European Journal of Operational Research 304 (2023) 381–394

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

An effective hybrid search algorithm for the multiple traveling repairman problem with profits

Jintong Ren[a,b], Jin-Kao Hao[c], Feng Wu[b], Zhang-Hua Fu[a,d,*]

[a] The Chinese University of Hong Kong, Shenzhen 518172, China
[b] University of Science and Technology of China, Hefei 230026, China
[c] LERIA, Université d'Angers, 2 boulevard Lavoisier, Angers 49045, France
[d] Shenzhen Institute of Artificial Intelligence and Robotics for Society, Shenzhen 518172, China

ARTICLE INFO

Article history:
Received 22 November 2021
Accepted 6 April 2022
Available online 12 April 2022

Keywords:
Combinatorial optimization
Multiple traveling repairman problem with profits
Arc-based crossover
Variable neighborhood search
Heuristics

ABSTRACT

The multiple traveling repairman problem with profits consists of multiple repairmen serving a subset of all customers to maximize the revenues collected through the visited customers. To address this problem, an effective hybrid search algorithm based on the memetic framework is proposed. In the proposed method, three features are integrated: a dedicated arc-based crossover to generate high-quality offspring solutions, a fast evaluation technique to reduce the complexity of navigating classical neighborhoods as well as a correcting step to ensure accurate evaluation of neighboring solutions. The performance of the algorithm on 470 benchmark instances were compared with those of the leading reference algorithms. The results show that the proposed algorithm outperforms the state-of-the-art algorithms by setting new records for 137 instances and matching the best-known results for 330 instances. The importance of the key search components of the algorithm was investigated.

1. Introduction

The traveling repairman problem with profits (TRPP) (Dewilde, Cattrysse, Coene, Spieksma, & Vansteenwegen, 2013) is a general model that can be stated as follows. Let $G(V, E)$ be a complete weighted graph, where V is the vertex set consisting of the depot 0 and the customer set $V_c = \{1, 2, \ldots, n\}$, and $E = \{(i, j) : i, j \in V\}$ is the edge set, where each edge (i, j) is associated with a symmetric weight $d_{i,j} = d_{j,i}$ (traveling time). A repairman begins his trip from the depot to collect a time-dependent revenue $p_i - l(i)$ by visiting each customer and stops his travels when there are no positive revenues. Here, p_i represents the profit, and $l(i)$ is the waiting time for each customer i ($l(0) = 0$). Each customer can be visited at most once. The objective of the TRPP is to determine an open Hamiltonian path such that the collected revenue $\sum_{i=0}^{m}[p_i - l(i)]^+$ is maximized, where m is the number of visited customers, and $[p_i - l(i)]^+$ is the larger value between $p_i - l(i)$ and 0.

The multiple TRPP (MTRPP) generalizes the TRPP by considering multiple repairmen (servers or vehicles) to service customers. In the MTRPP, all the repairmen start their trips from the depot and collect a time-dependent revenue independently. Let $K \geq 1$ be the number of repairmen, and a formal solution φ consists of K Hamiltonian paths (or routes) $\{X_1, X_2, \ldots, X_K\}$, where each path $X_k = (x_0^k, x_1^k, \ldots, x_{m_k}^k)$ contains m_k customers ($\bigcup_{k=1}^{K} X_i \subseteq V$ and $X_i \cap X_j = \{0\}$, $i \neq j$, $\forall i, j \in \{1, 2, \ldots, K\}$). The objective function can be defined as follows:

$$f(\varphi) = \sum_{k=1}^{K}\sum_{i=0}^{m_k}\left[p_{x_i^k} - l(x_i^k)\right]^+ \tag{1}$$

The aim of the MTRPP is then to find the solution φ^* with a maximal total collected revenue $f(\varphi^*)$.

If none of the collected revenue $p_{x_i^k} - l(x_i^k)$ is negative, then Eq. (1) can be rewritten as follows.

$$f(\varphi) = \sum_{k=1}^{K}\sum_{i=0}^{m_k} p_{x_i^k} - \sum_{k=1}^{K}\sum_{i=1}^{m_k}(m_k - i + 1)\cdot d_{x_{i-1}^k, x_i^k} \tag{2}$$

Eq. (2) is useful for the fast evaluation of our search algorithm.

The MTRPP is typically applied in humanitarian and emergency relief logistics. For instance, for post-disaster relief operations, K homogeneous rescue teams start their trips from the base to deliver emergency supplies and save survivors of damaged villages or cities. Assume that p_i persons are to be rescued for a village i and one person is lost with each time step. The objective of the rescue teams is to save as many lives as possible. This application scenario was also mentioned for the TRPP (Dewilde et al., 2013)

* Corresponding author.
E-mail address: fuzhanghua@cuhk.edu.cn (Z.-H. Fu).

https://doi.org/10.1016/j.ejor.2022.04.007
0377-2217/

with a single rescue team. The results revealed that the MTRPP is a convenient model for scenarios that require several rescue teams.

Existing studies on solving the MTRPP as well as some related problems are briefly reviewed as follows.

Two practical algorithms have been proposed for the MTRPP in the literature. In 2019, Lu, Benlic, Wu, & Peng (2019a) proposed the first memetic algorithm (MA-MTRPP) to solve the MTRPP. This algorithm uses a randomized greedy construction phase, variable neighborhood search, route-based crossover operator for solution initialization, local optimization, and solution recombination, respectively. MA-MTRPP outperformed the general CPLEX solver on the 240 benchmark instances introduced in that study. In the same year, Avci & Avci (2019) developed a mixed-integer linear programming model and suggested an adaptive large neighborhood algorithm (ALNS) search approach (ALNS-MTRPP) for the MTRPP, which incorporates a couple of problem-specific destroy operators and two new randomized repair operators. The authors proposed another set of 230 benchmark instances and a greedy randomized adaptive search procedure with iterated local search (GRASP-ILS), which was used as a reference heuristic. According to the experimental results, ALNS-MTRPP outperformed GRASP-ILS for most instances.

The closely related TRPP is a special case of the MTRPP with a single repairman ($K = 1$). Numerous heuristic algorithms were proposed to address the TRPP. In 2013, Dewilde et al. (2013) first proposed a tabu search algorithm incorporating multiple neighborhoods and a greedy initialization procedure. In 2017, Avci & Avci (2017) suggested a greedy randomized adaptive search procedure combined with iterated local search, which outperformed the previous algorithms by updating 46 best results. In 2019, Lu, Hao, & Wu (2019b) introduced a population-based hybrid evolutionary search algorithm that outperformed previous algorithms. In 2020, Pei, Mladenović, Urošević, Brimberg, & Liu (2020) developed a general variable neighborhood search approach integrating auxiliary data structures to improve search efficiency. This algorithm dominated all the previous algorithms by updating 40 best-known results and matching the best-known results for the remaining instances. As presented in this study, these auxiliary data structures can be beneficially extended to the MTRPP to design fast evaluation techniques for the generalized problem.

The team orienteering problem (TOP) (Chao, Golden, & Wasil, 1996) is another related problem, which states that a fixed number of homogeneous vehicles visit a subset of customers to maximize the collected profits within a traveling distance constraint. Unlike the MTRPP, the profits of customers in the TOP are time independent, and distance constraints exist for the vehicles. Various solution methods, including local search algorithms (Hammami, Rekik, & Coelho, 2020; Tsakirakis, Marinaki, Marinakis, & Matsatsinis, 2019; Vansteenwegen, Souffriau, Berghe, & Van Oudheusden, 2009), population-based algorithms (Bouly, Dang, & Moukrim, 2010; Dang, Guibadj, & Moukrim, 2013; Zettam & Elbenani, 2016), and exact methods based on branch-and-price and the cutting plane technique (Bianchessi, Mansini, & Speranza, 2018; Boussier, Feillet, & Gendreau, 2007; El-Hajj, Dang, & Moukrim, 2016; Poggi, Viana, & Uchoa, 2010), have been developed for the TOP. The cumulative capacitated vehicle routing problem is related to the MTRPP by considering capacity constraints for the K repairmen (or vehicles). Popular algorithms for this problem include evolutionary algorithm (Ngueveu, Prins, & Calvo, 2010), adaptive large neighborhood search heuristic (Ribeiro & Laporte, 2012), two-phase metaheuristic (Ke & Feng, 2013), iterated greedy algorithms (Nucamendi-Guillén, Angel-Bello, Martínez-Salazar, & Cordero-Franco, 2018), and brand-and-cut-and-price algorithm (Lysgaard & Wøhlk, 2014).

The MTRPP with multiple repairmen is a realistic model compared with the TRPP with a single repairman for real-life applications. However, contrary to the TRPP for which numerous solution methods exist, only two principal heuristics have been designed for the MTRPP. Thus, tools for addressing MTRPP should be enhanced. Moreover, the two existing algorithms for the MTRPP are sophisticated and involve many parameters (13 parameters for ALNS-MTRPP and 7 for MA-MTRPP). In addition, ALNS-MTRPP cannot satisfactorily handle large-scale instances (e.g., it requires 3 hours to solve 1000-customer instances).

In this study, we propose an easy-to-use (with only three parameters) and effective hybrid search algorithm based on the memetic framework to solve the MTRPP (named EHSA-MTRPP). We summarize the contributions as follows.

First, we propose an original arc-based crossover (*ABX*), which is inspired by experimental observation and backbone-based heuristics (Wang, Lü, Glover, & Hao, 2013; Zhang, 2004). ABX can be used to generate promising offspring solutions from high-quality parent solutions.

Second, to ensure a high computational effectiveness, we introduce an approximation method to reduce the complexities in examining the neighborhoods and prove that evaluating one neighboring solution in the underlying neighborhoods for the MTRPP can be performed in constant time. Moreover, in order to warrant an accurate evaluation, a correcting mechanism is executed during the search process.

Finally, we provide novel lower bounds for 137 instances out of the 470 benchmark instances in the literature. These bounds can be used for future studies on the MTRPP.

The rest of the paper is organized as follows. The next section is dedicated to the presentation of the proposed algorithm. In Section 3, we describe the experimental setup, parameter tuning, and computational results, followed by an investigation of the key components of the algorithm in Section 4. Conclusions and perspectives are provided in the last section.

2. Method

2.1. Main scheme

The proposed hybrid search algorithm for the MTRPP is based on the framework of the memetic algorithm (Moscato, 1999) and relies on five search components, namely a population initialization procedure (*IniPool*), a variable neighborhood search procedure (*VNS*) to perform the local refinement, a perturbation procedure (*Spert*) to help escape from the local optimum, an arc-based crossover (*ABX*) to generate high-quality offspring solutions, and a pool updating procedure (*UpdatingPool*) to manage the population with newly obtained solutions.

Algorithm 1 presents the general scheme of the EHSA-MTRPP algorithm. First, the algorithm calls *IniPool* (See Section 2.2) to create the population P, where each solution φ_i is improved by *VNS* (See Section 2.3) and the best one is recorded in φ^* (lines 8–12). Next, the algorithm enters the main search procedure (lines 13–32). For the while loop, we set C to 0 (line 14), randomly select two solutions φ_a and φ_b from the population P, and generate an offspring solution φ (lines 15–16) with *ABX* (See Section 2.5). After recording φ by φ_{lb}, the algorithm enters the inner loop (lines 18–27) to investigate the new solutions by iterating the *VNS* procedure and the *Spert* procedure. For each inner loop, the current solution φ is first improved by *VNS* (line 19) and then used to update the local best solution φ_{lb}. If φ is superior to φ_{lb}, φ_{lb} is updated, and the counter C is reset to 0 (lines 20–22). Otherwise, C is incremented by 1 (lines 23–25). Then, the perturbation procedure *Spert* is triggered to displace the search from the local optimum (line 26). The aforementioned procedures are repeated until C reaches the search limit *Limi* (line 27), which indicates that the search is exhausted (and trapped in a deep local optimal solution).

Algorithm 1 General scheme of the EHSA-MTRPP algorithm.

1: **Input**: Input graph $G(V, E)$, population size Np, search limit $Limi$, objective function f and maximum allowed time $T_{\max}$
2: **Output**: Best found solution φ^*
3: /* *IniPool* is used to generate the initial population. */
4: /* *VNS* is used to perform the local refinement. */
5: /* *Spert* is used to modify (slightly) the input local optimum. */
6: /* *ABX* is used to generate promising offspring solutions. */
7: /* *UpdatingPool* is used to update the population. */
8: $P = \{\varphi_1, \ldots \varphi_p\} \leftarrow IniPool()$ // See Section 2.2
9: **for** $i \leftarrow 1$ to Np **do**
10: $\varphi_i \leftarrow VNS(\varphi_i)$ // See Section 2.3
11: **end for**
12: $\varphi^* \leftarrow \arg\max\{f(\varphi_i), i = 1, \ldots, Np\}$
13: **while** $T_{\max}$ is not reached **do**
14: $C \leftarrow 0$
15: $(\varphi_a, \varphi_b) \leftarrow$ RandomChoose(P)
16: $\varphi \leftarrow ABX(\varphi_a, \varphi_b)$ // See Section 2.5
17: $\varphi_{lb} \leftarrow \varphi$
18: **repeat**
19: $\varphi \leftarrow VNS(\varphi)$
20: **if** $f(\varphi) > (\varphi_{lb})$ **then**
21: $\varphi_{lb} \leftarrow \varphi$
22: $C \leftarrow 0$
23: **else**
24: $C \leftarrow C + 1$
25: **end if**
26: $\varphi \leftarrow Spert(\varphi)$ // See Section 2.4
27: **until** $C \geq Limi$
28: $UpdatingPool(\varphi_{lb}, P)$ // See Section 2.6
29: **if** $f(\varphi_{lb}) > (\varphi^*)$ **then**
30: $\varphi^* \leftarrow \varphi_{lb}$
31: **end if**
32: **end while**
33: **return** φ^*

After the inner loop, the local best solution φ_{lb} is used to upgrade the population (line 28) and to update the best solution φ^* (lines 29–31). When the cut-off time $T_{\max}$ is reached (line 13), the algorithm stops and returns the best recorded solution φ^* (line 33).

2.2. Initial population

The initial population is filled with two types of solutions: half of them are created with a randomized construction method, while the remaining solutions are generated with a greedy construction method.

For the randomized construction method, we first create a giant tour with all the customers in a random order. Next, we separate the giant tour into K routes, where each route has the same number of customers. This method leads to a complete solution φ.

We also use the greedy construction method proposed by Avci & Avci (2019). Starting from an empty solution φ with K routes and a vertex list $V_r = \{1, 2, \ldots, n\}$, the greedy construction method iteratively adds one vertex to the solution following a greedy randomized principle. At each step, we evaluate the objective variation of the solution φ for each operation $Ope(v, k)$, which represents adding $v \in V_r$ to the route k. Next, we construct a candidate set OPE_c consisting of the q operations (q is set to three here) with the largest contributions to the objective value. Finally, a random operation $Ope(v, k) \in OPE_c$ is performed to extend the partial solution, and the vertex v was removed from V_r. These steps are repeated until all the customers are added into the solution. For more details, please refer to Avci & Avci (2019).

2.3. Solution improvement by variable neighborhood search

For local optimization, we adopt the general variable neighborhood search (VNS) method (Mladenović & Hansen, 1997), which has proved to be successful for both the TRPP (Avci & Avci, 2017; Lu et al., 2019b; Pei et al., 2020) and the MTRPP (Lu et al., 2019a).

The proposed *VNS* procedure for the MTRPP is presented in Algorithm 2. In the outer loop (lines 6–16), we first initialize the recorded solution φ' with the current solution φ and the neighborhood set S_N with seven neighborhoods N_1-N_7 (lines 7–8). After a local search procedure based on N_{Add} with the current solution (line 9), the search enters the inner loop to explore the best local solutions by alternating between different neighborhoods (lines 10–15). For each inner loop, we randomly select a neighborhood $N \in S_N$ and use it to perform local optimization from the current solution (lines 11–12). Next, an additional local optimization based on N_{Drop} is performed, and the neighborhood N is removed from the neighborhood set S_N (lines 13–14). When the neighborhood set S_N is explored ($S_N = \emptyset$), the inner loop ends. These steps are repeated until there are no improving solutions in the neighborhoods (line 16) and φ is returned (line 17). It is worth emphasizing that the *VNS* procedure employs the first-improving strategy (accepting the first improving solution) and stops when there is no improving solution in the neighborhoods. As it only takes improving solution at each iteration, it is also called "Variable Neighborhood Descent (VND)". The solution obtained from this procedure corresponds usually to a deep local optimum. To escape from the trap, a perturbation phase (See Section 2.4) is triggered to displace the search to a new search area.

Algorithm 2 Local optimization with variable neighborhood search.

1: **Input**: Objective function f and current solution φ
2: **Output**: Local best solution φ
3: /* N_1, N_2, N_3, N_4 represent *Swap*, *Insert*, 2-*opt* and *Or-opt* neighborhoods, respectively. */
4: /* N_5, N_6, N_7 represent respectively *Inter-Swap*, *Inter-Insert* and *Inter*-2-*opt* neighborhoods. */
5: /* N_{Add}, N_{Drop} denote *Add* and *Drop* neighborhoods. */
6: **repeat**
7: $\varphi' \leftarrow \varphi$
8: $S_N \leftarrow \{N_1, N_2, N_3, N_4, N_5, N_6, N_7\}$
9: $\varphi \leftarrow LocalSearch(\varphi, N_{Add})$
10: **while** $S_N \neq \emptyset$ **do**
11: Randomly choose a neighborhood $N \in S_N$
12: $\varphi \leftarrow LocalSearch(\varphi, N)$
13: $\varphi \leftarrow LocalSearch(\varphi, N_{Drop})$
14: $S_N \leftarrow S_N \setminus \{N\}$
15: **end while**
16: **until** $f(\varphi') \geq f(\varphi)$
17: **return** φ

Our *VNS* procedure exploits three sets of nine neighborhoods where seven of them were also used in (Avci & Avci, 2019; Lu et al., 2019a). The first set of four neighborhoods changes the order of customers in one route as follows:

- *Swap* (N_1): The visiting positions of two customers in one route are interchanged.
- *Insert* (N_2): One customer is removed from its position and inserted to the position between two adjacent nodes in the same route.
- 2-*opt* (N_3): Two nonadjacent edges are removed and replaced with two new edges in the same route.

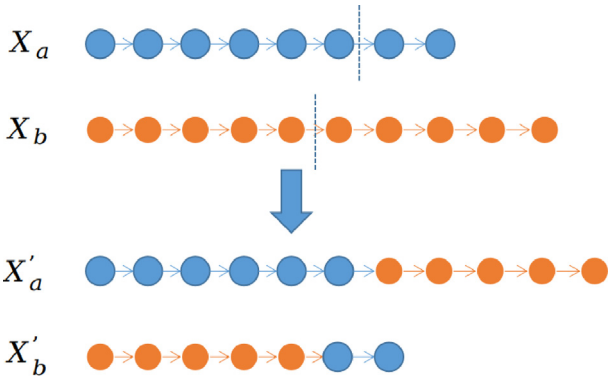

Fig. 1. Illustration of *Inter*-2-*opt*: suppose two routes X_a (marked in blue) and X_b (marked in orange) operating an *Inter*-2-*opt* produce two new routes X'_a and X'_b, where the blue dotted lines represent the edges to be removed. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

- *Or-opt* (N_4): A block of h ($h = 2, 3$) consecutive customers is removed from one route and inserted into two adjacent nodes in the same route.

The second set of three neighborhoods is designed to change the customers between different routes as follows:

- *Inter-Swap* (N_5): The positions of two customers are interchanged in two routes.
- *Inter-Insert* (N_6): One customer is removed from one route and inserted to the position between two adjacent nodes in another route.
- *Inter*-2-*opt* (N_7): Two edges are removed from two routes and replaced with two new edges. A simple illustration is presented in Fig. 1.

The third set of two neighborhoods changes the set of visited customers as follows:

- *Add* (N_{Add}): One unselected customer is added to some position in some route.
- *Drop* (N_{Drop}): One customer is removed from one route.

Notably, Pei et al. (2020) introduced a series of data structures to realize the fast evaluation of the neighboring solutions in the neighborhoods N_1-N_4, N_{Add} and N_{Drop} for solving the related TRPP, but they did not study the neighborhoods N_5-N_7. Here, we extend their method to the neighborhoods for the MTRPP. In practice, each neighboring solution in our algorithm can be evaluated in $O(1)$ (proof given in Appendix Appendix A), which is more efficient than the reference algorithms in the literature (Avci & Avci, 2019; Lu et al., 2019a). The detailed comparisons of the complexities in investigating various neighborhoods between the reference algorithms and the proposed algorithm are discussed in Section 2.7. The complexities of investigating the aforementioned neighborhoods are summarized as follows.

Proposition 1. *In the MTRPP, for the first set of four neighborhoods (N_1-N_4) and the third set of two neighborhoods (N_{Add} and N_{Drop}), the time complexity of evaluating each neighboring solution is $O(1)$. Let n be the number of all customers and m be the number of visited customers in the solution. The time complexities of investigating these neighborhoods are given as follows.*

(a) *Exploring the complete Swap neighborhood requires $O(m^2)$.*
(b) *Exploring the complete Insert neighborhood requires $O(m^2)$.*
(c) *Exploring the complete 2-opt neighborhood requires $O(m^2)$.*
(d) *Exploring the complete Or-opt neighborhood requires $O(m^2 \cdot h)$.*
(e) *Exploring the complete Add neighborhood requires $O(m \cdot (n - m))$.*
(f) *Exploring the complete Drop neighborhood requires $O(m)$.*

Proposition 2. *For the second set of three neighborhoods (N_5-N_7), each neighboring solution can be evaluated in $O(1)$. Let m be the number of visited customers in the solution. The time complexities of exploring these neighborhoods are summarized as follows.*

(a) *Exploring the complete Inter-Swap neighborhood can be finished in $O(m^2)$.*
(b) *Exploring the complete Inter-Insert neighborhood can be finished in $O(m^2)$.*
(c) *Exploring the complete Inter-2-opt neighborhood can be finished in $O(m^2)$.*

Detailed proofs of Propositions 1 and 2 are presented in Appendix Appendix A. With Eq. (2) and a special array in Eq. (A.1), we can efficiently investigate the aforementioned neighborhoods. Notably, the fast evaluation techniques in Propositions 1 and 2 are actually an approximation based on Eq. (2), which is not strictly equivalent to Eq. (1) due to the existence of negative revenue nodes. Therefore, a correcting step is applied to ensure an accurate evaluation of each neighboring solution by applying the *Drop* operator within the local optimization procedure (line 13 in Algorithm 2). The influence of this correcting step is experimentally investigated in Section 4.2.

2.4. Perturbation procedure

To help the search escape from deep local optimum, we apply two operators *Insert* and *Add* to perturb the local optimum. We first perform *St* times the *Insert* operation by randomly selecting a route and inserting some customers to a random position in the route. For the *Add* operation, we randomly add an unvisited customer to the tail of a random route. This procedure is repeated until all the unvisited customers are added to the solution. The parameter *St* is determined by the experiments in Section 3.1. We also tested other perturbation methods, but the proposed method proved to be better.

2.5. Arc-based crossover

In memetic algorithms, crossovers are used to generate diversified offspring solutions from parent solutions at each generation. Generally, a meaningful crossover is expected to inherit useful attributes of the parent solutions and maintain diversity with respect to the parents (Hao, 2012).

Preliminary experiments revealed that the same arcs frequently appear in high-quality solutions (See Section 4.5), which naturally encourages us to preserve these shared arcs (meaningful components) in the offspring solution[1] Following this observation, we propose a dedicated arc-based crossover for the MTRPP.

For a given solution φ with K paths $\{X_1, \ldots, X_K\}$, where each path $X_k = (x^k_0, \ldots, x^k_{m_k})$ contains m_k customers, the corresponding arc set A is defined as follows:

$$A = \left\{ \left(x^k_i, x^k_{i+1}\right) : x^k_i, x^k_{i+1} \in X_k, i \in [0, m_k - 1], k \in [1, K] \right\} \quad (3)$$

Given two parent solutions φ_s and φ_t, let V_s and V_t represent the set of selected customers, and A_s and A_t represent their corresponding arc sets, respectively. The arc-based crossover first copies one parent solution (say φ_s) to the offspring solution, and then randomly inserts 50% of nonshared arcs of φ_t ($A_t \setminus A_s$) into the

[1] Note that preserving and transferring the shared components from the parents to the offspring is the basis of backbone-based crossovers (Wang et al., 2013; Zhang, 2004).

offspring solution, and finally removes the duplicated vertices if needed.

The proposed *ABX* crossover is presented in Algorithm 3. First,

Algorithm 3 Arc-based crossover (*ABX*).

1: **Input**: Input graph $G(V, E)$, parent solutions φ_s and φ_t, the corresponding arc sets of the parents solutions A_s and A_t
2: **Output**: Offspring solution φ_o
3: /* RandomSel-Half-Arcs randomly selects 50% of arcs from a given set of arcs. */
4: /* V_o is the set of the selected customers for φ_o. */
5: /* V_f is the set of nodes, which will be not removed or inserted to other positions in future operations. */
6: $\varphi_o \leftarrow \varphi_s$
7: $V_o \leftarrow V_s$
8: $V_f = \emptyset$
9: $A_u \leftarrow$ RandomSel-Half-Arcs$(A_t \setminus A_s)$
10: **for** Each arc $(a, b) \in A_s \cap A_t$ **do**
11: $V_f \leftarrow V_f \cup \{a, b\}$
12: **end for**
13: **for** Each arc $(a, b) \in A_u$ **do**
14: **if** $a \notin V_o$ and $b \notin V_o$ **then**
15: Insert (a, b) to the tail of some route in φ_o
16: **else if** $a \in V_o$ and $b \notin V_o$ **then**
17: Insert b to the position after a in φ_o
18: **else if** $a \notin V_o$ and $b \in V_o$ **then**
19: Insert a to the position before b in φ_o
20: **else if** $b \notin V_f$ **then**
21: Remove b from φ_o
22: Insert b to the position after a in φ_o
23: **else if** $a \notin V_f$ and $b \in V_f$ **then**
24: Remove a from φ_o
25: Insert a to the position before b in φ_o
26: **end if**
27: $V_o \leftarrow V_o \cup \{a, b\}$
28: $V_f \leftarrow V_f \cup \{a, b\}$
29: **end for**
30: **return** φ_o

φ_s is copied to φ_o, V_s is copied to V_o, V_f is initialized as empty, and an arc set A_u is generated by randomly selecting 50% arcs from $A_t \setminus A_s$ (lines 6–9). To preserve the shared arcs $(a, b) \in A_s \cap A_t$ in the offspring solution φ_o, we add the vertices of these arcs into the set V_f (lines 10–12). The vertices in V_f are not considered in the future operations. Next, we insert each arc $(a, b) \in A_u$ into the offspring solution and remove the duplicated vertices (lines 13–29) according to the following conditions:

(1) If both a and b are not included in V_o, the arc (a, b) is added to the tail of some route (lines 14–15).
(2) If only a is contained in V_o, node b is inserted at the position after a in φ_o (lines 16–17).
(3) If only b belongs to V_o, node a is inserted at the position before b in φ_o (lines 18–19).
(4) If the two nodes are already in V_o and b is not in V_f, we remove b from φ_o and insert it at the position after a (lines 20–22).
(5) If the two nodes are already in V_o and a is not in V_f, we remove a from φ_o and insert it at the position before b (lines 23–26).

Both a and b are added into the set V_o and V_f after the aforementioned operations (lines 27–28), and the whole loop ends when all the arcs $(a, b) \in A_u$ are added into the offspring. Finally, the new generated offspring φ_o is returned (line 30).

Fig. 2 displays an illustrative example of the proposed crossover.

2.6. Pool updating

After the improvement of the offspring solution by the local refinement procedure, the population is updated by the improved offspring solution φ_{lb} (See line 28 in Algorithm 1). In this study, we use a simple strategy: if φ_{lb} differs from all the solutions in the population and is better than the worst solution in terms of the objective value, φ_{lb} replaces the worst solution in the population. Otherwise, φ_{lb} is abandoned. We tested a diversity-quality updating strategy, which considers not only the quality of the newly obtained solution but also its average distance to the other solutions to determine whether to accept φ_{lb} into the population. However, the proposed simple updating strategy exhibited a superior performance.

2.7. Discussion

EHSA-MTRPP differs from the reference algorithms (Avci & Avci, 2019; Lu et al., 2019a) in two aspects.

EHSA-MTRPP employs fast neighborhood evaluation techniques in its local optimization procedure for the MTRPP for the first time. These evaluation techniques ensure a higher computational efficiency of neighborhood examination than those of the existing algorithms such as ALNS-MTRPP (Avci & Avci, 2019) and MA-MTRPP (Lu et al., 2019a). To illustrate this point, Table 1 summarizes the various neighborhoods as well as the complexities in investigating each neighborhood in ALNS-MTRPP and MA-MTRPP.

From Table 1, we clearly remark that the proposed algorithm investigates the used neighborhoods efficiently. Notably, evaluating one neighboring solution in the *Inter-Or-opt* neighborhood and the *Double-bridge* neighborhood (*Double-bridge* is a popular operator for the traveling salesman problem (Lin & Kernighan, 1973)) could also be done in $O(1)$ using Eq. (2) and the auxiliary data structures. However, these neighborhoods are not helpful in improving the performance of our algorithm. Therefore, these two neighborhoods are not used in the proposed algorithm. As these two neighborhoods are widely used in related routing problems, we give the detailed proof of their complexities in Appendix Appendix A.

Additionally, the proposed algorithm adopts a dedicated arc-based crossover, which can generate new offspring solutions inheriting meaningful components (the shared arcs) and diversified from the parent solutions. MA-MTRPP (Lu et al., 2019a) applies a route-based crossover (RBX) (Potvin & Bengio, 1996), which simply copies one parent solution to the offspring solution, replaces some route of the offspring solution with a route from another parent solution, and removes the duplicated vertices if required. Our experiments and observations revealed that the key components of the solutions are the "arcs" and not the "routes," rendering ABX more appropriate than RBX for solving the MTRPP. Experimental results in Section 4.4 confirm these observations and demonstrate the effectiveness of the proposed algorithm with ABX compared to its variant with RBX.

3. Computational results and comparative study

This section presents computational experiments over the benchmark instances in the literature to evaluate the EHSA-MTRPP algorithm.

3.1. Instances, reference algorithms and parameter setting

Our computational experiments are based on two groups of 470 benchmark instances from Avci & Avci (2019) (230 instances denoted by Ins_Avci) and Lu et al. (2019a) (240 instances denoted by Ins_Lu)[2]

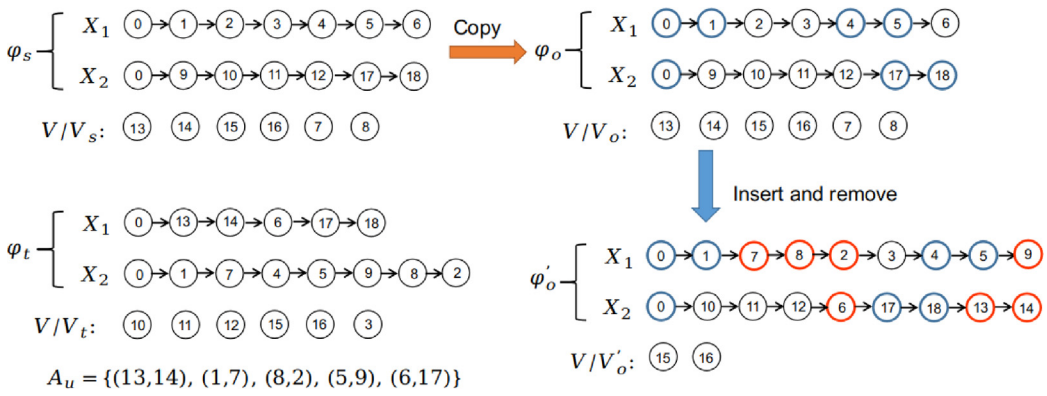

Fig. 2. Illustration of *ABX* on a 18-customer instance with two routes. φ_s and φ_t are two parent solutions, φ_o is the solution copied from φ_s, and φ'_o is the generated offspring solution. A_u is the set of arcs, which are randomly selected from the nonshared arcs of φ_t. Here, $V \setminus V_i$ represents the set of unselected customers for the solution φ_i, which can be φ_s, φ_t, φ_o, and φ'_o. The nodes of the shared arcs between φ_s and φ_t are marked in blue, while the nodes involved in inserting and removing are marked in red. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 1
Summary of the neighborhood structures as well as their complexities in the reference algorithms and the proposed algorithm, where n depicts the number of customers, m is the number of selected customers, and h is the number of consecutive customers in the block for N_4.

Neighborhood	ALNS-MTRPP (Avci & Avci, 2019)		MA-MTRPP (Lu et al., 2019a)		EHSA-MTRPP	
	Employment	Complexity	Employment	Complexity	Employment	Complexity
Swap	√	$O(m^2 \lg n)$	√	$O(m^3)$	√	$O(m^2)$
Insert	√	$O(m^2 \lg n)$	√	$O(m^3)$	√	$O(m^2)$
2-opt	√	$O(m^2 \lg n)$	√	$O(m^3)$	√	$O(m^2)$
Or-opt	√	$O(m^2 \lg n)$	√	$O(m^3)$	√	$O(m^2 \cdot h)$
Inter-Swap	√	$O(m^2 \lg n)$	√	$O(m^3)$	√	$O(m^2)$
Inter-Insert	√	$O(m^2 \lg n)$	√	$O(m^3)$	√	$O(m^2)$
Inter-2-opt	√	$O(m^2 \lg n)$	√	$O(m^3)$	√	$O(m^2)$
Inter-Or-opt	√	$O(m^2 \lg n)$	√	$O(m^3)$	×	-
Add	×	-	×	-	√	$O(m \cdot (n-m))$
Drop	×	-	×	-	√	$O(m)$
Double-bridge	×	-	√	$O(m^3)$	×	-

The 230 Ins_Avci instances were divided into 14 sets of instances according to the number of customers and servers (repairmen). The first ten sets of instances were converted from instances of the TRPP (Dewilde et al., 2013) (n=10, 20, 50, 100, 200) by considering two and three servers. As each set of instances of the TRPP is composed of 20 instances, the authors have created $5 \times 2 \times 20 = 200$ instances. The other four sets of instances are of larger sizes, including 10 instances with 500 customers and 10 servers, 10 instances with 500 customers and 20 servers, five instances with 750 customers and 100 servers, and five instances with 1000 customers and 50 servers.

The 240 Ins_Lu instances are also based on instances of the TRPP (Dewilde et al., 2013) and were divided into 12 sets of instances (with n=20, 50, 100, 200 and two, three, and four servers), where each set is composed of 20 instances. Unlike the Ins_Avci instances, Lu et al. (2019a) adjusted the profit for each customer of the instances to ensure a high-quality solution to hold approximately 75% to 95% of all the customers. Therefore, each customer i was assigned a non-negative profit p_i, which is a random integer between $\lceil d_{0,i} \rceil$ and $\lceil \frac{n}{k} \times \frac{\sum_{(i,j)\in E} d_{i,j}}{|E|} \rceil$, where n is the number of all customers and k is the number of servers.

The proposed EHSA-MTRPP algorithm was programmed in C++ and compiled with the g++ 7.5.0 compiler and the -O3 optimization flag[3] All the experiments reported in this work were performed on a computer with Intel Xeon(R) E5-2695 processor (2.1 GHz CPU and 2 GB RAM). The experimental environments of the reference algorithms are listed as follows.

- The ALNS-MTRPP algorithm (Avci & Avci, 2019) was coded in Matlab 9.1.0, and run on a personal computer equipped with Intel(R) Core (TM) i7-5500U processor (2.4 GHz and 8 GB RAM).
- The MA-MTRPP algorithm (Lu et al., 2019a) was programmed in C++ and compiled with g++. The experiments were executed on a computer with an Intel Xeon(R) CPU E5-2695 processor (2.1 GHz CPU and 2 GB RAM).

The reference results for the ALNS-MTRPP algorithm are extracted from Avci & Avci (2019) for the 230 Ins_Avci instances, while the results for the MA-MTRPP algorithm are from Lu et al. (2019a) for the 240 Ins_Lu instances. Unfortunately, the source codes of these reference algorithms are not available. Therefore, to ensure a fair comparison, we performed two experiments on the two groups of instances. Following the experimental setup in the literature, EHSA-MTRPP was run independently five times with different seeds on each Ins_Avci instance while 10 times on the Ins_Lu instances. The stopping condition in the literature is a prefixed maximum number of iterations, whereas in EHSA-MTRPP a maximum cut-off time is used. The average running time of ALNS-MTRPP (Avci & Avci, 2019) was typically several hours for large instances, and the average time to get the best solution for MA-MTRPP (Lu et al., 2019a) was approximately 300 and 500 seconds for the 200 customer instances. For fair comparisons, we set our cut-off time $T_{\max}$ to be twice the number of customers (in seconds). In practice, EHSA-MTRPP can attain superior solutions than

[2] These instances are available from: https://github.com/REN-Jintong/MTRPP.
[3] The code of our algorithm will be available at the Github page of footnote [2].

Table 2
Parameters of EHSA-MTRPP tuned with the Irace package.

Parameter	Description	Type	Value range
Limi	Search limit	Integer	[0, 30]
St	Strength of the *Insert* perturbation	Integer	[0, 100]
Np	Number of population	Categorical	{6, 8, 10, 20, 50, 100}

Table 3
Results of the reference algorithm ALNS-MTRPP(Avci & Avci, 2019) and EHSA-MTRPP on the 230 Ins_Avci instances. Each instance was solved five times according to Avci & Avci (2019). The optimal solutions for the instances of "Size $= 10, K = 2, 3$" are known, but their timing information is not available.

Size	K	ALNS-MTRPP			EHSA-MTRPP			p-value	δ	W	M	F
		Best	Average	Tavg	Best	Average	Tavg					
10	2	2114.85	2114.85	0.00	2114.85	2114.85	0.03	NA	0.000%	0	20	0
10	3	2230.60	2230.60	0.00	2230.60	2230.60	0.03	NA	0.000%	0	20	0
20	2	9074.60	9074.60	3.15	9074.60	9074.60	0.06	NA	0.000%	0	20	0
20	3	9450.45	9450.45	3.10	9450.45	9450.45	0.06	NA	0.000%	0	20	0
50	2	55469.15	55468.55	35.50	55469.15	55469.15	0.82	NA	0.000%	0	20	0
50	3	57184.85	57184.45	30.85	**57185.35**	**57185.35**	0.78	3.17×10^{-1}	0.001%	1	19	0
100	2	226899.95	226895.80	346.45	**226900.95**	**226900.47**	22.96	1.02×10^{-1}	0.000%	0	20	0
100	3	231954.05	231947.30	551.05	**231958.70**	**231954.23**	29.80	1.80×10^{-1}	0.002%	1	19	0
200	2	893183.35	892864.45	3600.00	**893513.85**	**893374.88**	263.23	1.20×10^{-4}	0.037%	19	0	1
200	3	907775.35	907611.55	3600.00	**907950.35**	**907841.50**	258.94	8.84×10^{-5}	0.019%	20	0	0
500	10	1428716.30	1422361.10	10800.00	**1437256.40**	**1436265.76**	898.97	5.06×10^{-3}	0.598%	10	0	0
500	20	692074.30	688804.60	10800.00	**694406.60**	**694114.40**	897.70	5.06×10^{-3}	0.337%	10	0	0
750	100	4000199.00	3966184.40	43200.00	**4000585.60**	**4000541.60**	1352.60	4.31×10^{-2}	0.010%	5	0	0
1000	50	5186645.80	5066567.40	43200.00	**5191726.40**	**5191527.76**	1757.60	4.31×10^{-2}	0.098%	5	0	0
Avg.		500212.50	496401.17	3527.83	**500848.55**	**500765.52**	195.88					

the reference algorithms with less time for most of the benchmark instances.

To determine the parameters listed in Table 2, we used an automatic parameter tuning tool Irace (López-Ibáñez, Dubois-Lacoste, Cáceres, Birattari, & Stützle, 2016). In this experiment, we selected 10 large and difficult instances as the training instances and set the maximum number of runs (tuning budget) to 2000.

According to the tuning experiment, the parameters determined by Irace are $Limi = 2$, $St = 11$, and $Np = 10$. We used this parameter setting for EHSA-MTRPP for all our computational experiments.

3.2. Comparative studies

This section presents the experimental results obtained by EHSA-MTRPP with respect to the reference algorithms (Avci & Avci, 2019; Lu et al., 2019a) over the two groups of 470 benchmark instances.

Table 3 lists the overall results of the reference algorithm ALNS-MTRPP and our EHSA-MTRPP algorithm on the 230 Ins_Avci instances (better results are indicated in bold). Columns "Size" and "K" display the numbers of customers and servers. Columns "Best," "Average," and "Tavg" (columns 3–5) denote the best found results, average found results, and average time to obtain the best found solutions, respectively, for ALNS-MTRPP. The following three columns depict the same information for EHSA-MTRPP (all the aforementioned values are averaged over the instances of each set). Column "p-value" lists the results of the Wilcoxon signed rank tests of the best found results (column "Best") between ALNS-MTRPP and EHSA-MTRPP, where "NA" indicates no difference between the two groups of results. Next, column "δ" presents an improvement in the percentage of the best objective value found by EHSA-MTRPP over the best objective value of ALNS-MTRPP. The last three columns list the number of instances for which the EHSA-MTRPP algorithm improved ("W"), matched ("M"), or failed ("F') to attain the best found results reported in Avci & Avci (2019). Finally, row "Avg." depicts the average values of the corresponding indicators.

From row "Avg." of Table 3, we remark that EHSA-MTRPP outperformed ALNS-MTRPP in terms of the best found results and the average found results. The two algorithms exhibited the same performance for the first five sets of instances (instances of small sizes), whereas for the remaining nine sets of instances, EHSA-MTRPP outperformed the reference algorithm ALNS-MTRPP both in terms of solution quality ("Best" and "Average") and running time (column "Tavg"). In particular, the results of the Wilcoxon signed rank test (column "p-value") revealed that a considerable difference exists between the best found results between ALNS-MTRPP and EHSA-MTRPP over the last six set of instances (p-value$<$0.05). Overall, EHSA-MTRPP clearly dominated ALNS-MTRPP by updating the best records (new lower bounds) for 71 instances, matching the best-known results for 158 instances, and only missing one best-known result.

Using similar column headings as Table 3, Table 4 summarizes the overall results of MA-MTRPP and EHSA-MTRPP on the 240 Ins_Lu instances.

Row "Avg." in Table 4 reveals that EHSA-MTRPP achieved a superior performance (column "Best" and "Average") to that of MA-MTRPP with a shorter average time (66.45 seconds compared to the 102.64 seconds achieved by MA-MTRPP). For each set of instances, EHSA-MTRPP exhibited a superior or equal performance in terms of the best found results and average found results. In particular, the proposed algorithm outperformed MA-MTRPP on the last three sets of large instances confirmed by the Wilcoxon signed rank test (p-value $<$ 0.05). In addition, EHSA-MTRPP required less time (column "Tavg") than MA-MTRPP did to attain the best found solutions for each set of instances. Overall, EHSA-MTRPP updated 66 best records (new lower bounds), matched the best-known results for 172 instances, and missed only two best-known results.

In summary, EHSA-MTRPP provided considerably superior results than the reference algorithms on the 470 benchmark instances by establishing 137 new record results (29%) and matching best-known results for 330 instances (70%). The detailed comparisons between the reference algorithms (Avci & Avci, 2019; Lu et al., 2019a) and EHSA-MTRPP are available at https://github.com/REN-Jintong/MTRPP.

Table 4
Results of the reference algorithm MA-MTRPP (Lu et al., 2019a) and EHSA-MTRPP on the instances of Ins_Lu. Each instance was solved 10 times according to Lu et al. (2019a). The optimal solutions for small instances ("Size = 20, $K = 2, 3, 4$") are known.

Size	K	MA-MTRPP			EHSA-MTRPP			p-value	δ	W	M	F
		Best	Average	Tavg	Best	Average	Tavg					
20	2	3937.60	3937.60	1.31	3937.60	3937.60	0.05	NA	0.000%	0	20	0
20	3	2399.20	2399.20	1.29	2399.20	2399.20	0.05	NA	0.000%	0	20	0
20	4	1733.40	1733.40	1.23	1733.40	1733.40	0.06	NA	0.000%	0	20	0
50	2	27172.15	27172.15	7.38	**27173.55**	**27173.55**	1.02	1.09×10^{-1}	0.005%	3	17	0
50	3	17523.55	17523.55	6.26	17523.55	17523.55	0.66	NA	0.000%	0	20	0
50	4	13049.05	13049.05	5.72	**13049.25**	**13049.25**	0.75	1.80×10^{-1}	0.002%	2	18	0
100	2	113566.35	113560.76	46.13	**113567.10**	**113566.60**	21.44	1.80×10^{-1}	0.001%	2	18	0
100	3	76976.35	76972.48	37.85	**76976.65**	**76976.24**	23.31	3.17×10^{-1}	0.000%	1	19	0
100	4	57188.40	57186.69	32.55	**57188.55**	**57188.53**	21.07	3.17×10^{-1}	0.000%	1	19	0
200	2	472301.40	472002.08	455.39	**472499.25**	**472354.94**	254.16	1.03×10^{-4}	0.042%	19	0	1
200	3	321136.55	320912.21	358.27	**321278.75**	**321175.57**	245.66	8.86×10^{-5}	0.044%	20	0	0
200	4	236694.15	236539.09	278.37	**236805.20**	**236720.93**	229.22	1.55×10^{-4}	0.047%	18	1	1
Avg.		111973.18	111915.69	102.64	**112011.00**	**111983.28**	66.45					

Table 5
Results of EHSA-MTRPP-NoFast and EHSA-MTRPP on large benchmark instances. Each instance was solved 10 times, and the cut-off time was set to be twice the number of customers.

Size	K	EHSA-MTRPP-NoFast			EHSA-MTRPP			p-value	δ	W	M	F
		Best	Average	Tavg	Best	Average	Tavg					
Ins_Avci												
200	2	893247.90	892891.18	169.58	**893513.85**	**893374.88**	263.23	1.03×10^{-4}	0.030%	19	0	1
200	3	907732.95	907532.35	169.48	**907950.35**	**907841.50**	258.94	8.86×10^{-5}	0.024%	20	0	0
500	10	1431017.90	1428561.78	500.05	**1437256.40**	**1436265.76**	898.97	5.06×10^{-3}	0.436%	10	0	0
500	20	692814.30	691839.16	500.21	**694406.60**	**694114.40**	897.70	5.06×10^{-3}	0.230%	10	0	0
750	100	3999139.60	3948547.72	754.74	**4000585.60**	**4000541.60**	1352.60	4.31×10^{-2}	0.036%	5	0	0
1000	50	5180266.20	5116501.24	1003.76	**5191726.40**	**5191527.76**	1757.60	4.31×10^{-2}	0.221%	5	0	0
Ins_Lu												
200	2	472354.10	471965.12	325.62	**472499.25**	**472354.94**	254.16	7.80×10^{-4}	0.031%	19	0	1
200	3	321165.95	320919.83	327.09	**321278.75**	**321175.57**	245.66	1.32×10^{-4}	0.035%	19	1	0
200	4	236709.90	236544.92	330.46	**236805.20**	**236720.93**	229.22	1.32×10^{-4}	0.040%	19	1	0

4. Additional results

This section first presents additional results to demonstrate the critical roles of the fast evaluation technique, the *Drop* operator and the arc-based crossover for the proposed algorithm. Furthermore, we experimentally compared the proposed algorithm to the variant with RBX and revealed the rationale behind the proposed crossover.

4.1. Influence of the fast evaluation technique in the neighborhood structure

To investigate the influence of the fast evaluation technique on our algorithm, we created a variant of EHSA-MTRPP where the fast evaluation technique (named EHSA-MTRPP-NoFast) was disabled. We used the parameters in Section 3.1 and ran both EHSA-MTRPP-NoFast and EHSA-MTRPP independently 10 times on each large-size instance ($n \geq 200$). The cut-off time was set to be twice the number of customers. Using similar column headings as Table 4, Table 5 summarizes the comparative results of EHSA-MTRPP-NoFast and EHSA-MTRPP over large instances from Ins_Avci and Ins_Lu (Better results are marked in bold).

From columns "Best" and "Average" in Table 5, one can conclude that EHSA-MTRPP outperformed EHSA-MTRPP-NoFast for each set of instances, which was also confirmed by the Wilcoxon signed rank tests (p-value< 0.05).

To illustrate the effectiveness of the fast evaluation technique, we performed another experiment by running both algorithms independently 10 times on nine instances of various sizes and recorded the numbers of the visited neighboring solutions. The cut-off time was also set to be twice the number of customers. Fig. 3 displays the average ratio of the visited solutions of EHSA-MTRPP over EHSA-MTRPP-NoFast for the instances of different sizes. EHSA-MTRPP visited more neighboring solutions than EHSA-MTRPP-NoFast did for all selected instances. The dominance of our algorithm with the fast evaluation technique becomes even more clear as the size of the instance increases. In summary, the results in Table 5 and in Fig. 3 demonstrate that the fast evaluation technique can help the proposed algorithm to efficiently explore the search space and contributes to the performance of the proposed algorithm.

4.2. Influence of the Drop operator in the neighborhood structure

As shown in Section 2.3, the *Drop* operator (line 13 in Algorithm 2) is applied after the local optimization with a neighborhood to eliminate the negative revenue nodes and get more accurate fitness function by the fast evaluation technique. This section investigates the influences of the *Drop* operator on the performance of the proposed algorithm, by focusing on the following questions: what happens if the *Drop* operator is disabled or if it is applied very frequently after each solution transition?

To answer these questions, we first studied the impacts of the *Drop* operator over the VNS procedure. For this, we extracted the VNS procedure from EHSA-MTRPP to create two variants: VNS-NoDrop by deleting the *Drop* operator (line 13 in Algorithm 2) and VNS-EachDrop by applying the *Drop* operator after each move (i.e., after each solution transition) in the local optimization. We ran the two variants VNS-NoDrop and VNS-EachDrop to solve 9 representative instances (100 runs per variant and per instance). The best found solutions and the running time were recorded.

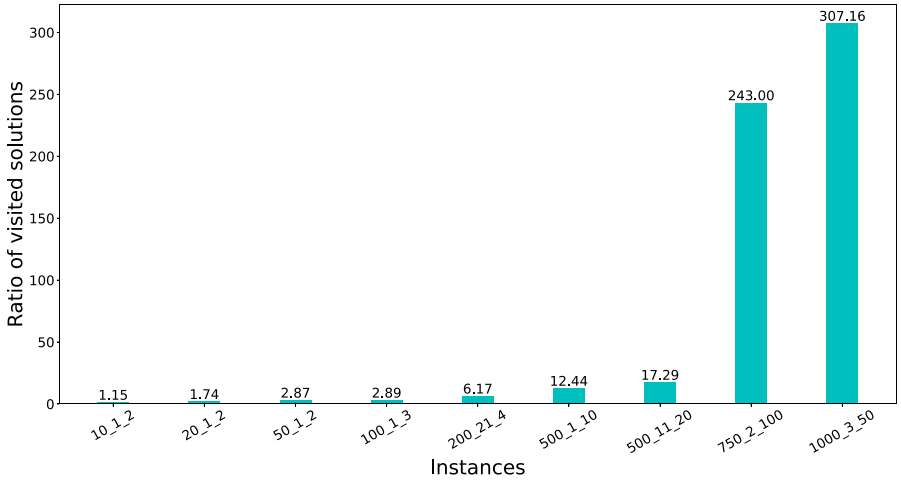

Fig. 3. Average ratio of the visited solutions of EHSA-MTRPP over EHSA-MTRPP-NoFast for nine instances of different sizes. "Size_index_K" for each instance indicates the number of customers, the instance index, and the number of routes. Each instance was solved 10 times independently, and the cut-off time was set to twice the number of customers.

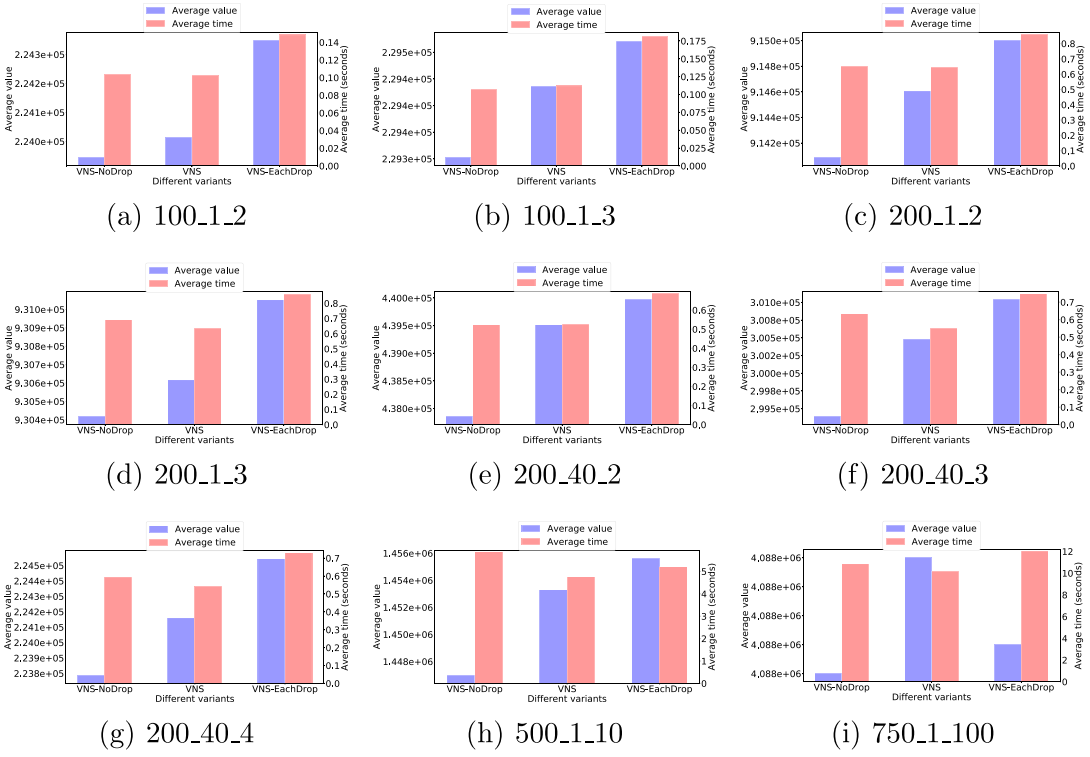

Fig. 4. Bar charts of VNS-NoDrop, VNS and VNS-EachDrop for solving 9 representative instances. The results were averaged over 100 independent executions of each compared algorithm.

Fig. 4 summarizes the corresponding bar charts that describe the averaged best objective values (blue bars) and average running time (red bars) of VNS with respect to its variants VNS-NoDrop and VNS-EachDrop. We observe that VNS and VNS-EachDrop outperform VNS-NoDrop in terms of the best found solutions for all the cases. In particular, VNS uses equal or less time to achieve better results than VNS-NoDrop. Between VNS and VNS-EachDrop, VNS-EachDrop obtains better results (except for 750_1_100) than VNS but uses more time. In summary, the *Drop* operator plays a positive role to the local optimization procedure. By more frequently applying the *Drop* operator (eliminating the influence of the negative revenue nodes), VNS-EachDrop finds better results than VNS while requiring more computation time.

To assess the impacts of the *Drop* operator within the EHSA-MTRPP algorithm, we created an algorithmic variant EHSA-MTRPP-EachDrop by replacing VNS with VNS-EachDrop in Algorithm 1 (line 19) and tested EHSA-MTRPP-EachDrop on the large benchmark instances using the same experimental setup in Section 3.1. The comparative results of EHSA-MTRPP-EachDrop and EHSA-MTRPP are shown in Table 6. For the best found results, EHSA-MTRPP-EachDrop performed better than EHSA-MTRPP over 3 sets of instances, but the statistically significant differences were not confirmed by the Wilcoxon signed rank tests (p-value$>$0.05). On the other hand, EHSA-MTRPP had a better performance over the remaining instances, and dominated EHSA-MTRPP-EachDrop on three sets of them (confirmed by p-value$<$0.05). For the av-

Table 6
Results of EHSA-MTRPP-EachDrop and EHSA-MTRPP on large benchmark instances. Each instance was solved 10 times, and the cut-off time was set to be twice the number of customers.

Size	K	EHSA-MTRPP-EachDrop			EHSA-MTRPP			p-value	δ	W	M	F
		Best	Average	Tavg	Best	Average	Tavg					
Ins_Avci												
200	2	**893515.30**	893323.11	130.83	893513.85	**893374.88**	263.23	7.06×10^{-1}	-0.000%	7	6	7
200	3	907921.85	907815.46	118.07	**907950.35**	**907841.50**	258.94	2.31×10^{-2}	0.003%	11	7	2
500	10	1436975.30	1435948.48	464.07	**1437256.40**	**1436265.76**	898.97	3.86×10^{-1}	0.020%	6	0	4
500	20	694387.30	694093.98	453.27	**694406.60**	**694114.40**	897.70	9.59×10^{-1}	0.003%	4	0	6
750	100	4000562.40	4000520.92	708.07	**4000585.60**	**4000541.60**	1352.60	4.31×10^{-2}	0.001%	5	0	0
1000	50	**5191788.20**	**5191568.52**	822.56	5191726.40	5191527.76	1757.60	3.45×10^{-1}	-0.001%	2	0	3
Ins_Lu												
200	2	**472508.80**	472309.32	220.51	472499.25	**472354.94**	254.16	9.53×10^{-1}	-0.002%	5	11	4
200	3	321278.25	321140.49	213.01	**321278.75**	**321175.57**	245.66	8.59×10^{-1}	0.000%	6	11	3
200	4	236794.90	236682.46	206.16	**236805.20**	**236720.93**	229.22	4.69×10^{-2}	0.004%	7	10	3

Table 7
Results of ILS-MTRPP and EHSA-MTRPP on large instances from the benchmark. Each instance was solved 10 times, and the cut-off time was set to be twice the number of customers.

Size	K	ILS-MTRPP			EHSA-MTRPP			p-value	δ	W	M	F
		Best	Average	Tavg	Best	Average	Tavg					
Ins_Avci												
200	2	893225.25	892997.62	106.16	**893513.85**	**893374.88**	263.23	8.86×10^{-5}	0.032%	20	0	0
200	3	907796.10	907666.30	97.70	**907950.35**	**907841.50**	258.94	8.84×10^{-5}	0.017%	20	0	0
500	10	1435567.80	1434542.62	213.20	**1437256.40**	**1436265.76**	898.97	5.06×10^{-3}	0.118%	10	0	0
500	20	693796.30	693456.04	215.12	**694406.60**	**694114.40**	897.70	5.06×10^{-3}	0.088%	10	0	0
750	100	4000456.40	4000414.68	320.81	**4000585.60**	**4000541.60**	1352.60	4.31×10^{-2}	0.003%	5	0	0
1000	50	5191550.60	5191326.92	520.71	**5191726.40**	**5191527.76**	1757.60	4.31×10^{-2}	0.003%	5	0	0
Ins_Lu												
200	2	472282.15	472016.28	198.16	**472499.25**	**472354.94**	254.16	2.93×10^{-4}	0.046%	19	0	1
200	3	321140.15	320982.75	201.67	**321278.75**	**321175.57**	245.66	8.84×10^{-5}	0.043%	20	0	0
200	4	236703.05	236592.84	199.80	**236805.20**	**236720.93**	229.22	8.84×10^{-5}	0.043%	20	0	0

erage results, EHSA-MTRPP dominated the EHSA-MTRPP-EachDrop almost over all sets of instances.

Finally, we investigated the impacts of the *Drop* operator over the EHSA-MTRPP algorithm in terms of time-consumption. We observe that the *Drop* operator generally consumed more time with EHSA-MTRPP-EachDrop than with EHSA-MTRPP. As an example, the *Drop* operator consumed 3.45% of the total time for EHSA-MTRPP-EachDrop against 0.20% for EHSA-MTRPP, when they were used to solve the large instance 1000_1_50. Nevertheless, calling the *Drop* operator after each move (in EHSA-MTRPP-EachDrop) doesn't excessively increase the time given that *Drop* has a linear time complexity. Instead, the most time-consuming component in both algorithms is the local search with the other neighborhoods (line 12, Algorithm 2).

To sum, EHSA-MTRPP has a slightly better performance than EHSA-MTRPP-EachDrop on the benchmark instances. However, EHSA-MTRPP-EachDrop also provides a number of results better than EHSA-MTRPP. We conclude that EHSA-MTRPP-EachDrop is a viable alternative for solving the MTRPP.

4.3. Influence of the crossover operator

This section explores the contributions of the arc-based crossover to our algorithm. We created an EHSA-MTRPP variant (ILS-MTRPP) by disabling ABX (lines 15–16) in Algorithm 1. Using the same experimental setup in Section 3.1, another experiment was performed on the benchmark instances of large size, and the results are presented in Table 7 with the same column headings as Table 5 (better results are marked in bold).

Columns "Best" and "Average" in Table 7 indicate that EHSA-MTRPP clearly dominated ILS-MTRPP for each set of instances (only missing one instance in "Size=200, K=2" in Ins_Lu). The dominance is confirmed by the results of the Wilcoxon signed rank tests (p-value<0.05). This experiment revealed that ABX contributed positively to the performance of our EHSA-MTRPP algorithm.

4.4. Compared to the route-based crossover (RBX) in the literature

This section compares the arc-based crossover and the route-based crossover, which is used in the reference algorithm MA-MTRPP (Lu et al., 2019a). We created an EHSA-MTRPP variant (EHSA-MTRPP-RBX) by replacing ABX with RBX (line 16 in Algorithm 1). On each large-size instance, both algorithms were independently run 10 times using the parameters in Section 3.1. The cut-off time was always set to be twice the number of customers. The results are summarized in Table 8, which uses the same column headings as Table 5 (Better results are indicated in bold).

The results revealed that with the exception of a few cases, EHSA-MTRPP outperformed EHSA-MTRPP-RBX on all sets of instances in terms of the best found results (column "Best") and the average found results (column "Average"), and the Wilcoxon signed rank tests (p-value<0.05) indicate that significant differences exist for eight sets of results (except for the instances of "Size=750, K=100"). This experiment confirmed that the ABX crossover is more appropriate than RBX for the MTRPP.

4.5. Rationale behind the arc-based crossover

We experimentally investigated the rationale behind ABX by analyzing the structural similarities between high-quality solutions. For two given solutions φ_1 and φ_2 with their corresponding arc sets A_1 and A_2, their similarity is defined by $Sim(\varphi_1,\varphi_2)=\frac{|A_1\cap A_2|}{|A_1\cup A_2|}$. Generally, the larger the similarity between two solutions is, the more arcs they share.

We ran the EHSA-MTRPP algorithm 100 times on each of the selected 16 instances (in different sizes) while recording the best

Table 8
Results of EHSA-MTRPP-RBX and EHSA-MTRPP on large instances from the benchmark. Each instance was solved 10 times, and the cut-off time was set to be twice the number of customers.

Size	K	EHSA-MTRPP-RBX			EHSA-MTRPP			p-value	δ	W	M	F
		Best	Average	Tavg	Best	Average	Tavg					
Ins_Avci												
200	2	893420.60	893254.29	156.76	**893513.85**	**893374.88**	263.23	2.35×10^{-4}	0.010%	19	0	1
200	3	907886.65	907784.43	154.88	**907950.35**	**907841.50**	258.94	1.28×10^{-3}	0.007%	18	1	1
500	10	1436503.90	1435606.38	379.31	**1437256.40**	**1436265.76**	898.97	5.06×10^{-3}	0.052%	10	0	0
500	20	694289.30	693986.20	406.29	**694406.60**	**694114.40**	897.70	5.06×10^{-3}	0.017%	10	0	0
750	100	4000559.80	4000520.52	641.77	**4000585.60**	**4000541.60**	1352.60	2.25×10^{-1}	0.001%	4	0	1
1000	50	5191587.40	5191416.32	751.44	**5191726.40**	**5191527.76**	1757.60	4.31×10^{-2}	0.003%	5	0	0
Ins_Lu												
200	2	472466.60	472286.13	300.66	**472499.25**	**472354.94**	254.16	3.76×10^{-3}	0.007%	17	1	2
200	3	321230.30	321127.52	297.78	**321278.75**	**321175.57**	245.66	3.40×10^{-4}	0.015%	18	1	1
200	4	236781.60	236703.33	302.65	**236805.20**	**236720.93**	229.22	7.37×10^{-4}	0.010%	16	2	2

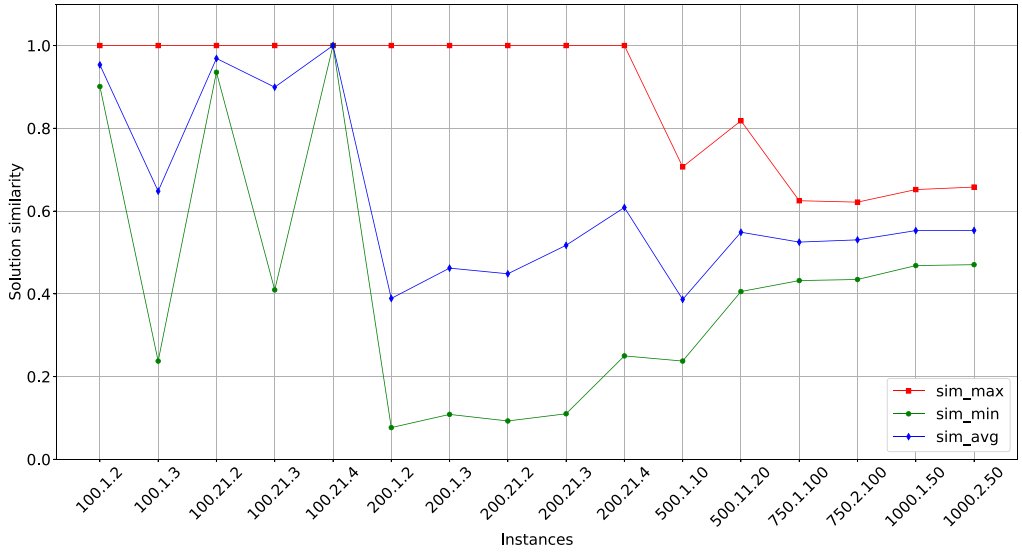

Fig. 5. Similarity between high-quality solutions for 16 instances of different sizes. Each instance was solved 100 times independently with a cut-off time per run set to twice the number of customers.

found solution in each run, and the cut-off time per run was always set to twice the number of customers. For each instance, we calculated the maximum similarity (denoted *sim*_max) between any two solutions by $sim\_\max = \max_{1\le i<j\le 100} Sim(\varphi_i, \varphi_j)$, the minimum similarity (denoted *sim*_min) between any two solutions by $sim\_\min = \min_{1\le i<j\le 100} Sim(\varphi_i, \varphi_j)$, and the average similarity (denoted *sim_avg*) between any two solutions by $sim\_avg = \frac{1}{4950} \cdot \sum_{1\le i<j\le 100} Sim(\varphi_i, \varphi_j)$. Fig. 5 displays the results of the solution similarities for various instances.

From Fig. 5, one concludes that a high similarity exists between high-quality solutions. In particular, the maximum similarity of the 100 high-quality solutions was more than 0.6, and the average similarity was over 0.4 for each instance. Thus, numerous arcs frequently appeared in high-quality solutions, which provides a solid foundation for the design of the arc-based crossover in this work. The maximum similarities for the last six largest instances ($n \ge 500$) were not as high as the other instances ($n \le 200$). This phenomenon could be attributed to the unsatisfactory results for these difficult instances ($n \ge 500$).

5. Conclusions

An effective hybrid search algorithm for the MTRP with profit was proposed under the framework of the memetic algorithm. The proposed algorithm is unique from the existing algorithms in terms of three key features, namely its fast neighborhood evaluation techniques designed to quickly and approximately examine the neighborhoods, a correcting procedure to ensure an accurate evaluation of the neighboring solutions by using the *Drop* operator and the dedicated arc-based crossover that generates diversified and meaningful offspring solutions.

The assessment on the 470 benchmark instances in the literature revealed that the performance of the proposed algorithm competed favorably with the existing algorithms by updating the best records (new lower bounds) for 137 instances (29%) and matching the best-known results for 330 instances (70%) within a reasonable time. Additional experiments revealed that the fast evaluation technique, the *Drop* operator and the arc-based crossover play positive roles in terms of influencing the performance of the algorithm. We analyzed both formally and experimentally the reduced complexities of neighborhood examinations and explored the influence of the correcting step (the frequency of calling *Drop* operator) on the algorithm performance. Besides that, we also provided experimental evidences (high similarity between high-quality solutions) to support the design of the arc-based crossover. The source code of our algorithm will be made available upon the publication of this paper. It can be used to solve practical applications and adapted to related problems. In the future, we will develop efficient algorithms based on the arc-based crossover for some other related problems such as the TOP.

Acknowledgement

We are grateful to the reviewers for their useful comments and suggestions which helped us to significantly improve the paper. This work was partially supported by the Shenzhen Science and Technology Innovation Commission (grant no. JCYJ20180508162601910), the National Key R&D Program of China (grant no. 2020YFB1313300), and the Funding fromthe Shenzhen Institute of Artificial Intelligence and Robotics for Society (grant no. AC01202101110). We also thank Chenyou Fan for his help.

Appendix A. Proof of the complexity of neighborhood exploration

Proof of Proposition 1. (Section 2.3). Pei et al. (2020) has proved that for the related TRPP, evaluating one neighboring solution of *Insert*, 2-*opt*, *Or-opt*, *Add* and *Drop* can be finished in $O(1)$ by using some specific data structures. It is easy to find that these conclusions are equally applicable in the MTRPP because the operations involved in N_1-N_4 as well as N_{Add} and N_{Drop} are confined inside one route, which is the same situation as in the TRPP.

For the *Swap* operator, the proof is given as follows. Let φ be a solution composed of K routes $\{X_1, X_2, \ldots, X_K\}$, where $X_k = (x_0^k, \ldots, x_{i-1}^k, x_i^k, x_{i+1}^k, \ldots, x_{j-1}^k, x_j^k, x_{j+1}^k, \ldots, x_{m_k}^k)$ is one route with m_k selected customers. Swapping x_i^k and x_j^k $(0 < i < j \le m_k)$ leads to a neighboring solution φ' whose k-th route is $X_k' = (x_0^k, \ldots, x_{i-1}^k, x_j^k, x_{i+1}^k, \ldots, x_{j-1}^k, x_i^k, x_{j+1}^k, \ldots, x_{m_k}^k)$. Using Eq. (2), the move gain $\Delta_f = f(\varphi') - f(\varphi)$ can be reached by

(1) If x_i^k and x_j^k are not adjacent, then

$$\begin{aligned}\Delta_f = &(m_k - i + 1) \cdot \left(d_{x_{i-1}^k, x_i^k} - d_{x_{i-1}^k, x_j^k}\right)\\ &+(m_k - i) \cdot \left(d_{x_i^k, x_{i+1}^k} - d_{x_j^k, x_{i+1}^k}\right)\\ &+(m_k - j + 1) \cdot \left(d_{x_{j-1}^k, x_j^k} - d_{x_{j-1}^k, x_i^k}\right)\\ &+(m_k - j) \cdot \left(d_{x_j^k, x_{j+1}^k} - d_{x_i^k, x_{j+1}^k}\right)\end{aligned}$$

(2) If x_i^k and x_j^k are adjacent, then

$$\begin{aligned}\Delta_f = &(m_k - i + 1) \cdot \left(d_{x_{i-1}^k, x_i^k} - d_{x_{i-1}^k, x_j^k}\right)\\ &+(m_k - j) \cdot \left(d_{x_j^k, x_{j+1}^k} - d_{x_i^k, x_{j+1}^k}\right)\end{aligned}$$

Thus, any neighboring solution in N_1 can be evaluated in $O(1)$ and the complexity of examining the N_1 neighborhood is bounded by $O(m^2)$. □

Proof of Proposition 2. (Section 2.3). Let φ be a solution composed of K routes $\{X_1, X_2, \ldots, X_K\}$, where $X_k = (x_0^k, x_1^k, \ldots, x_{m_k}^k)$ is one route with m_k selected customers $(k = 1, 2, \ldots, K)$. As the set of selected customers does not change, we only consider the change of the accumulated distance according to Eq. (2).

(a) For the *Inter-Swap* neighborhood, we suppose two routes in the solution φ

$$X_a = \left(x_0^a, x_1^a, \ldots, x_{i-1}^a, x_i^a, x_{i+1}^a, \ldots, x_{m_a}^a\right)$$

$$X_b = \left(x_0^b, x_1^b, \ldots, x_{j-1}^b, x_j^b, x_{j+1}^b, \ldots, x_{m_b}^b\right)$$

Exchanging x_i^a $(0 < i \le m_a)$ and x_j^b $(0 < j \le m_b)$ leads to a new solution φ' whose a-th and b-th routes are:

$$X_a' = \left(x_0^a, x_1^a, \ldots, x_{i-1}^a, x_j^b, x_{i+1}^a, \ldots, x_{m_a}^a\right)$$

$$X_b' = \left(x_0^b, x_1^b, \ldots, x_{j-1}^b, x_i^a, x_{j+1}^b, \ldots, x_{m_b}^b\right)$$

By Eq. (2), the move gain $\Delta_f = f(\varphi') - f(\varphi)$ can be achieved by

$$\begin{aligned}\Delta_f = &(m_a - i + 1) \cdot \left(d_{x_{i-1}^a, x_i^a} - d_{x_{i-1}^a, x_j^b}\right)\\ &+(m_a - i) \cdot \left(d_{x_i^a, x_{i+1}^a} - d_{x_j^b, x_{i+1}^a}\right)\\ &+(m_b - j + 1) \cdot \left(d_{x_{j-1}^b, x_j^b} - d_{x_{j-1}^b, x_i^a}\right)\\ &+(m_b - j) \cdot \left(d_{x_j^b, x_{j+1}^b} - d_{x_i^a, x_{j+1}^b}\right)\end{aligned}$$

Therefore, each neighboring solution in N_4 can be evaluated in $O(1)$, leading to the complexity of $O(m^2)$ for exploring the *Inter-Swap* neighborhood.

(b) For the *Inter-Insert* neighborhood, we are given two routes for the solution φ.

$$X_a = \left(x_0^a, x_1^a, \ldots, x_{i-1}^a, x_i^a, x_{i+1}^a, \ldots, x_{m_a}^a\right)$$

$$X_b = \left(x_0^b, x_1^b, \ldots, x_j^b, x_{j+1}^b, \ldots, x_{m_b}^b\right)$$

Inserting x_i^a $(0 < i \le m_a)$ into the position between x_j^b and x_{j+1}^b $(0 \le b \le m_b)$ produces a neighboring solution φ', whose two corresponding routes are:

$$X_a' = \left(x_0^a, x_1^a, \ldots, x_{i-1}^a, x_{i+1}^a, \ldots, x_{m_a}^a\right)$$

$$X_b' = \left(x_0^b, x_1^b, \ldots, x_j^b, x_i^a, x_{j+1}^b, \ldots, x_{m_b}^b\right)$$

The move gain Δ_f can be obtained by

$$\begin{aligned}\Delta_f = &Vsd_a(i-1) + (m_a + 1 - i) \cdot d_{x_{i-1}^a, x_i^a}\\ &+(m_a - i) \cdot \left(d_{x_i^a, x_{i+1}^a} - d_{x_{i-1}^a, x_{i+1}^a}\right)\\ &-Vsd_b(j) - (m_b + 1 - j) \cdot d_{x_j^b, x_i^a}\\ &-(m_b - j) \cdot \left(d_{x_i^a, x_{j+1}^a} - d_{x_j^b, x_{j+1}^b}\right)\end{aligned}$$

where $Vsd_a(i)$ and $Vsd_b(i)$ are two auxiliary arrays used to accelerate the evaluation procedure. For the k-th route in the solution, the auxiliary array is defined as follows.

$$Vsd_k(i) = \sum_{t=1}^{i} d_{x_{t-1}^k, x_t^k}. \tag{A.1}$$

The auxiliary arrays in Eq. (A.1) are pre-calculated and updated for each iteration (the complexity of updating these auxiliary arrays is $O(n)$). Therefore, each neighboring solution can be assessed in $O(1)$ while the complete *Inter-Insert* neighborhood can be examined in $O(m^2)$.

(c) For the *Inter-2-opt* neighborhood, we suppose two routes X_a and X_b.

$$X_a = \left(x_0^a, x_1^a, \ldots, x_i^a, x_{i+1}^a, \ldots, x_{m_a}^a\right)$$

$$X_b = \left(x_0^b, x_1^b, \ldots, x_j^b, x_{j+1}^b, \ldots, x_{m_b}^b\right)$$

Removing two edges (x_i^a, x_{i+1}^a) and (x_j^b, x_{j+1}^b) and replacing them with two other edges lead to a new solution φ', which has two corresponding routes X_a' and X_b'.

$$X_a' = \left(x_0^a, x_1^a, \ldots, x_i^a, x_{j+1}^b, \ldots, x_{m_b}^b\right)$$

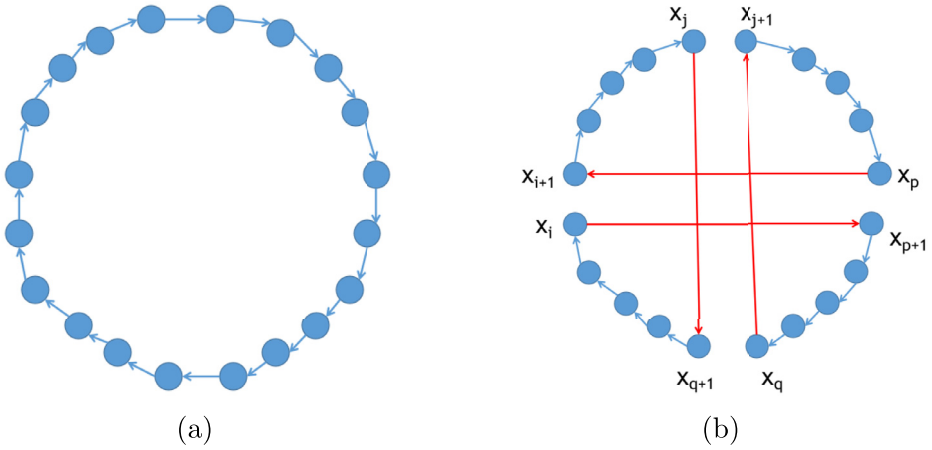

Fig. A.1. Simple illustration of the *Double-bridge* operation: a) one route before the operation; b) the route after the operation and the lines in red are the new reconnecting edges.. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

$$X'_b = \left(x^b_0, x^b_1, \ldots, x^b_j, x^a_{i+1}, \ldots, x^a_{m_a}\right)$$

The move gain Δ_f can be obtained as follows.

$$\begin{aligned}\Delta_f = {} & (m_a - i)\cdot d_{x^a_i, x^a_{i+1}} - (m_b - j)\cdot d_{x^a_i, x^b_{j+1}} \\ & + (m_a - m_b - i + j)\cdot Vsd_a(i) \\ & + (m_b - j)\cdot d_{x^b_j, x^b_{j+1}} - (m_a - i)\cdot d_{x^b_j, x^a_{i+1}} \\ & + (-m_a + m_b + i - j)\cdot Vsd_b(j)\end{aligned}$$

The complexity of evaluating one neighboring solution is thus $O(1)$ and exploring the complete *Inter-2-opt* neighborhood requires $O(m^2)$.

Additional results (Section 2.7). This section gives the descriptions of the *Double-bridge* neighborhood and *Inter-Or-opt* neighborhood, which were employed in the reference algorithm (Lu et al., 2019a) but not applied in the proposed algorithm. As these two neighborhoods are widely applied in the algorithms for the related problems, here we provide the detailed proof of their complexities of exploring the complete neighborhoods using the fast evaluation techniques.

- *Double-bridge* (Lin & Kernighan, 1973): Four edges in the same route are deleted and four new sub-tours are reconnected without changing the orientation of the four sub-tours. A simple illustration is presented in Fig. A.1.
- *Inter-Or-opt*: A block of h (h = 2, 3) consecutive customers is removed from one route and inserted into two adjacent nodes in another route.

Similar to other neighborhoods, the complexities of exploring their complete neighborhoods are summarized as follows.

- Exploring the complete *Double-bridge* neighborhood can be finished in $O(m^4)$.
- Exploring the complete *Inter-Or-opt* neighborhood can be finished in $O(m^2)$.

(1) For the *Double-bridge* neighborhood, we suppose a solution φ with K routes $\{X_1, X_2, \ldots, X_K\}$, where $X_k = (x^k_0, \ldots, x^k_i, x^k_{i+1}, \ldots, x^k_j, x^k_{j+1}, \ldots, x^k_p, x^k_{p+1}, \ldots, x^k_q, x^k_{q+1}, \ldots, x^k_{m_k})$ is one route with m_k selected customers. For giving a general case, we random select four positions i, j, p, q $(0 \le i,\ i+1 < j,\ j+1 < p,\ p+1 < q$ and $q+1 \le m_k)$ to perform a double-bridge operation. This results in a neighboring solution φ' whose k-th route is $X_k = (x^k_0, \ldots, x^k_i, x^k_{p+1}, \ldots, x^k_q, x^k_{j+1}, \ldots, x^k_p, x^k_{i+1}, \ldots, x^k_j, x^k_{q+1}, \ldots, x^k_{m_k})$ (See Fig. A.1). Using Eqs. (2) and (A.1), the move gain $\Delta_f = f(\varphi') - f(\varphi)$ could be obtained as follows.

$$\begin{aligned}\Delta_f = {} & (q - j)\cdot(Vsd_k(j) - Vsd_k(i+1)) + (i - p) \\ & \cdot(Vsd_k(q) - Vsd_k(p+1)) \\ & + (i + q - j - p)\cdot(Vsd_k(p) - Vsd_k(j+1)) \\ & + (m_k - i)\cdot d_{x^k_i, x^k_{i+1}} + (m_k - j)\cdot d_{x^k_j, x^k_{j+1}} \\ & + (m_k - p)\cdot d_{x^k_p, x^k_{p+1}} + (m_k - q)\cdot d_{x^k_q, x^k_{q+1}} \\ & - (m_k - i)\cdot d_{x^k_i, x^k_{p+1}} - (m_k - i - q + p)\cdot d_{x^k_q, x^k_{j+1}} \\ & - (m_k - i - q + j)\cdot d_{x^k_p, x^k_{i+1}} - (m_k - q)\cdot d_{x^k_j, x^k_{q+1}}\end{aligned}$$

Thus, any neighboring solution in the *Double-bridge* neighborhood can be evaluated in $O(1)$ and the complexity of examining the complete neighborhood is bounded by $O(m^4)$.

(2) For the *Inter-Or-opt* neighborhood, only the change of the accumulated distance is taken into consideration to obtain the move gain of the new neighboring solution. We suppose two routes in the solution φ.

$$X_a = \left(x^a_0, x^a_1, \ldots, x^a_{i-1}, x^a_i, x^a_{i+1}, \ldots, x^a_{i+h-1}, x^a_{i+h} \ldots, x^a_{m_a}\right)$$

$$X_b = \left(x^b_0, x^b_1, \ldots, x^b_j, x^b_{j+1}, \ldots, x^b_{m_b}\right)$$

Inserting the block $(x^a_i, x^a_{i+1}, \ldots, x^a_{i+h-1})$ $(0 < i,\ 0 < i + h \le m_a,\ h = 2, 3)$ to the position between x^b_j and x^b_{j+1} $(0 < j \le m_b)$ leads to a new solution φ' whose a-th and b-th routes are:

$$X'_a = \left(x^a_0, x^a_1, \ldots, x^a_{i-1}, x^a_{i+h} \ldots, x^a_{m_a}\right)$$

$$X'_b = \left(x^b_0, x^b_1, \ldots, x^b_j, x^a_i, x^a_{i+1}, \ldots, x^a_{i+h-1}, x^b_{j+1}, \ldots, x^b_{m_b}\right)$$

By Eqs. (2) and (A.1), the move gain $\Delta_f = f(\varphi') - f(\varphi)$ can be achieved by

$$\begin{aligned}\Delta_f = {} & h\cdot Vsd_a(i-1) - (m_a - h - i + 1)\cdot d_{x^a_{i-1}, x^a_{i+h}} \\ & + (m_a - i - h + 1)\cdot d_{x^a_{i+h-1}, x^a_{i+h}} + (m_a - i + 1)\cdot d_{x^a_{i-1}, x^a_i} \\ & + (m_b - j)\cdot d_{x^b_j, x^b_{j+1}} - h\cdot Vsd_b(j) \\ & - (m_b + h - j)\cdot d_{x^b_j, x^a_i} - (m_b - j)\cdot d_{x^a_{i+h-1}, x^b_{j+1}} \\ & + (m_a - m_b - h - i + j + 1)\cdot(Vsd_a(i+h-1) - Vsd_a(i))\end{aligned}$$

Therefore, each neighboring solution in the *Inter-Or-opt* neighborhood can be evaluated in $O(1)$, leading to the complexity of $O(m^2)$ for exploring the complete neighborhood. □

References

Avci, M., & Avci, M. G. (2017). A GRASP with iterated local search for the traveling repairman problem with profits. *Computers & Industrial Engineering, 113*, 323–332.

Avci, M. G., & Avci, M. (2019). An adaptive large neighborhood search approach for multiple traveling repairman problem with profits. *Computers & Operations Research, 111*, 367–385.

Bianchessi, N., Mansini, R., & Speranza, M. G. (2018). A branch-and-cut algorithm for the team orienteering problem. *International Transactions in Operational Research, 25*, 627–635.

Bouly, H., Dang, D.-C., & Moukrim, A. (2010). A memetic algorithm for the team orienteering problem. *4OR, 8*, 49–70.

Boussier, S., Feillet, D., & Gendreau, M. (2007). An exact algorithm for team orienteering problems. *4OR, 5*, 211–230.

Chao, I.-M., Golden, B. L., & Wasil, E. A. (1996). The team orienteering problem. *European Journal of Operational Research, 88*, 464–474.

Dang, D.-C., Guibadj, R. N., & Moukrim, A. (2013). An effective pso-inspired algorithm for the team orienteering problem. *European Journal of Operational Research, 229*, 332–344.

Dewilde, T., Cattrysse, D., Coene, S., Spieksma, F. C. R., & Vansteenwegen, P. (2013). Heuristics for the traveling repairman problem with profits. *Computers & Operations Research, 40*, 1700–1707.

El-Hajj, R., Dang, D.-C., & Moukrim, A. (2016). Solving the team orienteering problem with cutting planes. *Computers & Operations Research, 74*, 21–30.

Hammami, F., Rekik, M., & Coelho, L. C. (2020). A hybrid adaptive large neighborhood search heuristic for the team orienteering problem. *Computers & Operations Research, 123*, 105034.

Hao, J. K. (2012). Memetic algorithms in discrete optimization. In *Handbook of memetic algorithms* (pp. 73–94). Springer.

Ke, L., & Feng, Z. (2013). A two-phase metaheuristic for the cumulative capacitated vehicle routing problem. *Computers & Operations Research, 40*, 633–638.

Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research, 21*, 498–516.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, 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., & Peng, B. (2019a). Memetic algorithm for the multiple traveling repairman problem with profits. *Engineering Applications of Artificial Intelligence, 80*, 35–47.

Lu, Y., Hao, J.-K., & Wu, Q. (2019b). Hybrid evolutionary search for the traveling repairman problem with profits. *Information Sciences, 502*, 91–108.

Lysgaard, J., & Wøhlk, S. (2014). A branch-and-cut-and-price algorithm for the cumulative capacitated vehicle routing problem. *European Journal of Operational Research, 236*, 800–810.

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research, 24*, 1097–1100.

Moscato, P. (1999). Memetic algorithms: A short introduction. In *New ideas in optimization* (pp. 219–234). GBR: McGraw-Hill Ltd., UK.

Ngueveu, S. U., Prins, C., & Calvo, R. W. (2010). An effective memetic algorithm for the cumulative capacitated vehicle routing problem. *Computers & Operations Research, 37*, 1877–1885.

Nucamendi-Guillén, S., Angel-Bello, F., Martínez-Salazar, I., & Cordero-Franco, A. E. (2018). The cumulative capacitated vehicle routing problem: new formulations and iterated greedy algorithms. *Expert Systems with Applications, 113*, 315–327.

Pei, J., Mladenović, N., Urošević, D., Brimberg, J., & Liu, X. (2020). Solving the traveling repairman problem with profits: a novel variable neighborhood search approach. *Information Sciences, 507*, 108–123.

Poggi, M., Viana, H., & Uchoa, E. (2010). The team orienteering problem: Formulations and branch-cut and price. In *10th workshop on algorithmic approaches for transportation modelling, optimization, and systems (ATMOS'10)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Potvin, J.-Y., & Bengio, S. (1996). The vehicle routing problem with time windows part ii: Genetic search. *INFORMS Journal on Computing, 8*, 165–172.

Ribeiro, G. M., & Laporte, G. (2012). An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. *Computers & Operations Research, 39*, 728–735.

Tsakirakis, E., Marinaki, M., Marinakis, Y., & Matsatsinis, N. (2019). A similarity hybrid harmony search algorithm for the team orienteering problem. *Applied Soft Computing, 80*, 776–796.

Vansteenwegen, P., Souffriau, W., Berghe, G. V., & Van Oudheusden, D. (2009). A guided local search metaheuristic for the team orienteering problem. *European Journal of Operational Research, 196*, 118–127.

Wang, Y., Lü, Z., Glover, F., & Hao, J. K. (2013). Backbone guided tabu search for solving the ubqp problem. *Journal of Heuristics, 19*, 679–695.

Zettam, M., & Elbenani, B. (2016). A novel randomized heuristic for the team orienteering problem. In *2016 3rd international conference on logistics operations management (GOL) (pp. 1–6)*. IEEE.

Zhang, W. (2004). Configuration landscape analysis and backbone guided local search.: Part i: Satisfiability and maximum satisfiability. *Artificial Intelligence, 158*, 1–26.