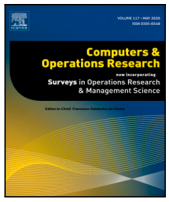
Computers & Operations Research 143 (2022) 105772

Contents lists available at ScienceDirect

Computers and Operations Research

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



An effective iterated two-stage heuristic algorithm for the multiple Traveling Salesmen Problem

Jiongzhi Zheng, Yawei Hong, Wenchang Xu, Wentao Li, Yongfu Chen *

School of Mechanical Science and Engineering, Huazhong University of Science and Technology, China

ARTICLE INFO

Keywords:
Multiple Traveling Salesmen Problem
Combinatorial optimization
Variable neighborhood search
Heuristic
Local search

ABSTRACT

The multiple Traveling Salesmen Problem (mTSP) is a general extension of the famous NP-hard Traveling Salesmen Problem (TSP), that there are m ($m > 1$) salesmen to visit the cities. In this paper, we address the mTSP with both the *minsum* objective and *minmax* objective, which aims at minimizing the total length of the m tours and the length of the longest tour among all the m tours, respectively. We propose an iterated two-stage heuristic algorithm called ITSHA for the mTSP. Each iteration of ITSHA consists of an initialization stage and an improvement stage. The initialization stage aims to generate high-quality and diverse initial solutions. The improvement stage mainly applies the variable neighborhood search (VNS) approach based on our proposed effective local search neighborhoods to optimize the initial solution. Moreover, some local optima escaping approaches are employed to enhance the search ability of the algorithm. Extensive experimental results on a wide range of public benchmark instances show that ITSHA significantly outperforms the state-of-the-art heuristic algorithms in solving the mTSP on both the objectives.

1. Introduction

The multiple Traveling Salesmen Problem (mTSP) is a general extension of the famous NP-hard Traveling Salesmen Problem (TSP), that there are m ($m > 1$) salesmen to visit the cities. In this paper, we consider the mTSP with a single depot, that is, the m salesmen travel from the same depot to visit all the cities exactly once without overlapping and finally return to the depot. We address the mTSP with both the *minsum* and *minmax* objectives, which aims at minimizing the total length of the m tours and the length of the longest tour among all the m tours, respectively. The mTSP is not only a natural but also a more practical extension of the TSP, that finds many practical applications in the real world (Cheikhrouhou and Khoufi, 2021). For example, the well-known Vehicle Routing Problem (VRP) (Nagata et al., 2010; Arnold and Sörensen, 2019), production schedules (Tang et al., 2000), the school bus routing problem (Miranda et al., 2018), printing press schedules (Carter and Ragsdale, 2002), task allocation (Vandermeulen et al., 2019), etc.

Typical methods for the mTSP are mainly exact algorithms (França et al., 1995; Kara and Bektas, 2006), approximation algorithms (Frederickson et al., 1978), and heuristics (Venkatesh and Singh, 2015; Soylu, 2015; Lu and Yue, 2019). The exact algorithms may be difficult for large instances and the approximation algorithms may suffer from weak optimality guarantees. Heuristics are known to be the most efficient and effective approaches for solving the mTSP.

Population-based meta-heuristics are the most popular and effective heuristic algorithms for the *minsum* and *minmax* mTSP recently. Some of them address both of the two objectives of mTSP. For example, the genetic algorithms (GA) (Carter and Ragsdale, 2006; Singh and Baghel, 2009; Yuan et al., 2013), artificial bee colony (ABC) algorithms (Venkatesh and Singh, 2015), ant colony optimization (ACO) algorithms (Lu and Yue, 2019; Liu et al., 2009), evolution strategy (ES) algorithm (Karabulut et al., 2021), and other population-based approaches (Venkatesh and Singh, 2015). Among these algorithms, the ABC algorithms as well as the invasive weed optimization (IWO) algorithm proposed by Venkatesh and Singh (2015), the ACO algorithm (Lu and Yue, 2019), and the ES algorithm (Karabulut et al., 2021) are some of the state-of-the-art heuristics for both the *minsum* and *minmax* mTSP. In addition, some studies focus on one of the two objectives of mTSP. The genetic algorithm called GAL (Lo et al., 2018) and the memetic algorithm called MASVND (Wang et al., 2017) are two effective heuristics for the *minsum* and *minmax* mTSP, respectively. They tested their algorithms on the mTSP instances with more than one thousand cities.

Local search is another type of heuristic algorithm, which is widely used in some famous combinatorial optimization problems such as the TSP (Lin and Kernighan, 1973; Helsgaun, 2000), satisfiability (Selman et al., 1992), and maximum satisfiability (Luo et al., 2017). However,

* Corresponding author.
E-mail address: chenyf@hust.edu.cn (Y. Chen).

https://doi.org/10.1016/j.cor.2022.105772
Received 24 May 2021; Received in revised form 26 February 2022; Accepted 28 February 2022
Available online 12 March 2022

the local search technique is mainly applied to improve the population-based mTSP meta-heuristics by incorporating with them in the related studies (Venkatesh and Singh, 2015; Karabulut et al., 2021; Wang et al., 2017), and few researches employ the local search method to directly solve the standard *minsum* and *minmax* mTSP. Among the local search algorithms, the general variable neighborhood search (GVNS) algorithm proposed by Soylu (2015) is one of the best-performing based on variable neighborhood search (VNS) for the *minsum* and *minmax* mTSP.

This work aims at making up the lack of effective local search heuristics for solving the *minsum* and *minmax* mTSP by introducing an iterated two-stage heuristic algorithm, denoted as ITSHA. The first stage (the initialization stage) of ITSHA is to generate an initial solution by the fuzzy c-means (FCM) clustering algorithm (Dunn, 1973; Bezdek et al., 1984) and a random greedy heuristic. The FCM algorithm and the random greedy heuristic help the algorithm escape from the local optima by providing diverse initial solutions. In the second stage (the improvement stage), a VNS approach based on our proposed neighborhoods is employed to improve the initial solution produced in the first stage. We define a candidate set for each city that records several other nearest cities in ascending order of the distance to reduce the search scope and improve the efficiency of the proposed neighborhoods. The solution can be adjusted several times by exchanging the positions of several cities during the improvement stage to escape from the local optima and find better solutions. ITSHA repeats these two stages until a stopping condition is met.

There are some related studies that apply clustering algorithms to solve the mTSP (Latah, 2016; Lu et al., 2016; Xu et al., 2018). These studies all combine clustering algorithms with the population-based algorithms including ACO (Latah, 2016) and GA (Lu et al., 2016; Xu et al., 2018) to solve the mTSP. Specifically, they apply clustering algorithms to divide the cities into m groups, then use the population-based algorithms to find the shortest m tours that each tour consists of the cities in each group. Obviously, the quality of the results of these algorithms depends too much on the clustering results, since the cities of each tour are fixed according to the clustering result. Their abandoning of the intra-tour improvements results in poor performance. These algorithms also did not compare with other state-of-the-art heuristics on widely used mTSP benchmark instances.

In our proposed ITSHA algorithm, the clustering algorithm is applied to generate high-quality and diverse initial solutions, which will be improved by the VNS approach in the improvement stage. Both inter-tour and intra-tour improvements are considered by our method. Thus our method can make up for the shortcomings of the related studies that apply clustering algorithms to solve the mTSP (Latah, 2016; Lu et al., 2016; Xu et al., 2018) described above. Moreover, we tested our algorithm on public *minsum* and *minmax* mTSP benchmarks with up to more than 1000 cities. The results show that our ITSHA algorithm significantly outperforms the state-of-the-art heuristics in solving the mTSP on both the objectives.

The main contributions of this work are as follows:

- We propose an iterated two-stage heuristic algorithm called ITSHA to solve the *minsum* and *minmax* mTSP with a single depot. ITSHA significantly outperforms the state-of-the-art mTSP heuristics, yields 32 new records among 38 public *minsum* mTSP instances and 22 new records among 44 public *minmax* mTSP instances.
- We propose three effective and efficient local search operators, called 2-*opt*, *Insert*, and *Swap*, based on the candidate sets. The proposed operators are significantly better than the local search neighborhoods used in existing mTSP heuristics.
- We propose applying the fuzzy clustering algorithm, adjusting the solution and the candidate edges to help the algorithm escape from the local optima and find better results.
- The proposed local search neighborhoods and the strategies for escaping from the local optima could be applied to other combinatorial optimization problems, such as various variants of the TSP and VRP.

The rest of this paper is organized as follows. Section 2 formulates the *minsum* and *minmax* mTSP. Section 3 describes our proposed ITSHA algorithm. Section 4 presents experimental results and analyses. Section 5 contains the concluding remarks.

2. Problem definition

Given a complete undirected graph $G(V, E)$, where $V = \{1, \dots, n\}$ denotes the set of the cities (note that city 1 is the depot), and E is the pairwise edges $\{e_{ij} | i, j \in V\}$. c_{ij} represents the cost of edge e_{ij} (usually equals to the distance of traveling from city i to city j). The mTSP is to determine a set of m routes that cover each city exactly once and minimize the objective function (*minsum* or *minmax*).

Let x_{ijk} be a three-index variable that $x_{ijk} = 1$ when salesman k visits city j immediately after city i, otherwise $x_{ijk} = 0$, and u_i be a variable that indicates the visiting rank of city i in order ($u_1 = 0$). The flow based formulation (Christofides et al., 1981; Bektas, 2006) of the *minsum* and *minmax* mTSP with the Miller–Tucker–Zemlin (MTZ) (Miller et al., 1960) sub-tour elimination constraints is given as follows:

minsum mTSP:

$$\text{Minimize} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} \sum_{k=1}^{m} x_{ijk} \tag{1}$$

minmax mTSP:

$$\begin{aligned} \text{Minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} x_{ijk_{max}}, \\ & k_{max} = \arg\max_{k\in\{1,\dots,m\}} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} x_{ijk} \end{aligned} \tag{2}$$

Subject to:

$$\sum_{i=1}^{n}\sum_{k=1}^{m} x_{ijk} = 1, \ j = 1, \dots, n \tag{3}$$

$$\sum_{i=1}^{n} x_{ilk} - \sum_{j=1}^{n} x_{ljk} = 0, \ k = 1, \dots, m, \ l = 1, \dots, n \tag{4}$$

$$\sum_{j=1}^{n} x_{1jk} = 1, \ k = 1, \dots, m \tag{5}$$

$$u_i - u_j + p\sum_{k=1}^{m} x_{ijk} \le p - 1, \ i \ne j = 2, \dots, n \tag{6}$$

$$x_{ijk} \in \{0, 1\}, \ \forall i, j, k. \tag{7}$$

As shown in Eqs. (1) and (2), the *minsum* and *minmax* mTSP aim at minimizing the total length of the m tours and the length of the longest tour among all the m tours, respectively. Constraints (3) state that each city should be visited exactly once and (4) are the flow conservation constraints that ensure that once a salesman visits a city, then he must also depart from the same city. Constraints (5) ensure that exactly m salesmen depart from the depot. Constraints (6) are the extensions of the MTZ (Miller et al., 1960) sub-tour elimination constraints to a three-index model, where p denotes the maximum number of cities that any salesman can visit.

3. The proposed algorithm

The proposed iterated two-stage heuristic algorithm (ITSHA) consists of the initialization stage and the improvement stage. We apply the fuzzy c-means (FCM) clustering algorithm (Dunn, 1973; Bezdek et al., 1984) and a random greedy heuristic to generate an initial solution in the initialization stage. The initialization stage actually helps the algorithm escape from the local optima by providing high-quality and diverse initial solutions. The VNS method based on our proposed neighborhoods is employed in the improvement stage to improve the initial solution. The solution can be adjusted several times by randomly exchanging the positions of several cities to find better solutions. The ITSHA algorithm repeats these two stages until a cut-off time is reached.

This section first describes the main process of the proposed ITSHA algorithm, then introduces the details of the two stages in ITSHA, respectively.

3.1. Main process of ITSHA

The main flow of ITSHA is presented in Algorithm 1. ITSHA first initializes the candidate set of each city (line 1). We denote CS as the collection of the candidate sets, CS_i as the candidate set of city i, and C_{max} (10 by default) as the number of cities in each candidate set, i.e., the initial candidate set of each city records C_{max} other nearest cities in ascending order of the city distance. The search scope of the proposed neighborhoods is significantly reduced by the candidate sets. Thus the efficiency of the algorithm is significantly improved. The approach of applying candidates to reduce the search scope is widely used in the famous TSP heuristics (Helsgaun, 2000; Nagata and Kobayashi, 2013).

Algorithm 1 The ITSHA algorithm

Input: the maximum number of the candidate cities: C_{max}, the number of times to adjust the solution: A_t, the number of the adjusted cities: A_c, the cut-off time t_{max}, the maximum number of cities that can be visited by any salesman: p, the weighting exponent in FCM: w, the termination criterion in FCM: ϵ

Output: a solution: S_{best}

1: CS := Initialize_Candidates(C_{max})
2: Initialize $L(S_{best}) := +\infty$, $L(S)$ is the objective value of solution S (Eq. (1) or Eq. (2))
3: **while** the cut-off time t_{max} is not reached **do**
4: Initialize $L(S_{better}) := +\infty$
% *the initialization stage*
5: NC := FCM(w, ϵ, p), NC_i is the group (cluster) that city i belongs to
6: S := Random_Greedy(NC, CS, C_{max}, S_{best})
% *the improvement stage*
7: Initialize $num_t := 0$
8: **while** $num_t < A_t + 1$ **do**
9: S := Adjust_Solution(S, A_c, p) if $num_t > 0$
10: S := VNS(S, CS, C_{max}, p)
11: $S_{better} := S$ if $L(S) < L(S_{better})$
12: $num_t := num_t + 1$
13: **end while**
14: **if** $L(S_{best}) \neq +\infty$ **then**
15: CS := Adjust_Candidates(CS, S_{better}, S_{best})
16: **end if**
17: $S_{best} := S_{better}$ if $L(better) < L(S_{best})$
18: **end while**

As shown in Algorithm 1, ITSHA repeats the initialization stage (lines 5–6) and the improvement stage (lines 7–13) until the cut-off time t_{max} is reached. The initialization stage generates an initial solution by the FCM algorithm (line 5) and the random greedy function (line 6). The improvement stage applies the VNS method (line 10) based on our proposed neighborhoods to improve the initial solution. The solution obtained during the improvement stage can be adjusted for A_t (3 by default) times to escape from the local optima (line 9). The approach of the solution adjustment function is to randomly delete A_c (5 by default) cities in S, then randomly insert these cities into S. Note that the adjusted solution should satisfy the constraint that each salesman can visit at most p cities, which is guaranteed by forbidding inserting cities to a tour with p cities.

Moreover, at the end of each iteration of ITSHA (except the first iteration), the candidate set of each city will be adjusted (lines 14–16). Specifically, if edge (i, j) appears in both S_{better} and S_{best}, the last candidate city in CS_i will be replaced with city j if $j \notin CS_i$. The method of adjusting candidate cities allows the search neighborhoods to change adaptively to enhance the algorithm's robustness and search ability. The experimental results also demonstrate that adjusting candidate cities can improve the performance of our ITSHA algorithm.

3.2. The initialization stage of ITSHA

This subsection introduces the initialization stage of ITSHA. We describe the process of the FCM algorithm and the random greedy function, respectively.

3.2.1. Fuzzy C-means clustering

The fuzzy c-means clustering (FCM) algorithm (Dunn, 1973; Bezdek et al., 1984) is an important vehicle to cope with overlapping clustering, which uses a membership matrix $U = [u_{ij}] \in \mathbb{R}^{N\times c}$ to represent the result of clustering N elements into c clusters. The membership degree u_{ij} of the element i in the cluster j is subjected to the following constraints: (1) $u_{ij} \in [0, 1]$. (2) $\sum_{j=1}^{c} u_{ij} = 1$.

In our ITSHA algorithm, we apply FCM to cluster the $n-1$ cities (all the n cities except the depot) into m clusters according to their positions, so as to assign the cities of each cluster to a salesman. With the help of the clustering algorithm, the positions of the cities assigned to each salesman (in each cluster) are close. We employ the FCM algorithm rather than the common c-means algorithm used in Latah (2016), Lu et al. (2016) and Xu et al. (2018) since the randomness and robustness of FCM are better than those of c-means. In other words, the diversity of the initial solutions generated by FCM is better than by c-means. The goal of the FCM algorithm in ITSHA is to minimize the following objective function J:

$$J = \sum_{i=2}^{n}\sum_{j=1}^{m} u_{ij}^{w}\left\|x_i - v_j\right\|^2, \tag{8}$$

where w is the weighting exponent, x_i is the position of city i, v_j is the cluster center of the cluster j. The FCM is carried out through iterative minimization of the objective function J through updating the cluster center v_j and the membership degrees u_{ij} according to the following formulas:

$$v_j = \frac{\sum_{i=2}^{n} u_{ij}^{w} x_i}{\sum_{i=2}^{n} u_{ij}^{w}},\ j = 1, \ldots, m \tag{9}$$

and

$$u_{ij} = \frac{1}{\sum_{k=1}^{m} \left(\frac{\left\|x_i - v_j\right\|}{\left\|x_i - v_k\right\|}\right)^{\frac{2}{w-1}}},\quad i = 2, \ldots, n,\ j = 1, \ldots, m. \tag{10}$$

The flow of the fuzzy clustering procedure in our ITSHA algorithm is shown in Algorithm 2. The FCM algorithm first randomly initializes the membership matrix U (line 3). The random initialized membership matrix can help the initialization stage of ITSHA generate diverse initial solutions. FCM then repeatedly updates the clustering centers and membership matrix until the membership matrix converges (lines 4–7). Lines 9–12 can prevent the clustering algorithm from obtaining empty clusters. In general, city i will be assigned to the salesman (cluster) t

that the membership degree u_{it} is the largest among $u_{ij}, j \in \{1, \ldots, m\}$ (lines 17–18). A cluster with $p-1$ cities cannot be assigned more cities since each salesman can visit at most p cities, including the depot (line 17).

Algorithm 2 Fuzzy clustering procedure

Input: the weighting exponent in FCM: w, the termination criterion in FCM: ϵ, the maximum number of cities that can be visited by any salesman: p

Output: a vector that represents the cluster each city belongs to: NC

1: Initialize $NC_i = 0$ for each $i \in \{2, ..., n\}$
2: Let $C_{num} := [C_{num_j}], j \in \{1, ..., m\}$ be a vector that C_{num_j} represents the number of the cities belong to cluster j
3: Let $k := 1$, and randomly initialize the membership matrix U^k
4: **repeat**
5: Compute $v_j, j \in \{1, ..., m\}$ according to Eq. (9)
6: Compute the updated membership matrix U^{k+1} according to Eq. (10), $k := k + 1$
7: **until** $\max_{i,j} \{|u_{ij}^k - u_{ij}^{k-1}|\} < \epsilon$
8: $U := U^k$
9: **for** $j := 1$ to m **do**
10: $t := \arg\max_{i \in \{2,...,n\} \wedge NC_i = 0} u_{ij}$
11: $NC_t := j$, $C_{num_j} := 1$
12: **end for**
13: **for** $i := 2$ to n **do**
14: **if** $NC_i \neq 0$ **then**
15: **continue**
16: **end if**
17: $t := \arg\max_{j \in \{1,...,m\} \wedge C_{num_j} < p-1} u_{ij}$
18: $NC_i := t$, $C_{num_t} := C_{num_t} + 1$
19: **end for**

3.2.2. Random greedy function

We propose a random greedy function to generate a feasible initial solution for the mTSP based on the clustering results of FCM. The process of the random greedy function is shown in Algorithm 3. The random greedy function actually determines a connected sequence of the cities in each cluster coupled with the depot (i.e., C^j in line 1) to produce a solution. For generating each of the m tours, a random city sc is chosen firstly (line 4), and the current city cc is set to be sc. Then, as long as not all cities in C^j have been chosen, choose the next city nc to follow cc in the current tour, and set cc equal to nc. nc is chosen as follows:

(1) If possible, choose nc such that (nc, cc) is an edge of the best solution S_{best} (lines 8–13).

(2) Otherwise, if possible, choose nc from the candidate set of cc, CS_{cc} (lines 14–18).

(3) Otherwise, choose nc at random among those cities not already chosen in C^j (lines 19–23).

When more than one city may be chosen, the city is chosen at random among the alternatives (a one-way list of cities, as shown in line 24 in Algorithm 3). The m sequences of chosen cities constitute the initial solution S of mTSP.

In summary, the random greedy function constructs the initial solution iteratively. In each iteration, the procedure tries to let the salesman follow the current best tour or select the next cities from the candidate sets. That is, the global structure of the mTSP instance (i.e., the candidate sets) and the best solution obtained during the procedure of ITSHA are utilized to improve the quality of the initial solution. The randomness of FCM and the random greedy function guarantees the diversity of the initial solutions. In a word, the initialization stage of ITSHA can provide high-quality and diverse initial solutions to help the algorithm escape from the local optima and yield better results.

Algorithm 3 Random greedy procedure

Input: the clustering results: NC, the candidate sets: CS, the maximum number of the candidate cities: C_{max}, the best solution: S_{best}

Output: a solution: S

1: Let $C^j = \{1\} \cup \{i | i \in \{2, ..., n\} \wedge NC_i = j\}$ be a set of the cities assigned to salesman j (i.e., the cities in cluster j coupled with the depot), $j \in \{1, ..., m\}$
2: Initialize $Se \in \{0\}^n$, $Se_i = 1$ indicates city i has been selected, otherwise $Se_i = 0$
3: **for** $j := 1$ to m **do**
4: $Se_1 := 0$
5: Randomly select a starting city sc in C^j, $Se_{sc} := 1$, $C^j := C^j \backslash \{sc\}$, set current city $cc := sc$
6: **repeat**
7: Initialize a set of the alternative cities $AC := \emptyset$
8: **if** $L(S_{best}) \neq +\infty$ **then**
9: Let a_1, a_2 be the two cities connected with cc in S_{best}.
10: **for** $i := 1$ to 2 **do**
11: If $NC_{a_i} = j \wedge Se_{a_i} = 0$, $AC := AC \cup \{a_i\}$
12: **end for**
13: **end if**
14: **if** $|AC| = 0$ **then**
15: **for** city $k \in CS_{cc}$ **do**
16: If $NC_k = j \wedge Se_k = 0$, $AC := AC \cup \{k\}$
17: **end for**
18: **end if**
19: **if** $|AC| = 0$ **then**
20: **for** city $k \in C^j$ **do**
21: If $NC_k = j \wedge Se_k^j = 0$, $AC := AC \cup \{k\}$
22: **end for**
23: **end if**
24: Randomly select next city $nc \in \{AC\}$, $Se_{nc} := 1$, $C^j := C^j \backslash \{nc\}$, connect city cc and nc in the solution S, $cc := nc$
25: **until** $C^j = \emptyset$
26: Connect city cc and sc in the solution S
27: **end for**

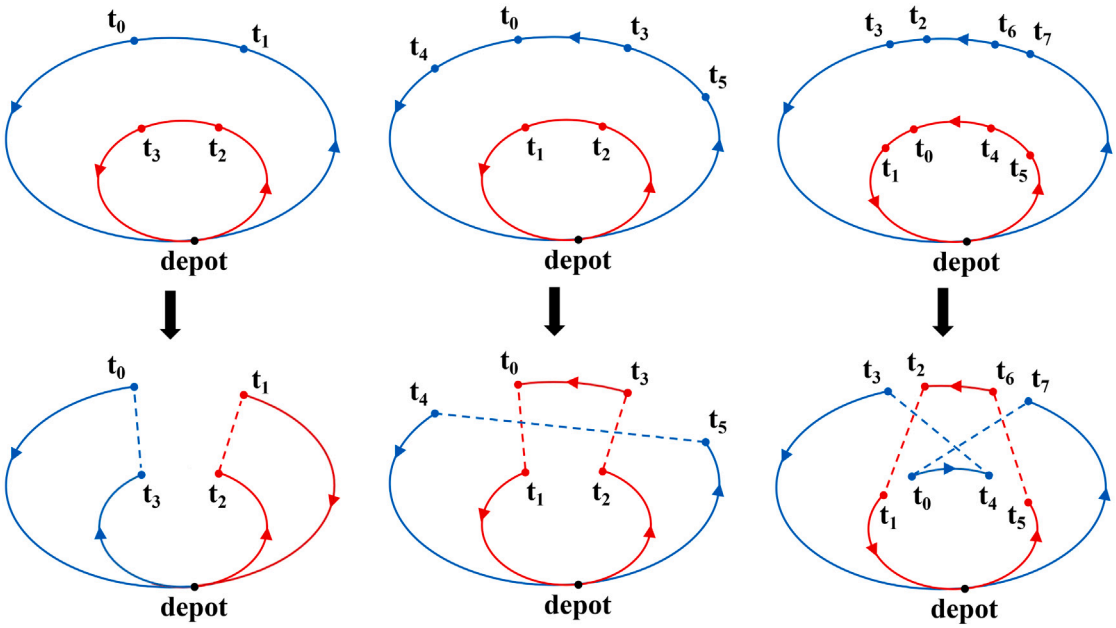
3.3. The improvement stage of ITSHA

This subsection mainly describes the process of the variable neighborhood search (VNS) in the improvement stage of ITSHA. We first introduce the neighborhoods used in our algorithms, then present the flow of the VNS.

3.3.1. Neighborhoods used in ITSHA

The VNS approach (Mladenovic and Hansen, 1997) is widely used in the routing problems include TSP (Hore et al., 2018), VRP (Kytöjoki et al., 2007; Defryn and Sörensen, 2017), and mTSP (Soylu, 2015; Wang et al., 2017). The performance of VNS strongly depends on the design of the neighborhood structures. However, the existing neighborhoods used in the state-of-the-art mTSP heuristics (Soylu, 2015; Wang et al., 2017; Karabulut et al., 2021) are inefficient, since they contain lots of low-quality operators that should not be considered. To handle this problem, we introduce three neighborhoods, 2-*opt*, *Insert* and *Swap*, applied in our ITSHA algorithm that are significantly more effective and efficient than the neighborhoods used in Soylu (2015), Karabulut et al. (2021) and Wang et al. (2017). The three neighborhoods are illustrated in Fig. 1.

In Fig. 1, the routes of different salesmen are in different colors. It is worth mentioning that, an arc with an arrow is a part of a tour, that may contain multiple cities. An arc without an arrow is an edge of the tour. The detailed description of the three neighborhoods is as follows.



(a) 2-*opt* move (b) *Insert* move (c) *Swap* move

Fig. 1. An Illustration of the neighborhoods used in the VNS process of ITSHA. Note that the routes of different salesmen are in different colors. An arc with an arrow is a part of a tour, that may contain multiple cities. An arc without an arrow is an edge of the tour.

(1) 2-*opt*: The 2-*opt* operator is shown in Fig. 1(a). It tries to replace two edges in the current mTSP tour, (t_0, t_1) and (t_2, t_3), with two new edges, (t_0, t_3) and (t_1, t_2), to improve the solution. However, performing an exhaustive search of 2-*opt* is too time consuming. Therefore, we restrict that t_2 must be selected in the candidate set of t_1, which can reduce the search scope of t_2 from about n to C_{max}.

(2) *Insert*: The *Insert* operator is shown in Fig. 1(b). It tries to insert a sequence of cities between t_0 and t_3 into an edge (t_1, t_2) to improve the solution. We restrict that t_1 must be selected in the candidate set of t_0, and t_3 must be selected in the candidate set of t_2. Such an restriction can reduce the search scope from about n^2 to C_{max}^2. It is worth mentioning that, the operators *One-point* move, *Or-opt*2 move, *Or-opt*3 move in Soylu (2015) and Wang et al. (2017), and the *Or-opt*4 move in Wang et al. (2017) are the special cases of our *Insert* operator when C_{max} is set to n and the sequence between t_0 and t_3 contains one, two, three, four cities respectively.

(3) *Swap*: The *Swap* operator is shown in Fig. 1(c). It tries to improve the solution by swapping two sequences of cities, one of them contains the cities between t_0 and t_4, another contains the cities between t_2 and t_6. We restrict that t_2 must be selected in the candidate set of t_1, t_4 must be selected in the candidate set of t_3, and t_6 must be selected in the candidate set of t_5. Such an restriction can reduce the search scope from about n^3 to C_{max}^3. It is worth mentioning that, the operators *Two-point* move and *Three-point* move in Soylu (2015) and Wang et al. (2017) are the special cases of our *Swap* operator when C_{max} is set to n, t_0 is equal to t_4, and the sequence between t_2 and t_6 contains one, two cities respectively.

Moreover, each of the three operators used in ITSHA can be applied for an inter-tour or an intra-tour improvement. Such a mechanism is more effective than the VNS approaches in the state-of-the-art heuristics (Soylu, 2015; Wang et al., 2017) that only apply the 2-*opt* operator to perform inter-tour improvements, and other operators to perform intra-tour improvements.

3.3.2. The process of VNS

The process of the VNS local search in ITSHA is shown in Algorithm 4. The VNS sequentially applies the three operators to improve the input solution to a local optimum. Specifically, the function *Insert*() in line 3 improves the input solution S_{old} to a local optimum for the *Insert* operator, i.e., *Insert*() tries to improve the solution until no *Insert* operator can be found to improve the current solution. The *Swap* operator and the 2-*opt* operator are applied in the same way (lines 4–5). Note that for the *minsum* mTSP, a solution is considered to be improved if the objective value (Eq. (1)) is reduced. For the *minmax* mTSP, a solution is considered to be improved if the objective value (Eq. (2)) is reduced, or the objective value is unchanged and the total length of the m tours is reduced. The solutions obtained during the VNS process are all feasible, i.e., they all satisfy the constraint that each salesman can visit at most p cities. The feasibility of the solutions is guaranteed by abandoning the operators that result in infeasible solutions. The VNS process terminates when the current solution cannot be improved by any of the three operators (line 6).

Algorithm 4 VNS in ITSHA

Input: a solution: S, the candidate sets: CS, the maximum number of the candidate cities: C_{max}, the maximum number of cities that can be visited by any salesman: p
Output: a solution: S
1: **repeat**
2: $S_{old} := S$
3: $S_{old} := Insert(S_{old}, CS, C_{max}, p)$
4: $S_{old} := Swap(S_{old}, CS, C_{max}, p)$
5: $S := 2\text{-}opt(S_{old}, CS, C_{max}, p)$
6: **until** $S = S_{old}$

With the help of the restriction of the search scope based on the candidate sets, the operators are refined by abandoning lots of low-quality moves. As a result, the computational complexity of improving a solution to the local optimum for each of the three operators (2-*opt*, *Insert*, *Swap*) is $O(n)$ (i.e., the computational complexity of line 3/4/5 in Algorithm 4 is $O(n)$). This is much more efficient than the neighborhoods used in Soylu (2015) and Wang et al. (2017), where the computational complexity of applying an operator once is $O(n^2)$.

In summary, the VNS process in our ITSHA algorithm is significantly better than the VNS used in Soylu (2015) and Wang et al. (2017) because: (1) Our neighborhoods are much more efficient than theirs, since

the low-quality moves can be refined by applying the candidate sets. (2) Our neighborhoods are much more effective than theirs, since our operators *Insert* and *Swap* can move a sequence of cities, which leads to a wide and deep search region. Thus our method can find higher-quality solutions. (3) All of our neighborhoods can be used to perform inter-tour or intra-tour improvements, while they only apply the 2-*opt* operator to perform inter-tour improvements, and other operators to perform intra-tour improvements.

4. Experimental results

Experimental results provide insight on why and how the proposed approach ITSHA is effective, suggesting that the VNS based on the proposed neighborhoods is efficient and effective. The fuzzy clustering algorithm, the solution adjustment process and the approach of adjusting candidate sets can help the algorithm escape from the local optima and find better solutions.

In this section, we first present the benchmark instances, baseline algorithms, experimental setup and various variants of ITSHA, then present and analyze the experimental results.

4.1. Benchmark instances

We test our ITSHA algorithm in solving the *minsum* and *minmax* mTSP on the benchmark instances used in the state-of-the-art heuristics (Venkatesh and Singh, 2015; Soylu, 2015; Lu and Yue, 2019; Karabulut et al., 2021; Lo et al., 2018; Wang et al., 2017), a total of 38 for the *minsum* mTSP with the number of cities ranges from 11 to 1002, and 44 for the *minmax* mTSP with the number of cities ranges from 11 to 1173. Note that for all the tested instances, the number in an instance name indicates the number of cities in that instance, and the first city of an instance is set to be the depot. We divide these instances into the following four sets:

- *Set I*: This set contains a total of 8 instances. Among them, there are three instances with $n = 128$ and $m = 10, 15, 30$ (denoted as *128*), and five small instances called *11a*, *11b*, *12a*, *12b*, and *16* with $m = 3$. Instances *11a*, *12a* and *16* comprise the first 11, 12, and 16 cities of the $n = 51$ instance of Carter and Ragsdale (2006) respectively, whereas *11b*, *12b*, and *128* instances are derived from sp11, uk12, and sgb128 data sets.[1] The 8 instances in this set are used in Venkatesh and Singh (2015), Soylu (2015) and Karabulut et al. (2021) for the *minsum* and *minmax* mTSP.
- *Set II*: This set contains a total of 12 instances that consist of three symmetric TSP instances *eil51*, *kroD100* and *mTSP150* in the TSPLIB.[2] Among these 12 instances, there are three instances with $n = 51$ and $m = 3, 5, 10$, four instances with $n = 100$ and $m = 3, 5, 10, 20$, and five instances with $n = 150$ and $m = 3, 5, 10, 20, 30$. This set of instances is widely used in Venkatesh and Singh (2015), Soylu (2015), Lu and Yue (2019) and Karabulut et al. (2021) for the *minsum* and *minmax* mTSP.
- *Set III*: This set contains a total of 18 instances that consist of six symmetric TSP instances *pr76*, *pr152*, *pr226*, *pr299*, *pr439*, and *pr1002* in the TSPLIB. The number of salesmen is set to be $m = 5, 10, 15$ for these six TSP instances. The maximum number of cities that can be visited by any salesman is set to be $p = 20/40/50/70/100/220$ for the instances in this set with $n = 76/152/226/299/439/1002$. This set of instances is used in Lo et al. (2018) for the *minsum* mTSP.
- *Set IV*: This set contains a total of 24 instances that consist of six symmetric TSP instances *ch150*, *kroA200*, *lin318*, *att532*, *rat783*, and *pcb1173* in the TSPLIB. The number of salesmen is set to be $m = 3, 5, 10, 20$ for these six TSP instances. This set of instances is used in Karabulut et al. (2021) and Wang et al. (2017) for the *minmax* mTSP.

[1] https://people.sc.fsu.edu/%7Ejburkardt/datasets/cities/cities.html
[2] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95

4.2. Baseline algorithms

We compare the ITSHA with the state-of-the-art mTSP heuristics. For example, the artificial bee colony (ABC) algorithms proposed by Venkatesh and Singh (2015), denoted as ABC(FC) and ABC(VC), which represent that a parameter in the ABC algorithm is fixed or variable, respectively. Venkatesh and Singh (2015) also propose an invasive weed optimization (IWO) algorithm. Moreover, the general variable neighborhood search (GVNS) heuristic proposed by Soylu (2015), the ant colony optimization (ACO) algorithm proposed by Lu and Yue (2019), the evolution strategy approach called ES proposed by Karabulut et al. (2021). The above six algorithms can be applied to solve both of the *minsum* and the *minmax* mTSP. In addition, there are some state-of-the-art heuristics aim at solving one of the *minsum* and the *minmax* mTSP. For example, the genetic algorithm for the *minsum* mTSP called GAL proposed by Lo et al. (2018), and the memetic algorithm for the *minmax* mTSP called MASVND proposed by Wang et al. (2017).

In summary, we compare the ITSHA with the above eight heuristics: ABC(FC), ABC(VC), IWO (Venkatesh and Singh, 2015), GVNS (Soylu, 2015), ACO (Lu and Yue, 2019), ES (Karabulut et al., 2021), GAL (Lo et al., 2018), and MASVND (Wang et al., 2017). Note that the results of these algorithms are all from the literature.

The information of these eight algorithms includes the objectives of the problems (*minsum* and *minmax*) that they can solve, the benchmarks they used, the stopping criteria, as well as the systems on which the algorithms run is concluded in Table 1.

4.3. Experimental setup

Our proposed algorithm ITSHA was coded in the C++ Programming Language and was performed on a server with Intel® Xeon® E5-2640 v2 2.00 GHz 8-core CPU and 64 GB RAM. Note that the machine we used is worse than the machines listed in Table 1. The parameters in ITSHA are set as follows: $C_{max} = 10$, $A_t = 3$, $A_c = 5$, $\omega = 2$, $\epsilon = 0.0001$. All these parameter values have been chosen empirically. To have a fair comparison with the baseline algorithms, ITSHA has been run for 10 independent replications for the instances in *Sets I* and *II* as Soylu (2015) did, and 20 independent replications for the instances in *Sets III* and *IV* as Karabulut et al. (2021), Lo et al. (2018) and Wang et al. (2017) did. The cut-off time of the instances in *Sets I*, *II*, and *III* is set to n seconds as Soylu (2015) and Karabulut et al. (2021) did, and $n/5$ s as Karabulut et al. (2021) and Wang et al. (2017) did.

4.4. Various variants of ITSHA

This subsection presents various variant algorithms of ITSHA for comparison and analysis. The variant algorithms are as follows:

- **ITSHA-2opt**: A variant of ITSHA using only the operator 2-*opt* in VNS.
- **ITSHA-Insert**: A variant of ITSHA using only the operator *Insert* in VNS.
- **ITSHA-Swap**: A variant of ITSHA using only the operator *Swap* in VNS.
- **ITSHA-Fix**CS: A variant of ITSHA without adjusting the candidate sets at the end of each iteration.
- **ITSHA-Zero**A_t: A variant of ITSHA without adjusting the solutions during the improvement stage.
- **ITSHA-NoFCM**: A variant of ITSHA without the fuzzy clustering process during the initialization stage.
- **ITSHA-inter-intra**: A variant of ITSHA that restricts that the operator 2-*opt* can only perform inter-tour improvements, and the operators *Insert* and *Swap* can only perform intra-tour improvements.
- **ITSHA-Operator1**: A variant of ITSHA that replaces the proposed operators with the operators in Karabulut et al. (2021).

Table 1
Information of the baseline algorithms and our ITSHA algorithm.

Method	Objectives	Benchmarks	Stopping criterion	Computer
ABC(FC)	*minsum* and *minmax*	*Sets I* and *II*	The maximum number of iterations is 1000	2.83 GHz Core 2 Quad system
ABC(VC)	*minsum* and *minmax*	*Sets I* and *II*	The maximum number of iterations is 1000	2.83 GHz Core 2 Quad system
IWO	*minsum* and *minmax*	*Sets I* and *II*	The maximum number of iterations is 1000	2.83 GHz Core 2 Quad system
GVNS	*minsum* and *minmax*	*Sets I* and *II*	The cut-off time is n seconds	2.4 GHz workstation
ACO	*minsum* and *minmax*	*Set II*	The maximum number of iterations is $10n$	–
ES	*minsum* and *minmax*	*Sets I*, *II* and *IV*	The cut-off time is n seconds for *Sets I* and *II*, $n/5$ s for *Set IV*	Intel Core2 Quad Q9400 CPU with 2.66 GHz
GAL	*minsum*	*Set III*	When the population converges	Intel Core i7-3370 CPU @3.4 GHz
MASVND	*minmax*	*Set IV*	The cut-off time is $n/5$ s	Intel Core i7-3770 CPU @ 3.40 GHz

Table 2
Comparison of ITSHA and the baseline algorithms in solving the *minsum* and *minmax* mTSP on the instances of *Set I*. Best results appear in bold.

Instance	m	ABC(FC)	ABC(VC)	IWO	GVNS	ES	ITSHA	
							Best	Average
minsum								
11a	3	198	198	198	198	198	**196**	196.0
11b	3	**135**	**135**	**135**	**135**	**135**	**135**	135.0
12a	3	199	199	199	199	199	**198**	198.0
12b	3	**2 295**	**2 295**	**2 295**	**2 295**	**2 295**	**2 295**	2295.0
16	3	242	242	242	242	242	**241**	241.0
128	10	30 799	26 482	24 514	22 647	21 354	**21 113**	21160.3
	15	32 777	28 405	26 368	25 204	23 962	**23 838**	23896.6
	30	43 599	41 754	39 579	37 383	36 871	**36 655**	36726.3
minmax								
11a	3	**77**	**77**	**77**	**77**	**77**	**77**	77.0
11b	3	**73**	**73**	**73**	**73**	**73**	**73**	73.0
12a	3	**77**	**77**	**77**	**77**	**77**	**77**	77.0
12b	3	**983**	**983**	**983**	**983**	**983**	**983**	983.0
16	3	**94**	**94**	**94**	**94**	**94**	**94**	94.0
128	10	4 872	4 660	4 450	2 980	2 921	**2 547**	2583.7
	15	3 819	3 958	3 665	2 305	2 406	**2 053**	2072.7
	30	3 456	3 811	3 494	1 980	2 064	**1 859**	1896.3

- **ITSHA-Operator2**: A variant of ITSHA that replaces the proposed operators with the operators in Wang et al. (2017). Note that the operators in Soylu (2015) are contained by those in Wang et al. (2017).
- **ITSHA-k**: A variant of ITSHA that $C_{max} = k$ (we tested $k = 5, 20, 50, n$ in experiments), note that ITSHA is equal to ITSHA-10, and ITSHA-n indicates that the candidate set of each city contains all the other cities.

We tested our ITSHA algorithm on all the instances in the four sets described in Section 4.1, but mainly compared ITSHA with its variants on the most popular and widely used mTSP instances of *Sets I* and *II* to evaluate the effectiveness of the components in ITSHA.

4.5. Comparison on ITSHA and the baselines

Finally, we compare our ITSHA algorithm with the baseline algorithms. Tables 2, 3, 4, and 5 show the results of ITSHA and the baseline algorithms in solving the instances of *Sets I*, *II*, *III*, and *IV*, respectively. In Tables 2, 3, and 5, we provide the best and average solutions of ITSHA, and the best solutions of the other algorithms. Table 4 compares the best and the average solutions of ITSHA and GAL (Lo et al., 2018).

From the results in Tables 2, 3, 4, and 5, we observe that our ITSHA algorithm significantly outperforms other state-of-the-art heuristic algorithms in solving both the *minsum* and *minmax* mTSP. Specifically, ITSHA yields 6/3 new best-known solutions for all the 8 instances of *Set I* with the *minsum/minmax* objective, 9/4 new best-known solutions for all the 12 instances of *Set II* with the *minsum/minmax* objective, 17 new best-known solutions for all the 18 *minsum* mTSP instances of *Set III*, and 15 new best-known solutions for all the 24 *minmax* mTSP instances of *Set IV*. In summary, for all the 38 tested *minsum* mTSP instances, ITSHA can yield better results than the best-known solutions in the literature on 32 instances. For all the 44 tested *minmax* mTSP instances, ITSHA can yield better results than the best-known solutions in the literature on 22 instances. The results demonstrate that our proposed ITSHA algorithm is powerful and effective in solving the mTSP on both the objectives.

4.6. Comparison on local search operators

We first compare the performance of the three local search operators used in ITSHA, including 2-*opt*, *Insert*, and *Swap*. Fig. 2 shows the results of ITSHA, ITSHA-2opt, ITSHA-Insert, and ITSHA-Swap in solving the instances in *Sets I* and *II* except the instances with $n = 11, 12, 16, 51$ (since they are too simple to distinguish the performance of the algorithms). The results are expressed by the ratio of the best solutions obtained in 10 runs of the four algorithms to the best-known solutions in the literature (Venkatesh and Singh, 2015; Soylu, 2015; Lu and Yue, 2019; Karabulut et al., 2021).

From the results in Fig. 2, we can see that:

(1) The order of the three local search operators with decreasing performance is: *Insert*, *Swap*, and 2-*opt*. Therefore, we order *Insert* first, then *Swap*, and finally 2-*opt* in the VNS process of ITSHA.

(2) The performance of ITSHA is better than that of the other three variants, indicating that the VNS method can make use of the complimentary of the three local search operators in searching for better solutions and improve the performance.

(3) Our algorithm shows better performance for solving the instance *128* than the other instances from the TSPLIB (i.e., *kroD100* and *mTSP150*). This might be because the structure of instance *128* is different from that of the instances from the TSPLIB, which is difficult for the baseline algorithms. ITSHA can still solve the instance *128* well, indicating the good robustness of our method for solving instances from various datasets.

Table 3
Comparison of ITSHA and the baseline algorithms in solving the *minsum* and *minmax* *m*TSP on the instances of *Set II*. Best results appear in bold.

Instance	*m*	ABC(FC)	ABC(VC)	IWO	GVNS	ACO	ES	ITSHA	
								Best	Average
minsum									
eil51	3	446	446	446	446	446	446	**443**	443.0
	5	475	472	472	472	472	472	**468**	468.0
	10	580	580	581	580	580	580	**577**	577.0
kroD100	3	21 798	21 798	21 798	21 879	21 798	21 798	**21 796**	21796.0
	5	23 238	23 182	23 294	23 175	23 296	23 175	**23 173**	23173.0
	10	27 023	26 961	26 961	27 008	26 966	26 927	**26 925**	26925.0
	20	39 509	38 333	**38 245**	38 326	**38 245**	**38 245**	38 248	38248.0
mTSP150	3	38 276	38 066	37 957	38 430	37 958	38 072	**37 914**	37947.9
	5	39 309	38 979	**38 714**	39 171	38 729	38 907	38 718	38782.7
	10	43 038	42 441	42 234	42 703	42 234	**42 203**	42 206	42255.7
	20	54 279	53 603	53 475	53 576	53 475	53 343	**53 310**	53386.9
	30	69 048	68 865	68 541	68 558	68 541	68 606	**68 445**	68527.3
minmax									
eil51	3	160	160	160	160	160	160	**159**	159.0
	5	**118**	**118**	**118**	**118**	**118**	**118**	**118**	118.0
	10	**112**	**112**	**112**	**112**	**112**	**112**	**112**	112.0
kroD100	3	8 577	8 509	8 509	8 509	8 511	8 509	**8 507**	8507.0
	5	6 785	6 768	6 767	6 767	6 845	**6 766**	6 770	6774.1
	10	**6 358**	**6 358**	**6 358**	**6 358**	**6 358**	**6 358**	**6 358**	6358.0
	20	**6 358**	**6 358**	**6 358**	**6 358**	**6 358**	**6 358**	**6 358**	6358.0
mTSP150	3	13 896	13 461	13 168	13 376	13 169	13 151	**13 084**	13236.5
	5	8 889	8 678	8 479	8 467	8 467	8 466	**8 465**	8543.6
	10	5 803	5 728	5 594	5 674	5 565	**5 557**	**5 557**	5604.0
	20	**5 246**	**5 246**	**5 246**	**5 246**	**5 246**	**5 246**	**5 246**	5246.0
	30	**5 246**	**5 246**	**5 246**	**5 246**	5 247	**5 246**	**5 246**	5246.0

Table 4
Comparison of ITSHA and the baseline algorithms in solving the *minsum* *m*TSP on the instances of *Set III*. Column *p* indicates the maximum number of cities that can be visited by any salesman. Best results appear in bold.

Instance	*p*	*m*	GAL		ITSHA	
			Best	Average	Best	Average
pr76	20	5	**152 278**	166138.0	152 972	153237.0
		10	177 806	182381.0	**175 676**	175764.0
		15	218 901	223927.0	**216 294**	216294.0
pr152	40	5	116 620	131674.0	**113 887**	114023.3
		10	132 917	141993.0	**122 732**	122863.8
		15	154 249	164741.0	**143 631**	143702.4
pr226	50	5	148 040	156629.0	**144 560**	147462.5
		10	167 782	171338.0	**154 188**	158838.4
		15	180 431	188489.0	**170 320**	174099.2
pr299	70	5	73 177	77676.0	**69 329**	70220.9
		10	75 450	78999.0	**71 803**	72611.3
		15	84 266	87490.0	**79 991**	80740.2
pr439	100	5	141 180	147389.0	**133 197**	133930.8
		10	144 527	151392.0	**135 930**	137000.1
		15	149 649	155512.0	**140 948**	141707.5
pr1002	220	5	332 652	338580.0	**303 142**	308753.2
		10	347 126	360284.0	**317 752**	320886.5
		15	379 677	383360.0	**339 448**	343863.1

4.7. Comparison on different size of candidate sets

We further compare the ITSHA algorithm with different sizes of candidate sets. Fig. 3 shows the results of ITSHA, ITSHA-5, ITSHA-20, ITSHA-50, and ITSHA-*n* in solving the same 12 instances in Fig. 2. The results are expressed by the ratio of the best solution obtained in 10 runs by these five algorithms to the best-known solutions in the literature (Venkatesh and Singh, 2015; Soylu, 2015; Lu and Yue, 2019; Karabulut et al., 2021).

From the results in Fig. 3, we can see that:

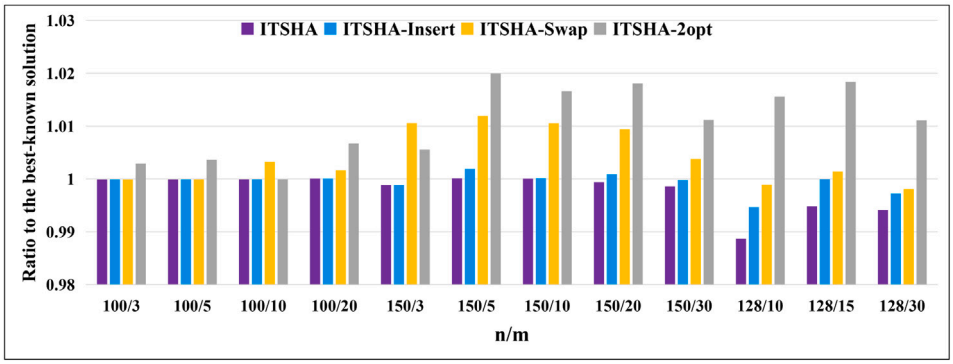
(1) ITSHA with $C_{max} = 10$ (the default value) is the best-performing algorithm among the five algorithms in Fig. 3. The decrease of C_{max} may reduce the search ability of the algorithm, and the increase of C_{max} may reduce the efficiency of the algorithm.

(2) The performance of ITSHA-*n* is much worse than the other four algorithms in Fig. 3, demonstrating that the candidate sets can significantly improve the performance of the VNS process in ITSHA. The results also indicate that the search neighborhoods in ITSHA are much more efficient than those in Soylu (2015), Karabulut et al. (2021) and Wang et al. (2017), since they do not use the candidate sets to refine the search region, but traverse all the possible operators.

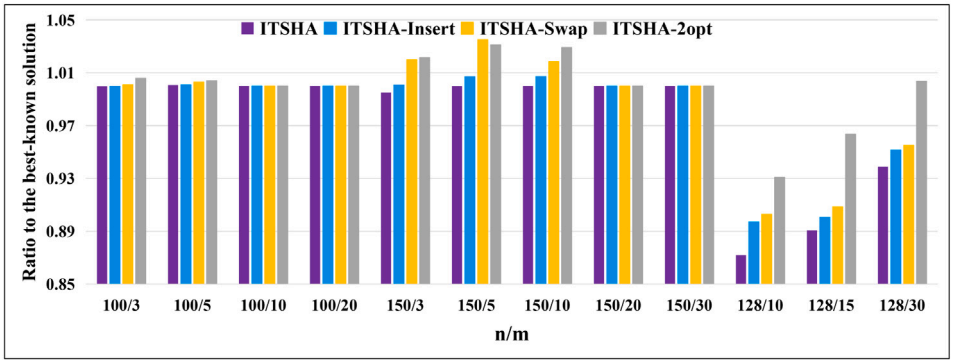
(3) The results demonstrate again that ITSHA shows excellent performance in solving the instance *128*, as ITSHA-k with $k = (5, 20)$ can yield solutions better than the best-known solutions of instance *128* with the *minsum* objective, and ITSHA-k with $k = (5, 20, 50, n)$ can yield solutions better than the best-known solutions of instance *128* with the *minmax* objective.

Table 5
Comparison of ITSHA and the baseline algorithms in solving the *minmax* *m*TSP on the instances of *Set IV*. Best results appear in bold.

Instance	*m*	ABC(FC)	ABC(VC)	IWO	GVNS	MASVND	ES	ITSHA	
								Best	Average
ch150	3	2454.40	2437.04	2413.24	2427.77	2429.49	2407.59	**2405.94**	2435.25
	5	1797.32	1764.65	1752.11	1830.50	1758.08	1741.61	**1740.63**	1765.62
	10	1563.39	1557.94	1554.64	1554.64	1554.64	1554.64	**1554.33**	1554.33
	20	1554.64	1554.64	1554.64	1554.64	1554.64	1554.64	**1554.33**	1554.33
kroA200	3	10976.60	10933.11	10814.18	10898.96	10831.66	10768.10	**10760.69**	10892.27
	5	7795.41	7595.41	7493.24	7836.21	**7415.54**	7572.32	7470.78	7547.11
	10	6291.01	6294.24	6237.00	**6223.22**	**6223.22**	**6223.22**	**6223.22**	6223.22
	20	**6223.22**	**6223.22**	**6223.22**	**6223.22**	**6223.22**	**6223.22**	**6223.22**	6223.22
lin318	3	17062.22	16707.02	16200.21	16861.99	16206.25	16273.80	**15918.24**	16237.07
	5	12449.06	12088.64	11730.03	12210.40	11752.41	11604.20	**11548.44**	11811.14
	10	10061.30	9983.23	9845.72	9826.77	**9731.17**	**9731.17**	**9731.17**	9731.17
	20	**9731.17**	**9731.17**	**9731.17**	**9731.17**	**9731.17**	**9731.17**	**9731.17**	9731.17
att532	3	35138.50	34401.24	32988.99	36395.54	32403.10	33597.40	**32223.24**	32882.79
	5	25033.97	24564.33	23519.68	24866.28	22619.66	23089.70	**22372.68**	22867.24
	10	19949.41	19584.52	19136.52	19278.83	18390.46	**18059.70**	18091.15	18313.77
	20	18332.84	18156.62	17850.80	17822.23	**17641.16**	17641.20	**17641.16**	17700.95
rat783	3	3622.79	3530.31	3457.97	3518.08	3279.16	3369.40	**3158.34**	3227.53
	5	2413.85	2317.66	2273.80	2325.47	2092.77	2127.99	**2024.27**	2077.57
	10	1626.69	1587.43	1542.05	1515.03	1432.34	**1360.89**	1367.98	1393.77
	20	1375.16	1350.96	1311.30	1578.87	1260.88	**1231.69**	**1231.69**	1235.00
pcb1173	3	24748.31	24384.17	24008.47	24988.00	22443.22	22601.70	**20292.61**	20675.14
	5	16590.98	16222.91	16057.19	15494.12	14557.30	14099.50	**12952.97**	13227.20
	10	10965.66	10652.46	10517.94	10386.45	9222.92	8160.25	**7864.11**	8000.58
	20	8373.09	8228.66	8063.17	8311.38	7063.23	6549.14	**6528.86**	6584.69



(a) Comparison on ITSHA with different local search operators in solving the *minsum* *m*TSP



(b) Comparison on ITSHA with different local search operators in solving the *minmax* *m*TSP

Fig. 2. Evaluation of ITSHA with different local search operators in solving the *minsum* and *minmax* *m*TSP.

4.8. Analyses on local optima escaping approaches

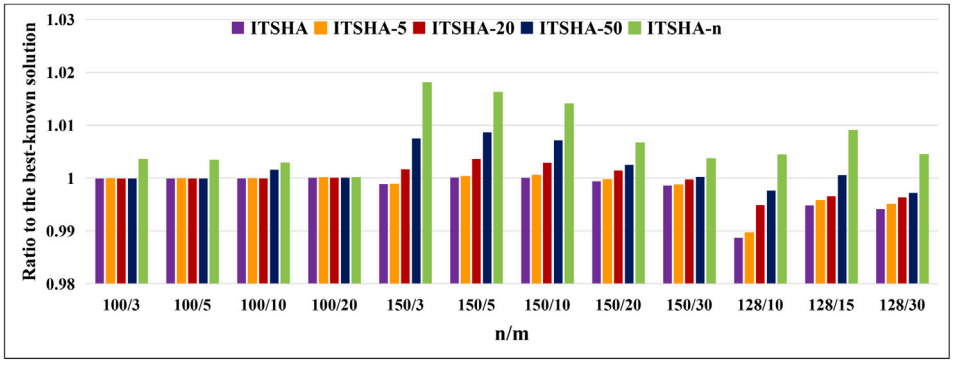
We then analyze the effectiveness of the approach of adjusting candidate sets, the solution adjustment process, the fuzzy clustering algorithm in our ITSHA algorithm. Table 6 shows the results of ITSHA-FixCS, ITSHA-ZeroA_t, ITSHA-NoFCM, and ITSHA in solving the same 12 instances in Fig. 2. The results are expressed by the best solutions obtained in 10 runs by these four algorithms.

From the results we can observe that:

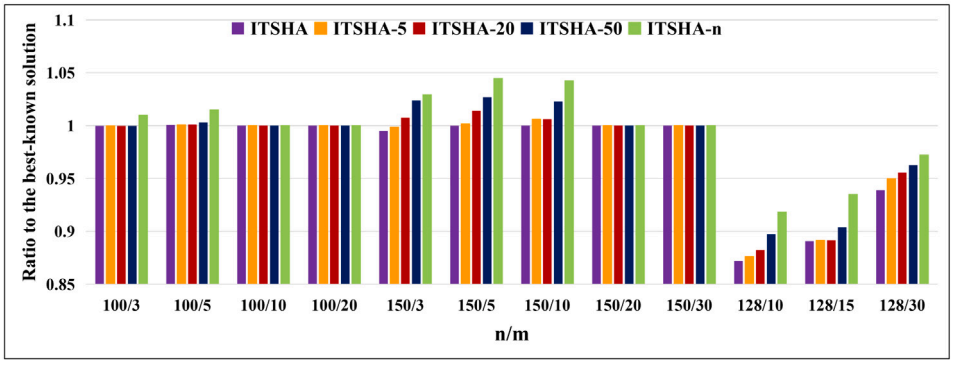
(1) ITSHA outperforms ITSHA-FixCS, indicating that the method of adjusting the candidate sets can improve the performance of ITSHA by helping the algorithm escape from the local optima.

Table 6
Comparison of ITSHA with its variants ITSHA-FixCS, ITSHA-ZeroA_t, and ITSHA-NoFCM.

Instance	kroD100				mTSP150					128		
m	3	5	10	20	3	5	10	20	30	10	15	30
minsum												
ITSHA-FixCS	21 796	23 173	26 925	38 248	37 914	38 720	42 217	53 411	68 552	21 144	23 882	36 706
ITSHA-ZeroA_t	21 796	23 173	26 925	38 248	37 914	38 778	42 206	53 366	68 453	21 123	23 884	36 668
ITSHA-NoFCM	21 796	23 173	26 925	38 248	37 947	38 751	42 208	53 310	68 459	21 145	23 862	36 681
ITSHA	21 796	23 173	26 925	38 248	37 914	38 718	42 206	53 310	68 445	21 113	23 838	36 655
minmax												
ITSHA-FixCS	8 507	6 773	6 358	6 358	13 177	8 493	5 591	5 246	5 246	2 582	2 055	1 884
ITSHA-ZeroA_t	8 507	6 770	6 358	6 358	13 131	8 507	5 591	5 246	5 246	2 556	2 058	1 874
ITSHA-NoFCM	8 507	6 772	6 358	6 358	13 168	8 481	5 593	5 246	5 246	2 563	2 068	1 870
ITSHA	8 507	6 770	6 358	6 358	13 084	8 465	5 557	5 246	5 246	2 547	2 053	1 859



(a) Comparison on different size of candidate sets in solving the *minsum* *m*TSP



(b) Comparison on different size of candidate sets in solving the *minmax* *m*TSP

Fig. 3. Evaluation of ITSHA with different sizes of candidate sets in solving the *minsum* and *minmax* *m*TSP.

(2) ITSHA outperforms ITSHA-ZeroA_t, indicating that the solution adjustment process can also improve the performance. Our ITSHA algorithm does not allow the current solution to be worse than the previous one. Adjusting the solution in each iteration can improve the flexibility and search ability of the algorithm.

(3) ITSHA outperforms ITSHA-NoFCM, indicating that the fuzzy clustering process can improve the algorithm by providing higher-quality initial solutions, since the clustering algorithm can allocate the cities with close positions to one salesman.

4.9. Analyze the superiority of proposed operators

In order to demonstrate the advantages of the proposed operators, we compared ITSHA with its variants, including ITSHA-inter-intra, ITSHA-Operator1, and ITSHA-Operator2, for solving the 12 instances in Fig. 2. Table 7 compares the best solutions obtained in 10 runs by these four algorithms.

From the results we can see that:

(1) ITSHA significantly outperforms ITSHA-inter-intra, demonstrating that our search neighborhoods that can perform both inter-tour and intra-tour improvements are effective. Such a mechanism is one of the factors that explain the success of our proposed operators.

(2) ITSHA significantly outperforms ITSHA-Operator1 and ITSHA-Operator2, indicating that our proposed operators are much better than the operators in Soylu (2015), Karabulut et al. (2021) and Wang et al. (2017). The reasons why our operators show much better performance are as follows. First, we apply the candidate sets to refine the search region. Second, the operators *Insert* and *Swap* can move a sequence of cities, which can help the algorithm find high-quality solutions. Third, our operators can perform both inter-tour and intra-tour improvements.

Table 7
Comparison of ITSHA with its variants ITSHA-inter-intra, ITSHA-Operator1, and ITSHA-Operator2.

Instance	kroD100				mTSP150					128		
m	3	5	10	20	3	5	10	20	30	10	15	30
minsum												
ITSHA-inter-intra	21 796	23 173	26 961	38 248	38 001	39 046	42 572	53 719	68 754	21 202	23 966	36 791
ITSHA-Operator1	22 033	23 404	27 504	39 228	39 034	40 279	43 832	55 828	70 617	29 934	33 495	45 385
ITSHA-Operator2	21 796	23 173	26 925	38 248	38 037	39 129	42 640	53 752	68 926	23 447	26 187	38 515
ITSHA	21 796	23 173	26 925	38 248	37 914	38 718	42 206	53 310	68 445	21 113	23 838	36 655
minmax												
ITSHA-inter-intra	8 507	6 772	6 358	6 358	13 380	8 616	5 640	5 246	5 246	2 618	2 103	1 927
ITSHA-Operator1	8 677	6 806	6 358	6 358	13 883	8 790	5 696	5 246	5 246	3 549	2 701	2 208
ITSHA-Operator2	8 507	6 772	6 358	6 358	13 314	8 568	5 595	5 246	5 246	2 925	2 305	1 969
ITSHA	8 507	6 770	6 358	6 358	13 084	8 465	5 557	5 246	5 246	2 547	2 053	1 859

(3) ITSHA-Operator1 is the worst among the four compared algorithms, because the operators in the ES algorithm (Karabulut et al., 2021) are too simple and ES mainly depends on the evolution method to find high-quality solutions.

5. Conclusion

This paper proposes an iterated two-stage heuristic algorithm, called ITSHA, for the *minsum* and *minmax* multiple Traveling Salesmen Problem (*m*TSP). Each iteration of ITSHA consists of an initialization stage and an improvement stage. The initialization stage containing the fuzzy clustering algorithm and a proposed random greedy function is used to generate high-quality and diverse initial solutions. The improvement stage mainly applies the variable neighborhood search (VNS) approach based on our proposed neighborhoods (2-*opt*, *Insert*, and *Swap*) to improve the initial solution.

Our proposed neighborhoods are effective and efficient, which benefit from the employment of the candidate sets. We further apply some approaches to help the local search algorithm escape from the local optima, for example, the fuzzy clustering algorithm and the random greedy function in the initialization stage, the solution adjustment during the improvement stage, and the method of adjusting candidate sets at the end of each iteration of ITSHA.

In summary, the ITSHA algorithm benefits from the effective and efficient VNS local search and the approaches for escaping from the local optima. Experimental results on 38 *minsum* *m*TSP and 44 *minmax* *m*TSP benchmarks demonstrate that our proposed ITSHA algorithm significantly outperforms the state-of-the-art heuristic algorithms in solving both the *minsum* and *minmax* *m*TSP. Moreover, the proposed search neighborhoods and the local optima escaping approaches could be applied to other combinatorial optimization problems, such as the TSP, VRP, and their various variants.

CRediT authorship contribution statement

Jiongzhi Zheng: Conceptualization, Methodology, Software, Writing – Original Draft, Data Curation. **Yawei Hong:** Investigation, Writing – original draft. **Wenchang Xu:** Investigation, Writing – original draft. **Wentao Li:** Investigation, Writing – original Draft. **Yongfu Chen:** Writing – reviewing and editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Arnold, F., Sörensen, K., 2019. Knowledge-guided local search for the vehicle routing problem. Comput. Oper. Res. 105, 32–46.

Bektas, T., 2006. The multiple traveling salesman problem: an overview of formulations and solution procedures. Omega 34 (3), 209–219.

Bezdek, J.C., Ehrlich, R., Full, W., 1984. FCM: The fuzzy c -means clustering algorithm. Comput. Geosci. 10 (2–3), 191–203.

Carter, A.E., Ragsdale, C.T., 2002. Scheduling pre-printed newspaper advertising inserts using genetic algorithms. Omega 30 (6), 415–421.

Carter, A.E., Ragsdale, C.T., 2006. A new approach to solving the multiple traveling salesperson problem using genetic algorithms. European J. Oper. Res. 175 (1), 246–257.

Cheikhrouhou, O., Khoufi, I., 2021. A comprehensive survey on the multiple traveling salesman problem: Applications, approaches and taxonomy. Comput. Sci. Rev. 40, 100369.

Christofides, N., Mingozzi, A., Toth, P., 1981. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. Math. Program. 20 (1), 255–282.

Defryn, C., Sörensen, K., 2017. A fast two-level variable neighborhood search for the clustered vehicle routing problem. Comput. Oper. Res. 83, 78–94.

Dunn, J.C., 1973. A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. J. Cybern. 3 (3), 32–57.

França, P.M., Gendreau, M., Laporte, G., Müller, F.M., 1995. The *m*-traveling salesman problem with minmax objective. Transp. Sci. 29 (3), 267–275.

Frederickson, G.N., Hecht, M.S., Kim, C.E., 1978. Approximation algorithms for some routing problems. SIAM J. Comput. 7 (2), 178–193.

Helsgaun, K., 2000. An effective implementation of the lin-kernighan traveling salesman heuristic. European J. Oper. Res. 126 (1), 106–130.

Hore, S., Chatterjee, A., Dewanji, A., 2018. Improving variable neighborhood search to solve the traveling salesman problem. Appl. Soft Comput. 68, 83–91.

Kara, I., Bektas, T., 2006. Integer linear programming formulations of multiple salesman problems and its variations. European J. Oper. Res. 174 (3), 1449–1458.

Karabulut, K., Öztop, H., Kandiller, L., Tasgetiren, M.F., 2021. Modeling and optimization of multiple traveling salesmen problems: An evolution strategy approach. Comput. Oper. Res. 129, 105192.

Kytöjoki, J., Nuortio, T., Bräysy, O., Gendreau, M., 2007. An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. Comput. Oper. Res. 34 (9), 2743–2757.

Latah, M., 2016. Solving multiple TSP problem by K-means and crossover based modified ACO algorithm. Int. J. Eng. Tech. Res. 5 (2), 430–434.

Lin, S., Kernighan, B.W., 1973. An effective heuristic algorithm for the traveling-salesman problem. Oper. Res. 21 (2), 498–516.

Liu, W., Li, S., Zhao, F., Zheng, A., 2009. An ant colony optimization algorithm for the multiple traveling salesmen problem. In: IEEE Conference on Industrial Electronics & Applications. pp. 1533—1537.

Lo, K.M., Yi, W.Y., Wong, P.-K., Leung, K.-S., Leung, Y., Mak, S.-T., 2018. A genetic algorithm with new local operators for multiple traveling salesman problems. Int. J. Comput. Intell. Syst. 11 (1), 692–705.

Lu, L.-C., Yue, T.-W., 2019. Mission-oriented ant-team ACO for min-max MTSP. Appl. Soft Comput. 76, 436–444.

Lu, Z., Zhang, K., He, J., Niu, Y., 2016. Applying K-means clustering and genetic algorithm for solving MTSP. In: Proceedings of BIC-TA (2) 2016. Vol. 682. pp. 278–284.

Luo, C., Cai, S., Su, K., Huang, W., 2017. CCEHC: an efficient local search algorithm for weighted partial maximum satisfiability. Artificial Intelligence 243, 26–44.

Miller, C.E., Tucker, A.W., Zemlin, R.A., 1960. Integer programming formulation of traveling salesman problems. J. ACM 7 (4), 326–329.

Miranda, D.M., de Camargo, R.S., Conceição, S. Vieira, Porto, M.F., Nunes, N.T.R., 2018. A multi-loading school bus routing problem. Expert Syst. Appl. 101, 228–242.

Mladenovic, N., Hansen, P., 1997. Variable neighborhood search. Comput. Oper. Res. 24 (11), 1097–1100.

Nagata, Y., Bräysy, O., Dullaert, W., 2010. A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. Comput. Oper. Res. 37 (4), 724–737.

Nagata, Y., Kobayashi, S., 2013. A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. INFORMS J. Comput. 25 (2), 346–363.

Selman, B., Levesque, H.J., Mitchell, D.G., 1992. A new method for solving hard satisfiability problems. In: Proceedings of AAAI 1992. pp. 440–446.

Singh, A., Baghel, A.S., 2009. A new grouping genetic algorithm approach to the multiple traveling salesperson problem. Soft Comput. 13 (1), 95–101.

Soylu, B., 2015. A general variable neighborhood search heuristic for multiple traveling salesmen problem. Comput. Ind. Eng. 90, 390–401.

Tang, L., Liu, J., Rong, A., Yang, Z., 2000. A multiple traveling salesman problem model for hot rolling scheduling in shanghai baoshan iron & steel complex. European J. Oper. Res. 124 (2), 267–282.

Vandermeulen, I., Groß, R., Kolling, A., 2019. Balanced task allocation by partitioning the multiple traveling salesperson problem. In: Proceedings of AAMAS 2019. pp. 1479–1487.

Venkatesh, P., Singh, A., 2015. Two metaheuristic approaches for the multiple traveling salesperson problem. Appl. Soft Comput. 26, 74–89.

Wang, Y., Chen, Y., Lin, Y., 2017. Memetic algorithm based on sequential variable neighborhood descent for the minmax multiple traveling salesman problem. Comput. Ind. Eng. 106, 105–122.

Xu, X., Yuan, H., Liptrott, M., Trovati, M., 2018. Two phase heuristic algorithm for the multiple-travelling salesman problem. Soft Comput. 22 (19), 6567–6581.

Yuan, S., Skinner, B., Huang, S., Liu, D., 2013. A new crossover approach for solving the multiple travelling salesmen problem using genetic algorithms. European J. Oper. Res. 228 (1), 72–82.