e-03)	10.27	**1066.3** (5.47e-05)
R104-1.0	10.53	1106.0	10.57	1095.3 (7.72e-02)	10.57	**1085.3** (9.61e-04)
R201-0.0	4.70	1297.4	4.63	1319.6 (1.04e-01)	4.70	**1275.4** (9.67e-02)
R201-0.1	4.53	1307.1	4.60	1302.0 (7.15e-01)	4.53	**1291.6** (2.59e-01)
R201-0.5	4.00	1370.3	4.00	1360.5 (4.21e-02)	4.00	**1328.7** (1.05e-11)
R201-1.0	4.00	1387.2	4.00	1385.2 (7.45e-01)	4.00	**1367.4** (2.21e-03)
R202-0.0	4.00	1190.8	4.00	**1176.7** (1.14e-03)	4.00	1182.2 (1.10e-01)
R202-0.1	4.00	1189.1	4.00	**1174.2** (6.97e-03)	4.00	1176.7 (2.77e-02)
R202-0.5	4.00	1233.5	4.00	**1209.5** (3.53e-04)	4.00	1211.4 (6.14e-03)
R202-1.0	4.00	1251.2	4.00	1239.1 (2.71e-02)	4.00	**1230.4** (4.38e-04)
Average	8.20	1231.1	8.20	1219.6	8.19	**1208.9**

Table 4
Comparison results for the static problems.

Instance	ACS	ACS-KM	DCG [10]
R103-0.0	1246.5	**1230.4**	1231.3
R104-0.0	1037.7	**1028.9**	1059.4
R201-0.0	1239.1	1224.1	**1220.8**
R202-0.0	1157.3	1141.6	**1072.8**
C101-0.0	**828.9**	**828.9**	**828.9**
C102-0.0	834.6	**828.9**	829.7
C201-0.0	**591.6**	**591.6**	**591.6**
C202-0.0	**591.6**	**591.6**	**591.6**
RC101-0.0	1649.8	**1643.2**	1720.7
RC102-0.0	1501.7	**1487**	1544.5
RC202-0.0	1245.4	**1200.1**	1295.8
RC203-0.0	1011.6	**997.2**	1099.8
Average	1078.0	**1066.1**	1090.6

different dynamicity levels, ACS-KM can always generate better results than ACS. It achieves the best results on 40 out of 48 instances. This proves that our method would have broad adaptability in dealing with dynamic variants of VRPs.

Our approaches still have some shortcomings to be overcome. Firstly, KM algorithm has strong convergence and may likely trap into local optimum. We plan to include some perturbation mechanisms into ACS-KM to diversify the search. Secondly, KM is still time-consuming. Some fast heuristics can be adopted to accelerate its execution. In future work, we can also try to incorporate machine learning techniques to capture the dynamic features of the environment and derive promising solutions in a real-time manner.

CRediT authorship contribution statement

Yi Teng: Model setup, Methodology, Conceptualization, Writing
Jinbiao Chen: Methodology, Conceptualization, Software
Shiyuan Zhang: Model setup, Methodology, Software
Jiahai Wang: Validation, Writing-Reviewing and Editing
Zizhen Zhang: Conceptualization, Model setup, Validation, Writing-Reviewing and Editing

Declaration of competing interest

There is no competing interest.

Acknowledgements

The work described in this article was supported by National Natural Science Foundation of China (No. 62172452), Research platforms and research projects of ordinary universities in Guangdong Province (No. 2021KQNCX061). Department of Education of Guangdong Province 2022 Higher Education Special Project (2022GXJK287). Natural Science Fund of Guangdong Province (No. 2019A1515011169, 2021A1515011301).

References

[1] AbdAllah AMF, Essam DL, Sarker RA. On solving periodic re-optimization dynamic vehicle routing problems. Appl Soft Comput 2017;55:1–12.
[2] Bianchessi N, Irnich S. Branch-and-cut for the split delivery vehicle routing problem with time windows. Transp Sci 2019;53(2):442–62.
[3] Boussaïd I, Chatterjee A, Siarry P, Ahmed-Nacer M. Hybridizing biogeography-based optimization with differential evolution for optimal power allocation in wireless sensor networks. IEEE Trans Veh Technol 2011;60(5):2347–53.
[4] Braekers K, Ramaekers K, Van Nieuwenhuyse I. The vehicle routing problem: state of the art classification and review. Comput Ind Eng 2016;99:300–13.
[5] Brandão J. Iterated local search algorithm with ejection chains for the open vehicle routing problem with time windows. Comput Ind Eng 2018;120:146–59.
[6] Bräysy O, Gendreau M. Vehicle routing problem with time windows, part I: route construction and local search algorithms. Transp Sci 2005;39(1):104–18.
[7] Bräysy O, Gendreau M. Vehicle routing problem with time windows, part II: metaheuristics. Transp Sci 2005;39(1):119–39.
[8] Cai X, Jiang L, Guo S, Huang H, Du H. TLHSA and SACA: two heuristic algorithms for two variant VRP models. J Comb Optim 2021:1–27.
[9] Chen H, Cai X, Gu F, Xu S. An improved ACO algorithm to vehicle routing problem with time windows and uncertainty. In: 2021 17th international conference on computational intelligence and security (CIS). IEEE; 2021. p. 50–4.
[10] Chen Z-L, Xu H. Dynamic column generation for dynamic vehicle routing with time windows. Transp Sci 2006;40(1):74–88.
[11] Clarke G, Wright JW. Scheduling of vehicles from a central depot to a number of delivery points. Oper Res 1964;12(4):568–81.
[12] Dang Y, Allen TT, Singh M. A heterogeneous vehicle routing problem with common carriers and time regulations: mathematical formulation and a two-color ant colony search. Comput Ind Eng 2022;168:108036.
[13] Dantzig GB, Ramser JH. The truck dispatching problem. Manag Sci 1959;6(1):80–91.
[14] Dellaert N, Dashty Saridarq F, Van Woensel T, Crainic TG. Branch-and-price–based algorithms for the two-echelon vehicle routing problem with time windows. Transp Sci 2019;53(2):463–79.
[15] Desrochers M, Desrosiers J, Solomon M. A new optimization algorithm for the vehicle routing problem with time windows. Oper Res 1992;40(2):342–54.
[16] Dorigo M, Gambardella LM. Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans Evol Comput 1997;1(1):53–66.
[17] Faiz TI, Vogiatzis C, Noor-E-Alam M. A column generation algorithm for vehicle scheduling and routing problems. Comput Ind Eng 2019;130:222–36.
[18] Gambardella LM, Taillard É, Agazzi G. MACS-VRPTW: a multiple colony system for vehicle routing problems with time windows. In: New ideas in optimization. Citeseer; 1999.

[19] Gao K, Cao Z, Zhang L, Chen Z, Han Y, Pan Q. A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems. IEEE/CAA J Autom Sin 2019;6(4):904–16.
[20] Gupta A, Srivastava S. Comparative analysis of ant colony and particle swarm optimization algorithms for distance optimization. Proc Comput Sci 2020;173:245–53. https://doi.org/10.1016/j.procs.2020.06.029. https://www.sciencedirect.com/science/article/pii/S1877050920315337. International conference on smart sustainable intelligent computing and applications under ICITETM2020.
[21] He M, Wei Z, Wu X, Peng Y. An adaptive variable neighborhood search ant colony algorithm for vehicle routing problem with soft time windows. IEEE Access 2021;9:21258–66.
[22] Huamán A, Huancahuari M, Wong L. Multiphase model based on k-means and ant colony optimization to solve the capacitated vehicle routing problem with time windows. In: Annual international conference on information management and big data. Springer; 2022. p. 141–57.
[23] Kilby P, Prosser P, Shaw P. Dynamic VRPs: a study of scenarios. Technical report, University of Strathclyde; 1998.
[24] Kritikos MN, Ioannou G. The balanced cargo vehicle routing problem with time windows. Int J Prod Econ 2010;123(1):42–51.
[25] Lesch V, König M, Kounev S, Stein A, Krupitzer C. Tackling the rich vehicle routing problem with nature-inspired algorithms. Appl Intell 2022:1–25.
[26] Luo H, Dridi M, Grunder O. An ACO-based heuristic approach for a route and speed optimization problem in home health care with synchronized visits and carbon emissions. Soft Comput 2021;25(23):14673–96.
[27] Molina JC, Salmeron JL, Eguia I. An ACS-based memetic algorithm for the heterogeneous vehicle routing problem with time windows. Expert Syst Appl 2020;157:113379.
[28] Montemanni R, Gambardella LM, Rizzoli AE, Donati AV. Ant colony system for a dynamic vehicle routing problem. J Comb Optim 2005;10(4):327–43.
[29] Munkres J. Algorithms for the assignment and transportation problems. J Soc Ind Appl Math 1957;5(1):32–8.
[30] Necula R, Breaban M, Raschip M. Tackling dynamic vehicle routing problem with time windows by means of ant colony system. In: 2017 IEEE congress on evolutionary computation (CEC). IEEE; 2017. p. 2480–7.
[31] Niccolai A, Grimaccia F, Mussetta M, Zich R. Optimal task allocation in wireless sensor networks by means of social network optimization. Mathematics 2019;7(4):315.
[32] Pan B, Zhang Z, Lim A. A hybrid algorithm for time-dependent vehicle routing problem with time windows. Comput Oper Res 2021;128:105193.
[33] Pan B, Zhang Z, Lim A. Multi-trip time-dependent vehicle routing problem with time windows. Eur J Oper Res 2021;291(1):218–31.
[34] Pérez-Rodríguez R, Hernández-Aguirre A. A hybrid estimation of distribution algorithm for the vehicle routing problem with time windows. Comput Ind Eng 2019;130:75–96.
[35] Pillac V, Gendreau M, Guéret C, Medaglia AL. A review of dynamic vehicle routing problems. Eur J Oper Res 2013;225(1):1–11.
[36] Psaraftis HN. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. Transp Sci 1980;14(2):130–54.
[37] Ratanavilisagul C. A novel modified ant colony optimization algorithm by resetting and updating pheromone for vehicle routing problem with time windows. In: 2021 7th international conference on computer and communications (ICCC). IEEE; 2021. p. 1194–8.
[38] Savelsbergh MW. The vehicle routing problem with time windows: minimizing route duration. ORSA J Comput 1992;4(2):146–54.
[39] Schneider M, Schwahn F, Vigo D. Designing granular solution methods for routing problems with time windows. Eur J Oper Res 2017;263(2):493–509.
[40] da Silva Junior OS, Leal JE, Reimann M. A multiple ant colony system with random variable neighborhood descent for the dynamic vehicle routing problem with time windows. Soft Comput 2021;25(4):2935–48.
[41] Soeffker N, Ulmer MW, Mattfeld DC. Stochastic dynamic vehicle routing in the light of prescriptive analytics: a review. Eur J Oper Res 2022;298(3):801–20.
[42] Solomon MM. Algorithms for the vehicle routing and scheduling problems with time window constraints. Oper Res 1987;35(2):254–65.
[43] Stützle T, Hoos HH. MAX–MIN ant system. Future Gener Comput Syst 2000;16(8):889–914.
[44] Suppan M, Hanne T, Dornberger R. Ant colony optimization to solve the rescue problem as a vehicle routing problem with hard time windows. In: Proceedings of international joint conference on advances in computational intelligence. Springer; 2022. p. 53–65.
[45] Ticha HB, Absi N, Feillet D, Quilliot A. Multigraph modeling and adaptive large neighborhood search for the vehicle routing problem with time windows. Comput Oper Res 2019;104:113–26.
[46] Veen Bv, Emmerich M, Yang Z, Bäck T, Kok J. Ant colony algorithms for the dynamic vehicle routing problem with time windows. In: International work-conference on the interplay between natural and artificial computation. Springer; 2013. p. 1–10.
[47] Vidal T, Laporte G, Matl P. A concise guide to existing and emerging vehicle routing problem variants. Eur J Oper Res 2020;286(2):401–16.
[48] Wang F, Liao F, Li Y, Yan X, Chen X. An ensemble learning based multi-objective evolutionary algorithm for the dynamic vehicle routing problem with time windows. Comput Ind Eng 2021;154:107131.
[49] Xu H, Pu P, Duan F. Dynamic vehicle routing problems with enhanced ant colony optimization. Discrete Dyn Nat Soc 2018;2018.
[50] Yang Z, Emmerich M, Bäck T. Ant based solver for dynamic vehicle routing problem with time windows and multiple priorities. In: 2015 IEEE congress on evolutionary computation (CEC). IEEE; 2015. p. 2813–9.
[51] Yao Y, Zhu X, Dong H, Wu S, Wu H, Tong LC, et al. ADMM-based problem decomposition scheme for vehicle routing problem with time windows. Transp Res, Part B, Methodol 2019;129:156–74.
[52] Yassen ET, Ayob M, Nazri MZA, Sabar NR. An adaptive hybrid algorithm for vehicle routing problems with time windows. Comput Ind Eng 2017;113:382–91.