Computers & Industrial Engineering 142 (2020) 106241

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

A learning-based memetic algorithm for the multiple vehicle pickup and delivery problem with LIFO loading

Bo Peng[a], Yuan Zhang[a], Zhipeng Lü[b,*], T.C.E. Cheng[c], Fred Glover[d]

[a] School of Business Administration, Southwestern University of Finance and Economics, Chengdu 610074, PR China
[b] SMART, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, PR China
[c] Department of Logistics and Maritime Studies, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong
[d] ECEE, College of Engineering & Applied Science, University of Colorado, Boulder, CO 80309, USA

ARTICLE INFO

Keywords:
Pickup and delivery problem
Routing
Traveling salesman
Memetic algorithms
Hybrid heuristics
Learning mechanisms

ABSTRACT

The multiple vehicle pickup and delivery problem is a generalization of the traveling salesman problem that has many important applications in supply chain logistics. One of the most prominent variants requires the route durations and the capacity of each vehicle to lie within given limits, while performing the loading and unloading operations by a last-in-first-out (LIFO) protocol. We propose a learning-based memetic algorithm to solve this problem that incorporates a hybrid initial solution construction method, a learning-based local search procedure, an effective component-based crossover operator utilizing the concept of structured combinations, and a longest-common-subsequence-based population updating strategy. Experimental results show that our approach is highly effective in terms of both computational efficiency and solution quality in comparison with the current state-of-the-art, improving the previous best-known results for 132 out of 158 problem instances, while matching the best-known results for all but three of the remaining instances.

1. Introduction

The travelling salesman problem with pickup and delivery (TSPPD) is a generalization of the well-known traveling salesman problem with many important applications (Azadian, Murat, & Chinnam, 2017; Pavone, 2013; Yu, Tang, Li, Sun, & Wang, 2016). TSPPD consists of determining a minimum cost circuit travelled by a vehicle to service several predefined requests to transport items from a specified pickup location to a specified delivery location. The vehicle starts from the depot and returns to it after all the requests have been serviced.

The TSPPD problem constitutes an instance of the widely studied vehicle routing problem that complicates the travelling salesman problem by introducing multiple vehicles and multiple routes, and embodies many variants by considering different constraints (such as vehicle capacities, time windows, and two or three-dimensional loading constraints (Mack & Bortfeldt, 2012). We first sketch a background of recent research in this field and then identify distinguishing characteristics of the TSPPD problem and its applications.

1.1. Brief background of recent vehicle routing research

Bortfeldt (2012) introduced an efficient hybrid heuristic to solve the capacitated vehicle routing problem by considering both vehicle capacity limits and three-dimensional loading constraints and employing the tabu search algorithm for routing together with a tree search algorithm for loading. Bortfeldt and Homberger (2013) proposed a two-stage heuristic method to solve the vehicle routing and loading problem by combining vehicle routing, possibly with time windows, and three-dimensional loading, as well as some packing constraints. The proposed two-stage heuristic is following a “packing first, routing second“ approach, i.e, the packing of goods and the routing of vehicles is carried out in two strictly separated stages. Wei, Zhang, and Lim (2014) developed an adaptive variable neighborhood search for the heterogeneous fleet vehicle routing problem with three-dimensional loading constraints (3L-HFVRP) by utilizing an extreme point based first fit heuristic to find a feasible loading pattern for each route, and designed two strategies to accelerate the loading and routing processes. Turky, Moser, and Aleti (2017) presented an iterated local search with guided perturbation for the 3L-HFVRP with time window constraints. Experimental results show that their proposed perturbation procedure significantly enhances the performance of the iterated local search algorithm on such constrained problems. Pace, Turky, Moser, and Aleti (2015) considered the 3L-HFVRP with capacity constraints and applied local search approach to test its performance based on data obtained

* Corresponding author.
E-mail addresses: pengbo@swufe.edu.cn (B. Peng), zhipeng.lv@hust.edu.cn (Z. Lü), Edwin.Cheng@polyu.edu.hk (T.C.E. Cheng).

https://doi.org/10.1016/j.cie.2019.106241
Received 31 May 2019; Received in revised form 17 December 2019; Accepted 19 December 2019
Available online 03 February 2020
0360-8352/

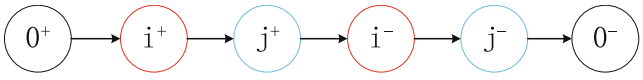
from their industry partner. Koch, Bortfeldt, and Wäscher (2018) considered vehicle routing problems with backhauls, time windows, simultaneous delivery and pickup and three dimensional loading constraints (3L-VRPSDPTW), proposing a hybrid algorithm consisting of an adaptive large neighborhood search for the routing and different packing heuristics for the loading problem. Reil, Bortfeldt, and Mönch (2018) considered the vehicle routing problems with backhauls, time windows and three dimensional loading constraints, studying different backhaul variants, including clustered backhauls, mixed linehauls and backhauls, and variants with simultaneous delivery and pickup and with divisible delivery and pickup. They employed another two-phase method to tackle this problem, in which the first phase of their method carried out the packing of goods for solving a 3D strip packing problem for each customer using tabu search. The second phase frist used a multi-start evolutionary strategy to minimize the number of vehicles and then a tabu search to minimize the total travel distance.

1.2. Distinguishing characteristics of the TSPPD problem and its applications

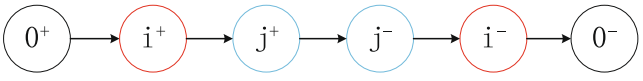
In the TSPPD problem, there exist two ways in which the loading and unloading operations corresponding to the pickup and delivery activities, respectively, are performed, namely first-in-first-out (FIFO) and last-in-first-out (LIFO), which correspond to two variants of TSPPD, called TSPPD with LIFO and TSPPD with FIFO. The FIFO policy implies that when a pickup node is visited, its corresponding item is loaded in a linear queue and an item can only be delivered if it is the first item of the queue, while the LIFO policy utilizes the mechanism of stack instead of queue, i.e., an item can be delivered if it is on the top of the stack. Fig. 1 depicts the two different policies, in which 0^+ and 0^- represent the depot at the beginning and end of the two routes, and i^+ and i^- represent the pickup and the delivery nodes for item i (and similarly for item j), where Fig. 1(a) and (b) show the FIFO and LIFO loadings, respectively.

In practice, TSPPD with FIFO exists in many real-life applications such as the dial-a-ride system where the major concern is fairness, i.e., the passengers (such as patients) picked up earlier must be dropped off earlier. Previous contributions to solve this problem include a branch-and-bound algorithm by Carrabs, Cerulli, and Cordeau (2007), a branch-and-cut algorithm by Cordeau, DellAmico, and Iori (2010) that can solve instances with up to 25 requests, and two effective heuristics based on probabilistic tabu search and iterated local search by Erdogan, Cordeau, and Laporte (2009). Recently, Lu, Benlic, and Wu (2018) proposed a multi-restart iterative search approach based on combined utilization of six move operators to tackle this problem.

On the other hand, TSPPD with LIFO likewise occurs in many applications, such as the transport of bulky, fragile, or hazardous items. Cordeau, Iori, Laporte, and Salazar González (2010) proposed a branch-and-cut algorithm that can solve instances with up to 17 requests, while Li, Lim, Oon, Qin, and Tu (2011) proposed a variable neighborhood search heuristic based on a tree representation to improve the previous results in the literature. In general, TSPPD and its variants have been extensively researched in the literature, where the recent studies include (Chami, Manier, & Manier, 2017; Furtado, Munari, & Morabito, 2017; Montero, Miranda-Bront, & Mndez-Daz, 2017; Naccache, Côté, & Coelho, 2018; Veenstra, Cherkesly, Desaulniers, & Laporte, 2017).



(a) FIFO loading.



(b) LIFO loading.

Fig. 1. The FIFO and LIFO loadings in the pickup and delivery problem.

In this paper we focus on the pickup and delivery problem under the LIFO policy, utilizing a general framework that can also be applied to address the problem under the FIFO policy. Specifically, we extend the TSPPD problem to involve multiple vehicles, enabling the single vehicle problem to be handled as a special case.

Over the past decades, several state-of-the-art algorithms have been proposed for solving TSPPD with the multiple vehicle extension. Kachitvichyanukul, Sombuntham, and Kunnapapdeelert (2015) proposed two new solution representations, SD2 and SD3 for PSO algorithm which they used in conjunction with a variant of PSO involving multiple social learning terms in application to the generalized multi-depot vehicle routing problem with multiple pickup and delivery requests. Cherkesly, Desaulniers, and Laporte (2015) proposed a population-based metaheuristic to address the multiple vehicle pickup and delivery problem with LIFO loading and time windows, called the MPDPL with time windows. The authors combined local search with a genetic algorithm to produce high-quality solutions within reasonable computing times. Cheang, Gao, Lim, Qin, and Zhu (2012) considered the case where the route length of each vehicle cannot exceed a maximum limit and the vehicles have unlimited capacity, called MPDPL with distance constraints, abbreviated as PDPLD. They proposed a two-stage approach for solving the problem to minimize the total distance and the number of vehicles, employing simulated annealing and ejection pool in the first stage, and variable neighborhood search and probabilistic tabu search in the second stage. Benavent, Landete, Mota, and Tirado (2015) addressed MPDPL with distance constraints (PDPLD) as a special case of MPDPL with maximum time (which is called the pickup and delivery problem with limited time, abbreviated as PDPLT), observing that minimizing the total distance is equivalent to minimizing the total time and that minimizing the number of vehicles as the primary objective can be addressed by adding a large number to the travel times of the arcs leaving the depot. However, the exact method proposed by Benavent et al. (2015) can only solve instances with up to 60 nodes, while their proposed tabu search can solve larger instances with up to 400 nodes.

This difference between the exact and metaheuristic methods motivates us to employ a metaheuristic approach to tackle large-size instances of the PDPLT problem. The main contributions of our study are as follows:

- A learning-based memetic algorithm (LMA) for solving PDPLT, which introduces a hybrid initial solution construction method by incorporating the splitting approach and the Lin-Kernighan heuristic (LKH) for the asymmetric travelling salesman problem (ATSP), a subproblem of PDPLT, to generate a random initial population with high quality.
- A reward and punishment mechanism inspired by reinforcement learning to manage the multiple neighborhood moves and guide the search.
- A component-based crossover operator and a longest-common-subsequence-based (LCS-based) population updating strategy to obtain a better trade-off between intensification and diversification of the search.
- Our experimental results demonstrate that the performance of our LMA is highly effective compared to state-of-the-art approaches in the literature by improving the previous best-known results for 131 out of 158 problem instances (including both PDPLD and PDPLT instances), while matching the best-known results for all but three of the remaining instances.

We organize the rest of the paper as follows: Section 2 introduces the PDPLT problem and Section 3 presents the proposed memetic algorithm for solving PDPLT in detail. Section 4 presents and discusses the proper setting of the key parameters and examine the performance

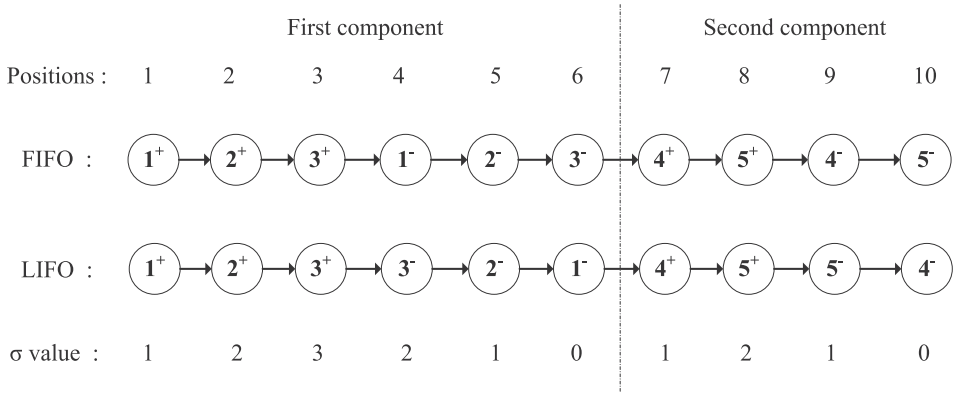


Fig. 2. Example of components.

of the proposed algorithm against the current best performing algorithms for both PDPLT and PDPLD. In Section 5 we analyze the main strategic components of our algorithm. Finally, we conclude the paper and suggest topics for future research in Section 6.

2. Problem description and definitions

2.1. Problem description

In PDPLT, we are given a set $N = \{1, \cdots, n\}$ of n requests, each of which concerns the transport of an item with a load from pickup vertex i^+ to delivery vertex i^- $(1 \le i \le n)$. There are several vehicles with limited capacity that starts from a depot vertex 0^+ and return to a depot vertex 0^- with the objective of minimizing the total travel time incurred by all the vehicles. The vehicles must fulfill all the requests by visiting each pickup vertex to pick up the indicated load and travel to the corresponding delivery vertex to deliver the load in accordance with the LIFO policy. Specifically, PDPLT is defined on a complete weighted undirected graph $G = (V, E)$ with the following features.

- $V = P \cup D \cup O$ denotes the set of nodes, where $P = \{1^+, 2^+, ..., n^+\}$ denotes the set of pickup nodes, $D = \{1^-, 2^-, ..., n^-\}$ is the set of delivery nodes, O denotes the set of starting and ending nodes $\{0^+, 0^-\}$, also called depots, and $E = \{(u, v): u, v \in V, u \neq v\}$ is the edge set.
- Each item must be picked at $i^+ \in P$ and delivered at $i^- \in D$, where the load of the item is denoted by d_i.
- The service time at each pickup node or delivery node $u \in P \cup D$ is denoted by st_u, and the travel time to traverse the arc $(u, v) \in E$ is denoted by $tt_{u,v}$.
- The maximum capacity of each vehicle is .
- The maximum duration of each route including the service time and traversal time is MD.

Let R be one route in a solution S and $R = \{u_0 = 0^+, u_1, u_2, \cdots, u_m = 0^-\}$, where u_k is the kth node visited in R $(1 \le k \le m)$. If the visited node is a pickup node (i.e., $u_k \in P$), the corresponding load capacity of the vehicle after visiting it is $l(u_k) = l(u_{k-1}) + d_{u_k}$, while the corresponding load of the vehicle after visiting it is equal to $l(u_k) = l(u_{k-1}) - d_{u_k}$ if the visited node is a pickup node (i.e., $u_k \in D$). For each node in the route, the corresponding load cannot exceed the given maximum capacity, i.e., $l(u_k) \le MC$. We denote by $DT(R) = \sum_{k=0}^{m-1} tt_{u_k,u_{k+1}}$ the total traversal time and $ST(R) = \sum_{k=1}^{m-1} st_k$ the total service time. The corresponding duration of each route including the traversal time and service time cannot exceed the given maximum duration, i.e., $DT(R) + ST(R) \le MD$. In addition, the LIFO policy is followed for both pickup and delivery operations. A feasible solution to this problem is a set of vehicle routes that satisfy three constraints, i.e., the maximum capacity, maximum duration, and LIFO constraints. The objective is to find a feasible solution with the minimum total travel time as follows:

$$Minimizef(S) = \sum_{i=1}^{|S|} DT(R_i) + ST(R_i), \tag{1}$$

We refer the reader to benavent2015multiple for more details of the mathematical formulation of the problem.

2.2. Definitions

A pair consisting of the pickup vertex i^+ and the corresponding delivery vertex i^- is called a *couple*, i.e., request. A *component* is a set of vertices, before the beginning and at the end of which there are no requests being transported by the vehicle. Erdogan et al. (2009) first defined the concept of *component* for FIFO and we extend this concept for both FIFO and LIFO policies in this study. In particular, we introduce a term $\sigma(k)$ to denote the number of uncompleted requests when the kth vertex is visited in a route. If the value of $\sigma(k)$ for the pickup vertex k is equal to 1, then k is the beginning of its component. If the $\sigma(k)$ for the pickup vertex k is equal to 0, then k is the end of its component. Fig. 2 is an example in which there are two components in the route. The vertices in positions 1 and 7 identify the beginnings of the two components as their σ values are equal to 1, i.e., there are no requests transported by the vehicle before visiting these two nodes. The delivery vertices in positions 6 and 10 respectively correspond to the ends of the components as their σ values are equal to 0. There are no requests for the vehicle after serving these two vertices. Hence, the paths from positions 1 to 6 and positions 7 to 10 denote two different components.

3. Memetic algorithm

3.1. Main framework

A memetic algorithm is a general-purpose metaheuristic approach that typically combines a local search optimization procedure with a population-based framework, which has been successfully applied to tackle many classical combinatorial optimization problems, including the quadratic assignment problem (Benlic & Hao, 2015), which provides a different generalization of the traveling salesman problem. The purpose of combining local search and population-based strategies is to take advantage of both the crossover operator as a diversification mechanism for discovering promising unexplored regions of the search space and the local optimization as an intensification procedure to obtain high-quality solutions within a search region. We outline our proposed memetic algorithm for PDPLT in Algorithm 1. At the beginning of the algorithm, we iteratively employ a hybrid heuristic method to generate the initial population (line 1). Following this, we employ a learning-based local search to optimize the solutions in the population

(lines 2–4). Later, we iteratively combine two parent solutions randomly selected from the population to generate offspring solutions using a component-based crossover operator under the LIFO policy until the stopping criterion, i.e., maximum computing time, is satisfied (lines 5–7). After each use of the crossover operator, we improve the generated offspring solution using a learning-based local search to guide the search to promising regions (line 8). During this process, S^* records the best solution found so far (lines 9–11). We then apply the longest-common-sequence-based (LCS-based) population updating strategy to possibly replace the worst individual in the population with the improved offspring solution (lines 12–15).

Algorithm 1. Framework of the memetic algorithm for solving PDPLT

Require: Benchmark instance (B); the maximum computing time (T_{max})
Ensure: Best-found solution (S^*)
/* Generate np feasible solutions as an initial population (Section 3.2) */
1: $P_c = \{S_1, \cdots, S_{np}\} \leftarrow Hybird\_inital\_solution(B)$
/* Improve each individual S_i in the population with a learning-based local search (Section 3.3) */
2: **for** $i = 1, \cdots, np$ **do**
3: $S_i \leftarrow Learning\_based\_localsearch(S_i)$
4: **end for**
5: **while** The maximum computing time T_{max} is not reached **do**
6: Randomly select parent solutions S_i and S_j from P where $1 \le i, j \le np$ and $i \ne j$
/* Generate offspring S_c from S_i and S_j (Section 3.4) */
7: $S_c \leftarrow S_i \oplus S_j = Component\_based\_crossover(S_i, S_j)$
/* Improve S_c with a learning-based local search (Section 3.3) */
8: $S_c \leftarrow Learning\_based\_localsearch(S_c)$
9: S_c is better than S^*
10: $S^* \leftarrow S_c$
11: **end if**
/* The longest-common-subsequence based population updating strategy (Section 3.5) */
12: Determine the worst individual S_w where the goodness value $GS(S_w, P_c) = \min\{GS(S_k, P_c)\}$, $1 \le k \le np$ (see Eq. (7))
13: **if** $GS(S_c, P_c \cup S_c) > GS(S_w, P_c \cup S_c)$ **then**
14: $P_c \leftarrow P_c \cup S_c \setminus S_w$
15: end if
16: **end while**
17: **return** (S^*)

3.2. Hybrid initial solutions

We construct the initial solutions by iteratively using a hybrid initial procedure based on a splitting approach that is able to obtain high-quality initial solutions within short computing time. A similar hybrid initial procedure has been successfully employed to tackle various vehicle routing problems (VRPs), e.g., multi-depot VRP (Escobar, Linfati, Toth, & Baldoquin, 2014) and multi-route VRP (Azi, Gendreau, & Potvin, 2014). In order to generate high-quality initial solutions, we first adapt the splitting mechanism for our problem by employing the Lin-Kernighan heuristic (LKH) for the ATSP subproblem in PDPLT to improve the solution quality of the initial solutions. The steps of the construction procedure are presented in Algorithm 2 and can be summarized in the following steps:

Algorithm 2. Hybrid initial solution

Require: Benchmark instance (B)
Ensure: The initial solution (S_0)
/* Step 1: Generate a set of components by randomly setting k couples as one component, with satisfying the maximum capacity constraint*/
1: t ← 0
2: **while** The request set N is not empty, i.e., $N \ne \emptyset$ **do**
3: Construct each component by randomly selecting k requests (i.e., $i_1, \cdots, i_k$) from the request set N, by satisfying the maximum capacity constraint, i.e., $\sum_{s=1}^{k} d_{is} <= MC$
4: $N \leftarrow N \setminus \{i_1, \cdots, i_k\}$
5: t ← t + 1
6: $C_t \leftarrow \{i_1^+, \cdots, i_k^+, i_k^-, \cdots, i_1^-\}$
7: **end while**
/* Step 2: Construct a composite ATSP tour containing all the components generated in Step 1 by employing the well-known LKH as implemented */
8: S ← LKH_ATSP($(C_1, C_2, \cdots, C_t)$) /* t denotes the number of components generated in Step 1 */
/* Step 3: Split the composite ATSP tour into several routes with satisfying the maximum duration constraint */
9: $R_m \leftarrow 0$, $m \leftarrow 1$
10: **for** $i = 1, \cdots, t$ **do**
11: **if** DT($R_m \oplus C_i$) + ST($R_m \oplus C_i$) ≤ MD **then**
12: $R_m \leftarrow R_m \oplus C_i$
13: **else**
14: m ← m + 1
15: $R_m \leftarrow C_i$
16: **end if**
17: **end for**
/* Step 4: Use LKH to optimize each route */
18: **for** $i = 1, \cdots, m$ **do**
19: LKH_ATSP(R_i)
20: **end for**
21: $S_0 \leftarrow (R_1, \cdots, R_m)$
22: **return** (S_0)

- Step 1. Generate a set of components C by randomly setting k ($1 \le k \le 3$) couples $\{i_1^+, \cdots, i_k^+, i_k^-, \cdots, i_1^-\}$ as one component with satisfying both the LIFO constraint and maximum capacity constraint (see lines 1–6 in Algorithm 2). For example, in Fig. 3, there exist six components, i.e., C1-C6.
- Step 2. Construct a composite ATSP tour by employing the well-known LKH procedure (Helsgaun, 2000) to optimize all the components ($C_1, C_2, \cdots, C_t$) generated in Step 1 (line 8). The reason why we deploy the ATSP model to obtain high-quality route stems from the fact that the traveling time from component C1 to component C2 is not necessarily equal to the traveling time from C2 to C1.
- Step 3. Split the composite ATSP tour into several routes, by iteratively inserting each component C into the current route R if the current route satisfies the maximum duration constraint. Otherwise,

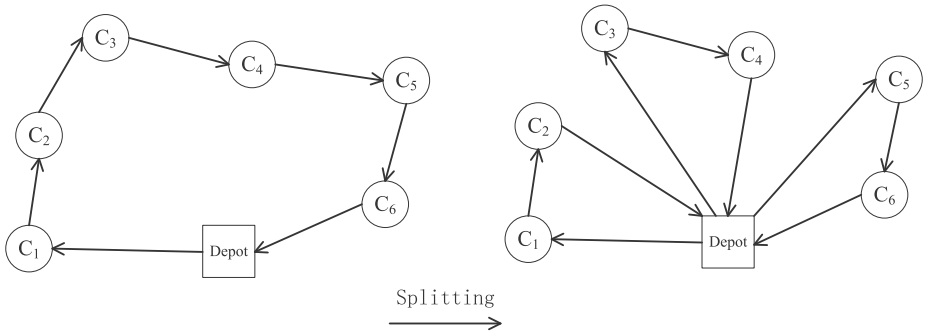


Fig. 3. Illustration of the construction mechanism for initial solutions.

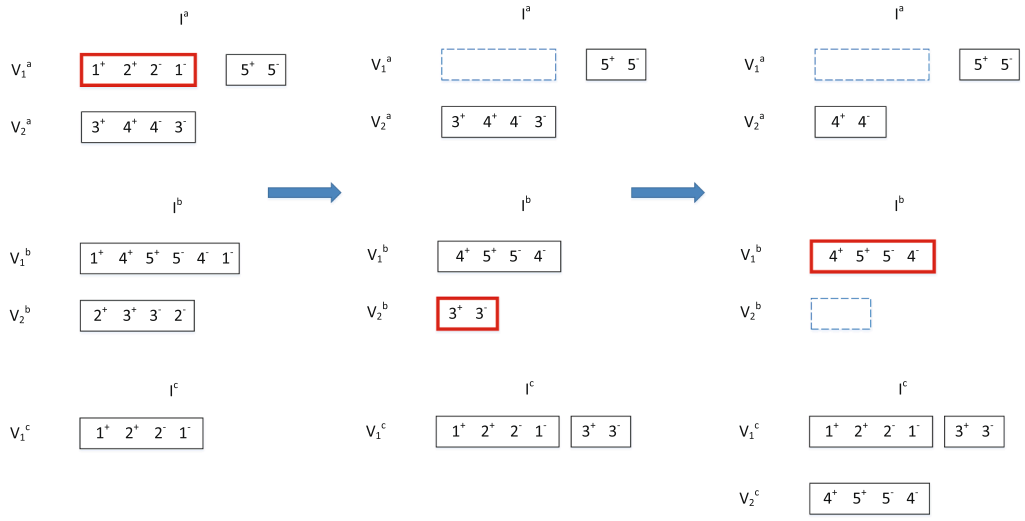


Fig. 4. Illustration of the crossover operator with respect to the LIFO policy.

Table 1
Settings of some important parameters for LMA.

Parameter	Description	Candidate values	Final value
α	The reaction factor that controls how quickly the score adjustment function reacts to changes according to the performance of the moves	0.1, 0.2, 0.3	0.1
β_1	The reward parameter if a move produces a new best solution	1, 5, 10	5
β_2	The reward parameter if a move improves the current solution	1, 2, 5	1
γ	The punishment parameter if a generated solution is worse than the current solution	0.8, 0.9, 0.99	0.9
ζ	The parameter of the ratio of the number of request nodes deleted in the perturbation strategy (n/ζ)	2,4,6	4
ε	The threshold used to employ the dedicated perturbation ($n*\varepsilon$)	1, 5, 10	5
np	The number of individuals in the population	10, 20, 30	10
δ	The constant parameter to balance the objective value and the distance in the goodness value	0.5, 0.7, 0.9	0.7

we insert it into a new route (lines 9–17). For example, in Fig. 3, the route of the whole ATSP tour is split into three routes to satisfy the maximum duration constraint.

- Step 4. For each route R, we obtain an ATSP tour by using LKH to minimize the total travelling cost for visiting the customers belonging to the route (lines 18–21). Finally, the generated solution S_0 is returned as the initial solution (line 22).

The hybrid initial solution construction procedure proposed above is usually able to generate feasible solutions that satisfy the three constraints, i.e., the maximum duration, maximum capacity, and LIFO constraints. In addition, high-quality solutions are generated within reasonable computing times. In Section 5.1 we demonstrate the efficacy of our procedure in comparison with other constructive methods proposed in the literature.

3.3. Learning-based local search

One of the key components of our memetic algorithm is the learning-based local search procedure that plays the critical role of intensifying the search. With the exception of tabu search, traditional local search utilizes a set of moves to search the solution regions without maintaining a memory of the process, while the local search based on our reinforcement learning mechanism is able to effectively exploit memory to manage the neighborhood moves and guide the search to promising regions.

3.3.1. Neighborhood moves

Our algorithm employs both intra-route moves (performed in the same route) and inter-route moves (performed between two different routes), as follows:

- Request-Insertion (M_1): A request (e.g., $\{i^+, i^-\}$) is removed from its current route and inserted either in a different position of the same route or in a different route.
- Request-Swap (M_2): Two requests (in the same route or in different routes) exchange their positions.
- Double-Request-Swap (M_3): Two pairs of consecutive requests exchange their positions. This operator extends the Request-Swap move by considering consecutive requests i and j defined to be consecutive if the pickup node j^+ of request j is the closest to pickup node i^+ of request i in the route.
- Component-Insertion (M_4): A component is removed from its current route and inserted either in a different position of the same route or in a different route.
- Component-Swap (M_5): Two components (in the same route or in different routes) exchange their positions.
- Double-Component-Swap (M_6): Two pairs of consecutive components exchange their positions. This operator extends the Component-Swap move by considering two pairs of consecutive components.

In our local search, we only consider neighborhood moves that can satisfy all the three constraints, i.e., the maximum capacity, maximum duration, and LIFO constraints. Obviously, in cases where the constraints are tight, it is very difficult to find feasible solutions in a single move as confirmed in benavent2015multiple. Hence, the main difference of our approach from that proposed in Benavent et al. (2015), where only the Request-Insertion move is used, is that we employ six different neighborhood moves to expand the search space in order to find more promising feasible solutions, instead of relaxing the maximum duration constraint and penalizing violations.

The size and complexity of the neighborhood structure significantly

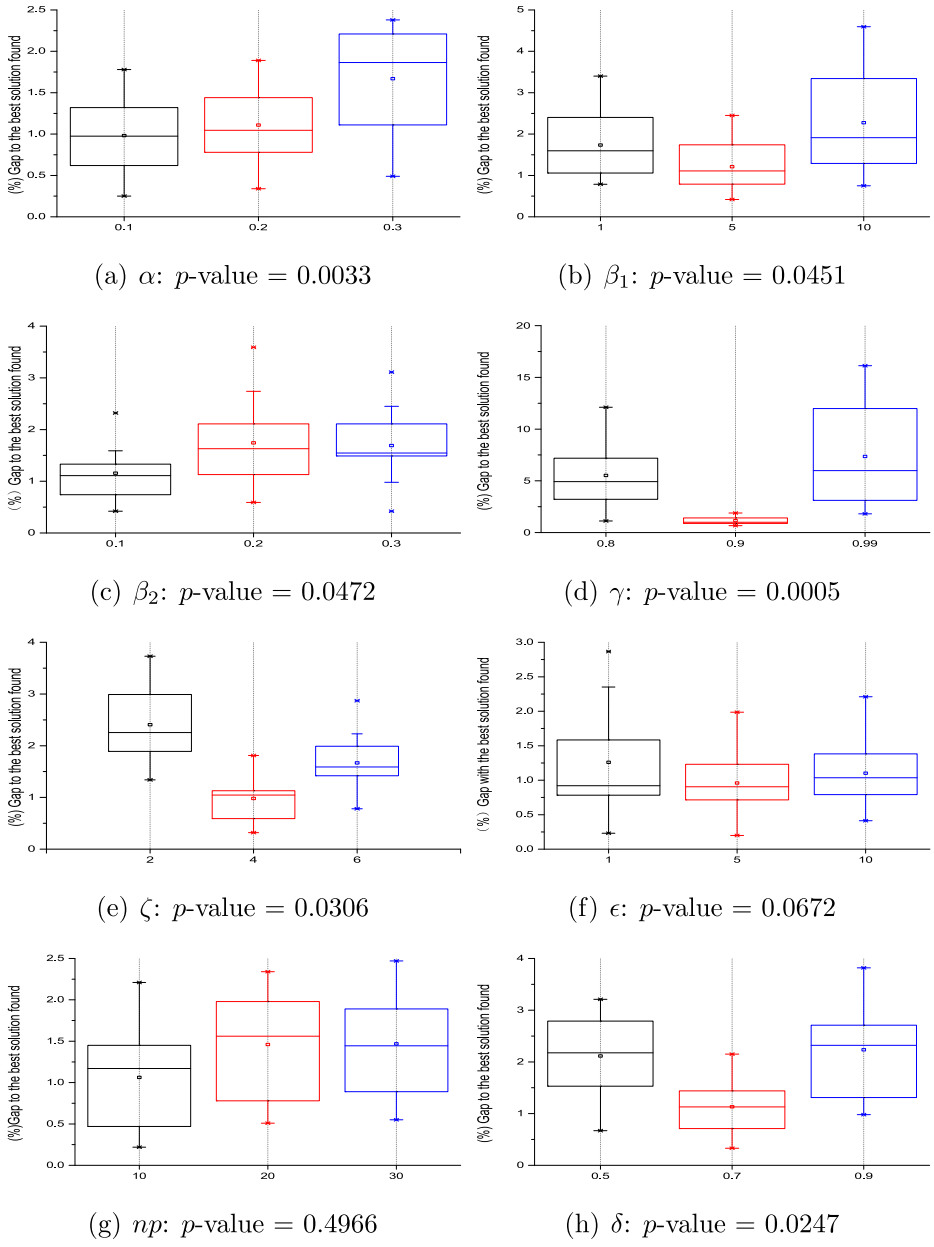


Fig. 5. Box-and-whisker plots of the gaps between the objective values and the best found solutions for the considered parameters.

affect the performance of the algorithm. The request-insertion neighborhood operator (M_1) has n candidate requests to be removed, and $n_k \times (2n_k - 1)$ possible positions for each candidate pickup and its corresponding delivery node, where n_k denotes the number of vehicles in route k. Hence the size of M_1 neighborhood is $n \times \sum_{k=1}^{k=K} (n_k \times (2n_k - 1))$, where K denotes the number of routes. For request-swap neighborhood operator (M_2), the size of the neighborhood is the same as the number of two candidate requests combinations, which is equal to $n \times (n - 1)/2$. For double-request-swap neighborhood operator (M_3), the size of the neighborhood is $(n - 2) \times (n - 3)/2$. As for the component based neighborhood operators (i.e., the component-insertion operator (M_4), component-swap (M_5) and double-component-swap (M_6)n), the size of each neighborhood is no more than its corresponding request-based neighborhood operators, since a component usually consists of several requests. Therefore, the complexity of all the six neighborhood operators employed in our method is bounded by $O(n^3)$.

3.3.2. Learning mechanism

Reinforcement learning is an area of machine learning concerned with how an agent should take actions in an environment so as to maximize cumulative reward. The intuition underlying reinforcement learning is that actions that lead to large rewards should be made more likely to recur.

We employ a reward and penalty strategy to dynamically manage the neighborhood moves and guide the search based on the expectation that different neighborhood moves may be preferable for different problem instances or search landscapes. Consequently, we keep track of a score for each neighborhood move, which measures how well the move has performed for the current instance or landscape, adopting the perspective that alternating among different moves based on the proposed learning mechanism may yield more robust performance.

To select moves, we assign scores to different moves and use the roulette wheel selection principle. If we have n moves with scores sc_i ($i \in 1, 2, ...,n$), move k is selected with probability λ_k, where

Table 2
Results for solving public benchmark PDPLT instances with 100–110 nodes.

	MS-ITS			LMA					MS-ITS			LMA			
Instances	f_{best}	Veh	Time	f_{best}	Veh	f_{avg}	Time	Instances	f_{best}	Veh	Time	f_{best}	Veh	f_{avg}	Time
lc101	9867.61	10	65.91	**9867.29**	9	9868.01	16.31	lc201	9825.64	3	125.00	**9824.76**	3	9825.73	28.75
lc102	9862.70	9	59.19	9862.70	9	9862.70	27.87	lc202	9860.47	3	115.66	**9860.24**	3	9861.17	16.07
lc103	9867.84	9	56.47	9867.84	9	9867.98	10.69	lc203	9873.71	3	107.53	9873.71	3	9874.13	17.30
lc104	9857.95	10	62.22	**9857.80**	9	9857.95	41.32	lc204	9832.31	3	121.44	**9830.38**	3	9831.28	16.94
lc105	9848.78	9	54.31	**9848.01**	9	9848.83	4.94	lc205	9794.84	3	103.66	9794.84	3	9794.84	15.84
lc106	9866.18	9	60.28	9866.18	9	9866.18	10.71	lc206	9882.15	3	116.55	9882.15	3	9882.15	17.49
lc107	9874.48	9	62.25	9874.48	9	9876.24	33.08	lc207	9802.90	3	98.61	9802.90	3	9802.90	16.25
lc108	9862.42	9	60.89	9862.42	9	9863.11	48.03	lc208	9803.00	3	99.86	9803.00	3	9803.00	15.31
lc109	9855.31	9	52.73	9855.31	9	9855.32	22.76								
lr101	2140.15	11	61.36	**2110.53**	10	2120.47	33.78	lr201	1967.24	3	188.69	**1948.63**	2	1955.15	16.49
lr102	2125.61	10	62.24	**2114.12**	10	2126.19	13.96	lr202	2130.35	3	188.00	**2124.35**	3	2129.76	10.86
lr103	2113.11	10	49.53	**2111.05**	10	2113.66	38.62	lr203	2102.09	3	208.83	**2100.92**	3	2101.99	18.68
lr104	2072.80	10	47.72	**2064.99**	10	2076.81	46.35	lr204	2184.12	3	189.06	**2179.69**	3	2185.68	12.80
lr105	2114.97	10	51.80	**2109.91**	10	2110.01	11.52	lr205	2079.14	3	202.53	**2065.66**	3	2085.43	23.56
lr106	2147.80	10	58.52	**2126.12**	10	2139.85	38.58	lr206	2111.64	3	168.78	**2110.17**	3	2111.32	19.20
lr107	2188.71	11	52.13	**2173.23**	10	2193.84	11.41	lr207	2146.86	3	188.17	**2144.38**	3	2148.59	75.53
lr108	2150.33	10	45.59	**2148.11**	10	2151.92	39.51	lr208	2090.03	3	170.94	**2082.15**	3	2093.70	16.28
lr109	2165.48	11	60.38	**2154.84**	10	2163.55	45.93	lr209	2066.29	3	211.49	**2045.65**	3	2058.91	21.55
lr110	2041.50	10	54.05	2041.50	10	2042.16	42.09	lr210	2073.73	3	193.95	**2068.44**	3	2070.08	97.72
lr111	2114.81	10	60.55	**2110.61**	10	2115.86	21.61	lr211	2027.38	3	216.20	**2027.38**	3	2027.98	16.87
lr112	2102.96	10	53.89	**2098.09**	10	2100.47	19.24								
lrc101	2289.98	10	50.88	**2265.37**	10	2287.39	18.55	lrc201	2203.00	3	161.02	**2168.34**	3	2187.96	17.20
lrc102	2336.21	10	48.58	**2331.00**	10	2340.18	14.82	lrc202	2250.50	3	158.69	**2204.66**	3	2235.81	38.47
lrc103	2213.32	11	61.28	**2195.80**	10	2207.54	26.77	lrc203	2192.66	3	157.36	**2174.51**	3	2189.39	50.17
lrc104	2191.28	10	57.22	**2190.22**	10	2198.88	35.40	lrc204	2018.41	3	158.92	2018.41	3	2019.13	16.05
lrc105	2335.36	11	60.52	**2305.44**	10	2321.17	25.32	lrc205	2261.11	3	171.17	**2225.98**	3	2249.28	18.26
lrc106	2238.28	10	57.77	**2226.83**	10	2237.95	17.59	lrc206	2281.59	3	145.84	**2263.27**	3	2279.38	10.88
lrc107	2218.77	10	50.31	**2204.83**	10	2220.36	43.51	lrc207	2426.00	4	169.05	**2366.94**	3	2388.16	14.88
lrc108	2201.42	10	51.73	**2201.10**	10	2202.11	19.21	lrc208	2176.96	3	154.55	**2104.36**	3	2149.66	18.86
avg	4560.90	9.93	56.22	**4553.30**	9.69	4559.89	26.88		4424.60	3.04	158.95	**4410.96**	2.96	4420.09	24.38
#better				22								21			
#equal				7								6			
#worse				0								0			

Table 3
Results for solving public benchmark PDPLT instances with 402–422 nodes.

	MS-ITS			LMA					MS-ITS			LMA			
Instances	f_{best}	Veh	Time	f_{best}	Veh	f_{avg}	Time	Instances	Best	Veh	Time	f_{best}	Veh	f_{avg}	Time
LC1_4_1	42336.75	30	405.08	**42237.43**	30	42340.19	307.99	LC2_4_1	40714.04	12	732.05	**40670.71**	12	40691.66	210.03
LC1_4_2	42366.74	30	382.77	**42346.62**	30	42410.56	357.37	LC2_4_2	41140.66	14	737.39	**40866.52**	12	40919.77	394.92
LC1_4_3	42599.50	30	402.63	**42569.57**	30	42578.89	386.35	LC2_4_3	41316.85	13	741.09	**41014.62**	12	41210.58	278.97
LC1_4_4	42482.09	30	417.38	**42402.31**	30	42456.06	355.22	LC2_4_4	41035.84	13	764.05	**40870.53**	12	40981.32	324.30
LC1_4_5	42312.18	30	395.00	**42213.65**	30	42289.31	358.27	LC2_4_5	40996.57	13	772.70	**40735.31**	12	40810.29	297.03
LC1_4_6	42284.78	29	437.97	**42271.70**	30	42299.72	388.61	LC2_4_6	40727.48	12	744.81	**40594.96**	12	40661.76	223.93
LC1_4_7	42257.38	30	391.61	**42163.86**	30	42231.43	303.67	LC2_4_7	41001.45	13	752.27	**40687.18**	12	40883.25	385.95
LC1_4_8	42409.92	30	400.83	**42381.73**	30	42399.15	385.35	LC2_4_8	40925.04	13	737.17	**40667.13**	12	40760.51	247.25
LC1_4_9	42657.31	30	377.06	**42645.50**	30	42660.04	371.78	LC2_4_9	40810.30	13	788.56	**40745.74**	12	40781.96	275.21
LC1_4_10	42539.79	30	406.77	**42524.90**	30	42539.11	376.14	LC2_4_10	41060.15	14	735.99	**40873.01**	13	40987.03	366.40
LR1_4_1	10502.64	18	840.63	**9851.86**	14	9917.68	274.60	LR2_4_1	12095.03	7	1598.11	**11186.32**	4	11199.84	231.22
LR1_4_2	10624.45	18	827.59	**9845.99**	14	9941.57	272.03	LR2_4_2	12087.73	10	1528.31	**11126.38**	4	11564.79	351.06
LR1_4_3	10515.92	18	805.61	**9784.51**	14	9856.29	284.36	LR2_4_3	11890.15	7	1578.91	**10876.17**	4	11239.41	314.20
LR1_4_4	9910.92	17	850.89	**9147.95**	14	9403.33	399.28	LR2_4_4	11445.21	10	1612.69	**10371.95**	4	10685.31	296.48
LR1_4_5	10190.46	16	762.41	**9602.53**	13	9820.47	374.79	LR2_4_5	11753.71	6	1622.72	**11076.18**	6	11542.49	399.52
LR1_4_6	10414.40	16	829.00	**9952.60**	14	10231.68	396.33	LR2_4_6	11549.06	6	1540.38	**10666.13**	4	10869.72	155.28
LR1_4_7	10321.30	20	851.63	**9495.14**	13	9853.49	298.65	LR2_4_7	11454.11	8	1658.30	**10383.31**	4	10534.97	398.30
LR1_4_8	9290.86	15	856.17	**8922.60**	12	9127.58	393.68	LR2_4_8	11684.98	6	1472.44	**10688.82**	5	10898.28	162.62
LR1_4_9	10447.86	18	827.13	**9801.38**	13	9910.65	373.73	LR2_4_9	11695.28	9	1574.20	**10786.47**	4	11349.26	136.39
LR1_4_10	10041.44	17	848.27	**9417.07**	13	9823.10	316.42	LR2_4_10	12228.19	8	1589.20	**11064.19**	4	11901.38	159.49
LRC1_4_1	9700.01	17	818.39	**9178.79**	14	9320.28	389.15	LRC2_4_1	10733.52	6	1617.91	**9656.87**	4	9810.79	349.17
LRC1_4_2	9651.76	16	838.09	**8871.73**	13	8913.31	**305.77**	LRC2_4_2	10444.30	5	1670.25	**9515.38**	4	9882.12	277.20
LRC1_4_3	9667.97	16	759.05	**9220.59**	14	9513.77	**389.64**	LRC2_4_3	10704.17	7	1577.23	**9673.14**	4	10106.65	361.75
LRC1_4_4	9015.70	14	772.53	**8823.12**	13	8948.09	**399.86**	LRC2_4_4	10857.97	12	1489.13	**9477.39**	5	9811.89	368.64
LRC1_4_5	9555.94	16	791.08	**8879.86**	13	9135.54	**292.87**	LRC2_4_5	10288.07	7	1612.36	**9484.97**	4	9879.17	256.31
LRC1_4_6	9363.88	17	814.67	**9023.08**	14	9136.72	**365.37**	LRC2_4_6	10596.35	10	1638.67	**9275.77**	4	9513.30	292.73
LRC1_4_7	9645.08	18	870.91	**8828.10**	13	9132.86	**362.44**	LRC2_4_7	11160.72	7	1551.36	**9861.02**	4	10391.22	385.31

(*continued on next page*)

Table 3 (*continued*)

	MS-ITS			LMA					MS-ITS			LMA			
Instances	f_{best}	Veh	Time	f_{best}	Veh	f_{avg}	Time	Instances	Best	Veh	Time	f_{best}	Veh	f_{avg}	Time
LRC1_4_8	9528.92	16	829.53	**8855.54**	13	9111.02	**386.94**	LRC2_4_8	10882.58	11	1478.17	**9441.73**	4	9689.08	358.19
LRC1_4_9	9634.73	17	837.19	**8949.42**	13	9095.33	**189.92**	LRC2_4_9	11054.00	7	1578.86	**9668.99**	4	9910.38	206.83
LRC1_4_10	9639.88	16	838.55	**8975.52**	13	9110.15	**398.25**	LRC2_4_10	10855.83	7	1590.75	**9786.51**	4	10213.11	389.08
avg	20730.35	21.17	682.89	**20306.15**	18.90	20450.25	348.83		21172.98	9.53	1302.87	**20393.11**	6.83	20656.04	295.13
#better				30								30			
#equal				0								0			
#worse				0								0			

$$\lambda_k = \frac{sc_k}{\sum_{i=1}^{n} sc_i},\ k = 1, 2, ...,n. \tag{2}$$

At the beginning of the search, each neighborhood move has the same score sc_0 and hence the same probability of being chosen. After each iteration j, the score of the neighborhood used is updated as follows:

$$sc_{i,j+1} = sc_{i,j} + \alpha * \beta_l,\ i, j = 1, 2, ...,n;\ l = 1, 2. \tag{3}$$

where the reaction factor α controls how quickly the score adjustment function reacts to changes according to the performance of the moves, and parameter β denotes the different incremental scores according to the following several situations. If one move can produce a new best solution, we reward this neighborhood move by choosing β_1 in Eq. (3). If one move can generate a better solution than the current solution, the neighborhood move would still be rewarded β_2. However, if the generated solution is worse than the current solution, then we punish the move by multiplying the score by γ as follows:

$$sc_{i,j+1} = \gamma * sc_{i,j},\ i, j = 1, 2, ...,n. \tag{4}$$

The learning-based local search phase proceeds with iterative exploitation of the six neighborhood moves as shown in Algorithm 3. In each iteration, one neighborhood move is picked with probability λ_i (lines 3–4). Then, if the neighborhood solution S' obtained by this neighborhood move cannot improve the current solution S, the next neighborhood move is chosen from those remaining; otherwise, the current solution S is replaced by the best neighborhood solution S' generated by current neighborhood move (lines 5–10). Subsequently, the score of the neighborhood move N_i is updated by Eqs. (3) and (4) (line 11). During this process, S^b records the best solution found in the local search, S^* preserves the best found solution so far, and *no_improv_iter* denotes the number of iterations without improving the best found solution S^b (lines 12–16). When none of the moves can improve the current best solution, we apply a simple perturbation strategy to achieve a better trade-off between diversification and intensification of the search (lines 17–19).

To apply the perturbation operator, we randomly delete part of the request nodes (n/ζ) from the current solution and re-insert the deleted request nodes into the partial solution based on a greedy strategy.

Algorithm 3. learning-based local search

Require: Initial current solution(S);
Ensure: Best found solution(S^b) during the search
/* The set of neighborhood moves denoted by I including the six neighborhood moves proposed in Sec. 3.3.1*/
1: $I = \{M_1, M_2, \cdots, M_6\}$, $I_0 \leftarrow I$, $S^b \leftarrow S$, *no_improv_iter* $\leftarrow 0$
2: while *no_improv_iter* $< n * \varepsilon$ **do**
3: Calculate the probability λ_i of each neighborhood move M_i by Eq. (2)
4: Randomly select one neighborhood move M_i from I with probability λ_i, where $1 \le i \le |I_0|$
5: Choose the best neighboring solution S' from the set of neighboring solutions generated by M_i move, (i.e., $S' \leftarrow S \oplus M_i$)
6: if S' is not better than S **then**
7: $I \leftarrow I \backslash \{M_i\}$
8: **else**
9: $S \leftarrow S'$, $I \leftarrow I_0$
10: **end if**
11: Update the score of the neighborhood move M_i by equations (3) and (4)
12: **if** S' is better than S^* (or S^b) **then**
13: S^*(or S^b) $\leftarrow S'$, *no_improv_iter* $\leftarrow 0$
14: **else**
15: *no_improv_iter* $\leftarrow$ *no_improv_iter* + 1
16: **end if**
17: **if** $|I| = 0$ **then**
18: $S \leftarrow$Perturbation(S^b), $I \leftarrow I_0$
19: **end if**
20: **end while**
21: **return** (S^b)

3.4. Component-based crossover operator

The crossover operator is another key component of our memetic algorithm. In practice, it is important to devise a dedicated re-combination operator that has strong "semantics" with respect to the studied problem. In the last few years, several kinds of crossover operator have been used in the literature. Li et al. (2011) introduced a complicated crossover operator based on the tree representation and Cherkesly et al. (2015) proposed an adapted-order-based crossover operator. Both operators can only be used for the LIFO policy and cannot deal with the FIFO constraint. In this study we propose a general crossover operator, which is different from the previous ones as follows: First, our crossover operator always generates feasible solutions with respect to all the constraints, i.e., the LIFO, maximum time, and maximum capacity constraints. Thus, it is unnecessary to employ the repair strategy to ensure feasibility of the generated solutions. Second, based on an iterated greedy construction mechanism, our crossover operator can obtain high-quality offspring solutions.

We design the proposed crossover operator such that the elite components are transferred from the parents to the offspring to a large extent, which follows the general principles underlying the structured combination approach introduced in Glover (1994), and operates in the following two sequential steps:

• Step 1. Generate components: We divide the two selected parents into their corresponding components. To illustrate this, Fig. 4 depicts an example with two parents $I^a = \{V_1^a, V_2^a\}$ and $I^b = \{V_1^b, V_2^b\}$, where V_1^a and V_2^a (V_1^b and V_2^b) denote the two routes of solution I^a (I^b). Obviously, there are three components in the two routes for I^a, i.e., (1^+, 2^+, 2^-, 1^-), (5^+, 5^-), and (3^+, 4^+, 4^-, 3^-), while there are only two components for I^b, i.e., (1^+, 4^+, 5^+, 5^-, 4^-, 1^-) and (2^+, 3^+, 3^-, 2^-).

• Step 2. Iterated greedy construction based on components: For each route, components are added sequentially according to a best saving criterion. The candidate insertion position of each component is either before or after one component expected for being inserted into one new route. After each insertion, it is necessary to remove the same couple of the inserted component from the other parent in order to

Table 4
Performance comparisons among VNS, PTS, and LMA for solving PDPLD instances.

		BKS		VNS			PTS			LMA					
Instance	Size	f_{best}	Veh	f_{avg}	Gap(%)	Time	f_{avg}	Gap(%)	Time	f_{best}	Gap(%)	Veh	f_{avg}	Gap(%)	Time
brd14051	25	5086	2	5086.0	0	1.62	5087.2	0.02	0.54	**5084**	−0.04	2	5084	−0.04	0.41
	51	**9352**	2	9352.0	0	7.27	9438.7	0.93	2.96	9354	0.02	2	9355.4	0.04	2.71
	75	**8543**	2	8543.4	0	19.05	8544.8	0.02	6.84	8544	0.01	2	8544.2	0.01	2.48
	101	11,169	2	11169.0	0	21.31	11169.0	0	6.42	**11156**	−0.12	2	11246.7	0.7	4.81
	251	27195	8	28196.1	3.68	172.9	28730.2	5.65	119.51	**26690**	−1.86	5	27372.6	0.65	89.89
	501	58028	8	59090.0	1.83	965.21	60699.8	4.6	907.25	**57419**	−1.05	8	57810.2	−0.38	330.32
	751	96068	12	96421.5	0.37	1345.17	97976.6	1.99	4559.2	**95167**	−0.94	13	95597.1	−0.49	471.86
d15112	25	**108207**	2	108207.0	0	1.45	108439.0	0.21	0.31	108208	0	2	108211.3	0	1.09
	51	**178863**	3	179086.0	0.12	5.29	179859.0	0.56	1.13	178866	0	3	178866.5	0	2.33
	75	**239511**	4	244600.7	2.13	13.82	251313.9	4.93	3.71	239516	0	4	239516.1	0	8.29
	101	325761	5	339504.3	4.22	14.19	346055.0	6.23	5.28	325761	0	5	325761.3	0	4.22
	251	700366	10	720770.3	2.91	98.15	727627.5	3.89	200.22	**697292**	−0.44	10	703117.9	0.39	65.85
	501	1145838	16	1165275.3	1.7	252.73	1188016.3	3.68	663.22	**1143485**	−0.21	17	1145970	0.01	330.32
	751	1596048	22	1622964.0	1.69	1036.52	1637573.1	2.6	1765.95	**1587880**	−0.51	22	1600147.6	0.26	979.01
d18512	25	5086	2	5086.0	0	1.63	5087.2	0.02	0.54	**5084**	−0.04	2	5084	−0.04	1.38
	51	**9245**	2	9256.8	0.13	6.67	9286.0	0.44	1.17	9250	0.05	2	9250.1	0.06	2.72
	75	10147	2	10148.8	0.02	16.85	10180.7	0.33	8.41	**10146**	−0.01	2	10146	−0.01	2.92
	101	11742	2	11765.6	0.2	31.13	11909.0	1.42	6.24	**11614**	−1.09	2	11747.2	0.04	18.53
	251	27945	5	28933.5	3.54	229.93	29314.2	4.9	206.38	**27757**	−0.67	5	28757.9	2.91	204.08
	501	56790	8	57787.4	1.76	733.32	58928.0	3.76	1241.01	**56575**	−0.38	8	58109.2	2.32	528.32
	751	91670	11	94016.2	2.56	2346.58	95612.5	4.3	3765.94	**90801**	−0.95	12	91862.9	0.21	2970.85
fnl4461	25	**2168**	1	2168.0	0	2.35	2168.0	0	0.40	2170	0.09	2	2170	0	0.21
	51	**4830**	2	4830.0	0	4.68	4830.0	0	1.11	4832	0.04	2	4832	0	2.49
	75	**7399**	3	7432.2	0.45	15.83	7466.5	0.91	8.22	7406	0.09	2	7406	0.09	6.19
	101	10608	4	10765.3	1.48	18.29	10897.8	2.73	7.32	**10604**	−0.04	4	10615.9	0.07	13.68
	251	30651	4	31334.1	2.23	255.07	31782.4	3.69	210.30	**30546**	−0.34	4	30842.9	0.63	39.84
	501	81994	7	83246.3	1.53	667.54	85081.8	3.77	1596.72	**81454**	−0.66	7	81702.6	−0.36	478.17
	751	135940	12	138626.0	1.98	2163.77	139106.8	2.33	3172.93	**135592**	−0.26	12	138392.2	1.8	1908.04
nrw1379	25	**3464**	2	3464.0	0	2.00	3466.1	0.06	0.58	3465	0.03	2	3465	0	0.22
	51	**5398**	2	5423.4	0.47	8.71	5499.3	1.88	2.43	5397	−0.02	2	5397	0	2.74
	75	**8207**	3	8352.2	1.77	20.23	8403.4	2.39	11.84	8219	0.15	3	8220.5	0.16	11.66
	101	11933	4	12220.9	2.41	33.06	12537.3	5.06	18.82	**11733**	−1.68	4	11989.7	0.48	23.78
	251	31075	7	31635.0	1.8	96.03	32217.6	3.68	186.17	**30957**	−0.38	7	31348.5	0.88	76.51
	501	68202	13	68962.5	1.12	293.56	70800.0	3.81	775.82	**67684**	−0.76	13	67914.8	−0.42	150.78
	751	122587	21	124711.0	1.73	858.91	125859.0	2.67	1740.60	**122179**	−0.33	22	123753.9	0.95	771.56
pr1002	25	**16221**	1	16221.0	0	1.75	16221.0	0	0.21	16221	0	1	**16221**	0	0.28
	51	47905	3	47989.0	0.18	4.60	47980.6	0.16	1.20	**47129**	−1.62	2	47293.5	−1.28	3.64
	75	64102	4	64889.0	1.23	11.94	65197.8	1.71	7.49	**63869**	−0.36	4	63960.4	−0.22	8.98
	101	87700	5	88260.8	0.64	13.78	88373.9	0.77	7.65	**87692**	−0.01	5	87717.8	0.02	19.01
	251	25798	8	263657.4	2.51	60.59	264251.3	2.74	157.55	**254460**	−1.06	8	256864.3	−0.13	10.12
	501	597464	14	611917.4	2.42	396.32	620169.5	3.8	776.39	**595095**	−0.4	15	599562.1	0.35	297.45
	751	1008027	19	1031423.3	2.32	1045.75	1034424.5	2.62	1723.79	**1000604**	−0.74	20	1017101.2	0.9	375.67
avg	250.71	174422.21	6.40	177923.54	1.27	316.56	179942.44	2.27	568.57	**173641.83**	−0.39	6.45	174960.23	0.26	243.41
#better										29					
#equal										2					
#worse										11					

Table 5
Comparisons of HIS with RCDP and RDPD on public benchmark PDPLT instances.

	RCPD			RPBD			HIS		
Instances Set	f_{best}	f_{avg}	V_{avg}	f_{best}	f_{avg}	V_{avg}	f_{best}	f_{avg}	V_{avg}
lc1	11666.49	11954.37	10	12992.87	13245.64	12	**10126.75**	10201.19	9
lc2	12191.51	12330.29	4	13329.51	14019.83	5	**10738.37**	10894.22	4
lr1	3662.08	3844.15	19	4196.58	4321.40	23	**2553.86**	2778.01	12
lr2	4083.73	4308.95	4	4527.36	4742.02	5	**2891.91**	2996.54	3
lrc1	4415.38	4764.21	22	4957.51	5119.48	28	**2836.88**	3008.18	13
lrc2	5007.25	5214.66	5	5556.38	5794.76	6	**3318.75**	3428.90	4
LC1	61759.62	67891.54	46	76303.72	80432.01	58	**44253.90**	47812.32	32
LC2	59966.55	65048.99	17	70925.37	74501.11	20	**44595.98**	48676.67	13
LR1	31547.26	35674.50	43	41113.11	46487.24	61	**13548.62**	15998.56	18
LR2	34561.74	37529.64	11	43408.86	48645.30	14	**17700.74**	19741.52	6
LRC1	31231.48	35671.47	46	42537.10	45789.69	68	**12493.77**	13987.12	17
LRC2	34043.98	38485.91	11	46199.36	49975.54	16	**16365.12**	17895.55	5
avg	24511.42	26893.22	19.83	30503.98	32756.17	26.33	**15118.72**	16451.56	11.33
#better							12		
#equal							0		
#worse							0		

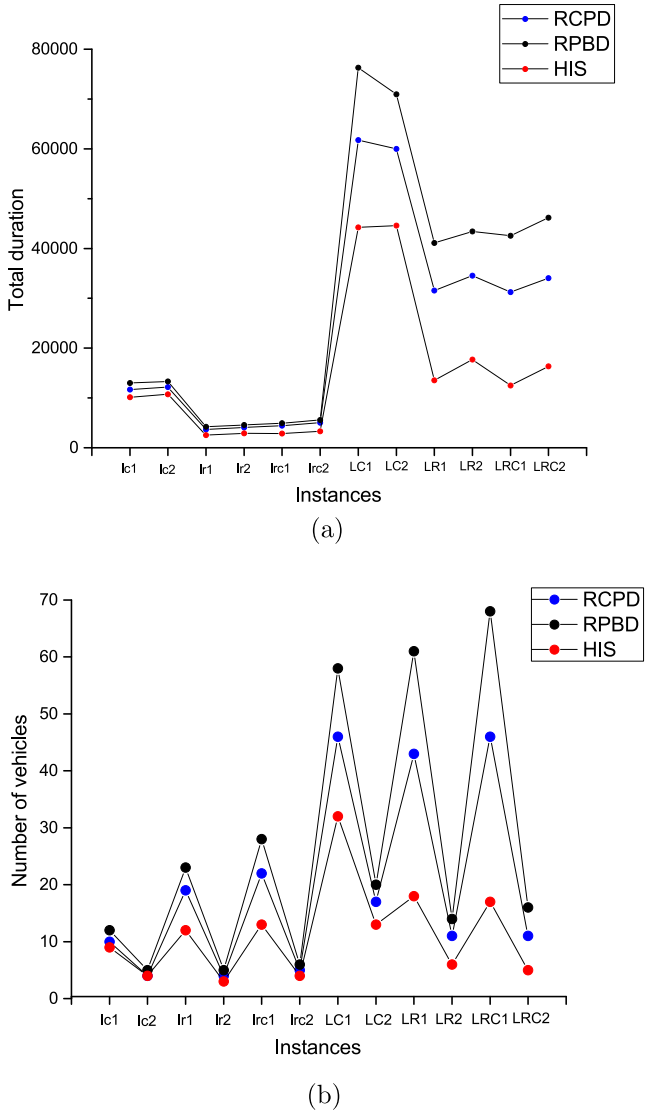


Fig. 6. Comparative results of HIS with RCPD and RPBD in terms of (a)average total duration (b)the corresponding number of vehicles, respectively.

avoid redundancy. The best insertion position of each component cp in the route is the one yielding the minimum value of f':

$$f' = \Delta(f)/\theta, \tag{5}$$

where $\Delta(f)$ represents the incremental objective value after inserting component cp in the route and θ denotes the number of couples in component cp. As shown in Fig. 4, we insert component $(1^+, 2^+, 2^-, 1^-)$ into route V_1^c and remove two couples $(1^+, 1^-)$ and $(2^+, 2^-)$ from I^a and I^b. Then, we insert the next component $(3^+, 3^-)$ with the best value f' into route V_1^c. Note that if all the cases for which one of the remaining components is inserted into route V_1^c will violate the maximum time constraint, then the selected component $(4^+, 5^+, 5^-, 4^-)$ will constitute one new route V_2^c.

3.5. LCS-based population updating strategy

To maintain healthy diversity of the population, we use the LCS-based population updating strategy introduced by Cheng and Peng (2016) to solve the job shop scheduling problem, to decide whether the improved solution should be inserted into the population or discarded. This population updating strategy simultaneously considers the solution quality and the distances among the individuals in the population to guarantee diversity of the population. The underlying idea is that the similarity of two solutions based on the longest common subsequence could clearly match the neighborhood moves based on the insert and swap operations. For this purpose, we first make two definitions as follows:

Definition 1 (*(Distance between a solution and its population)*.). *Given a solution S_i and the population $PP = \{S_1, S_2, \cdots, S_{np}\}$, the distance $Dist_{PP}(S_i)$ between the solution S_i and its population PP can be defined as follows*:

$$Dist_{PP}(S_i) = Min\{2 * n - lcs(S_i, S_j): 1 \le i, j \le np, i \ne j\}, \tag{6}$$

where $2 * n$ and $lcs(S_i, S_j)$ denote the number of all the pickup and delivery nodes and the length of the longest common subsequence between S_i and S_j, respectively. For example, given a solution $S_1 = (i^+, i^-, j^+, j^-, k^+, k^-)$, the candidate neighboring solution is represented by $S_2 = (j^+, j^-, k^+, k^-, i^+, i^-)$ after a request-insert move by insert request i after request k. The distance between S_1 and S_2 based on the longest common subsequence is only 2 while the distance between S_1 and S_2 based on the classic Hamming distance can be 6. With respect to the hamming distance of solutions S_1 and S_2, these two solutions are completely different solutions. However, both solutions can be converted by using an insert neighborhood move. Hence, the longest common subsequence mechanism can better match the insert and swap moves.

Definition 2 (*(Goodness score of a solution in the population)*.). *The goodness score $GS(S_i)$ of a solution S_i is defined by its objective function value, as well as its distance to the population, as follows*:

$$GS(S_i) = \delta \times \frac{f_{max} - f_{S_i}}{f_{max} - f_{min} + 1} + (1 - \delta) \times \frac{Dist(S_i) - Dist_{min}}{Dist_{max} - Dist_{min} + 1}, \tag{7}$$

where f_{max} and f_{min} denote the maximum and minimum objective values of the individuals in the population PP, and $Dist_{max}$ and $Dist_{min}$ are the maximum and minimum distances between a solution to the population, respectively. The δ is a constant parameter. Since it could be $Dist_{max} = Dist_{min}$, in the denominator, $Dist_{max}$ - $Dist_{min}$ + 1 was used.

In each generation, the offspring individual is inserted into the population if the goodness score of the offspring is better than the worst solution in the population according to the goodness score. Otherwise, the offspring individual is discarded. It is clear that the greater the goodness score $GS(S_i)$ is, the better is the solution S_i. It is noted that this goodness score function simultaneously considers the factors of solution quality and population diversity. On the one hand, we should maintain a population of elite solutions. On the other hand, we have to emphasize the importance of the diversity of the solutions to avoid premature convergence of the population.

4. Computational studies

In this section we report extensive computational studies conducted to assess the performance of the proposed learning-based memetic algorithm (LMA) with the state-of-the-art reference algorithms in solving public benchmark instances of both PDPLD and PDPLT. Note that PDPLD is a special case of PDPLT where the latter simply considers very high vehicle capacity and ignores the service times in the request nodes.

4.1. Benchmark instances and experimental protocols

For experimental evaluations, we use two data sets in Benavent et al. (2015) and Cheang et al. (2012). The first set of benchmark instances was first proposed in Li and Lim (2003) for the pickup and delivery problem with time windows (PDPTW), which consists of six classes of instances, namely lc1, lc2, lr1, lr2, lrc1, and lrc2, where the nodes in the lc instances are in clusters, in the lr instances are randomly distributed, and in the lrc instances are partially clustered and partially

Table 6
Comparison of the memetic algorithms with and without the learning-based mechanism on public benchmark PDPLT and PDPLD instances.

Instances Set	WLMA			LMA		
	f_{best}	f_{avg}	Time	f_{best}	f_{avg}	Time
lc1	9863.21	9865.69	20.59	**9862.57**	9862.93	23.37
lc2	9834.25	9842.23	17.72	**9834.01**	9834.40	17.84
lr1	2117.58	2136.90	29.54	**2113.59**	2121.18	30.80
lr2	2085.46	2315.71	35.14	**2082.07**	2288.05	33.25
lrc1	2246.18	2304.58	27.09	**2240.07**	2251.93	25.69
lrc2	2210.52	2227.36	20.11	**2190.81**	2212.35	23.10
LC1	42390.90	42551.95	385.68	**42375.73**	42420.45	360.08
LC2	40972.84	41056.29	312.42	**40772.57**	40868.81	300.34
LR1	9715.79	9801.76	323.63	**9582.16**	9788.584	338.38
LR2	11788.35	12183.44	281.18	**10822.59**	11178.55	260.46
LRC1	9210.61	9431.28	360.73	**9030.575**	9141.71	348.02
LRC2	9712.52	10233.04	345.24	**9584.18**	9920.77	324.52
brd14051	30510.23	30719.17	345.24	**30487.29**	30711.20	143.43
d15112	611889.62	619696.93	187.57	**611571.29**	614512.51	198.01
d18512	30384.21	30890.80	589.35	**30175.29**	30708.19	532.69
fnl4461	39077.55	40018.72	305.10	**38942.86**	39422.04	352.65
nrw1379	35723.52	36001.26	206.96	**35662.00**	36012.01	149.56
pr1002	296784.67	299954.69	235.22	**295010.00**	298385.43	110.43
avg	66473.22	67290.66	223.81	**66241.09**	66757.84	198.48
#better				18		
#equal				0		
#worse				0		

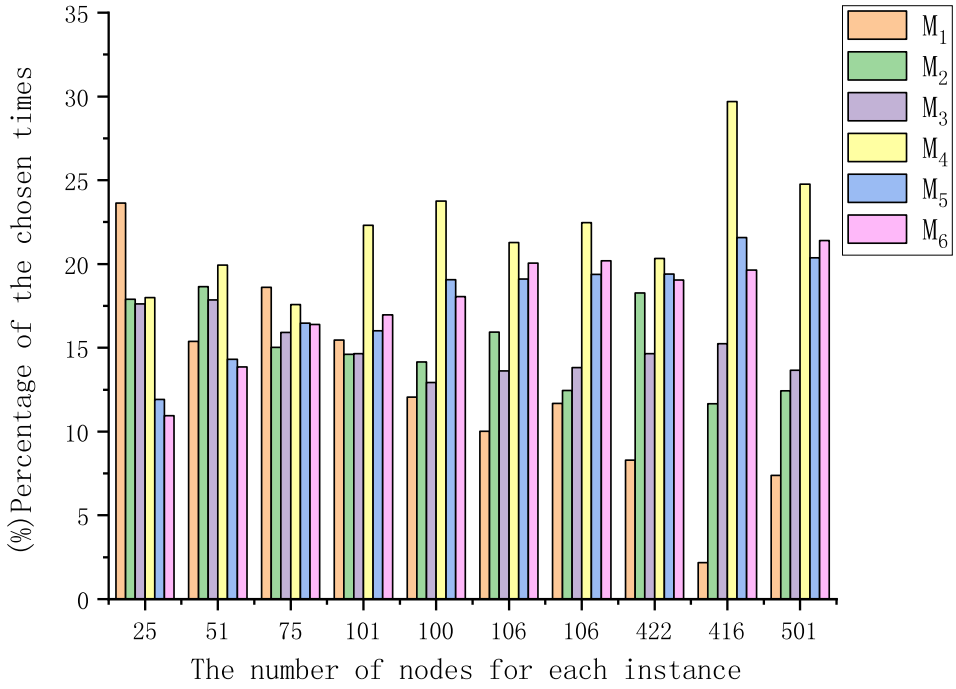
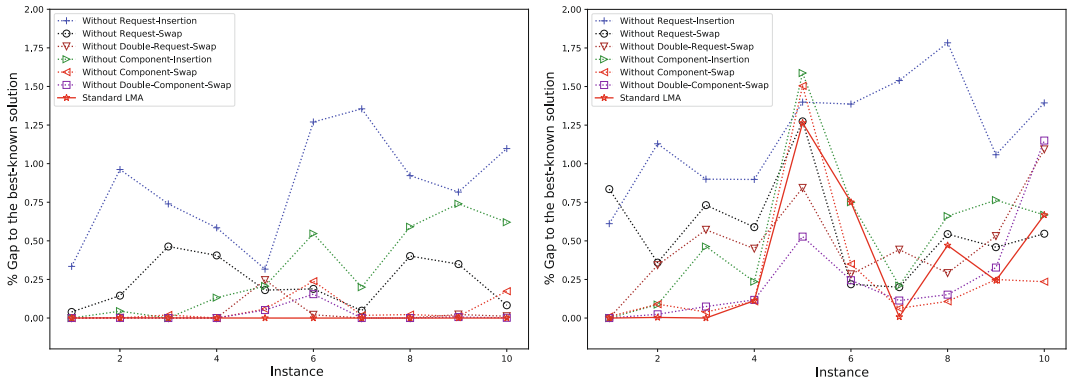


Fig. 7. Percentage of the chosen times for six neighborhood moves in LMA.

randomly distributed. Benavent et al. (2015) modified these original instances to match the new problem PDPLT by setting the maximum route duration equal to the width of the time window associated with the depot and ignoring all the other time windows. Hence, in our computational studies, we used the same benchmark instances as thosed used in Benavent et al. (2015). The instances can be divided into two subsets of 116 instances, including 56 small-size instances with 100–110 request nodes and 60 large-size instances with 402–422 request nodes.

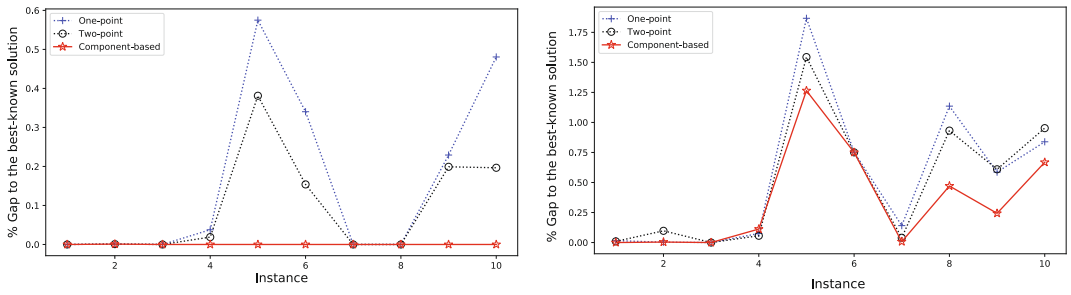
The second set of benchmark instances was used in Cheang et al. (2012) for PDPLD. Since PDPLD is a special case of PDPLT, we compare our learning algorithm LMA with the reference algorithms in the literature by setting a very high vehicle capacity limit for each vehicle and ignoring the service time in each request vertex. These instances were derived from the six TSP instances fn14461, brd14051, d15112, d18512, nrw1379, and pr1002 extracted from TSPLIB (Reinelt, 1991). For each of these TSP instances, subsets of vertices were selected with 25, 51, 75, 101, 251, 501, and 751 vertices. In addition, we imposed a travel distance limit d_{max} on each instance, where $d_{max} = max\{d(0^+, i^+) + d(i^+, i^-) + d(i^-, 0^-)\}$ $(i \in r)$, which is the largest distance for any route involving a single request. All the benchmark instances are publicly available on the website[1], as well as the executable files of our LMA method.

[1] https://github.com/283224262/PDPLT.



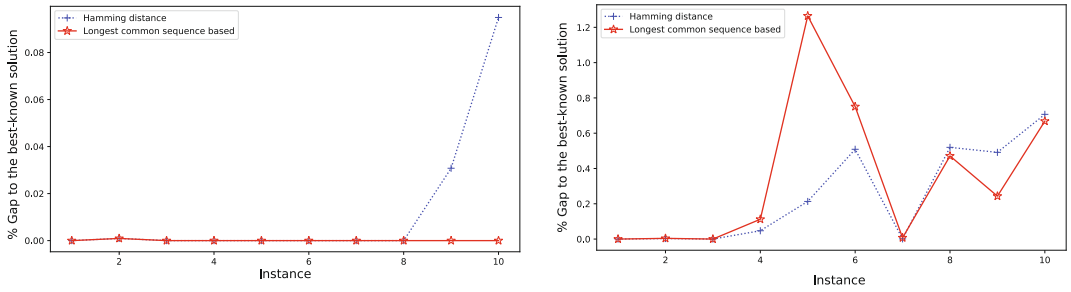
(a) Best results (b) Average results

Fig. 8. Comparisons of LMA with its variants that exclude six different neighborhoods.



(a) Best results (b) Average results

Fig. 9. Comparisons of LMA with its variants that use two crossover operator (i.e., one-point crossover and two-point crossover).



(a) Best results (b) Average results

Fig. 10. Comparisons of LMA with its variant that uses the Hamming distance in the population updating procedure.

In this study we coded our LMA algorithm in C++ and ran it on a PC with a AMD Athlon 3.0 GHz CPU and 2 Gb RAM operating under the Windows 7 operating system. To evaluate the performance of LMA, we compare it with the following state-of-the-art heuristics from the literature:

- The multi-start iterated tabu search (MS-ITS) for PDPLT proposed by Benavent et al. (Benavent et al., 2015), who implemented it on a 2.66 GHz Core 2Quad processor with 3 Gb RAM under the Windows XP operating system.
- Eight heuristics including a Variable Neighborhood Search (VNS) algorithm, six reduced versions of the VNS heuristic, and the Probabilistic Tabu Search (PTS) algorithm for PDPLD proposed in Cheang et al. (2012), who tested their performance on a Dell server with an Intel Xeon E5520 2.26 GHz CPU and 8 GB RAM operating under the Linux operating system. The computing times reported are in CPU seconds on this server.

In order to achieve relatively fair comparisons, we apply the method proposed in Chen, Hao, and Glover (2016) to scale the computing times reported for the mentioned heuristics in the corresponding studies. The procedure is based on the assumption that the CPU speed is approximately linearly proportional to the CPU frequency. Specifically, we performed ten independent runs of our LMA for each instance, with the maximum time limit per run set to equal the scaled CPU times by multiplying the computing time reported by the current best-performing algorithms from Benavent et al. (2015) and Cheang et al. (2012) with the ratio values (2.66/3.0) and (2.26/3.0), respectively.

4.2. Parameter tuning and parameter sensitivity analysis

Table 1 presents the settings of the LMA parameters used in the reported experiments. We tuned the parameters (α, β_1, β_2, γ , ζ, np, ε, and δ) with the Iterated F-race (IFR) proposed by Birattari, Yuan, Balaprakash, and Stützle (2010) and an automated configure method that is part of the IRACE package from Lopez-Ibanez, Dubois Lacoste, Caceres, Birattari, and Stützle (2016). We performed the tuning process on instances LC1, LR1, LRC1, brd14051, d15112, d18512, fnl4461,

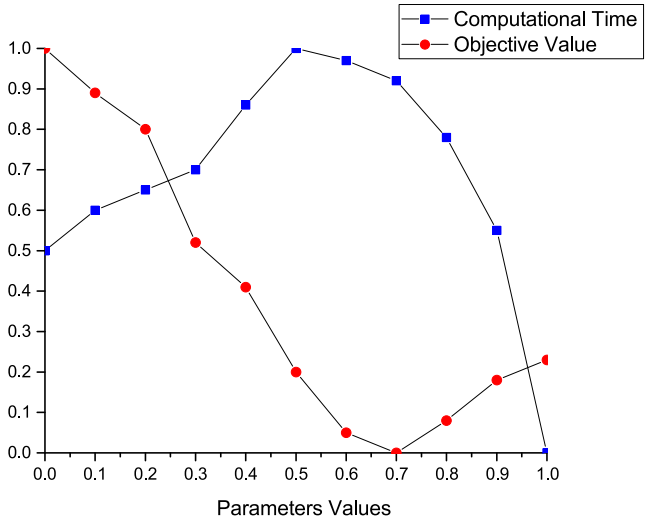


Fig. 11. The effect of the parameter.δ

nrw1379, and pr1002 with 402–440, 501, and 751 vertices. For each parameter, IFR requires a limited set of values as input to choose from the "candidate values" which are empirically determined and presented in Table 1. We set the total time budget for IRACE to 1,000 executions of LMA, with a time limit of 400 s for the PDPLT instances LC1, LR1, and LRC1, and 3,000 s for the PDPLD instances brd14051, d15112, d18512, fnl4461, nrw1379, and pr1002. We denote the parameters setting suggested by IFR as *Finalvalue* in Table 1.

To evaluate the sensitivity of each parameter, we performed a further tuning study on these representative medium and large instances. For each running, we evaluated each candidate value from Table 1 by fixing the remaining parameters to the corresponding values. We performed ten independent runs for each instance. We present the distribution and range of the average results for each parameter in a box-and-whisker plot in Fig. 5. In order to determine whether there exist statistically significant differences in the solution samples for different values of each candidate parameter, we conducted the the Friedman rank sum rest. We report the corresponding p-values in Fig. 5. The Friedman test reveals statistically significant differences in performance for parameters α (p-value = 0.0033), β_1 (p-value = 0.0451), β_2 (p-value = 0.0472), γ (p-value = 0.0005), ζ (p-value = 0.0306), and δ (p-value = 0.0247), while the remaining parameters do not exhibit significant differences. From the plots in Fig. 5, we further observe that the best-performing parameter values are almost the same as the best parameter values reported by IFR.

4.3. Computational results on the tested benchmark instances

In this section we evaluate the performance of LMA in solving PDPLT in comparison with the best-performing algorithm MS-ITS proposed by Benavent et al. (2015). Specifically, we applied LMA to solve each instance for ten times, terminating each run when the time reached the maximum time of 400 s. As depicted in Tables 2 and 3, the columns f_{best}, Veh, f_{avg}, and Time report the best objective value, i.e., the minimum travelled duration, the number of vehicles corresponding to the best solution, the average objective value over the ten runs, and the computing time, respectively. In addition, the rows *#best*, *#equal*, and *#worse* indicate respectively the numbers of instances for which the associated algorithm obtains better, equal, or worse objective values in terms of f_{best} compared with the reference results reported in the literature, and the row *avg* denotes the average value over all the instances in the set.

From Table 2, we observe that LMA outperforms the reference algorithm MS-ITS in terms of f_{best}, Veh, and Time. In particular, LMA achieves better results than the current best-performing algorithm MS-ITS in terms of obtaining the best objective value for 43 out of the 56 instances, while matching the current best-known results for the remaining 13 instances. Moreover, LMA is able to obtain the best objective value within a very short time, which is several times faster than the reference algorithm MS-ITS for the tested instances, i.e., MS-ITS's 56.22 and 158.95 versus LMA's 26.88 and 24.38.

To further compare the performance of LMA and the reference algorithm MS-ITS, we applied them to solve the large-size instances with 402–422 nodes. Table 3 presents the results in detail. It is evident that LMA is able to obtain better results in terms of both the best objective value and computing time for all tested instances. Furthermore, in terms of the number of vehicles corresponding to the best objective value, LMA usually outperforms MS-ITS for the tested instances, i.e., MS-ITS's 21.17 and 18.9 versus LMA's 9.53 and 6.83.

In view of its good performance, we further compare LMA with the eight heuristic algorithms for PDPLD instances in Cheang et al. (2012). As shown in Table 4, the first two columns present the instance identifiers and the number of the request vertices, respectively. The following two columns report the best objective value f_{best}, and the number of the vehicles *Veh* corresponding to the best known solution (BKS) found by the eight heuristics denoted as *BKS*, including a VNS algorithm, six reduced versions of the VNS heuristic, and the PTS algorithm for PDPLD in Cheang et al. (2012). We present the average results of the objective value f_{avg}, the gap of f_{avg} value to the BKS value

Table 7
Summary for the key components in LMA.

Key components	Description	Motivation	Influence and results
Hybrid initial solution (HIS) method	HIS adapts the splitting mechanism for PDPLT by employing the Lin-Kernighan heuristic (LKH) for the ATSP sub- problem in PDPLT to improve the solution quality of the initial solutions.	The similar initial generation method is successfully applied to other routing problems (such as multi-depot VRP and multi-route VRP).	Experimental results in Section 5.1 show that HIS can generate better initial solutions than two previous methods RCDP and RPBD.
Learning-based mechanism	Reinforcement learning mechanism is able to effectively exploit memory to manage the neighborhood moves.	(1) The structure of the problem instances affects the performance of neighborhood moves; (2) Alternating among different moves based on the proposed learning mechanism yield more robust performance.	Experimental results in Sections 5.2 and 5.3 show the effectiveness of the learning based-mechanism and six neighborhood moves.
Components-based crossover	Two steps based on the iterated greedy method to generate the feasible offspring solution.	(1) The classic one-point and two-point crossover usually generate a infeasible offspring solution; (2) The dedicated component-based crossover based on the problem structure may obtain a better performance.	Better performance in comparison with two crossover operators in Section 5.4 shows its importance.
Longest common subsequence (LCS) based population updating strategy	We define the distance and goodness score in the population, which are based on the longest common subsequence strategy	The similarity of two solutions based on the longest common subsequence could better match the neighborhood moves based on the insert and swap operations.	Compared with the Hamming distance based updating strategy, LCS-based updating strategy can obtain a slightly better performance (see Section 5.5).

in percentage (%), and computing time *Time* obtained by the VNS and PTS algorithms in ten executions in columns 5–10. The last six columns show that LMA obtains better results for 29 out of 42 instances, matches the current best-known results for 2 instances, and is slightly worse than MS-ITS for 11 instances. In addition, LMA outperforms VNS and PTS in terms of the average objective value, the average gap and the average computing time for most tested instances.

To summarize, the results of the above extensive computational studies demonstrate the efficacy of LMA in tackling PDPLT in terms of both solution quality and computational efficiency.

5. Analysis and discussions

In this section we analyze several key components and one important parameter of LMA, including the hybrid initial solution (HIS) method, the learning-based mechanism to manage the neighborhood moves, the six neighborhood moves, the component-based crossover operator, and the parameter δ for the goodness score, with a view to understanding their impacts on the performance of LMA.

5.1. Comparisons between the hybrid initial solution method and the previous construction methods

To study the effectiveness of the proposed HIS method, we compare it with the two effective reference methods, namely random solution with consecutive pickups and deliveries (RCPD), and random solution with pickups before deliveries (RPBD), proposed in Benavent et al. (2015) for PDPLT. Since all the initial solution generation methods only run within a very short time (in fact, less than one second) for all the benchmark instances, we ignore the comparison in terms of computing time. We performed ten runs of each method to solve each instance and recorded the best objective value (minimum duration) and the corresponding number of vehicles. Table 5 and Fig. 6 present the comparisons of the HIS method with the RCDP and RPBD methods on different classes of instances. Table 5 presents for each instance set its name, the average results in terms of the best objective value f_{best} and the average objective value f_{avg} for each instance set, and the average number of vehicles V_{avg} corresponding to the best solution, which is rounded to an integer. Fig. 6 graphically shows the comparisons of HIS with RCDP and RPBD, where the horizontal axis denotes the sets of instances, and the vertical axis denotes the total duration and the number of vehicles, respectively.

From Table 5 and Fig. 6, we observe that HIS is able to obtain better results than RCDP and RPBD in terms of the best objective value and the corresponding number of vehicles. As the size of the instances becomes larger, the advantage of LMA becomes more evident. In sum, HIS is a very effective initial solution generation method for PDPLT.

5.2. Analysis of the learning-based mechanism

To highlight the importance of the learning-based mechanism in LMA, we carried out the following computational studies. We use LMA and WLMA to denote the learning-based memetic algorithm and the memetic algorithm without the learning mechanism, respectively. In other words, under WLMA, the neighborhood moves are chosen in a token-ring fashion (i.e., $N_1, \cdots, N_6$) with the same fixed probability without the learning-based reward and punishment strategy, while other ingredients are kept unchanged.

We independently solved each instance for ten times by using LMA and WLMA. We report the computational results in Table 6, which shows the average result in terms of the best objective value and average objective value, f_{best} and f_{avg}, respectively, in each instance set, and the average computing time per successful run, i.e., *Time*, where we indicate the best objective values between the two algorithms in bold.

Table 6 shows that when the learning-based mechanism is used, LMA outperforms WLMA on all the public benchmark instances. In particular, in terms of the best and average objective values, LMA obtains better values than WLMA for all the tested instances, as illustrated by WLMA's 66473.22 and 67290.66 versus LMA's 66241.09 and 66757.84. As for the average running time, the two algorithms are very close to each other, i.e., WLMA's 223.81 versus LMA's 198.48. Specifically, LMA takes shorter (longer) average computing times to reach the average best results on 13 (5) sets of the tested instances, respectively, compared with WLMA.

In order to further study the characteristics of the learning mechanism. We obtain the percentage of the chosen times for six neighborhood moves during the search of LMA for ten representative instances with different scales[2]. We run each instance once with the maximum time limit of 360 s by using LMA. Fig. 7 shows the experimental results for the percentage of the chosen moves for all the six neighborhood moves in LMA. The probability of choosing small moves with no more than two requests (such as M_1 and M_2) clearly decreases as the size of the instances increases (for sizes ranging from 25 to 501). In contrast, the probability of choosing large moves based on components (such as M_4, M_5 and M_6) significantly increases. One can clearly observe that the large moves are preferred in the large scale instances. In general, the probability of each move being selected varies for difference instances, which are significantly different from the token-ring fashion for WLMA (with the same probability for each move) verifying the impact of our proposed learning mechanism.

The above results indicate that our learning-based mechanism plays a crucial role in boosting the performance of LMA in solving PDPLT.

5.3. Effectiveness of the neighborhoods

As described in Section 3.3.1, our LMA procedure employs six dedicated neighborhood structures (M_1-M_6). In this section, we investigate the influence of each neighborhood on the performance of the LMA algorithm. For this purpose, we propose six variants of LMA such that for each LMA variant, we disable one particular neighborhood while keeping other components unchanged. Along with the standard LMA algorithm, six LMA variants are tested on ten representative instances. Specifically, for each instance, all the reference methods were iteratively performed until a pre-fixed maximum time of LMA given in Tables 2–4 is reached. The best and the average performances are plotted in Fig. 8, where the y-axis indicates the percentage gap to the best-known solutions (see Tables 2–4). Fig. 8 shows that the manner in which removing a neighborhood deteriorates the search power of LMA, and confirms that all six neighborhoods contribute to the performance of the LMA algorithm. Apart from the usefulness of each individual neighborhood, their combined use within the LMA algorithm constitutes an important feature to ensure the performance of the whole algorithm.

5.4. Impact of component-based crossover operator

LMA applies the component-based crossover operator described in Section 3.4 to generate feasible and promising offspring solutions. To investigate the merit of this crossover operator, we compare LMA with two variants that use only one-point and two-point crossover operators respectively. Both operators are classic crossover operators used in scheduling and routing problems (Kellegöz, Toklu, & Wilson, 2008; Lu, Hao, & Wu, 2019), and can be described as follows. In the one-point crossover operator, the first step is to randomly select one cutting point. The node sub-sequence on one side of the cutting point is inherited from one parent and passed to the offspring. The other nodes are copied to the offspring in the order they appeared in the other parent. In the two-

[2] Ten representative instances consists of brd14051_25, d15112_51, d18512_75, fn14461_101, lc101, lr101, nrw1379_251, LC1_4_1, LR1_4_1, and pr1002_501.

point crossover operator, two cutting points are randomly selected to divide the parent solutions. By reference to the two parent solutions S_1 and S_2, the offspring is obtained by first transferring the sub-sequence between the two cutting points from S_1, and then copying in the same order the remaining nodes from S_2. In contrast to the component-based operator, both two reference operators usually generate infeasible solutions, violating the LIFO constraint. Hence, we employ the request-insertion neighborhood operator to reschedule the position of the conflicting requests to make the infeasible solution satisfy the maximum duration, maximum capacity, and LIFO constraints. For this analysis, we repeat the same experimental procedure as in the previous section. The best and the average performances are plotted in Fig. 9. In terms of the best and the average results, we observe a better performance of LMA when the component-based crossover operator is used, which highlights its importance in LMA.

5.5. Impact of longest common subsequence based updating strategy

To investigate the merit of the longest common subsequence mechanism, we compare LMA with a variant that uses the Hamming distance in the population updating procedure, as described in Section 3.5. For this analysis, we repeat the same experimental procedure as in the previous section. The best and the average performances are plotted in Fig. 10. In terms of the best and the average results, we observe a slightly better performance of LMA when the longest common subsequence is used. In particular, LMA is able to obtain better values for the best results than the LMA variant that uses the Hamming distance for two solutions, although both methods perform similarly in terms of the average results, which show the effectiveness of the longest common subsequence for LMA.

5.6. Importance of the parameter δ

The LCS-based population updating strategy considers both solution quality and the distances between solutions and the population in updating the search. The parameter δ is used to balance these two factors in order to achieve a better trade-off between intensification and diversification of the search. In order to ascertain the importance of δ in LMA, we conducted the following computational studies.

. Specifically, we selected eight representative sets of instances with different scales of the benchmark instances, namely lc2, lr2, lrc2, LC2, LR2, LRC2, fn14461 and pr1002. Taking into account the randomness of the algorithm, we ran it ten times for each parameter setting (0, 0.1, ⋯, 1.0). Fig. 11 presents the average results over all the tested instances, where the horizontal axis denotes the value of the parameter δ and the vertical axis denotes the computing time and the objective values normalized by the following normalized function:

$$\theta(z) = \frac{z - z_{min}}{z_{max} - z_{min}}. \quad (8)$$

Fig. 11 shows the trajectories of the computing time and objective value over different δ values. When δ is equal to 1, which gives all the weight to the objective value and none to the population distance, the corresponding algorithm is the fastest but cannot find high-quality solutions because the algorithm does not consider the diversity of the population, while the algorithm can obtain the best values with δ set at 0.6 to 0.8. The results demonstrate that premature convergence of the algorithm can be avoided by employing the LCS-based population updating strategy. We further observe that the best-performing parameter values are the same as the best parameter values analyzed in Fig. 5.

At last, we summarize and explain the key proposed components, as well as highlighting their corresponding motivations and influences in our LMA in Table 7.

6. Conclusion

Our learning-based memetic algorithm LMA for the pickup and delivery problem under the LIFO loading policy demonstrates the effectiveness of its key features, which include a hybrid initial solution construction method, a learning-based local search procedure, a component-based crossover operator utilizing the ideas of structured combinations, and a longest-common-subsequence-based population updating strategy.

Experimental evaluations on two sets of public benchmark instances show that our LMA performs very favourably compared to the current state-of-the-art reference algorithm MS-ITS benavent2015multiple and the eight heuristics including a VNS algorithm, its six reduced versions, and PTS for PDPLD proposed in Cheang et al. (2012). In particular, LMA is able to obtain highly competitive results in terms of both computational efficiency and solution quality for most of the PDPLT and PDPLD instances, improving the best-known results for 132 out of 158 of the tested problem instances, while matching the best-known results for all but three of the remaining instances. In addition, our computational studies demonstrate the effectiveness of the key strategies incorporated into our proposed LMA.

These outcomes motivate future research to extend our work in the following directions: First, it would be interesting to employ a powerful tabu search method (such as granular tabu search) to improve the search capability of the learning-based local search phase. Second, the design of our approach invites the development of related procedures that combine its strategies with those of other population-based frameworks like path-relinking (Glover, Laguna, & Marti, 2000) should be very promising. Finally, the success of these ideas for tackling the PDPLT problem suggests that it would be worthwhile to test their performance in dealing with other variants of the vehicle routing problem.

CRediT authorship contribution statement

Bo Peng: Conceptualization, Methodology, Writing - original draft. **Yuan Zhang:** Conceptualization, Methodology, Writing - original draft. **Zhipeng Lü:** Writing - review & editing, Project administration, Funding acquisition. **T.C.E. Cheng:** Writing - review & editing, Project administration, Funding acquisition. **Fred Glover:** Writing - review & editing.

Acknowledgments

This research was supported in part by the National Natural Science Foundation of China under grant numbers 71320107001, 61370183, 71871184, the Fundamental Research Funds under grant numbers JBK2001013 for the Central Universities of China, and the Programme for New Century Excellent Talents in Universities (NCET 2013).

References

Azadian, F., Murat, A., & Chinnam, R. B. (2017). An unpaired pickup and delivery problem with time dependent assignment costs: Application in air cargo transportation. *European Journal of Operational Research, 263*(1), 188–202.

Azi, N., Gendreau, M., & Potvin, J.-Y. (2014). An adaptive large neighborhood search for a vehicle routing problem with multiple routes. *Computers & Operations Research, 41*, 167–173.

Benavent, E., Landete, M., Mota, E., & Tirado, G. (2015). The multiple vehicle pickup and delivery problem with LIFO constraints. *European Journal of Operational Research, 243*(3), 752–762.

Benlic, U., & Hao, J. K. (2015). Memetic search for the quadratic assignment problem. *Expert Systems with Applications, 42*(1), 584–595.

Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T. (2010). F-race and iterated f-race: An overview.

Bortfeldt, A. (2012). A hybrid algorithm for the capacitated vehicle routing problem with three-dimensional loading constraints. *Computers & Operations Research, 39*(9), 2248–2257.

Bortfeldt, A., & Homberger, J. (2013). Packing first, routing seconda heuristic for the vehicle routing and loading problem. *Computers & Operations Research, 40*(3),

873–885.
Carrabs, F., Cerulli, R., & Cordeau, J.-F. (2007). An additive branch-and-bound algorithm for the pickup and delivery traveling salesman problem with LIFO or FIFO loading. *INFOR: Information Systems and Operational Research, 45*(4), 223–238.
Chami, Z. A., Manier, H., & Manier, M. A. (2017). A lexicographic approach for the bi-objective selective pickup and delivery problem with time windows and paired demands. *Annals of Operations Research, 2*, 1–19.
Cheang, B., Gao, X., Lim, A., Qin, H., & Zhu, W. (2012). Multiple pickup and delivery traveling salesman problem with last-in-first-out loading and distance constraints. *European Journal of Operational Research, 223*(1), 60–75.
Chen, Y., Hao, J. K., & Glover, F. (2016). A hybrid metaheuristic approach for the capacitated arc routing problem. *European Journal of Operational Research, 253*(1), 25–39.
Cheng, T. C. E., Peng, B. & Z. L (2016). A hybrid evolutionary algorithm to solve the job shop scheduling problem. Annals of Operations Research, vol. 242, 2 (pp. 223–237).
Cherkesly, M., Desaulniers, G., & Laporte, G. (2015). A population-based metaheuristic for the pickup and delivery problem with time windows and LIFO loading. *Computers & Operations Research, 62*, 23–35.
Cordeau, J. F., DellAmico, M., & Iori, M. (2010). Branch-and-cut for the pickup and delivery traveling salesman problem with FIFO loading. *Computers & Operations Research, 37*(5), 970–980.
Cordeau, J. F., Iori, M., Laporte, G., & Salazar González, J. J. (2010). A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with LIFO loading. *Networks, 55*(1), 46–59.
Erdogan, G., Cordeau, J.-F., & Laporte, G. (2009). The pickup and delivery traveling salesman problem with first-in-first-out loading. *Computers & Operations Research, 36*(6), 1800–1808.
Escobar, J. W., Linfati, R., Toth, P., & Baldoquin, M. G. (2014). A hybrid granular tabu search algorithm for the multi-depot vehicle routing problem. *Journal of Heuristics, 20*(5), 483–509.
Furtado, M. G. S., Munari, P., & Morabito, R. (2017). Pickup and delivery problem with time windows: A new compact two-index formulation. *Operations Research Letters, 45*(4), 334–341.
Glover, F. (1994). Tabu search for nonlinear and parametric optimization (with links to genetic algorithms). *Discrete Applied Mathematics, 49*(1–3), 231–255.
Glover, F., Laguna, M., & Marti, R. (2000). Fundamentals of scatter search and path relinking. *Control and Cybernetics, 29*(3), 653–684.
Helsgaun, K. (2000). An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research, 126*(1), 106–130.
Kachitvichyanukul, V., Sombuntham, P., & Kunnapapdeelert, S. (2015). Two solution representations for solving multi-depot vehicle routing problem with multiple pickup and delivery requests via pso. *Computers & Industrial Engineering, 89*, 125–136.
Kellegöz, T., Toklu, B., & Wilson, J. (2008). Comparing efficiencies of genetic crossover operators for one machine total weighted tardiness problem. *Applied Mathematics and Computation, 199*(2), 590–598.
Koch, H., Bortfeldt, A., & Wäscher, G. (2018). A hybrid algorithm for the vehicle routing problem with backhauls, time windows and three-dimensional loading constraints. *OR Spectrum, 40*(4), 1029–1075.
Li, H., & Lim, A. (2003). A metaheuristic for the pickup and delivery problem with time windows. *International Journal on Artificial Intelligence Tools, 12*(02), 173–186.
Li, Y., Lim, A., Oon, W. C., Qin, H., & Tu, D. (2011). The tree representation for the pickup and delivery traveling salesman problem with LIFO loading. *European Journal of Operational Research, 212*(3), 482–496.
Lopez-Ibanez, M., Dubois Lacoste, J., Caceres, L. P., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives, 3*, 43–58.
Lu, Y., Benlic, U., & Wu, Q. (2018). Multi-restart iterative search for the pickup and delivery traveling salesman problem with FIFO loading. Computers & Operations Research.
Lu, Y., Hao, J.-K., & Wu, Q. (2019). Hybrid evolutionary search for the traveling repairman problem with profits. *Information Sciences, 502*, 91–108.
Mack, D., & Bortfeldt, A. (2012). A heuristic for solving large bin packing problems in two and three dimensions. *Central European Journal of Operations Research, 20*(2), 337–354.
Montero, A., Miranda-Bront, J. J., & Mndez-Daz, I. (2017). An ILP-based local search procedure for the VRP with pickups and deliveries. *Annals of Operations Research, 259*(1–2), 327–350.
Naccache, S., Côté, J.-F., & Coelho, L. C. (2018). The multi-pickup and delivery problem with time windows. *European Journal of Operational Research, 269*(1), 353–362.
Pace, S., Turky, A., Moser, I., & Aleti, A. (2015). Distributing fibre boards: A practical application of the heterogeneous fleet vehicle routing problem with time windows and three-dimensional loading constraints. *Procedia Computer Science, 51*, 2257–2266.
Pavone, M. (2013). Asymptotically optimal algorithms for one-to-one pickup and delivery problems with applications to transportation systems. *IEEE Transactions on Automatic Control, 58*(9), 2261–2276.
Reil, S., Bortfeldt, A., & Mönch, L. (2018). Heuristics for vehicle routing problems with backhauls, time windows, and 3d loading constraints. *European Journal of Operational Research, 266*(3), 877–894.
Reinelt, G. (1991). TSPLIB–A traveling salesman problem library. *ORSA Journal on Computing, 3*(3), 376–384.
Turky, A., Moser, I., Aleti, A. (2017.) An iterated local search with guided perturbation for the heterogeneous fleet vehicle routing problem with time windows and three-dimensional loading constraints. In Australasian conference on artificial life and computational intelligence, Springer, unstr2017 (pp. 279–290).
Veenstra, M., Cherkesly, M., Desaulniers, G., & Laporte, G. (2017). The pickup and delivery problem with time windows and handling operations. *Computers & Operations Research, 77*(7), 127–140.
Wei, L., Zhang, Z., & Lim, A. (2014). An adaptive variable neighborhood search for a heterogeneous fleet vehicle routing problem with three-dimensional loading constraints. *IEEE Computational Intelligence Magazine, 9*(4), 18–30.
Yu, Y., Tang, J., Li, J., Sun, W., & Wang, J. (2016). Reducing carbon emission of pickup and delivery using integrated scheduling. *Transportation Research Part D Transport & Environment, 47*, 237–250.