Omega 54 (2015) 60–71

Contents lists available at ScienceDirect

Omega

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

A memetic algorithm for the patient transportation problem[☆]

Zhenzhen Zhang[a], Mengyang Liu[a,*], Andrew Lim[b,a,1]

[a] Department of Management Sciences, City University of Hong Kong, Tat Chee Ave, Kowloon Tong, Hong Kong
[b] School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore 637371, Singapore

ARTICLE INFO

Article history:
Received 12 August 2014
Accepted 21 January 2015
Available online 31 January 2015

Keywords:
Vehicle scheduling
Health service
Memetic algorithm
Dial-a-ride problem
Multi-trip

ABSTRACT

This paper addresses a real-life public patient transportation problem derived from the Hong Kong Hospital Authority (HKHA), which provides ambulance transportation services for disabled and elderly patients from one location to another. We model the problem as a multi-trip dial-a-ride problem (MTDARP), which requires designing several routes for each ambulance. A route is a sequence of locations, starting and terminating at the depot (hospital), according to which the ambulance picks up clients at the origins and delivers them to the destinations. A route is feasible only if it satisfies a series of side constraints, such as the pair and precedence constraints, capacity limit, ride time, route duration limit and time windows. Owing to the route duration limit, in particular, every ambulance is scheduled to operate several routes during the working period. To prevent the spread of disease, the interior of the ambulances needs to be disinfected at the depot between two consecutive trips. The primary aim of the problem investigated herein is to service more requests with the given resources, and to minimize the total travel cost for the same number of requests. In this paper, we provide a mathematical formulation for the problem and develop a memetic algorithm with a customized recombination operator. Moreover, the segment-based evaluation method is adapted to examine the moves quickly. The performance of the proposed algorithm is assessed using the real-world data from 2009 and compared with results obtained by solving the mathematical model. In addition, the proposed algorithm is adapted to solve the classic DARP instances, and found to perform well on medium-scale instances.

1. Introduction

In this paper, we investigate the public patient transportation service provided by the Hong Kong Hospital Authority (HKHA). This service transfers patients from one location to another to receive medical services, and requires that certain service requirements are fulfilled. As the services are all non-emergency in nature, they are reserved in advance.

The classic dial-a-ride problem (DARP) [13,12] is a special case of our problem. In DARP, every vehicle follows a route to provide transportation services for passengers. Along a given route, each vehicle picks up or drops off customers at required locations in pre-reserved time windows. In addition, the service level for each passenger, such as ride time and route duration constraints, must be taken into account.

There are three main differences between our problem and the classic DARP. First, each ambulance completes several short routes instead of just one within the working period. A restrictive duration limit ensures that the ambulances are disinfected regularly to prevent the spread of disease. Second, the lunch breaks of drivers and assistants need to be considered in the schedule. Thus, the scheduling of multiple trips and lunch breaks makes the problem complex to solve. Finally, owing to the limited number of vehicles, some clients may not be served. Here, our objective is to design a transportation plan to service as many patients as possible, and then to minimize the total travel cost for serving the same number of patients. The problem is modeled as a multi-trip dial-a-ride problem (MTDARP).

To solve the MTDARP, we have developed a memetic algorithm (MA) with the state-of-the-art framework proposed by Vidal et al. [55]. First, the initial population is filled by a constructive algorithm based on the *reg-k* regret insertion strategy. We apply a customized crossover operator to generate a new chromosome. The offspring chromosome is improved by a local search with different move types, and then placed into the population. When the size of the population exceeds a given threshold, an advanced diversity management scheme is invoked to perform survivor selection. To accelerate the process, the segment-based evaluation method is adapted to examine the temporal solutions, and a mechanism is proposed to avoid the duplicated checks of moves already examined. We evaluate the performance of the proposed

[☆]This manuscript was processed by Associate Editor Sterna.
[*] Corresponding author. Tel.: +852 64067667; fax: +852 34420188.
E-mail addresses: zhenzhenzhang222@gmail.com (Z. Zhang), liu.mengyang.hust@gmail.com (M. Liu).
[1] Currently on no pay leave from City University of Hong Kong.

http://dx.doi.org/10.1016/j.omega.2015.01.011
0305-0483/© 2015 Elsevier Ltd. All rights reserved.

algorithm based on real-world data collected by the HKHA for 2009 and compare it with results obtained by solving the mathematical model. In addition, the algorithm is adapted to solve the classic DARP instances and compared with recent methods.

This problem is particularly important and interesting. Theoretically, the classic DARP is an NP-hard problem, and thus the MTDARP is also NP-hard but of even greater complexity. In practice, it is becoming more difficult for decision makers to design schedules manually because of the increasing number of requests and non-emergency ambulances. We believe that our proposed algorithm can provide a superior, good-quality solution. Thus, our problem offers good alignment between research and real-life applications.

The remainder of the paper is organized as follows. In Section 2, we describe the relevant literature, including studies on DARP, multi-trip vehicle routing problems (MTVRP), and a state-of-the-art genetic algorithm (GA). We then provide a formal description and the mathematical formulation of the MTDARP in Section 3. In Section 4, we introduce the proposed algorithm in detail. In Section 5, we report a series of experiments to evaluate the performance of the proposed algorithm based on the real-world data and DARP instances. Finally, Section 6 concludes the paper.

2. Related work

To the best of our knowledge, the MTDARP has not previously been studied. However, the problem is related to several variants of the vehicle routing problem, including DARP, the MTVRP and its extension with time windows (MTVRPTW). Pickup and delivery (PD) as well as time windows (TW) are common characteristics in real applications [23,5,29]. Recently, the healthcare problems attract increasing interest from the researchers [36,27,26,51].

DARP can be seen as a variant of the pickup and delivery problem with time windows (PDPTW) which usually focuses on goods transportation. For more information on the widely used benchmark, the state-of-the-art meta-heuristic and exact algorithms of PDPTW, readers are referred to Li and Lim [25], Ropke and Pisinger [47], Baldacci et al. [4]. DARP addresses passenger transportation, and thus service quality must be taken into account. DARP usually arises in contexts in which passengers cannot easily use the regular public transportation system, e.g., rural areas [54,15], airports [43], and the health-care context [44,31,19,7]. Most of the DARP literature is derived from real-world applications, and accordingly the problem definitions usually differ slightly. The proposed algorithms for these problems include exact algorithms, constructive heuristics, and meta-heuristics. The exact algorithm can solve only small- or medium-scale instances to optimality, such as branch-and-cut [12] and branch-and-price-and-cut [17], whereas the aim of constructive methods is to generate a feasible solution quickly. Diana and Dessouky [15] proposed a regret insertion method in which the spatial and temporal aspects are considered simultaneously in selecting the seed customer for each route. Wolfler Calvo and Colorni [58] first solved the assignment problem on an auxiliary graph to create clusters of customers, and then designed the routes using those clusters. Cordeau and Laporte [13] developed a tabu search (TS) in which the infeasible solutions are allowed to enlarge the search space and an eight-step evaluation scheme is used to solve the maximum ride time constraint optimally. This eight-step scheme was often employed in subsequent studies. Recently, Kirchler and Wolfler Calvo [24] proposed a granular tabu search in which the assignment problem is solved to define the closeness of two customers. The variable neighborhood search (VNS) was addressed by Parragh et al. [38], who designed several novel neighborhood structures. Moreover, Jain and Hentenryck [21] developed the large neighborhood search (LNS), in which a tree search is used to complete the partial solutions. Guerriero et al. [18] presented the greedy randomized adaptive search programming (GRASP) to solve DARP with heterogeneous vehicles. A GA was also attempted by Jorgensen et al. [22] and Rekiek et al. [44]. In their studies, the chromosomes only record information on the clusters of customers, and the corresponding routes are constructed by an insertion method. Furthermore, Braekers et al. [8] proposed an exact branch-and-cut algorithm and a deterministic annealing (DA) meta-heuristic to extensively solve the homogeneous and the heterogeneous single depot DARP instances and the multi-depot heterogeneous DARP problem. Several authors have recently tried to design hybrid methods to exploit the advantages of both exact algorithms and meta-heuristics. For example, Parragh and Schmid [39] developed a hybrid column generation and large neighbourhood search method in which a new solution is obtained via column generation and improved by LNS. For a more detailed review of DARP, readers are referred to the survey of Cordeau and Laporte [14].

The multi-trip concept is actually very common in practice. A multi-trip arrangement usually results from the limits on route duration. The MTVRP was first proposed by Fleischmann [16], who proposed a route-first pack-second strategy. The author first constructed the routes via saving-based heuristics, and then assigned them to working shifts through a bin-packing algorithm. A number of other interesting algorithms were later developed, such as a constructive method [40], tabu search [9,10,1], adaptive memory programming [53,35], a genetic algorithm [48], and a memetic algorithm [11]. Furthermore, a detailed review has been carried out by Sen and Bülbül [50]. To date, the only exact algorithm is proposed by Mingozzi et al. [32].

A recently proposed extension of the MTVRP is the imposition of the MTVRPTW. The scheduling of routes is more difficult in the MTVRPTW. Thus, Azi et al. [2] developed a two-phase exact algorithm for the MTVRPTW with a single vehicle in which all non-dominated feasible routes are enumerated in the first phase, and some routes are then selected and scheduled to form a solution. A number of authors recently extended the framework to the problem with multiple vehicles [3,30,20]. For meta-heuristics, however, only Battarra et al. [6] have presented an adaptive guidance approach for solving the problem with the objective of minimizing the number of required vehicles.

In the proposed MTDARP model, assistant requirements and lunch breaks are taken into consideration. These constraints have been examined in ambulance routing problems [36,26]. For example, Parragh et al. [37] proposed a heterogeneous DARP that considers the requirements of assistants (referred to as attendants in their paper) and lunch break constraints. They implemented a branch-and-price algorithm to solve the problem. However, in their problem, a lunch break can be held in any vertex, whereas in the proposed model, staff have to return to the depot to have lunch. A similar lunch break constraint was investigated by Sze et al. [52]. In their work, a two-stage scheduling heuristic is implemented to solve the problem arising in in-flight food loading operations. Lim et al. [26] considered a problem where the time windows of lunch breaks for assistants are heterogeneous.

Memetic algorithms (MAs), which are also called hybrid genetic algorithms (hybrid GAs), are optimization techniques based on the synergistic combination of ideas taken from different algorithmic solvers, such as population-based searches (as in evolutionary techniques) and local searches (as in gradient-ascent techniques) [33]. These algorithms have been effectively applied to solve the vehicle routing problem (VRP) and its variants. For more detailed information, we refer readers to the recent survey by Potvin [41]. Less work has been reported on the PDPTW and its variants. Nagata and Kobayashi [34] stated that the main reason for this gap is the difficulty of designing an effective crossover operator for the problem with time windows, because of the fairly constrained search space. Recently, Vidal et al. [55] proposed a framework with advanced diversity management for the genetic algorithm, and demonstrated its effectiveness on a series of VRP variants [56,57].

In addition, Cattaruzza et al. [11] employed this advanced GA framework to solve the MTVRP and obtained good results. Motivated by these successes, we address a memetic algorithm with the advanced framework and a customized recombination operator to generate high-quality solutions for our problem.

3. Problem description and modelling

In this section, we provide a formal description and a mathematical model for the MTDARP.

3.1. Problem statement and notation

Let $H = \{1, \dots, n\}$ denote a set of requests to be serviced. Let $P = \{1, \dots, n\}$ represent the set of pickup nodes, and $D = \{n+1, \dots, 2n\}$ be the set of delivery nodes. Each request $h \in H$ consists of two nodes: the pickup node $h \in P$ and corresponding delivery node $n+h \in D$.

The MTDARP is defined on a complete undirected graph $G = (V, A)$ with a set of nodes $V = \{0\} \cup N = \{0, 1, \dots, 2n\}$ and a set of arcs $A = \{(i,j) | i, j \in V, i \neq j\}$. The depot is defined as node 0, and the set of locations as $N = P \cup D$. Each node $i \in N$ is associated with a seat demand q_i, a non-negative service time s_i, and a time window $[e_i, l_i]$. For each request h, the seat demand of pickup node, q_h, must be positive, and that of the corresponding delivery node is then defined by $q_{n+h} = -q_h$. Each request $h \in H$ has a maximum ride time f_h and assistant demand o_h. The nodes belonging to request h, i.e., nodes $h \in P$ and $n+h \in D$, share the same values of f_h and o_h as request h. Each arc $(i,j) \in A$ has a travel time t_{ij} satisfying the triangle inequality. In this study, the maximum ride time of each request h is defined as $f_h = 3t_{h,n+h}$.

Let K denote the set of available vehicles with the same capacity Q. Every vehicle is assigned a lunch break occurring during time window $[EL, LL]$ and with length T_{lunch}, allowing staff to have lunch in the hospital restaurant. The vehicles are available only during the period $[EV, LV]$. To keep drivers and staff in a suitable condition to provide a satisfactory service and avoid the spread of disease, the duration of the route, that is, the time between leaving the depot and returning to it, should not exceed the given maximum travel time T_{max}, and the vehicles must be disinfected immediately, which can be considered as a set-up time δ for the next trip. In this paper, T_{max} and δ are set as 1.5 h and 0.5 h, respectively.

Because of the maximum duration limit, a vehicle operates several routes to service various requests within a planning horizon. A route is a visit sequence of picking up and delivering clients, and starts and finishes at the depot. The route is feasible if it satisfies the following constraints: (i) a pickup node and its corresponding delivery node must be visited within the same route, and the pickup node must be visited prior to the delivery node; (ii) the total number of clients at each node in the route must not exceed capacity; (iii) every node must be visited within its time window, and thus if a vehicle arrives early, it must wait for the start of service; (iv) the ride time of each passenger, that is, the total time spent in the vehicle, cannot exceed his maximum ride time; (v) the duration of each route is strictly restricted by limit T_{max}, and the vehicle interior must be disinfected immediately after returning to the depot; and (vi) the schedule should take lunch breaks into account, and there should be no time overlap between the routes of the same vehicle. A feasible solution consists of a set of feasible routes satisfying all constraints.

The objective of the MTDARP is hierarchical. First, we need to generate a set of routes for the given vehicles to ensure that the number of serviced requests is maximized. Then, for the same number of requests, the total travel cost of the vehicles should be minimized. A simple example with 13 requests and 2 vehicles is provided in Fig. 1, where the beginning time of service and the vehicle load after node servicing are indicated in brackets. Every vehicle has a capacity of 9 and operates two routes. Request 6 is rejected owing to the time conflict. Note that the vehicles leave the depot with an adequate number of assistants on board.

3.2. Mathematical formulation

This section presents a mathematical model for the problem, adapted from the model of the pickup and delivery problem (PDP) proposed by Lu and Dessouky [28]. Within a single workday, each vehicle operates several routes in accordance with the maximum route duration limit. To represent those routes, we duplicate the depot node and construct depot set $V_d = \{2n+1, \dots, 2n+|R|, 2n+|R|+1\}$

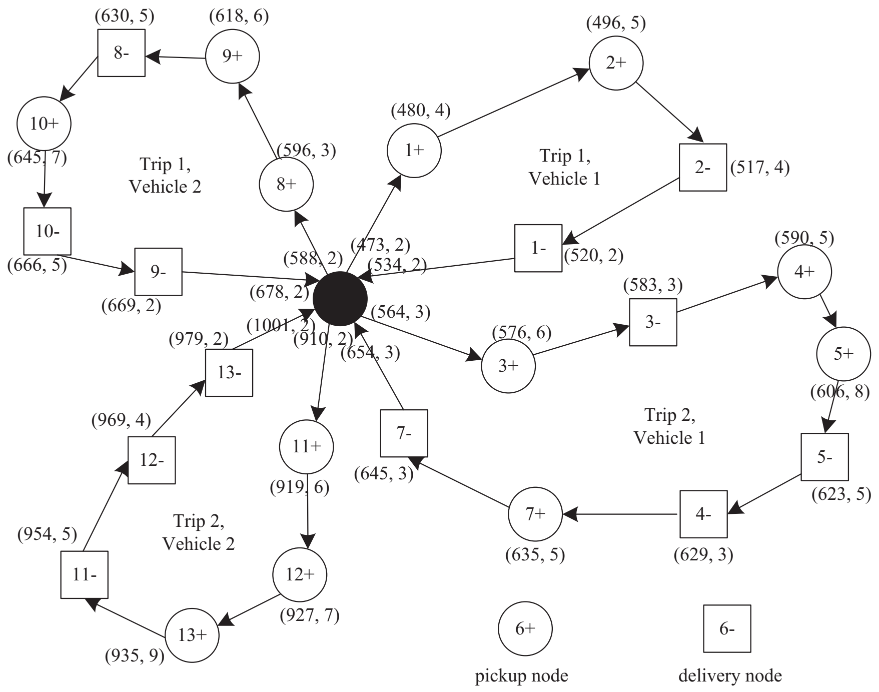

Fig. 1. A simple example for the MTDARP.

with the duplicated depot nodes. $R=\{1,\ldots|R|\}$ contains the index of routes a vehicle may take within a single workday. In the experiment, R is large enough to contain all of the routes operated by a vehicle within one workday. The nodes in V_d serve as the start and end depots for each route of any vehicle. For example, any vehicle's route r begins at depot $2n+r$ and ends at depot $2n+r+1$. Note that node $2n+1$ serves only as the starting node for route 1, whereas node $2n+|R|+1$ serves only as the ending node for route $|R|$.

Then, the mathematical model of the MTDARP is built on the new Graph $G'=\{V',A'\}$, where $V'=P\cup D\cup V_d$. Arc set $A'=\{(i,j)|i,j\in P\cup D, i\neq j\}\cup\{(2n+r,i)|i\in P, r\in R\}\cup\{(i,2n+r)|i\in D, 2\le r\le|R|+1\}\cup\{(2n+r,2n+r+1)|r\in R\}$. The variables used in the mathematical model are defined as follows:

- $x_{i,j,k}$: 1 if arc $(i,j)\in A'$ is used by vehicle k; and 0 otherwise.
- $b_{i,j}$: 1 if $i\in N$ is served before $j\in N$ and i,j are served by the same vehicle or the index of the route serving $i\in N$ comes before that of j; and 0 otherwise. For example, if node i is served by route r_1 of vehicle k_1 and node j by route r_2 of vehicle k_2 and $r_1<r_2$, then $b_{i,j}=1$.
- a_i: the time to begin service at node $i\in N$.
- $y_{k,r}$: 1 if vehicle k is assigned a lunch break after finishing route r.
- d_i: the vehicle load after visiting node $i\in N$.
- m_i: the number of assistants in the vehicle when serving node i.
- $u_{k,r}$: starting time of trip (k,r) $(k\in K, r\in R)$.
- $v_{k,r}$: completion time of trip (k,r) $(k\in K, r\in R)$.

The variables $b_{i,j}$ help to reduce an extra index in the arc flow variables to indicate the trips of each vehicle. Using these variables, and a quite large number M and the notation given in the previous subsection, this problem can be formulated as follows:

$$\min \sum_{k\in K}\sum_{(i,j)\in A'} t_{i,j}x_{i,j,k} - M\sum_{i\in P,j\in V',k\in K} x_{i,j,k} \tag{1}$$

$$\text{s.t.} \sum_{\{j|(i,j)\in A'\}} x_{i,j,k} = \sum_{\{j|(j,i)\in A'\}} x_{j,i,k}, \quad \forall i\in V'\setminus\{2n+1, 2n+|R|+1\},\ k\in K \tag{2}$$

$$\sum_{\{j|(j,i)\in A'\}} x_{j,i,k}=1, \quad \forall i\in V_d\setminus\{2n+1\},\ k\in K \tag{3}$$

$$\sum_{\{j|(2n+1,j)\in A'\}} x_{2n+1,j,k}=1, \quad \forall k\in K \tag{4}$$

$$\sum_{k\in K}\sum_{\{j|(i,j)\in A'\}} x_{i,j,k}\le 1, \quad \forall i\in P\cup D \tag{5}$$

$$\sum_{\{j|(i,j)\in A'\}} x_{i,j,k} = \sum_{\{j|(n+i,j)\in A'\}} x_{n+i,j,k}, \quad \forall i\in P,\ k\in K \tag{6}$$

$$b_{i,j}\ge\sum_{k\in K}x_{i,j,k}, \quad \forall (i,j)\in A'\setminus V_d\times V_d \tag{7}$$

$$b_{i,j}+b_{j,i}\le 1, \quad \forall (i,j)\in A' \text{ or } (j,i)\in A' \tag{8}$$

$$b_{h,i}\ge b_{h,j}+\sum_{k\in K}x_{i,j,k}+\sum_{k\in K}x_{j,i,k}-1, \quad \forall i,j,h\in N,\ h\neq i,\ h\neq j \tag{9}$$

$$b_{h,i}\ge b_{h,j}+\sum_{k\in K}x_{i,j,k}-1, \quad \forall i\in V_d, j,h\in N,\ h\neq j \tag{10}$$

$$b_{h,j}\ge b_{h,i}+\sum_{k\in K}x_{i,j,k}-1, \quad \forall i\in V_d, j,h\in N,\ h\neq j \tag{11}$$

$$b_{h,j}\ge b_{h,i}+\sum_{k\in K}x_{i,j,k}-1, \quad \forall j\in V_d, i,h\in N,\ h\neq i \tag{12}$$

$$b_{i,2n+r+1}\ge b_{i,2n+r}, \quad \forall i\in N,\ r\in R \tag{13}$$

$$b_{i,j}=b_{n+i,j}, \quad \forall i\in P,\ j\in V_d \tag{14}$$

$$a_j\ge(a_i+s_i+t_{i,j})-M_{i,j}\left(1-\sum_{k\in K}x_{i,j,k}\right), \quad \forall i,\ j\in N \tag{15}$$

$$a_i\ge(u_{k,r}+t_{2n+r,i})-M_{i,k,r}(1-x_{2n+r,i,k}), \quad \forall i\in P,\ k\in K,\ r\in R \tag{16}$$

$$v_{k,r}\ge(a_i+s_i+t_{i,2n+r+1})-M_{i,k,r}(1-x_{i,2n+r+1,k}), \quad \forall i\in D,\ k\in K,\ r\in R \tag{17}$$

$$e_i\le a_i\le l_i, \quad \forall i\in N \tag{18}$$

$$t_{i,n+i}\le a_{n+i}-(a_i+s_i)\le f_i, \quad \forall i\in P \tag{19}$$

$$v_{k,r}-u_{k,r}\le T_{max}, \quad \forall k\in K,\ r\in R \tag{20}$$

$$d_j\ge(d_i+q_j)-M_{i,j}\left(1-\sum_{k\in K}x_{i,j,k}\right), \quad \forall i,j\in N \text{ and } (i,j)\in A' \tag{21}$$

$$m_i+d_i\le Q, \quad \forall i\in P \tag{22}$$

$$m_i\le m_j+o_{max}\left(1-\sum_{k\in K}x_{i,j,k}-\sum_{k\in K}x_{j,i,k}\right), \quad \forall i,j\in N \tag{23}$$

$$u_{k,r}\le v_{k,r}, \quad \forall k\in K,\ r\in R \tag{24}$$

$$u_{k,r+1}\ge v_{k,r}+\delta(1-x_{2n+r,2n+r+1,k}), \quad \forall k\in K\ r=1,\ldots,|R|-1 \tag{25}$$

$$u_{k,r+1}\ge v_{k,r}+T_{lunch}*y_{k,r}, \quad \forall k\in K\ \ r=1,\ldots,|R|-1 \tag{26}$$

$$u_{k,r+1}\ge(EL+T_{lunch})*y_{k,r}, \quad \forall k\in K,\ r=1,\ldots,|R|-1 \tag{27}$$

$$v_{k,r}\le LL+M_{k,r}(1-y_{k,r}), \quad \forall k\in K,\ r=1,\ldots,|R|-1 \tag{28}$$

$$u_{k,1}\ge EL+T_{lunch}-M_k\sum_{r\in R}y_{k,r}, \quad \forall k\in K \tag{29}$$

$$x_{i,j,k}\in\{0,1\}, \quad \forall (i,j)\in A',\ k\in K \tag{30}$$

$$b_{i,j}\in\{0,1\}, \quad \forall i,j\in V \tag{31}$$

$$o_i\le m_i\le Q-q_i, \quad \forall i\in P \tag{32}$$

$$d_i\ge\max\{q_i,0\}, \quad \forall i\in N \tag{33}$$

$$LV\ge u_{k,r}, v_{k,r}\ge EV \quad \forall k\in K,\ r\in R \tag{34}$$

The objective function minimizes the weighted sum of the number of unserved requests and the total travel cost. Here, $\sum_{i\in P,j\in V',\ k\in K}x_{i,j,k}$ is the total number of serviced customers. This form of the objective function is in line with our primary goal of servicing as many requests as possible and then minimizing the travel cost for the same number of serviced requests.

Constraint (2) is the network constraint. Constraints (3) and (4) ensure that each depot node is used only once by each vehicle. Constraint (5) guarantees that each customer is served at most once. Constraint (6) is the pair constraint, which forces the pickup and delivery nodes of each request to be visited by the same vehicle.

Constraints (7) and (8) give the value and bound to variable $b_{i,j}$, respectively. Constraints (9)–(13) are copy constraints [28], of which constraints (9)–(11) force two connected nodes in N, or any depot node and its successors, to have the same value as the corresponding $b_{i,j}$. In other words, if nodes $i,j\in N$ are connected or $i\in N$ is served immediately after depot node $j\in V_d$, then all of the nodes served before node i must be served before j, and vice versa. Constraint 12 ensures that, for depot node j and its predecessor i, the nodes served before i must be served before j. Note that there is no constraint to ensure that the nodes served before j also must be served before i. The reason is as follows. As a depot node, j may have multiple predecessors. For example, there exist two different nodes $i_1, i_2 \in N$, which are served just before returning to depot node j, albeit by different vehicles. Then, for nodes i_1, i_2, there is no requirement concerning which one is served before the other. Constraint (13) copies the values among the depot nodes, and is needed in case some routes are empty.

Constraint (14) ensures that the pickup and delivery nodes of each request are in the same route. Remember that Constraint (6) forces each customer to be served by only one vehicle. Therefore, these two constraints together guarantee that the pickup and delivery nodes of each request are served by the same route of one vehicle.

Constraints (15)–(18) are the time window constraints. The $M_{i,j}$ in Constraint (15) satisfies the condition that $M_{i,j} \geq l_i + s_i + t_{i,j} - e_j$. The $M_{i,k,r}$ in Constraint (16) satisfies the condition that $M_{i,k,r} \geq LV + t_{2n+r,i} - e_i$, and the $M_{i,k,r}$ in Constraint (17) ensures that $M_{i,k,r} \geq l_i + s_i + t_{i,2n+r+1} - EV$. Constraint (19) constitutes the precedence and maximum ride time constraints, and Constraint (20) is the maximum duration constraint for each route.

Constraint (21) reflects the load relationship between two consecutive visited nodes. The $M_{i,j}$ in this constraint satisfies $M_{i,j} \geq d_i^{max} + q_j - d_j^{min}$, where d_i^{max} refers to the maximum load a vehicle may hold after serving node i, and d_j^{min} refers to the minimum load it may hold after serving node j. Constraint (22) is the capacity constraint. Note that the assistants in the vehicle also occupy seats, and therefore they are also considered when calculating the vehicle load. For any route, the number of assistants onboard remains the same throughout the entire trip, which is reflected by Constraint (23), where o_{max} refers to the maximum number of assistants required by clients.

Constraint (24) guarantees that the end time of a route is not earlier than its start time. This constraint is needed in case some routes are empty. Constraint (25) represents the requirement for a break between two consecutive routes. Note that when a route is empty, no break is required.

Finally, Constraints (26)–(29) force every vehicle to have a lunch during the workday. When a vehicle is scheduled for a lunch break after a certain route, the break time after that route should be adapted to lunch break time period T_{lunch}, as indicated by Constraint (26). The time window constraint of the lunch break is reflected by Constraint (28), where $M_{k,r} \geq LV - LL$. Constraint (29) means that a vehicle must either begin operation after the lunch break or have a lunch break during the working period. The M_k in this constraint satisfies $M_k \geq EL + T_{lunch} - EV$.

Although variable $b_{i,j}$ is a binary variable according to its definition shown in Constraint (31), only linear relaxation is needed, as Lu and Dessouky [28] proved. To tighten the linear relaxation of the problem, the following redundant constraints could be added. These constraints help speed up the process of the CPLEX solver

$$b_{i,j} = 1, \quad i = 2n+1 \text{ or } j = 2n+|R|+1 \tag{35}$$

$$b_{i,n+i} = 1, \quad \forall i \in P \tag{36}$$

$$b_{2n+i,2n+j} = 1, \quad \forall i,j \in R, \ i<j. \tag{37}$$

4. Algorithm description

The MA [33], which is also called the hybrid genetic algorithm, is now widely used as a synergy of the population-based evolutional approach with separate individual local improvement procedures. The MA incorporates the advantages of both algorithms, with the local search (LS) exploiting the search intensively and the population-based approach exploring the search space broadly by maintaining diversification. We propose a customized recombination operator for this tightly constrained problem that differs from that in the literature. In addition, we employed the advanced population diversity scheme proposed by Vidal et al. [55], which has proved quite effective for various VRP variants [56,57,11]. The MA framework is given in Fig. 2, and its components are explained in this section. Section 4.1 describes how to represent a chromosome and examine its feasibility. Then, the regret insertion method is presented in Section 4.2 to construct a solution. The crossover operator is given in Section 4.3 to generate an offspring chromosome, followed by an local search (4.4) to improve the solution. Finally, the advanced survivor selection is reported in Section 4.5.

4.1. Chromosome and feasibility examination

Each route is represented as a visit sequence of pickup and delivery nodes that starts and terminates at the depot. The route is feasible only if it satisfies the corresponding constraints. Also, for each vehicle, the routes are recorded according to the order in which they operate. The chromosome of the instance depicted in Fig. 1 is shown in Fig. 3.

This problem requires the assignment of requests to a route and then the scheduling of routes for the vehicles. Thus, it is not trivial to examine the feasibility of the solutions. To examine a solution, we first check whether every route is feasible in terms of the related constraints, such as capacity, time windows, duration limit, ride time constraint, and staff requirements. If all routes are feasible, we then attempt to identify a feasible schedule for each route. If a solution is examined as infeasible, it is discarded.

4.1.1. Feasibility check of single route

For a given route, there are usually many time instants at which the service could start, rendering the route feasible. However, for a multi-trip schedule, we need to calculate a more exact start-time window for each route in such a way that if the route begins at any instant of that window, the shortest waiting time is induced. Vidal et al. [56] recently developed an efficient segment-based evaluation scheme for the VRPTW. Because the operators considered in this paper are related to the removal and reinsertion of requests, the segment-based method can be adapted to calculate the start-time window and examine the constraints for the MTDARP.

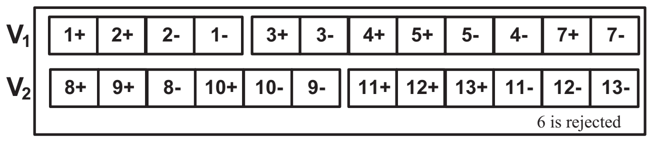

Fig. 3. Example of a chromosome representing a solution.

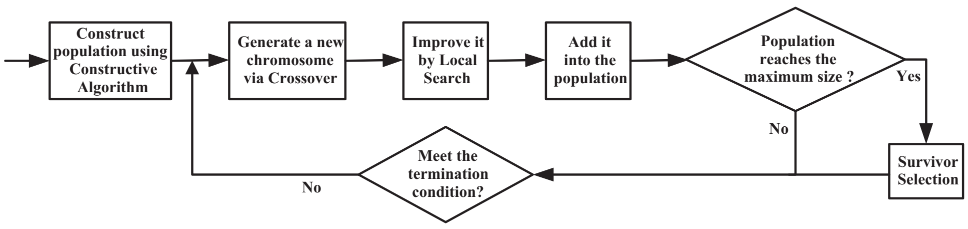
Construct population using Constructive Algorithm
Generate a new chromosome via Crossover
Improve it by Local Search
Add it into the population
Population reaches the maximum size ?
Yes
No
Survivor Selection
Meet the termination condition?
No

Fig. 2. Framework of the memetic algorithm.

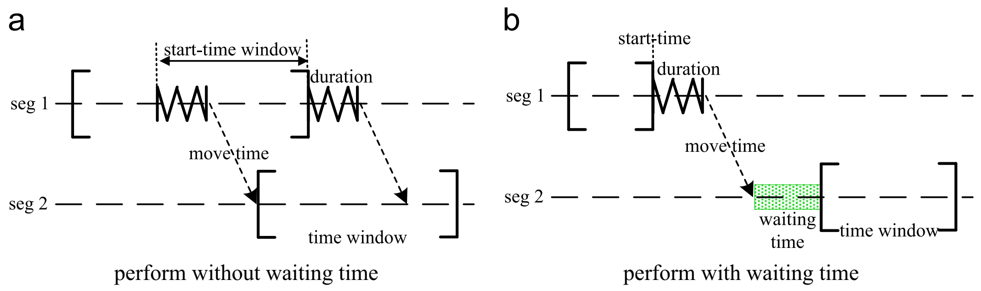

Fig. 4. Calculation of start-time window and duration: (a) perform without waiting time and (b) perform with waiting time.

A route can be seen as a series of concatenated segments. Let σ represent a segment of consecutive nodes, $TW(\sigma)$ indicate the start-time window for the segment with the shortest waiting time, and $D(\sigma)$ and $C(\sigma)$ be the corresponding minimum duration and travel cost, respectively. In addition, $Q(\sigma)$ is the vehicle load after the last node of the segment is serviced. Then, peak load $PQ(\sigma)$ indicates the maximum load at certain node, and $SF(\sigma)$ represents the number of assistants needed to service each node. At the beginning, these notations for segment $\sigma_0 = (v)$ with a single node v are initialized as follows: $TW(\sigma_0) = [e_v, l_v]$, $D(\sigma_0) = s_v$, $C(\sigma_0) = 0$, $Q(\sigma_0) = q_v$, $PQ(\sigma_0) = q_v$, and $SF(\sigma_0) = o_v$. Finally, two basic operators on time windows are then defined as follows:

$$TimeWindow[a, b] \pm d = [a \pm d, b \pm d] \tag{38}$$

$$TimeWindow[a, b] \cap TimeWindow[c, d] = \begin{cases} [b, b] & \text{if } b < c \\ \varnothing & \text{if } d < a \\ [\max\{a, c\}, \min\{b, d\}] & \text{otherwise} \end{cases} \tag{39}$$

Thus, for the two segments $\sigma = (\sigma_i, \ldots, \sigma_j)$ and $\sigma' = (\sigma'_i, \ldots, \sigma'_j)$, the values of the notations of concatenated segment $\sigma \oplus \sigma'$ are computed as follows, where $\oplus$ represents the concatenation operator:

$$TW(\sigma \oplus \sigma') = TW(\sigma) \cap (TW(\sigma') - t_{\sigma_j, \sigma'_i} - D(\sigma)) \tag{40}$$

$$D(\sigma \oplus \sigma') = D(\sigma) + D(\sigma') + t_{\sigma_j, \sigma'_i} + \Delta_{WT} \tag{41}$$

$$C(\sigma \oplus \sigma') = C(\sigma) + C(\sigma') + t_{\sigma_j, \sigma'_i} \tag{42}$$

$$Q(\sigma \oplus \sigma') = Q(\sigma) + Q(\sigma') \tag{43}$$

$$PQ(\sigma \oplus \sigma') = \max\{PQ(\sigma), Q(\sigma) + PQ(\sigma')\} \tag{44}$$

$$SF(\sigma \oplus \sigma') = \max\{SF(\sigma), SF(\sigma')\}, \tag{45}$$

where $\Delta_{WT} = \max\{TW(\sigma') \cdot e - (TW(\sigma) \cdot l + D(\sigma) + t_{\sigma_j, \sigma'_i}), 0\}$ is the waiting time. An intuitive explanation for the time equations is depicted in Fig. 4. The new segment $\sigma \oplus \sigma'$ is feasible if $TW(\sigma \oplus \sigma') \neq \varnothing$, $D(\sigma \oplus \sigma') \leq T_{max}$, and $PQ(\sigma \oplus \sigma') + SF(\sigma \oplus \sigma') \leq Q$. Moreover, for the relationship between the earliest start time $TW(\sigma \oplus \sigma') \cdot e$ and the latest start time $TW(\sigma \oplus \sigma') \cdot l$, the case of $TW(\sigma \oplus \sigma') \cdot e < TW(\sigma \oplus \sigma') \cdot l$ indicates that this route can be completed without any waiting time; otherwise, the route contains waiting time, and the best start time is a time instant.

The information on all segments of a solution can be stored in a preprocessing phase, and the operators involve the recombination of a small number of segments. Hence, every operator is evaluated in constant time, as noted by Vidal et al. [56,57].

After the aforementioned constraints are satisfied, the ride time constraint is examined. If the route contains waiting time, the service start time at each node needs to be decided more carefully. Thus, to solve the ride time constraint, we adapt the eight-step evaluation scheme proposed by Cordeau and Laporte [13], which uses the forward time slack introduced by Savelsbergh [49]. The forward time slack defines the maximum amount of time by which the start time can be delayed at one node without causing the other nodes to violate the time windows and ride time constraints [24]. For node n_i in the route $(n_0, n_1, \ldots, n_q)$, where n_0 and n_q represent the depot, the calculation is as follows:

$$F_{n_i} = \min_{i \leq j \leq q} \left(\sum_{i < p \leq j} W_{n_p} + \max\{\min\{l_{n_j} - B_{n_j}, f_{n_j} - P_{n_j}\}, 0\} \right), \tag{46}$$

where W_{n_p} is the waiting time at node n_p, B_{n_j} represents the start time of service at node n_j, and P_{n_j} denotes the ride time of a passenger whose delivery node is n_j and whose corresponding pickup node is before n_i on the route; otherwise, $P_{n_j} = -\infty$, which means that the ride time has no effect on the computation.

A detailed evaluation of ride time is provided in Algorithm 1. W_{n_p}, B_{n_i}, and arrival time A_{n_i} at node n_i are computed as follows:

$$A_{n_i} = B_{n_{i-1}} + s_{n_{i-1}} + t_{n_{i-1}, n_i}, \quad W_{n_i} = \max\{e_{n_i} - A_{n_i}, 0\}, \quad B_{n_i} = A_{n_i} + W_{n_i} \tag{47}$$

Note that Step 4 in the procedure is invoked only when waiting time exists at some nodes. Finally, if a feasible schedule exists, then the service start time at each node is specified.

Algorithm 1. Procedure for solving the ride time constraint.

```
RideTimeEvaluation (a given route)
1 Set $B_{n_0}$ as the specific start time of the route belonging to the start-time window
2 Compute $A_{n_i}$, $W_{n_i}$, and $B_{n_i}$ for each node $n_i$ using Eq. (47)
3 Compute all ride time $P_{n_i}$; if all $P_{n_i} \leq f_{n_i}$, return TRUE
4 for every node $n_j$ that is a pickup node.
      (a) Compute $F_{n_j}$
      (b) Let $W_{n_j} \leftarrow W_{n_j} + \min\{F_{n_j}, \sum_{j<p<q} W_{n_p}\}$, $B_{n_j} \leftarrow A_{n_j} + W_{n_j}$
      (c) Update $A_{n_i}$, $W_{n_i}$, and $B_{n_i}$ for each node $n_i$ after node $n_j$
      (d) Update ride time $P_{n_i}$ for each request whose delivery node is after $n_j$
      (e) If all $P_{n_i} \leq f_{n_i}$ for requests whose destinations lie after $n_j$, return TRUE
5 return FALSE
```

4.1.2. Feasibility check of multi-trip schedule

Following the foregoing computations, the start-time window and travel duration of each route can be obtained. For the set of routes assigned to a given vehicle, we need to check whether there is any order in which the vehicle could operate those routes without any time overlap. To address the need for a lunch break, a dummy route with the start-time window as the lunch window and the route duration as the lunch duration is added for each vehicle.

Assuming that $R(r_1, r_2, \ldots, r_u)$ is the set of routes assigned to a vehicle, we first quickly examine the schedule via a greedy method. If it fails to obtain a schedule plan, the dynamic programming (DP) method is invoked to address this subproblem exactly.

The greedy method sorts the routes by their earliest start times, breaking ties by the minimum route duration. The feasibility of the order obtained can be examined with a forward sweep as follows:

$$B(r_1) = TW(r_1).e, \quad B(r_i) = \max\{B(r_{i-1}) + D(r_{i-1}) + \delta', TW(r_i).e\}, \quad i = 2, \ldots, u, \tag{48}$$

where δ' equals disinfection time δ if neither route r_{i-1} nor r_i is the dummy route, and otherwise $\delta' = 0$. The schedule is feasible only if every $B(r_i) \le TW(r_i).l$ is respected. In most cases, a feasible schedule can be obtained through the greedy method, and its computational complexity is $O(u \log u)$, where u is the number of trips performed by the vehicle. Thus, it is quite an efficient way to obtain schedule plans. The DP method can determine whether a feasible schedule exists precisely. Its complexity is $O(2^u)$, which is much less than the complexity of enumerating all possible orders $O(u! \cdot u)$. The pseudo-code of the DP method is shown in Algorithm 2, where $F(R)$ indicates the earliest possible time that all routes in set R can be completed. $F(R)$ is calculated recursively, and the schedule is feasible if $F(R) < \infty$, which means that the vehicle can complete all routes without any time conflicts during the working period.

Algorithm 2. Pseudo-code of DP method.

```
DPSchedule(route set $R$)
1   $F(R) \leftarrow \infty$
2   if $R = \varnothing$
3       $F(R) \leftarrow 0$
4   else
5       for each route $r_i$ in set $R$
6           $R' \leftarrow R \setminus r_i$
7           DPSchedule($R'$)
8           if $F(R') + \delta' \le TW(r_i).l$
9               $F(R) \leftarrow \min\{(F(R), \max\{F(R') + \delta', TW(r_i).e\} + D(r_i))\}$
10  return $F(R)$
```

In this study, a route is not bound to a given vehicle. Inserting a request into a route may result in a vehicle being unable to complete its assigned routes successfully. When such a situation occurs, the new route can be shifted to another vehicle that can accommodate it rather than simply rejecting the insertion. This inherent flexibility increases the likelihood of identifying a feasible schedule, as shown in Fig. 5.

According to the foregoing procedures, a solution is feasible only if all routes are feasible and a feasible schedule of routes has been identified. In this study, feasible solutions alone are permitted to be added to the population. Thus, the search space is limited to the feasible region.

4.2. Constructive algorithm

The regret insertion (RI) method improves the basic greedy insertion method via a look-ahead scheme, which has proved quite effective for solving tightly constrained problems [42,15]. Accordingly, the RI method is used here to generate initial solutions.

We denote $\Delta C_{h,r}$ as the lowest insertion cost of accommodating request h in route r, and set $\Delta C_{h,r} = +\infty$ if the request cannot be inserted into r. The feasibility of insertion is examined using the mechanism presented in the previous section, and the route can be shifted to another vehicle if necessary. The basic greedy insertion method iteratively selects the request with the lowest

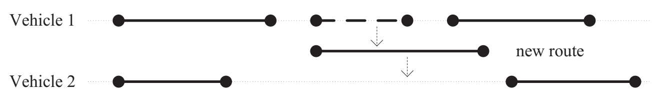

Fig. 5. Importance of shifting a new route.

insertion cost, and inserts it into the corresponding route. However, the RI method first inserts the request with the maximum regret value, which is defined as follows for request h:

$$RV = \sum_{i=1}^{k} (\Delta C_{h,r_i} - \Delta C_{h,r_1}), \tag{49}$$

where r_i indicates the route for which request h has the ith lowest insertion cost. Note that an empty trip is maintained, by which a request can be easily inserted into a new route if a lower cost is induced. The *reg-k* methods are obtained with a different value of k, and their performance is usually different. Note that the method with $k=1$ is the basic greedy insertion. Finally, all unserved requests are put into a bank.

To accelerate the process, the compatibility of pairs of requests is examined in the preprocessing stage and stored in a matrix. Then, before trying to insert a request into a route, we first retrieve the information from the matrix to check whether the request can coexist with those already in the route. If it cannot, all insertion attempts are abandoned. This strategy is quite effective for instances with narrow time windows.

4.3. Crossover

The crossover operator is a pivotal component of the evolutionary algorithm, which is designed to ensure that the offspring solution inherits the good characteristics of both parents. Designing an effective crossover operator for this problem is quite difficult because it has a fairly constrained search space. Here, we propose a customized crossover operator that combines the routes of two parents to generate an offspring solution. The offspring solution is generally obtained by replacing some of the routes of one parent with different routes from the other, and then repaired to satisfy all constraints.

First, two parents, P_1 and P_2, are selected through a binary tournament that randomly chooses two chromosomes and remains the one with the better objective value. Then, each route in parent P_1 is randomly chosen with a probability of 50% and cloned to the offspring solution. For the routes of parent P_2, the requests already present in the offspring are removed, which ensures that each request is serviced only once. Next, attempts are made to assign these routes to a vehicle of the offspring feasibly one by one. If no such vehicle exists, the route is discarded, and the requests are placed into a reject pool temporarily. Finally, the unserved requests are reinserted into the offspring via the *reg-k* method, where k is selected randomly. The crossover process is illustrated in Fig. 6, where the shaded requests in the parent sets are reserved to form new routes.

4.4. Local search

A local search is employed to improve the solution intensively. The move types include relocation and swap adapted from the traditional operators. The relocation operator shifts a request to another route, and inserts it into the best position with the lowest incremental cost. The request can be either from the bank or serviced in a route. The swap operator exchanges the places of two requests, one of which may be from the bank. The two requests are first erased from their original places, and then attempts are made to insert them into the previous place of the other request. Note that the insertion is not limited to the previous position of the other request, but to the best position with the lowest incremental travel cost. If a request cannot be inserted into a given route, it is put into the bank temporarily. Finally, the solution is further completed through the *reg-k* method with a random k. We can see that both move types have the ability to reduce the number of unserved requests.

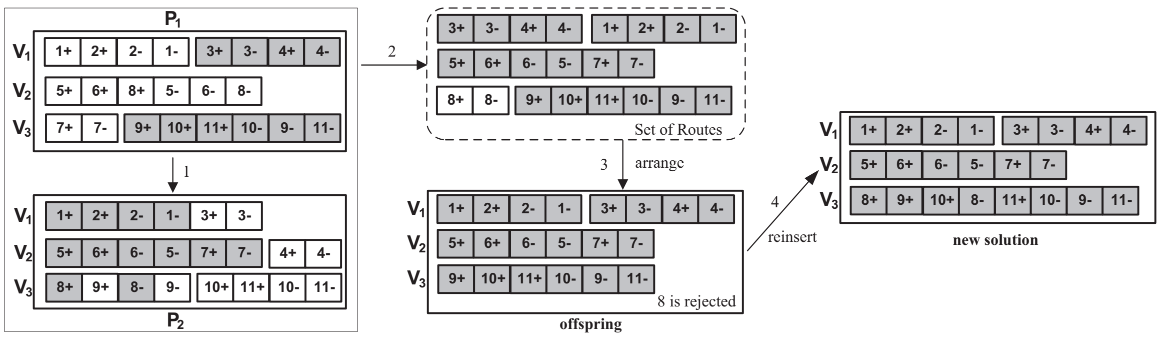

Fig. 6. Crossover operator used to obtain the offspring solution.

In the local search, the two move types are applied cyclically to systematically exploit the search space with the first improvement strategy that the first better solution encountered is immediately accepted as the current solution. The procedure terminates when no improving move is identified.

The feasibility check of a solution is clearly time-consuming. Thus, several strategies are employed to speed up the search process. The compatibility matrix described in Section 4.2 is also utilized here. In addition, information on moves is also recorded to avoid duplicate checks. More precisely, two matrices are used to store the feasibility information of moves, relocating a request to a route and swapping two requests, respectively. These matrices are initialized as feasible and then updated accordingly during the search process. If a move cannot generate a feasible solution, its corresponding information is updated as infeasible. Then, in the next attempt to identify an improving solution, moves marked as infeasible are ignored. When a move obtains an improving solution, the move is performed immediately to update the current solution, and the information on moves that relocate requests to updated routes and swap the requests belonging to those updated routes is reset as feasible. This procedure is repeated until all moves are marked as infeasible or no improving solution is identified. However, in the case of a multi-trip schedule, a move may become feasible even though it is not directly involved with the updated routes. Hence, to explore the search space exhaustively, the two matrices are reset as feasible, and the foregoing procedure is invoked again. If an improving solution can still not be obtained, the local search is terminated.

4.5. Population management

The initial population is formed by μ chromosomes, which are constructed by the *reg-k* methods and improved by the local search with 50% probability. More precisely, the first chromosome is generated with the *reg-2* method, and the number of routes is denoted as $|R|$. Then, the remaining $\mu-1$ chromosomes are obtained via the *reg-k* method with k uniformly selected from the interval $[1, |R|]$ at random.

In each iteration, an offspring solution is generated by the crossover operator and improved by the local search. Then, it is directly included in the population. If the size of the population reaches the maximum size limit $\mu+\lambda$, the survivor selection procedure proposed by Vidal et al. [55] is invoked to discard λ individuals.

Survivor selection takes into account both the objective cost of chromosomes and their diversity contributions to the population. The diversity contribution $f(S)$ of an individual S is defined as the average distance between S and its n_c closest neighbors in the population denoted as N_c, which is computed as follows:

$$f(S)=\frac{1}{n_c}\sum_{S_1\in N_c} D(S,S_1), \tag{50}$$

where $D(S,S_1)$ measures the common arcs between S and S_1. Then, the biased fitness $BF(S)$ is calculated for each chromosome as

$$BF(S)=r_o(S)+\left(1-\frac{n_e}{N_p}\right)r_f(S), \tag{51}$$

where N_p is the size of the current population, and $r_o(S)$ and $r_f(S)$ indicate the ranks of chromosome S in the population with respect to the objective cost and diversity contribution, respectively. Parameter n_e is the number of elite chromosomes carried over from the current generation to the next. The elitism properties are guaranteed during selection by Eq. (51), as proved by Vidal et al. [55]. In accordance with biased fitness, the following detailed selection procedure is performed. First, for chromosomes with the same objective cost, only the one with the minimum biased fitness is retained. Second, if the size of the population is still larger than μ, those with the maximum biased fitness are removed.

5. Experiments and analysis

To evaluate the performance of the proposed algorithm, we conducted several computational experiments on real-world test data contributed by public hospitals in Hong Kong. The experiments can be classified into four parts. First, we determine the parameters applied in the algorithm. Second, we compare the results obtained by the proposed algorithm with those obtained by solving the mathematical model on small instances. Third, the results for larger scale instances are reported for future studies. Finally, the proposed algorithm is evaluated on instances of the classic DARP. In this study, the mathematical model is solved with CPLEX 12.5, and the proposed MA is coded in C++ and performed on an Intel Xeon E5430 clocked at 2.66 GHz (Quad Core) with 8 GB RAM running the CentOS 5 Linux operating system. The runtime of the MA for each instance is limited by the number of requests n: 1800 CPU seconds for $n\le 50$, 3600 CPU seconds for $50<n\le 100$, 5400 CPU seconds for $100<n\le 150$, and 7200 CPU seconds for $n>150$. All experiments are run 10 times with random seed 1 to 10. The test instances and detailed solution plans can be downloaded at http://www.computational-logistics.org/orlib/mtdarp.

5.1. Computational data

As noted, we conducted our experiments using a set of real-world data from public hospitals in Hong Kong. The daily information forms an instance comprising three parts, namely, resource, request, and traffic data. The resource data provide the number of available ambulances, their capacity, and the lunch window. The request data include seat demand, assistant demand, time windows, and the attributes related to the request, such as the locations of the origin and destination. The traffic data constitute a matrix of the travel time between any two locations, which

Table 1
Calibration experiment results for parameters μ and λ.

μ	$\lambda=5$			$\lambda=10$			$\lambda=15$			$\lambda=20$		
	UR	*Cost*	T_{bst}	*UR*	*Cost*	T_{bst}	*UR*	*Cost*	T_{bst}	*UR*	*Cost*	T_{bst}
5	1.35	1127.23	2904.8	1.40	1129.02	2602.2	1.39	1130.95	2875.9	1.38	1132.09	2673.1
10	1.42	1129.28	2981.4	1.37	1131.39	2774.0	**1.34**	**1133.13**	2647.3	1.40	1130.95	2695.7
15	1.36	1134.43	2746.7	1.39	1133.07	2753.6	1.38	1133.67	2719.7	1.39	1134.95	2627.2
20	1.42	1131.59	2457.8	1.40	1132.72	2595.4	1.37	1134.24	2497.0	1.40	1132.63	2577.3

Table 2
Calibration experiment results for parameters p_c and p_e.

p_e	$p_c=0.2$			$p_c=0.4$			$p_c=0.6$			$p_c=0.8$		
	UR	*Cost*	T_{bst}	*UR*	*Cost*	T_{bst}	*UR*	*Cost*	T_{bst}	*UR*	*Cost*	T_{bst}
0.2	1.37	1130.36	2751.2	1.38	1130.49	2965.2	1.35	1132.36	2941.1	1.37	1130.07	2931.5
0.4	1.37	1129.17	2845.8	1.34	1133.13	2648.4	1.33	1131.35	2793.9	1.34	1130.96	2982.0
0.6	1.35	1130.73	2876.8	1.34	1130.01	3057.3	**1.32**	**1130.24**	3023.4	1.34	1131.63	2962.4
0.8	1.37	1132.00	2964.9	1.37	1131.53	3043.7	1.39	1131.61	2966.7	1.36	1132.23	3015.0

satisfies the triangle inequality. Owing to the operational arrangements of Hong Kong public hospitals, the instances can be classified into two types: *holiday* and *non-holiday* instances. *Holiday* instances have wide time windows and fewer requests. These time windows are usually the whole morning, the whole afternoon, or even the whole day, and the number of requests ranges from 29 to 92. In *non-holiday* instances, in contrast, the widths of most time windows are between 30 and 60 min, although some requests may have windows as wide as those in the *holiday* instances. The number of requests varies from 72 to 185. With regard to the vehicle information, the capacity ranges from 6 to 13, and the number of ambulances ranges from 2 to 11.

5.2. Parameter tuning

In this section, we report a series of experiments carried out to determine the parameters applied in the proposed algorithm. Parameter μ is the minimum size of the population, and λ decides when to invoke survivor selection. For the preliminary test, we choose 10 instances with different numbers of requests: 4 *holiday* instances (instances 5, 10, 15, 20) and 6 *non-holiday* instances (instances 5, 10, 15, 20, 25, 30). Four different values are tested for each parameter, giving us 16 pairs of parameter combinations. The average results obtained are reported in Table 1, where notations *UR*, *Cost*, and T_{bst} denote the average number of unserved requests, the average travel cost, and the average time used to obtain the final solutions. The best result is marked in bold. The parameters with $\mu=10$ and $\lambda=15$ achieved the best performance because they allow more requests to be serviced. Thus, these values are adopted in the following experiments.

Moreover, parameter n_c controls calculation of the diversity contribution, and n_e determines the number of elite solutions reserved for the next iteration. In the proposed algorithm, they are defined as $n_c=\mu p_c$ and $n_e=\mu p_e$, respectively. Thus, the combinations of different values of p_c and p_e are evaluated, with the results presented in Table 2. Better results are obtained when p_c and p_e are 0.6. Hence, the values of p_c and p_e are set as 0.6.

5.3. Comparison on small instances

To validate the performance of the proposed MA, we generate a number of small instances by picking several requests and storing the corresponding information. These instances are then solved to optimality with CPLEX solver 12.5 according to the proposed mathematical model in Section 3. In this model, there is quite a large number M in objective function (1), and each client can be served at most once (see Constraint (5)) to achieve the objective of the proposed MTDARP, i.e., to first serve as many client as possible and then reduce the travel cost as much as possible. Note that for small instances, all clients can be fully served. Therefore, we can remove M from objective function (1) and replace $\leq$ with $=$ in Constraint (5), i.e., requiring all clients to be served once. Our experiment shows that these changes allow more instances to be solved to optimality with the CPLEX solver. Because the main purpose of solving the small instances to optimality is to compare the optimal solution with that generated by the proposed MA, the changes are acceptable.

Because of the limitation imposed by 8 GB of physical memory, the maximum number of requests in the solved instances is 15. Here, we use 30 instances with 10–15 requests and 2 vehicles as the test data. The optimality and average results of the MA are shown in Table 3, where *Cost* is the travel cost, T_{tot} is the time required by the CPLEX solver, and T_{bst} is the time required by the MA to obtain the final results. In all of the following tables, K is the number of vehicles and n is the number of requests. Note that all requests can be fully served by two vehicles. We can see that the proposed MA can solve each small instance to optimality quickly for every run and that optimality can be achieved in less than 1 s for most instances.

5.4. Results for larger instances

The detailed results for the *holiday* and *non-holiday* instances, including the best result and the average result over 10 runs, are shown in Tables 4 and 5, respectively. The notations are as defined above. Table 5 shows that, for the *non-holiday* instances, the proposed algorithm exhibits stable performance regarding the number of unserved requests with the exception of two instances. However, for half of the *holiday* instances, the number of unserved requests varies in the different runs, as shown in Table 4. These results indicate that it is more difficult to solve instances with wide time windows. In addition, the costs over the 10 runs exhibit greater variance for larger instances.

Observing the solutions for the *holiday* and *non-holiday* instances obtained by the MA, we can see that the number of trips completed by a vehicle usually ranges from four to seven, but reaches eight for some *holiday* instances. The number of requests serviced in one trip generally ranges from one to five, but can reach six in some *non-holiday* solutions. However, the number of trips servicing a given number of requests varies in different instances. For example, trips servicing four requests are quite rare in certain instances. Moreover,

Table 3
Comparison for small instances.

Inst.	K	n	Optimality		MA		Inst.	K	n	Optimality		MA	
			Cost	T_{tot}	*Cost*	T_{bst}				*Cost*	T_{tot}	*Cost*	T_{bst}
1	2	10	133	10	133	0.02	16	2	12	200	16,481	200	0.02
2	2	10	157	6	157	0.00	17	2	12	160	7978	160	0.02
3	2	10	158	41	158	0.01	18	2	13	142	11,028	142	0.02
4	2	10	155	256	155	0.00	19	2	13	136	1146	136	0.01
5	2	11	137	1579	137	0.03	20	2	13	139	38	139	0.01
6	2	11	148	1598	148	0.02	21	2	13	155	12,948	155	0.03
7	2	11	162	799	162	0.00	22	2	13	155	3991	155	0.03
8	2	11	134	3428	134	0.02	23	2	13	139	104	139	0.01
9	2	11	159	858	159	0.00	24	2	14	124	134	124	0.01
10	2	11	184	1311	184	0.00	25	2	14	124	140	124	0.04
11	2	11	156	2321	156	0.00	26	2	14	139	3917	139	0.21
12	2	12	146	1581	146	0.04	27	2	14	156	3859	156	2.87
13	2	12	150	886	150	0.11	28	2	14	121	6249	121	0.01
14	2	12	142	15,178	142	0.02	29	2	15	137	10,008	137	0.01
15	2	12	151	2133	151	23.45	30	2	15	138	16,749	138	0.01

Table 4
Detailed results for *holiday* instances.

Inst.	K	n	Best		Average		T_{bst}	Inst.	K	n	Best		Average		T_{bst}
			UR	*Cost*	*UR*	*Cost*					*UR*	*Cost*	*UR*	*Cost*	
101	2	29	0	296	0	296	128.6	111	3	58	1	701	1.3	696.6	1943.4
102	2	32	0	286	0	288.7	659.9	112	3	66	3	584	3	600.4	2143.8
103	2	37	0	314	0	314	138.1	113	3	69	2	581	2.1	596	2793.0
104	2	39	1	522	1.6	497.9	584.7	114	4	70	0	637	0	641	2766.3
105	2	40	3	517	3	517	83.0	115	3	75	8	563	8.2	569.4	2130.0
106	2	44	1	374	1	376.1	1169.6	116	4	78	1	855	1	869.3	1957.4
107	2	46	1	485	1.9	432.1	1012.6	117	4	79	2	932	2.9	906.9	2643.6
108	3	50	0	408	0	413.2	805.3	118	5	82	0	834	0	841.3	2752.5
109	2	53	2	364	2	368.1	2148.4	119	4	86	2	863	2.5	857.3	2365.9
110	3	57	0	694	0.6	671.7	2768.7	120	5	92	0	1119	0.7	1079.5	2223.4

Table 5
Detailed results for *non-holiday* instances.

Inst.	K	n	Best		Average		T_{bst}	Inst.	K	n	Best		Average		T_{bst}
			UR	*Cost*	*UR*	*Cost*					*UR*	*Cost*	*UR*	*Cost*	
201	4	72	3	781	3	784.5	2349.3	216	6	114	1	1324	1	1338.5	3810.9
202	4	73	3	771	3	773.6	2746.2	217	8	125	3	1384	3	1416	2544.6
203	4	75	0	857	0	877.3	2399.7	218	8	127	3	1599	3	1623.4	3406.1
204	4	76	2	918	2	937.7	2808.5	219	9	128	0	1362	0	1373.2	4180.0
205	5	78	0	864	0	873	2797.3	220	8	128	1	1401	1	1416.8	4070.2
206	5	79	4	1067	4	1078.3	2952.5	221	8	129	3	1367	3	1384.2	4488.2
207	5	80	0	848	0	854.5	2845.8	222	6	130	3	1256	3.2	1264.2	3037.5
208	5	85	1	1127	1	1139.1	2650.0	223	9	150	0	1892	0	1910.3	3239.4
209	6	96	0	1132	0	1151.3	2750.5	224	9	153	5	1806	5	1841.6	5603.2
210	6	97	0	1024	0	1046.6	3169.9	225	10	160	0	1788	0	1819.8	4819.6
211	6	100	1	1372	1	1398.5	1938.6	226	10	164	2	1886	2	1916.6	3878.0
212	6	103	1	1184	1	1196	4412.2	227	9	167	2	1786	2	1819.2	4696.6
213	7	104	1	1354	1	1365.7	3462.6	228	9	170	1	1912	1	1953.5	4459.8
214	6	106	2	1296	2	1313	3410.4	229	11	175	0	2294	1.7	2133.9	4479.9
215	7	107	0	1379	0	1389.7	4116.7	230	11	185	0	1905	0	1926	3395.3

trips servicing only one request are more frequent in the *non-holiday* instances, and the sequence of pickup and delivery nodes in a trip also varies from instance to instance.

To investigate the convergence of the proposed MA, we perform an additional experiment with the random seed taken as 1. Because the method usually converges quickly in the early stage, we record the number of unserved requests (*UR*) and total cost (*Cost*) of the best solution found by the MA at times of 1, 2, 4,…,4096, 7200 s. The percentage gaps of these values over the final solutions are shown in Fig. 7, where the instances are divided into groups according to the number of requests *n*. It can be observed from Fig. 7(a) that the MA converges very quickly on *UR* for the early 512 s from more than 50% to 15% with the exception of one group, and converges to the final solution before 2048 s for all groups except the *holiday* group with $n > 50$. Similarly in Fig. 7(b), the MA converges quickly on *Cost* from more than 5% to less than 2% during

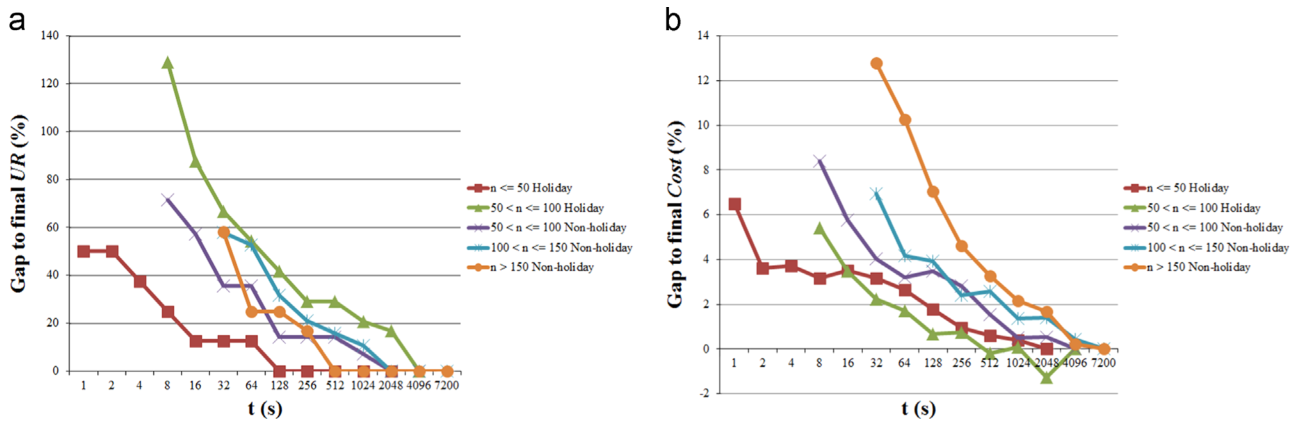

Fig. 7. Convergence of the MA on MTDARP instances: (a) convergence on the number of unserved requests and (b) convergence on cost.

Table 6
Comparison of average results on DARP instances.

Inst.		MA	Opt	GAP_{opt}	VNS	GAP_{VNS}	HLNS	GAP_{HLNS}	DP-LNS	GAP_{DP-LNS}	DA	GAP_{DA}
Medium	Type-a	683.71	671.68	1.79	–	–	672.40	1.68	672.75	1.63	–	–
	Type-b	737.42	727.33	1.39	–	–	727.97	1.30	729.14	1.14	727.36	1.38
Large	Type-a	566.49	–	–	535.34	5.82	533.36	6.21	538.31	5.24	532.80	6.32
	Type-b	528.01	–	–	491.76	7.37	491.54	7.42	506.16	4.32	491.32	7.47

the first 1024 s, and then converges to near final solutions before 2048 s for the groups with $n \le 100$. Note that the gaps in cost may fluctuate or become negative over time because when more requests are served the total cost may increase.

5.5. Comparison for DARP instances

To further evaluate the performance of the MA, the algorithm is slightly adapted to solve the classic DARP instances by setting route duration T_{max} to a very large value, the lunch time window to $[0,0]$, and the lunch duration to $T_{lunch}=0$. The parameters are kept the same as before. There are two groups of widely used benchmark data: the medium-size instances introduced by Cordeau [12] and Ropke et al. [46] which are usually tested by the exact methods, and the large-size instances introduced by Cordeau and Laporte [13] which are used to evaluate meta-heuristic algorithms. Both groups consist of two types: type-a has a tighter ride time constraint, whereas type-b has a less tight constraint. The medium-size group contains 21 type-a and 21 type-b instances, and the large-size group comprises 10 type-a and 10 type-b instances.

The results obtained by the MA are compared with those reported in the recent literature. More precisely, the optimal results of the medium-size instances are reported by Gschwind and Irnich [17], and the results for the large-size group are obtained by VNS [38], HLNS [39], DP-LNS [45], and DA [8]. Note that VNS and DP-LNS run the algorithm for 5–9 h and 3–5 h on some instances, which is much longer than our method. The comparisons of the average results are reported in Table 6, where the dash indicates that no results were reported. *Opt* means the optimal cost, and GAP_{opt} is the percentage gap of the MA over optimality. The notations in the other columns have similar meanings.

The MA easily obtains feasible solutions that service all requests. Table 6 shows that it performs well on medium-size instances. The percentage gaps over optimality are 1.79% and 1.39% for type-a and type-b, respectively. However, the MA performs worse on large-size instances, with gaps of more than 5%, particularly for the type-b instances. Possible explanations are as follows. The crossover operator is customized for the MTDARP whose solutions contain a number of short trips rather than a few long trips. Moreover, the parameters are not specially designed for DARP. Furthermore, the runtime is much shorter than the VNS and DP-LNS methods.

6. Conclusion

In this paper, we solved a patient transportation problem derived from a real-world healthcare transportation service provided by the Hong Kong Hospital Authority. In accordance with the characteristics of the problem, it is modeled as a multi-trip dial-a-ride problem (MTDARP), a new variant of the vehicle routing problem. We provide a formulation for the problem and develop a memetic algorithm with a customized recombination operator to solve it. The regret insertion method is used to construct the population, and the segment-based evaluation scheme is adapted to evaluate the temporal solutions. Moreover, the currently advanced population management framework is employed to maintain the population. The proposed algorithm was tested on the real-world data. The comparisons on small instances demonstrate the proposed algorithm's ability to solve them to optimality quickly. Finally, the results for larger instances are provided for future studies.

Acknowledgements

This work was partially supported by the Hong Kong Research Grants Council (Grant no: T32-102/14N).

References

[1] Alonso F, Alvarez MJ, Beasley JE. A tabu search algorithm for the periodic vehicle routing problem with multiple vehicle trips and accessibility restrictions. Journal of the Operational Research Society 2007;59:963–76.
[2] Azi N, Gendreau M, Potvin J-Y. An exact algorithm for a single-vehicle routing problem with time windows and multiple routes. European Journal of Operational Research 2007;178:755–66.
[3] Azi N, Gendreau M, Potvin J-Y. An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles. European Journal of Operational Research 2010;202:756–63.

[4] Baldacci R, Bartolini E, Mingozzi A. An exact algorithm for the pickup and delivery problem with time windows. Operations Research 2011;59:414–26.
[5] Bard JF, Jarrah AI. Integrating commercial and residential pickup and delivery networks: a case study. Omega 2013;41:706–20.
[6] Battarra M, Monaci M, Vigo D. An adaptive guidance approach for the heuristic solution of a minimum multiple trip vehicle routing problem. Computers and Operations Research 2009;36:3041–50.
[7] Beaudry A, Laporte G, Melo T, Nickel S. Dynamic transportation of patients in hospitals. OR Spectrum 2010;32:77–107.
[8] Braekers K, Caris A, Janssens GK. Exact and meta-heuristic approach for a general heterogeneous dial-a-ride problem with multiple depots. Transportation Research Part B: Methodological 2014;67:166–86.
[9] Brandão J, Mercer A. A tabu search algorithm for the multi-trip vehicle routing and scheduling problem. European Journal of Operational Research 1997;100:180–91.
[10] Brandão JCS, Mercer A. The multi-trip vehicle routing problems. The Journal of the Operational Research Society 1998;49:799–805.
[11] Cattaruzza D, Absi N, Feillet D, Vidal T. A memetic algorithm for the multi trip vehicle routing problem. European Journal of Operational Research 2014;236:833–48.
[12] Cordeau J-F. A branch-and-cut algorithm for the dial-a-ride problem. Operations Research 2006;54:573–86.
[13] Cordeau J-F, Laporte G. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. Transportation Research Part B: Methodological 2003;37:579–94.
[14] Cordeau J-F, Laporte G. The dial-a-ride problem: models and algorithms. Annals of Operations Research 2007;153:29–46.
[15] Diana M, Dessouky MM. A new regret insertion heuristic for solving large-scale dial-a-ride problems with time windows. Transportation Research Part B: Methodological 2004;38:539–57.
[16] Fleischmann B. The vehicle routing problem with multiple use of vehicles. Working paper; 1990.
[17] Gschwind T, Irnich S. Effective handling of dynamic time windows and its application to solving the dial-a-ride problem. Transportation Science 2015:1–20. http://dx.doi.org/10.1287/trsc.2014.0531 forthcoming.
[18] Guerriero F, Bruni ME, Greco F. A hybrid greedy randomized adaptive search heuristic to solve the dial-a-ride problem. Asia-Pacific Journal of Operational Research 2013;30.
[19] Hanne T, Melo T, Nickel S. Bringing robustness to patient flow management through optimized patient transports in hospitals. Interfaces 2009;39:241–55.
[20] Hernandez F, Feillet D, Giroudeau R, Naud O. A new exact algorithm to solve the multi-trip vehicle routing problem with time windows and limited duration. 4OR 2014;12:235–59.
[21] Jain S, Hentenryck PV. Large neighborhood search for dial-a-ride problems. In: Lee J, editor. Principles and practice of constraint programming, Lecture notes in computer science, vol. 6876. Berlin, Heidelberg: Springer; 2011. p. 400–13.
[22] Jorgensen RM, Larsen J, Bergvinsdottir KB. Solving the dial-a-ride problem using genetic algorithms. Journal of the Operational Research Society 2007;58:1321–31.
[23] Karaoglan I, Altiparmak F, Kara I, Dengiz B. The location-routing problem with simultaneous pickup and delivery: formulations and a heuristic approach. Omega 2012;40:465–77.
[24] Kirchler D, Wolfler Calvo R. A granular tabu search algorithm for the dial-a-ride problem. Transportation Research Part B: Methodological 2013;56:120–35.
[25] Li H, Lim A. A metaheuristic for the pickup and delivery problem with time windows. International Journal on Artificial Intelligence Tools 2003;12:173–86.
[26] Lim A, Zhang Z, Qin H. Pickup and delivery service with manpower planning in hong kong public hospitals. Transportation Science 2015, in press.
[27] Liu R, Xie X, Garaix T. Hybridization of tabu search with feasible and infeasible local searches for periodic home health care logistics. Omega 2014;47:17–32.
[28] Lu Q, Dessouky M. An exact algorithm for the multiple vehicle pickup and delivery problem. Transportation Science 2004;38:503–14.
[29] Ma H, Cheang B, Lim A, Zhang L, Zhu Y. An investigation into the vehicle routing problem with time windows and link capacity constraints. Omega 2012;40:336–47.
[30] Macedo R, Alves C, Valério de Carvalho J, Clautiaux F, Hanafi S. Solving the vehicle routing problem with time windows and multiple routes exactly using a pseudo-polynomial model. European Journal of Operational Research 2011;214:536–45.
[31] Melachrinoudis E, Ilhan AB, Min H. A dial-a-ride problem for client transportation in a health-care organization. Computers and Operations Research 2007;34:742–59.
[32] Mingozzi A, Roberti R, Toth P. An exact algorithm for the multitrip vehicle routing problem. INFORMS Journal on Computing 2013;25:193–207.
[33] Moscato P, Cotta C. A modern introduction to memetic algorithms. In: Gendreau M, Potvin J-Y, editors. Handbook of metaheuristics, International series in operations research & management science, vol. 146. US, Boston, MA: Springer; 2010. p. 141–83.
[34] Nagata Y, Kobayashi S. A memetic algorithm for the pickup and delivery problem with time windows using. In: Parallel problem solving from nature, PPSN XI. Berlin, Heidelberg: Springer; 2010. p. 536–45.
[35] Olivera A, Viera O. Adaptive memory programming for the vehicle routing problem with multiple trips. Computers and Operations Research 2007;34:28–47.
[36] Parragh S. Ambulance routing problems with rich constraints and multiple objectives [Ph.D. thesis]. uniwien; 2009.
[37] Parragh SN, Cordeau J-F, Doerner KF, Hartl RF. Models and algorithms for the heterogeneous dial-a-ride problem with driver-related constraints. OR Spectrum 2012;34:593–633.
[38] Parragh SN, Doerner KF, Hartl RF. Variable neighborhood search for the dial-a-ride problem. Computers and Operations Research 2010;37:1129–38.
[39] Parragh SN, Schmid V. Hybrid column generation and large neighborhood search for the dial-a-ride problem. Computers and Operations Research 2013;40:490–7.
[40] Petch RJ, Salhi S. A multi-phase constructive heuristic for the vehicle routing problem with multiple trips. Discrete Applied Mathematics 2003;133:69–92.
[41] Potvin J-Y. State-of-the-art review—evolutionary algorithms for vehicle routing. INFORMS Journal on Computing 2009;21:518–48.
[42] Potvin J-Y, Rousseau J-M. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. European Journal of Operational Research 1993;66:331–40.
[43] Reinhardt LB, Clausen T, Pisinger D. Synchronized dial-a-ride transportation of disabled passengers at airports. European Journal of Operational Research 2013;225:106–17.
[44] Rekiek B, Delchambre A, Saleh HA. Handicapped person transportation: an application of the grouping genetic algorithm. Engineering Applications of Artificial Intelligence 2006;19:511–20.
[45] Ritzinger U, Puchinger J, Hartl RF. Dynamic programming based metaheuristics for the dial-a-ride problem. Annals of Operations Research 2015:1–18. http://dx.doi.org/10.1007/s10479-014-1605-7 forthcoming.
[46] Ropke S, Cordeau J-F, Laporte G. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. Networks 2007;49:258–72.
[47] Ropke S, Pisinger D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science 2006;40:455–72.
[48] Salhi S, Petch R. A GA based heuristic for the vehicle routing problem with multiple trips. Journal of Mathematical Modelling and Algorithms 2007;6:591–613.
[49] Savelsbergh MWP. The vehicle routing problem with time windows: minimizing route duration. ORSA Journal on Computing 1992;4:146–54.
[50] Sen A, Bülbül K. A survey on multi trip vehicle routing problem. In: International logistics and supply chain congress. Istanbul, Türkiye; 2008. p. 401–7.
[51] Steiner MTA, Datta D, Steiner Neto PJ, Scarpin CT, Rui Figueira J. Multi-objective optimization in partitioning the healthcare system of Parana State in Brazil. Omega 2015;52:53–64.
[52] Sze S-N, Chiew K-L, Sze J-F. Multi-trip vehicle routing and scheduling problem with time window in real life. In: International conference of numerical analysis and applied mathematics, vol. 1479. AIP Publishing; 2012. p. 1151–4.
[53] Taillard ED, Laporte G, Gendreau M. Vehicle routing with multiple use of vehicles. The Journal of the Operational Research Society 1996;47:1065–70.
[54] Toth P, Vigo D. Heuristic algorithms for the handicapped persons transportation problem. Transportation Science 1997;31:60–71.
[55] Vidal T, Crainic TG, Gendreau M, Lahrichi N, Rei W. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. Operations Research 2012;60:611–24.
[56] Vidal T, Crainic TG, Gendreau M, Prins C. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. Computers and Operations Research 2013;40:475–89.
[57] Vidal T, Crainic TG, Gendreau M, Prins C. A unified solution framework for multi-attribute vehicle routing problems. European Journal of Operational Research 2014;234:658–73.
[58] Wolfler Calvo R, Colorni A. An effective and fast heuristic for the dial-a-ride problem. 4OR 2006;5:61–73.