A hybrid genetic algorithm for the min–max Multiple Traveling Salesman Problem

Sasan Mahmoudinazlou [a], Changhyun Kwon [b,*]

[a] *Department of Industrial and Management Systems Engineering, University of South Florida, Tampa, FL 33620, USA*
[b] *Department of Industrial and Systems Engineering, KAIST, Daejeon, 34141, Republic of Korea*

ARTICLE INFO

ABSTRACT

This paper proposes a hybrid genetic algorithm for solving the Multiple Traveling Salesman Problem (mTSP) to minimize the length of the longest tour. The genetic algorithm utilizes a TSP sequence as the representation of each individual, and a dynamic programming algorithm is employed to evaluate the individual and find the optimal mTSP solution for the given sequence of cities. A novel crossover operator is designed to combine similar tours from two parents and offers great diversity for the population. For some of the generated offspring, we detect and remove intersections between tours to obtain a solution with no intersections. This is particularly useful for the min–max mTSP. The generated offspring are also improved by a self-adaptive random local search and a thorough neighborhood search. Our algorithm outperforms all existing algorithms on average, with similar cutoff time thresholds, when tested against multiple benchmark sets found in the literature. Additionally, we improve the best-known solutions for 21 out of 89 instances on four benchmark sets.

1. Introduction

The Traveling Salesman Problem (TSP) is a widely recognized combinatorial optimization problem. This problem determines the minimal-cost Hamiltonian cycle – one that visits every city exactly once – using a given set of cities and their respective distances. There are numerous applications for the TSP in the fields of transportation, logistics, and manufacturing.

This paper studies the Multiple Traveling Salesman Problem (mTSP). As a variant of the TSP, an mTSP includes multiple salesmen who start and end their tours at the same depot. The following graph notation is used to represent the problem. We consider $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a complete undirected graph with n vertices representing the cities to be visited. We let m be the number of salesmen available for the task and d be the depot vertex from where each salesman starts and ends their tour. We denote the distance between vertex i and vertex j by c_{ij}.

The purpose of the mTSP is to find a set of m tours, one for each salesman, such that every city is visited exactly once by one of the salesmen. There are two common objective functions for solving the mTSP. The first objective involves minimizing the overall distance traveled by all the salesmen, a problem known as the min-sum mTSP (Lin and Kernighan, 1973). The alternative objective is to minimize the total distance covered by the salesperson with the longest tour, which is commonly referred to as the min–max objective function (França et al., 1995) In our paper, we focus on solving the min–max mTSP problem.

The mTSP solution can be represented using a set of m separate TSP tours, one for each salesman, starting and ending at the depot vertex d. In other words, we need to partition the set of vertices $\mathcal{V}$ into m disjoint subsets $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_m$, where each subset represents the cities visited by a single salesman. Each salesman's tour can be represented as a sequence of vertices that form a Hamiltonian cycle within the corresponding subset of cities. The overall solution can then be represented as a concatenation of the m Hamiltonian cycles in the order in which the salesmen visit the cities.

Note that mTSPs have similar applications as TSPs and are introduced in scenarios in which multiple agents need to cover a set of locations concurrently. The mTSP can, for example, be used to optimize delivery routes for multiple vehicles or drivers in logistics and transportation. mTSP can be used in network design to optimize the placement of facilities such as communication towers and data centers. In manufacturing, the mTSP can be used to optimize the routing of multiple machines or robots that need to work together to produce a product.

The mTSP belongs to the NP-hard class of problems, meaning that its exact solution is computationally impractical when dealing with large data sets. It is, therefore, necessary to use heuristic algorithms

---

* Corresponding author.
E-mail addresses: sasanm@usf.edu (S. Mahmoudinazlou), chkwon@kaist.ac.kr (C. Kwon).

in order to obtain near-optimal solutions within a reasonable period of time. One popular approach is to use genetic algorithms (GA), which are stochastic optimization algorithms inspired by the principles of natural selection and genetics.

In this paper, we propose a hybrid genetic algorithm for solving the mTSP. Our algorithm employs a divide-and-conquer approach, where each individual in the genetic algorithm represents a TSP solution. To determine the optimal mTSP solution for a given TSP sequence, we utilize an algorithm based on dynamic programming. To enhance the diversity of the population, we have developed a new crossover function. This function promotes the exchange of genetic information between similar tours from two parents, thereby introducing greater variation among the individuals. For refining the solutions, we employ a combination of local search methods. These methods aim to further optimize the tours by exploring the neighborhood of each solution. Additionally, we introduce a function specifically designed to detect and remove the intersections between tours. This function ensures that the generated solutions have no intersections, which is particularly beneficial for achieving high-quality solutions in the mTSP. Nevertheless, it is not applied to every individual in the population since there can exist intersections within an optimal solution of the mTSP.

Our paper makes four significant contributions. Firstly, we utilize a method that combines dynamic programming and genetic algorithms to tackle the mTSP. This approach leverages the strengths of both techniques, aiming to improve solutions in the context of this problem. Secondly, we introduce a new crossover function that offers two advantages: it is computationally inexpensive and highly effective in diversifying populations. This contributes to the exploration of a wider solution space and enhances the algorithm's performance. Thirdly, we show that the removal of intersections between tours can lead to better solutions in min–max mTSP. Lastly, we validate the effectiveness of our algorithm through extensive experiments on benchmark instances. Our results not only surpass those of existing algorithms but also demonstrate notable improvements in the best solutions found within a specified time period for multiple instances. These contributions collectively showcase the value and efficacy of our proposed approach for solving the mTSP.

The remainder of this paper is organized as follows. In Section 2, we provide a literature review of existing methods for solving the mTSP, highlighting their strengths and limitations. Section 3 provides a detailed description of our hybrid genetic algorithm, explaining the key components and their interplay in addressing the mTSP. In Section 4, we present the results of our experiments on benchmark instances. Finally, in Section 5, we summarize our findings and contributions, emphasizing the significance of our approach to solving the mTSP. We also discuss potential directions for future research, exploring avenues for further improvement and extension of our algorithm.

2. Literature review

The multiple traveling salesman problem (mTSP) has garnered significant attention in the literature, resulting in extensive research efforts. Among the various approaches proposed, the Genetic Algorithm (GA) has emerged as one of the most popular and successful methods for solving the mTSP. In this section, we review the existing literature that focuses on the utilization of GA for tackling the mTSP, emphasizing its efficiency and effectiveness in addressing the problem. Subsequently, we shift our discussion to explore other heuristics employed in the solution of the mTSP. Lastly, we delve into the variants of the mTSP and discuss the relevant literature dedicated to their study. Notably, many of these studies also employ evolutionary algorithms as a primary tool for addressing the specific problem variants. By providing a comprehensive review of the literature in these three categories, we aim to present a holistic understanding of the research landscape surrounding the mTSP and its various solution methodologies.

Genetic algorithms have emerged as the predominant approach for solving mTSP problems in the literature. Tang et al. (2000) address the mTSP as part of a rolling scheduling system and employs a chromosome encoding scheme that includes the sequence of nodes along with delimiters to separate the tours for each salesperson (see Fig. 1(a)). Using two-chromosome encoding, Park (2001) develops a GA to solve the Vehicle Routing Problem with Time Windows (VRPTW). The first chromosome represents the cities, and the second represents the salesmen who visit those cities (Fig. 1(b)). The study incorporates partially mapped crossover (PMX) and explores four different mutation functions. Carter and Ragsdale (2006) propose a genetic algorithm for solving the mTSP using a two-part chromosome encoding. The first part represents the sequence of cities, and the second part indicates the number of cities visited by each salesperson (see Fig. 1(c)). Furthermore, Chen and Chen (2011) conduct a design of experiments to explore different combinations of crossovers and mutations based on the aforementioned two-part chromosome encoding. This approach allows for the evaluation of various genetic operators' effectiveness in solving the mTSP.

In recent years, several studies have aimed to enhance genetic algorithms for solving the mTSP through the introduction of new crossovers or novel chromosome encodings. Notably, Yuan et al. (2013) propose a new crossover named TCX, specifically tailored for the two-part chromosome representation. TCX operates by treating each salesman separately during the crossover process in the first part of the chromosome, ultimately improving the effectiveness of the algorithm. An alternative approach to chromosome encoding is explored by Brown et al. (2007), which introduces the Grouping Genetic Algorithm (GGA) chromosome representation for encoding mTSP solutions (see Fig. 1(d)). In this encoding, each city carries information about the assigned salesman and its order within the salesman's tour. Singh and Baghel (2009) develop a grouping genetic algorithm using a many-chromosome representation, where each part of a chromosome represents a tour for a specific salesman (see Fig. 1(e)). They also devise a new crossover strategy: in each iteration, the parent is randomly selected, and the shortest tour from the selected parent is added to the offspring. The cities belonging to this tour are removed from the other parent. This process is repeated $m - 1$ times, followed by the addition of the remaining cities based on either a greedy or random approach. This study served as the initial inspiration for our own crossover development. Furthermore, Wang et al. (2017) develop a Memetic Algorithm with a many-chromosome representation utilizing Sequential Variable Neighborhood Descent (MASVND) to address the min–max mTSP. By incorporating these innovative approaches, these studies have contributed to the ongoing advancement and improvement of genetic algorithms for solving the mTSP.
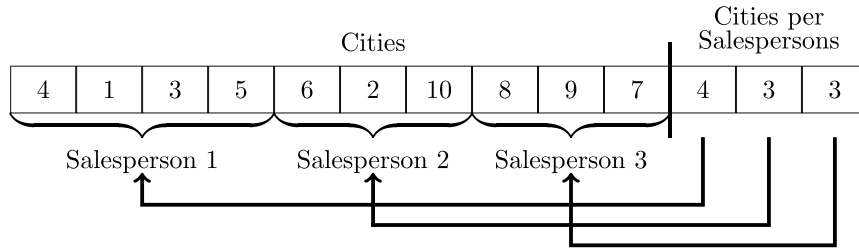
This study proposes a hybrid approach that combines a genetic algorithm, dynamic programming, and local searches to solve the mTSP. The motivation for using this approach stems from our previous work (Mahmoudinazlou and Kwon, 2023), where a divide and conquer approach is employed to solve the Traveling Salesman Problem with Drones (TSPD). In that study, a GA with chromosome representations incorporating the sequences of the truck and drone is used. By utilizing a dynamic programming-based algorithm called JOIN, the optimal location for the drone's launch and landing is determined. Building upon this previous work, the current study adopts a similar approach by utilizing a GA with a simple chromosome representation consisting of TSP sequences (see Fig. 1(g)). To obtain the mTSP solution, a dynamic programming algorithm called SPLIT is employed to determine the optimal delimiters for the given TSP sequence. This approach transfers some decision-making from the GA to the SPLIT algorithm, thereby enhancing decision-making effectiveness and reducing the complexity of the GA. This simplification allows the GA to reach convergence more quickly. Furthermore, this approach enables simultaneous searches in two different regions. The GA focuses on exploring the TSP neighborhood, while the local search functions aim to improve
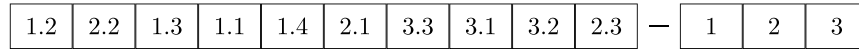
4	1	3	5	−1	6	2	10	−1	8	9	7

salesperson 1   salesperson 2   salesperson 3
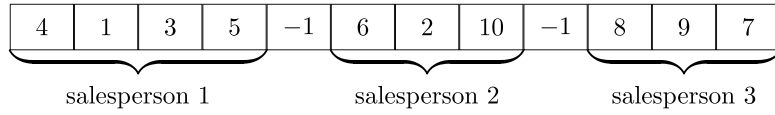
(a) Representation used in Tang et al. (2000). The tours of different salesmen are separated by −1.

Cities

6	4	1	8	2	3	9	5	7	10

Salespersons

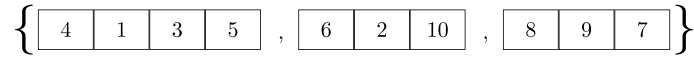| 2	1	1	3	2	1	3	1	3	2

(b) Two chromosome representation. The first chromosome is the sequence of cities, the second is the index of salesmen visiting them.
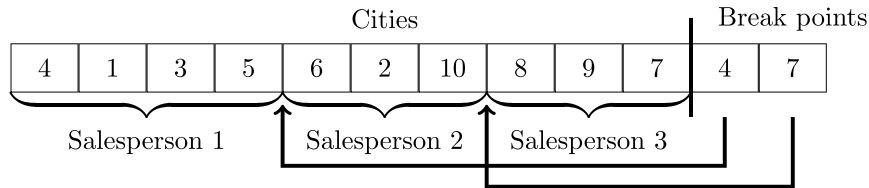
Cities   Cities per Salespersons

4	1	3	5	6	2	10	8	9	7	4	3	3

Salesperson 1   Salesperson 2   Salesperson 3

(c) Two-part chromosome representation.

| 1.2	2.2	1.3	1.1	1.4	2.1	3.3	3.1	3.2	2.3	—	1	2	3

(d) GGA chromosome encoding. The value of 3.2 in the gene position nine means that salesperson three visits city nine, and it is the second city on the route. The second part comprises the group section of the chromosome

$$\left\{ \boxed{4 \;\; 1 \;\; 3 \;\; 5} \;,\; \boxed{6 \;\; 2 \;\; 10} \;,\; \boxed{8 \;\; 9 \;\; 7} \right\}$$

(e) Many-chromosome representation.

Cities   Break points

4	1	3	5	6	2	10	8	9	7	4	7

Salesperson 1   Salesperson 2   Salesperson 3

(f) Two-part chromosome representation with breakpoints.

4	1	3	6	2	10	8	9	7
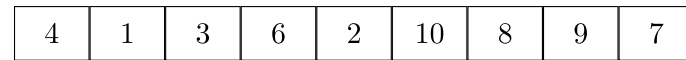
(g) Chromosome representation used in this study, which is a sequence of all cities.

Fig. 1. Different chromosome encodings used in the literature.

the mTSP solution within the mTSP region. By combining GA, dynamic programming, and local searches, this hybrid approach offers several advantages compared to existing methods, including improved convergence speed, optimal decision-making through dynamic programming, and simultaneous searches in multiple areas to enhance the overall solution quality.

Several alternative heuristics have been proposed for solving the mTSP. These methods offer different perspectives and strategies to tackle the problem. Junjie and Dingwei (2006) apply an Ant Colony Optimization (ACO) algorithm to solve the min-sum mTSP. Similarly, Liu et al. (2009) develop an Ant Colony system to address mTSP with multiple objectives. Venkatesh and Singh (2015) introduce two metaheuristic algorithms: one based on the Artificial Bee Colony (ABC) algorithm and another based on the Invasive Weed Optimization (IWO) algorithm. These algorithms are combined with a local search procedure that involves a 2-opt algorithm and a heuristic for reallocating cities from

the longest tour to other tours in a greedy manner. Soylu (2015) proposes a General Variable Neighborhood Search (GVNS) for the mTSP. The study utilizes six different local search functions, including one-point move, two-point move, Or-opt2 move, Or-opt3 move, three-point move, and 2-opt move, to explore the search space effectively. He and Hao (2022) introduce a Hybrid Search with Neighborhood Reduction (HSNR) algorithm for solving the mTSP. The HSNR algorithm follows a two-step iterative process, involving inter-tour and intra-tour searches, to achieve convergence. In the inter-tour step, two neighborhood search functions, an insert operator, and a cross-exchange are employed based on Tabu Search to explore the search space between the tours. The intra-tour optimization is performed using the TSP heuristic EAX, which focuses on improving the individual tours. Zheng et al. (2022) propose an iterated two-stage heuristic algorithm (ITSHA), which includes initialization as stage one and improvement as stage two. In the first stage, the authors create initial solutions by either using a random greedy process or employing C-means clustering. The improvement stage involves a variable neighborhood search that includes a 2-opt move, an insert move, and a swap move.

To the best of our knowledge, Zheng et al. (2022)'s mTSP results are the best on widely used benchmark sets at the time of writing this paper. However, it is worth noting that He and Hao (2023)'s Memetic Algorithm (MA) has outperformed their algorithm with a longer cutoff time. Nevertheless, there are no experimental results available for their MA using the commonly used cutoff time. Therefore, in Section 4, we compare our results primarily with the ITSHA algorithm.

A number of valuable studies have been conducted that utilize reinforcement learning to solve the mTSP. The following are some of these papers that we would like to mention. Hu et al. (2020) propose a multi-agent reinforcement learning algorithm that utilizes GNNs as the training tool to solve mTSP. A multi-agent reinforcement learning algorithm, ScheduleNet, is proposed by Park et al. (2023) for solving the min–max mTSP. Kim et al. (2023) employ GNNs in order to train an agent to use the cross-exchange neighborhood for the solution of the mTSP. Their algorithm is known as the Neuro Cross Exchange (NCE) algorithm, whose results on some benchmark sets appeared promising. Therefore, we chose NCE as one of the baseline algorithms for evaluating our method.

The mTSP family contains a number of variants that deserve attention. Alves and Lopes (2015) propose a bi-objective mTSP that attempts to minimize both the total distance traveled and the longest tour. There are two approaches presented, one that consists of a multi-objective GA and one that consists of a regular GA that combines two objectives. This study utilizes the two-chromosome encoding (Fig. 1(b)). As a result of their findings, multi-objective GA performs better for this bi-objective mTSP. In a variant of the mTSP introduced by Li et al. (2013), certain cities are restricted to a particular salesperson. This problem is referred to as mTSP*. The problem is also solved using GA with two-chromosome encoding. A similar problem is studied by Liu et al. (2021) under the name Visiting Constrained Multiple Traveling Salesman Problem (VCMTSP). According to this problem, each city is restricted to only being visited by certain salesmen who have been predetermined. An evolutionary algorithm with many-chromosome encoding (Fig. 1(e)) and five local search functions is proposed in this study.

There is also a variant of mTSP where there are multiple depots. This problem is known as multi-depot mTSP or M-mTSP. A review of some of the most recent M-mTSP studies is presented. A new ant colony optimization algorithm is proposed by Lu and Yue (2019) for solving M-mTSP. Jiang et al. (2020) propose a hybrid Partheno Genetic Algorithm (PGA) and an ant colony optimization algorithm to solve the M-mTSP. The solution is encoded using a two-part chromosome representation, with the second part representing the breakpoints (Fig. 1(f)). Wang et al. (2020) improve the PGA by incorporating the reproduction mechanism into the invasive weed algorithm. They refer to their algorithm as RIPGA, which uses two-part chromosomes with breakpoints as an encoding method. In order to solve the M-mTSP, Karabulut et al. (2021)

Table 1
Summary of heuristic algorithms solving mTSP.

Algorithm	Problem	Objective	Approach
Tang et al. (2000)	mTSP	min-sum	GA
Park (2001)	VRP	min-sum	GA
Carter and Ragsdale (2006)	mTSP	Both	GA
Junjie and Dingwei (2006)	mTSP	min-sum	AC
Brown et al. (2007)	mTSP	Both	GA
Singh and Baghel (2009)	mTSP	Both	GGA
Liu et al. (2009)	mTSP	Both	AC
Chen and Chen (2011)	mTSP	min-sum	GA
Yuan et al. (2013)	mTSP	Both	GA
Li et al. (2013)	mTSP*	min–max	GA
Venkatesh and Singh (2015)	mTSP	Both	BC/IWA
Soylu (2015)	mTSP	Both	GVNS
Alves and Lopes (2015)	mTSP	bi-objective	GA
Wang et al. (2017)	mTSP	min–max	MA
Lu and Yue (2019)	mTSP	min–max	AC
Jiang et al. (2020)	M-mTSP	min-sum	AC/PGA
Wang et al. (2020)	M-mTSP	min-sum	PGA/IWA
Karabulut et al. (2021)	M-mTSP	Both	ES
Liu et al. (2021)	VCMTSP	min-sum	GA
He and Hao (2022)	mTSP	Both	TS/EAX
Zheng et al. (2022)	mTSP	Both	VNS
He and Hao (2023)	M-mTSP	Both	MA
This study	mTSP	min–max	HGA/DP

propose an Evolution Strategy (ES) with many-chromosome representation. The study utilizes a self-adaptive random local search with four different neighborhoods in order to enhance the solutions. The article by He and Hao (2023) proposes a Memetic Algorithm (MA) that can be utilized to solve the M-mTSP problem by using a generalized edge assembly crossover algorithm (EAX). In order to improve the solution, a variable neighborhood descent is utilized, and a thorough post-optimization is performed to further improve the solution. It should be noted that all algorithms that are designed for M-mTSP are capable of solving mTSP when the number of depots is set to one.

Lastly, we have summarized all the reviewed studies in Table 1 for the convenience of the reader and listed them in chronological order.

3. A hybrid genetic algorithm

This section outlines the structure of our algorithm, starting with the genetic algorithm. The next step is to discuss the details of our dynamic programming-based SPLIT algorithm and how it evaluates an individual. We proceed to explore the various local search functions employed in the algorithm. Lastly, we examine the implementation of removing intersections between tours and highlight its benefits to the min–max mTSP.

Algorithm 1 Hybrid Genetic Algorithm

1: $\Omega = \texttt{initial\_population()}$ ▷ Section 3.2
2: **while** Stopping condition is not met **do**
3:    $\texttt{sort}(\Omega)$ ▷ Based on fitness and diversification factor
4:    Select ω_1 and ω_2 from Ω
5:    $c \leftarrow \texttt{crossover}(\omega_1, \omega_2)$ ▷ Section 3.3
6:    $\omega \leftarrow \textsc{Split}(c)$ ▷ Algorithms 2 and 3
7:    $\texttt{educate}(\omega)$ ▷ Section 3.4
8:    $\Omega_{\text{F}} \leftarrow \Omega_{\text{F}} \cup \{\omega\}$
9:    **if** $\texttt{size}(\Omega) = \mu + \lambda$ **then**
10:      $\texttt{select\_survivors}(\Omega)$
11:    **end if**
12:    **if** $\texttt{best}(\Omega)$ not improved for It_{DIV} iterations **then**
13:      $\texttt{diversify}(\Omega)$
14:    **end if**
15: **end while**
16: Return $\texttt{best}(\Omega)$

A simplified version of Vidal et al. (2012)'s GA structure is used in this study. While our GA does not have any infeasible regions,

the population control is similar to Vidal et al. (2012), in which the number of individuals varies between μ and $\mu + \lambda$. An overview of the procedure is presented in Algorithm 1. The initial population is generated by producing μ individuals as described in Section 3.2 (line 1). The following steps are repeated until the stopping condition is met (lines 2–15). The algorithm terminates when either the number of iterations without improvement reaches It_{NI} or the total running time of the algorithm reaches the predetermined cutoff time (line 2).

First, the population is sorted based on fitness and diversification factor (line 3), which is explained in Section 3.1. Using the Tournament Selection method, two individuals are selected as parents from the population (line 4). In order to select each parent, $k_{\text{TOURNAMENT}}$ individuals are randomly selected, and the best of them is chosen. Upon selection of two parents from the population, a Similar Tour Crossover (STX) is performed on them in order to create the child's chromosome (line 5). More detailed information about STX crossover can be found in Section 3.3. The generated chromosome is a permutation of all cities that must be converted into a feasible mTSP solution. The SPLIT algorithm is employed on the chromosome to divide the given sequence into m tours in an optimal manner (line 6). Next, the child undergoes a combination of local search functions and improvements before being added to the population (line 7). If the population size reaches $\mu + \lambda$, the individuals will be sorted in accordance with their fitness (min–max objective), the first μ individuals will survive, while the remainder will be discarded (lines 9–11). In order to prevent the algorithm from becoming trapped in a local optimum, when there are no improvements after It_{DIV} generations, an attempt is made to diversify the population by keeping n_{BEST} number of individuals and adding new chromosomes similar to the generation of the initial population to increase the population size to μ again (lines 12–14).

Following is a summary of the remainder of this section: In Section 3.1, we introduce our dynamic programming algorithm, SPLIT, which is specifically designed for individual evaluation. Section 3.2 describes the process of generating the initial population. Section 3.3 provides an explanation of the STX crossover. Lastly, in Section 3.4, we discuss the steps involved in educating the child.

3.1. Individual evaluation by Split

As previously mentioned, our GA utilizes a TSP sequence (excluding depot) as the chromosome representation. In this paper, we propose a dynamic programming algorithm that takes a TSP tour as input, identifies the delimiters, and then computes the optimal solution for the given sequence in the mTSP context. As illustrated in Fig. 2, each chromosome representation is a TSP tour after adding the depot at the start and end. Without changing the order of the cities, the Split algorithm divides the initial sequence into m distinct sequences, each of which would become a tour after the depot is added.

The SPLIT algorithm, initially introduced by Prins (2004) for VRP to minimize total travel distance, has been adapted and modified for min–max mTSP in our study. To facilitate understanding, we now provide the notation used in the algorithm:

- n: the number of customer nodes
- $S = \{S_1, S_2, \ldots, S_n\}$: the given TSP sequence
- m: the number of salesmen
- t_{S_i, S_j}: travel time from node S_i to node S_j
- k: position of the current node, $k \in \{1, 2, \ldots, n\}$
- r: the number of completed routes
- R_k: set of all possible numbers of completed routes at position k, where

$$R_k = \begin{cases} \{0\} & \text{if } k = 0, \\ \{1, \ldots, \min(k, m-1)\} & \text{if } 1 \leq k < n, \\ \{1, \ldots, m\} & \text{if } k = n \end{cases}$$

- V_r^k: the makespan (min–max objective) of the path from the beginning depot to the node in position k, while completing r routes
- P_r^k: the predecessor of k, while r routes has been completed

---

Algorithm 2 SPLIT for *min-max* mTSP

1: $V_0^0 \leftarrow 0$
2: $V_r^k \leftarrow +\infty \quad \forall k = 1$ to n , $\forall r$ in R_k
3: **for** $k = 1$ to n **do**
4:      **for** $r \in R_{k-1}$ **do**
5:          **if** $V_r^{k-1} < \infty$ **then**
6:              $T \leftarrow 0; j \leftarrow k$
7:              **while** $j \leq n$ **do**
8:                  **if** k=j **then**
9:                      $T \leftarrow t_{0, S_j} + t_{S_j, 0}$ ▷ 0 represents depot.
10:                  **else**
11:                      $T \leftarrow T - t_{S_{j-1}, 0} + t_{S_{j-1}, S_j} + t_{S_j, 0}$
12:                  **end if**
13:                  **if** $r + 1 \in R_j$ **then**
14:                      **if** $\max(V_r^{k-1}, T) < V_{r+1}^j$ **then**
15:                          $V_{r+1}^j \leftarrow \max(V_r^{k-1}, T)$
16:                          $P_{r+1}^j \leftarrow k - 1$
17:                      **end if**
18:                  **end if**
19:                  $j \leftarrow j + 1$
20:              **end while**
21:          **end if**
22:      **end for**
23: **end for**

Algorithm 3 Extracting the mTSP solution

1: **for** $r = 1$ to m **do**
2:      $tour(r) \leftarrow \emptyset$
3: **end for**
4: $r \leftarrow m$
5: $j \leftarrow n$
6: **while** $r > 1$ **do**
7:      $i \leftarrow P_r^j$
8:      **for** $k = i + 1 : j$ **do**
9:          $tour(r) \leftarrow tour(r) \cup S_k$
10:      **end for**
11:      $r \leftarrow r - 1$
12:      $j \leftarrow i$
13: **end while**

Algorithm 2 presents the details of the SPLIT algorithm. Our objective is to minimize the longest tour from the depot to the kth node in the given TSP sequence by constructing a series of r tours. For each step in the DP algorithm, we perform forward propagation on V_r^k to generate a new tour and update the value of V_{r+1}^j for all $j > k$. This process continues until all possible enumerations have been considered. The resulting V_m^n yields the objective of optimal mTSP solution for the given sequence. Additionally, we utilize P_r^k for extracting the mTSP solution through Algorithm 3.

The optimal objective value of SPLIT is the minimum time of the longest tour in the mTSP solution of an individual in the population. To promote diversity in the population, we calculate the fitness measure for each individual using the min–max objective function and the diversity factor. According to Vidal et al. (2012), the normalized Hamming distance $\delta^H(P_1, P_2)$ can be calculated as follows:

$$\delta^H(P_1, P_2) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(P_1[i] \neq P_2[i]),$$
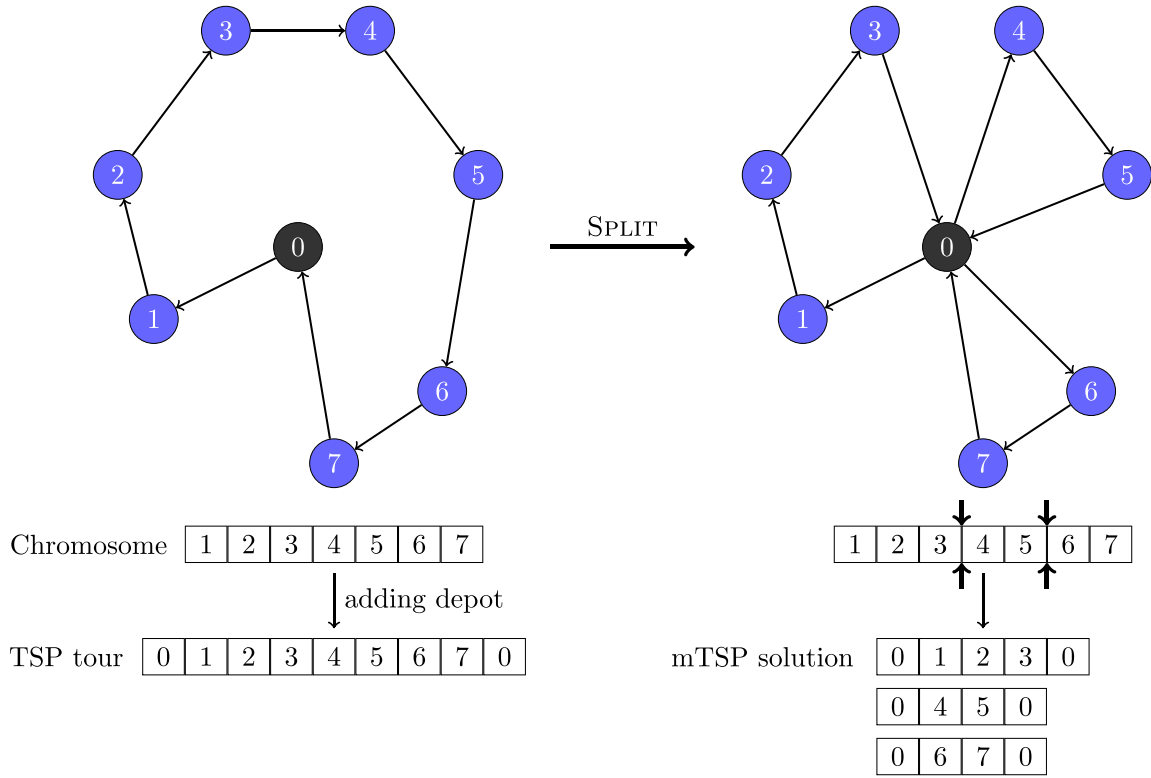
Fig. 2. Illustration of SPLIT algorithm.

where $\mathbf{1}(\cdot)$ equals 1 if the condition specified within the parentheses is true and 0 otherwise, and n represents the number of customer nodes. The distance is measured between zero and one, where a value of one indicates that the representations of two individuals are completely different, and a value of zero indicates that the representations are similar. To quantify diversity, we calculate the diversity contribution $\Delta(P)$ based on the average Hamming distance between an individual P and its two closest neighbors in the population. The fitness function of each individual is calculated using the following formula:

$$\text{fitness}(P) = \text{min–max}(P) \times \left(1 - \frac{n_{\text{ELITE}}}{n_{\text{POPULATION}}}\right)^{\Delta(P)},$$

where we let min–max(P) be the optimal minimized value of the longest tour found by the SPLIT algorithm for individual P.

3.2. Initial population

The SPLIT algorithm can be employed on a TSP tour to generate an mTSP solution. It is expected that different TSP tours will result in different mTSP solutions, and, thus a more diverse population. This study utilizes four TSP algorithms: an exact algorithm implemented in *Concorde* (Applegate et al., 2002) and three simple heuristic algorithms of *nearest insertion, farthest insertion*, and *cheapest insertion*. To create the remainder of the initial population, we randomly select one of the available TSP tours, modify it, and then apply the SPLIT algorithm to obtain the individual. Changing the TSP tour is accomplished in one of the following ways:

- By selecting two randomly chosen positions on the tour, the nodes between these positions are inversed.
- Take the TSP tour and randomly divide it into m subtours, and then shuffle the order of the subtours.
- Select a random number r between 2 and $\frac{n}{2}$, then randomly select r positions and shuffle their order.

Until μ individuals are present in the initial population, this process is repeated.

3.3. Similar tour crossover (STX)

In order to replicate the mTSP solution from two parents, we develop a novel crossover function. Our proposed crossover works as follows: A tour is randomly chosen from the first parent, and then we select a tour from the second parent that has the maximum number of mutual cities with it. Afterward, a simple two-point crossover is implemented between selected tours, and the resulting tour is added to the child. In other words, two positions within the range of both tours are selected at random, and the resulting tour comprises the cities in between on the tour with a smaller number of cities along with the cities outside the chosen positions from the other tour. This process should be repeated until the child has m tours. At the end of the process, some cities may appear more than once, and some may be absent from the solution. The repeating cities will be removed from the child's tour. In the final step, all remaining cities will be assigned to the child's tour based on a greedy approach. There should be an examination of all possible insertions for each city except for the longest tour, and the city should then be placed in the position with the least amount of increase. This crossover combines tours that share the most similarities in cities; hence the name Similar Tour Crossover (STX) is chosen. Fig. 3 illustrates a detailed example of the STX, which shows how it is employed to generate children from two parents. By conducting experiments on different instances, Section 4.6 examines the effectiveness of STX crossover.

3.4. Chromosome education

Our genetic algorithm is referred to as *hybrid* because each offspring undergoes operations designed to improve its quality and, in general, to enhance the quality of the gene pool of the population. Three sequential layers are involved in the improvement of the offspring. At the first level, attention is paid to the geometry of the solution and the removal of intersections between different tours. The second layer attempts to make improvements on different tours regardless of the objective
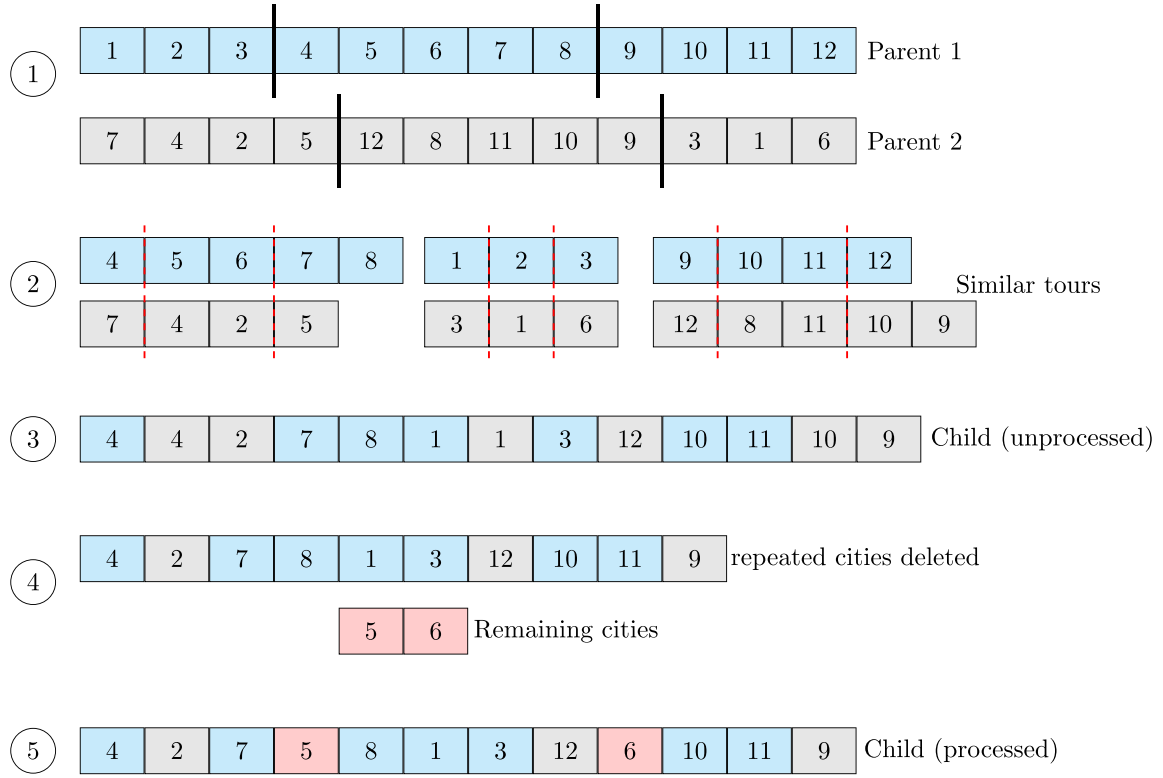
Fig. 3. Illustration of STX crossover for a problem with size $n = 12$ and $m = 3$.

function, which is minimizing the longest tour. Lastly, the third layer focuses on reducing the length of the longest tour.

Removing intersections

During this study, we discovered that for min–max mTSP, tours of optimal or near-optimal solutions tend to be separate and have the fewest intersections, particularly when the depot is in the center. Therefore, before beginning the local searches, with probability P_{REMOVE}, a function is employed on the generated offspring that detects the intersections and removes them by exchanging sub-sequences of the intersected tours. An illustration of the implementation of this function can be observed in Fig. 4, which represents the intermediate and final solutions for the problem MTSP-150 with $m = 5$ salesmen, from instance *Set III* (Section 4.3). A diagram of the generated offspring in an intermediate generation, where the tours intersect, can be seen in Fig. 4(a). As an example, tour 1 and tour 3 intersect between cities $[38, 67]$ from tour 1 and $[84, 49]$ from tour 3. When this intersection is located, the function shifts the sub-sequence $[84, 144, 24, 60]$ from tour 3 to tour 1 as well as the sub-sequence $[67, 72, 68, 2, 107]$ from tour 1 to tour 3.

The result of repeating this procedure for all intersections is an mTSP solution with no intersections, as shown in Fig. 4(b). It is important to note that it is very likely that the objective will worsen as a result of this function being employed. Nevertheless, implementing this technique once in a while may contribute to the gene pool moving toward the (near) optimal solution shown in Fig. 4(c). In addition, this function contributes to a more diverse solution, even for problems involving intersections in the optimal solution. Considering the drastic changes it makes to the solution, it can be considered a mutation. Getting out of the local optima can be accomplished by the sudden changes between tours that no local search can induce. A number of experiments are conducted in Section 4.6 in order to assess the contribution of removing the intersections.

Local searches

We now discuss how our algorithm works to perform local searches. The most commonly used neighborhoods in the mTSP (and VRP) can be divided into two categories, inter-tour neighborhoods and intra-tour neighborhoods. In the inter-tour neighborhoods, improvements are made by making changes between tours, while in the intra-tour neighborhoods, improvements are made within the tours. From inter-tour neighborhoods, we use 1-shift move and 1-swap move in this study. The 1-shift move involves taking a city from one tour and relocating it to another tour for possible improvement (Fig. 5(a)). In a 1-swap move, two cities from two different tours are exchanged (Fig. 5(b)).

Our study utilizes Reinsert, Exchange, Or-opt2, Or-opt3, and 2-opt intra-tour neighborhoods out of many available. There is only one tour involved in these neighborhood functions. In a Reinsert move, one city is removed and then placed in a new location (Fig. 6(a)). The Exchange move swaps two cities within a tour (Fig. 6(b)). The Or-opt2 and Or-opt3 moves are similar to Reinsert, except they remove two and three adjacent cities, respectively, and place them in a different location (Figs. 6(c) and 6(d)). The 2-opt move reverses a sub-sequence within the tour (Fig. 6(e)).

The critical point now is how and in what order these neighborhoods are utilized. The improvement process is conducted sequentially using two different functions. The first function, which is the second layer of improvement, involves taking each tour and finding its neighbors based on the distances between their cities. Each city from the chosen tour is investigated for a 1-shift move as well as a 1-swap move for all possible positions and cities from neighboring tours. When the 1-shift move is utilized, it decreases the length of the tour from which the city is removed and increases the length of the tour to which the chosen city is added. When the amount of decrease is greater than the amount of increase, and the length of the increased tour does not exceed the maximum tour length, the 1-shift move is implemented. The 1-swap move is implemented if both tour lengths decrease. Until no further improvements are achieved, this process is repeated. Upon completion

(a) Before removing the intersections.

(b) After removing the intersections.

(c) Best Known solution.
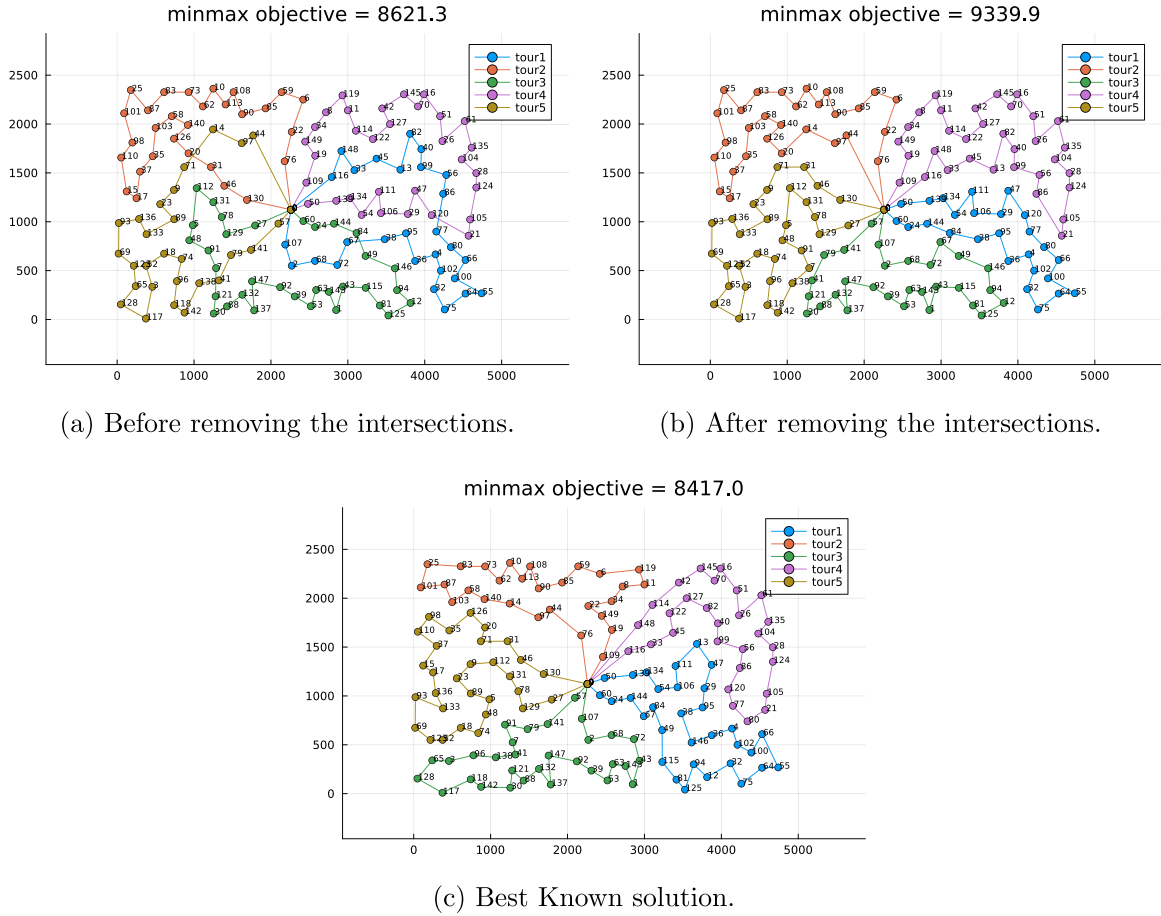
Fig. 4. The importance of removing the intersections on instance MTSP-150 with 5 salesmen.

(a) *1-shift* move.
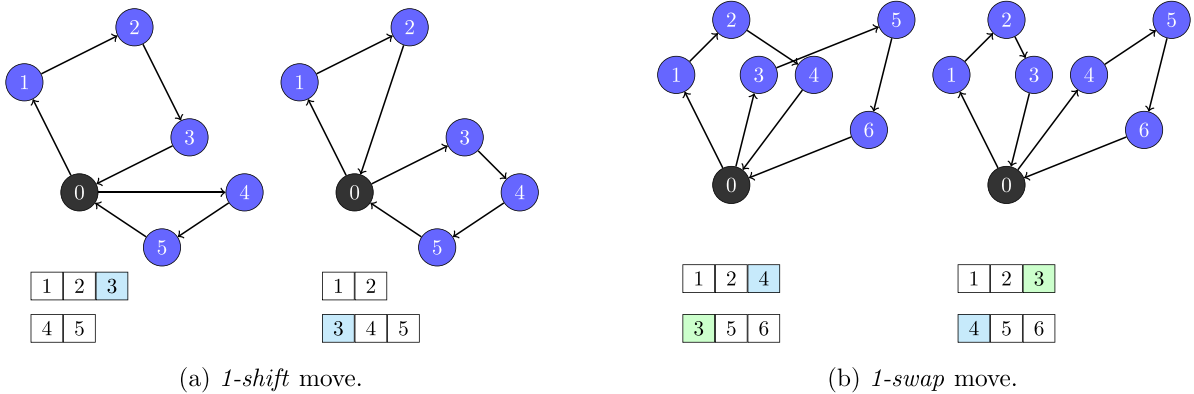
(b) *1-swap* move.

Fig. 5. Inter-route neighborhoods.

of the 1-shift and 1-swap moves, we employ the 2-opt move on all tours for all pairs of cities until no further gains can be achieved. We refer to the first function as Enriching the chromosome since it adds value to the genes of an individual.

All that remains is to explain the third layer of offspring improvement. As soon as the enrichment process is completed, the second function is activated for further improvement. The second function employs the Reinsert, Exchange, Or-opt2, and Or-opt3 moves in a self-adaptive manner. By using a roulette wheel, the number of improvements made by each of these moves is counted, and the probability of choosing a move depends on the outcome of the roulette wheel. By dividing the number of improvements occurring in each move by the total number of improvements, the roulette wheel probabilities are

calculated. As an example, if the number of improvements by the search functions is 100, 200, 200, and 500, then the roulette wheel would have probabilities of 0.1, 0.2, 0.2, and 0.5. As a starting point, the counts of improvements for all the moves are initialized to 100 in order to create fair competition. Choosing a tour from the solution is required in order to implement each of these moves. Since the longest tour plays a critical role in the achievement of the min–max objective, improving the longest tour is of greater importance than improving the others. Improvements in non-dominant tours can, however, also contribute to the enhancement of the gene pool. As a result, when implementing any of these intra-tour moves, we either choose the longest tour or a random tour. Using this strategy, we are certain that at least half of the time will be spent on improving the objective function. The second

(a) *Reinsert* move.

(b) *Exchange* move.

(c) *Or-opt2* move.

(d) *Or-opt3* move.

(e) *2-opt* move.

Fig. 6. Intra-route neighborhoods.

function is called Improving the chromosome. Since we only examine one move randomly in each implementation, the improving function repeats the roulette wheel and local search process multiple times for each child. It should be noted, however, that there is a potential trade-off here. A strong implementation of local search at the beginning would prevent GA from diversifying the population and would result in premature convergence. In contrast, 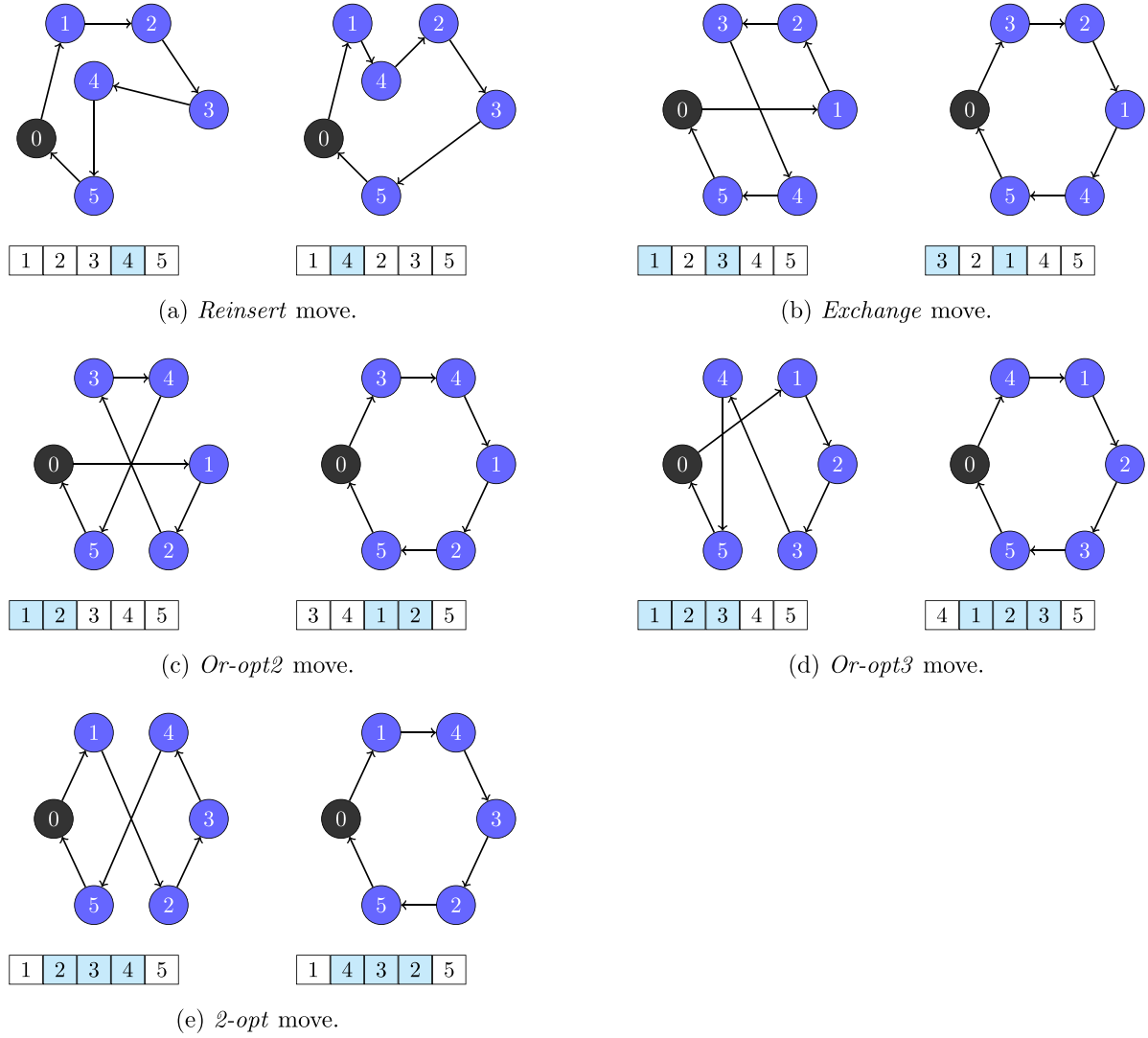if local search is not employed sufficiently, local optima are less likely to be escaped. Thus, if the number of generations with no improvement is less than n_{IMPRV}, the improving process must be repeated n^1_{LOCAL} times; otherwise n^2_{LOCAL} times, where $n^1_{\text{LOCAL}} < n^2_{\text{LOCAL}}$. As a final note, it is important to note that the process of selecting cities for potential relocations is not based solely on randomness. Our method only selects randomly from the nearest n_{CLOSE} cities instead of all of them in order to save time.

4. Computational results

In order to evaluate the effectiveness of our HGA, four benchmark sets of instances are used. Using Julia programming language, the algorithm is implemented on a Mac computer with 16 GB of RAM and an Apple M1 processor. The parameters used in our HGA are $\mu = 10$, $\lambda = 20$, $k_{\text{TOURNAMENT}} = 2$, $It_{\text{DIV}} = 1000$, $n_{\text{BEST}} = 0.2\,\mu$, $n_{\text{ELITE}} = 0.2 n_{\text{POPULATION}}$, $P_{\text{REMOVE}} = 0.1$, $n_{\text{IMPRV}} = 100$, $n^1_{\text{LOCAL}} = 100$, $n^2_{\text{LOCAL}} = 1000$, $n_{\text{CLOSE}} = 0.1n$.

The following benchmark instances are used for evaluating our HGA and comparing it with the best existing methods in the literature:

- *Set I*: The number of cities in instances of this set is 50, 100, and 200. There are three different numbers of salesmen for each size, resulting in nine total settings. Cities are randomly generated nodes.
- *Set II*: A total of 16 instances are included in this set, as defined by Necula et al. (2015). There are four TSP instances, eil51, berlin52, eil76, and rat99, from TSPLIB that are solved with $m = 2, 3, 5$ and 7 numbers of salesmen.
- *Set III*: A widely studied set of instances defined by Carter and Ragsdale (2006), which includes three Euclidean two-dimensional symmetric TSP instances with 51, 100, and 150 cities. Solving each instance for different numbers of salesmen (from $m = 3, 5, 10, 20$ and 30) gives us 12 instances in total.
- *Set IV*: A set of 24 instances defined by Wang et al. (2017) that consists of six TSP instances, ch150, kroA200, lin318, att532, rat783 and cb1173 from TSPLIB. In this set, $m = 3, 5, 10$ and 20 salesmen are used.

4.1. Comparative results for Set I

This set of instances contains three sizes, 50, 100, and 200 cities (including the depot). The problem is solved with $m = 5, 7$ and 10 for $n = 50$. In the case of $n = 100$ and $n = 200$, the number of

Table 2
Results of HGA for random instances (*Set I*).

N	Method	5			7			10		
		Cost	Gap (%)	Time (s)	Cost	Gap (%)	Time (s)	Cost	Gap (%)	Time (s)
50	LKH-3	**2.00**	0.00	187.46	1.95	0.00	249.31	1.91	0.00	170.20
	OR-tools	2.04	2.00	3.24	1.96	0.51	3.75	1.91	0.00	3.67
	ScheduleNet	2.17	8.50	1.60	2.07	6.15	1.67	1.98	3.66	1.90
	NCE-mTSP	2.02	1.00	2.48	1.96	0.51	2.50	1.91	0.00	3.44
	HGA-avg	**2.00**	0.00	2.75	**1.92**	−1.53	2.39	**1.82**	**−4.71**	2.31

	Method	5			10			15		
		Cost	Gap (%)	Time (s)	Cost	Gap (%)	Time (s)	Cost	Gap (%)	Time (s)
100	LKH-3	**2.20**	0.00	262.85	1.97	0.00	474.78	1.98	0.00	378.90
	OR-tools	2.41	9.55	35.47	2.03	3.05	45.40	2.03	2.53	48.86
	ScheduleNet	2.59	17.73	14.84	2.13	8.12	16.22	2.07	4.55	20.02
	NCE-mTSP	2.24	1.82	16.36	1.97	0.00	13.00	1.98	0.00	23.37
	HGA-avg	2.21	0.45	8.06	**1.95**	**−1.02**	6.51	**1.96**	**−1.01**	6.58

	Method	10			15			20		
		Cost	Gap (%)	Time (s)	Cost	Gap (%)	Time (s)	Cost	Gap (%)	Time (s)
200	LKH-3	2.04	0.00	1224.40	**2.00**	0.00	1147.13	**1.97**	0.00	908.14
	OR-tools	2.33	14.22	675.79	2.33	16.50	604.31	2.37	20.30	649.17
	ScheduleNet	2.45	20.10	193.41	2.24	12.00	213.07	2.17	10.15	225.50
	NCE-mTSP	2.06	0.98	84.96	**2.00**	0.00	84.28	2.02	2.54	108.91
	HGA-avg	**2.02**	**−0.98**	27.99	**2.00**	0.00	23.32	1.98	0.51	26.66

Table 3
Results of HGA for *Set II*. The 'best gap' is the gap between HGA-best and BKS. The 'average gap' is the gap between HGA-avg and NCE-mTSP.

Instance	eil51				berlin52				eil76				rat99			
m	2	3	5	7	2	3	5	7	2	3	5	7	2	3	5	7
CPLEX	**222.7**	**159.6**	124.0	**112.1**	**4110.2**	3244.4	2441.4	**2440.9**	280.9	197.3	150.3	139.6	728.8	587.2	469.3	443.9
LKH3	**222.7**	**159.6**	124.0	**112.1**	**4110.2**	3244.4	2441.4	**2440.9**	280.9	197.3	150.3	139.6	728.8	587.2	469.3	443.9
OR-Tools	243.3	170.5	127.5	**112.1**	4665.5	3311.3	2482.6	**2440.9**	318.0	212.4	143.4	128.3	762.2	552.1	473.7	442.5
NCE	235.0	170.3	121.6	**112.1**	4110.0	3274.0	2660.0	**2441.0**	285.5	211.0	144.6	**127.6**	695.8	527.8	458.6	441.6
NCE-mTSP	226.1	166.3	119.9	**112.1**	4128.0	3191.0	2474.0	**2441.0**	282.1	197.5	147.2	**127.6**	666.0	533.2	462.2	443.9
BKS	222.7	159.6	119.9	112.1	4110.2	3191.0	2441.4	2440.9	280.9	197.3	143.4	127.6	666.0	527.8	458.6	441.6
HGA-best	**222.7**	**159.6**	118.1	**112.1**	**4110.2**	3069.6	2440.9	**2440.9**	280.9	196.7	**142.9**	**127.6**	666.0	517.7	450.3	**436.7**
HGA-avg	222.9	**159.6**	118.3	**112.1**	**4110.2**	3073.8	2440.9	**2440.9**	281.6	196.7	143.5	**127.6**	668.0	517.8	450.3	437.3
Time (s/run)	4.0	3.2	3.7	2.7	4.1	3.3	2.9	2.8	5.2	5.0	6.4	4.7	8.0	6.7	9.7	16.2
Best gap	0.00%	0.00%	−1.50%	0.00%	0.00%	−3.80%	−0.02%	0.00%	0.00%	−0.30%	−0.35%	0.00%	0.00%	−1.91%	−1.81%	−1.11%
avg gap	−1.42%	−4.03%	−1.33%	0.00%	−0.43%	−3.67%	−1.34%	0.00%	−0.18%	−0.41%	−2.51%	0.00%	0.30%	−2.89%	−2.57%	−1.49%

salesmen is 5, 10, 15 and 10, 15, 20, respectively. 100 instances are generated randomly for each combination using a uniform distribution over $[0, 1] \times [0, 1]$. Table 2 presents the average results of all 100 instances. The HGA is stopped when there are no improvements observed after 2500 generations ($It_{NI} = 2500$). HGA has been implemented 10 times for each instance; therefore, HGA-avg represents the average performance of HGA. LKH-3 (Helsgaun, 2017), OR-tools (Perron and Furnon, 2023), ScheduleNet (Park et al., 2023) and NCE (Kim et al., 2023) are used as baseline algorithms for comparison. HGA has obtained the best results in five problem sizes out of nine and equal results in two. Only in two problem sizes does HGA fail to defeat LKH-3. The difference in computational time, however, is extremely significant.

4.2. Comparative results for Set II

In this set of instances, four TSPLIB-based instances are solved with different numbers of salesmen: $m = 2, 3, 5,$ and 7. The stopping condition for HGA is the same as Section 4.1. Ten runs of the HGA have been conducted for each instance, and the average and best results are reported. The baseline algorithms used for comparison include CPLEX, LKH-3, OR-tools, NCE, and NCE-mTSP (Kim et al., 2023). Table 3 presents the results of the HGA and baseline algorithms. The CPLEX results represent the best bounds determined by CPLEX. The results for the baseline algorithms are reported based on Park et al. (2023) and Kim et al. (2023). Since the running times of the baseline algorithms were not provided, only the running time of the HGA is reported. Among the 16 problems in this set, the HGA's solution is equal

to or better than the baseline algorithms in all cases and strictly better in 8 cases. To the best of our knowledge, the reported BKS (Best Known Solution) row in Table 3 represents the best-known solutions to these problems. Therefore, the HGA has improved the BKS for exactly half of the problems in this set.

4.3. Comparative results for Set III

This set of instances, proposed by Carter and Ragsdale (2006), consists of three instances named MTSP-51, MTSP-100, and MTSP-150. MTSP-51 is solved with $m = 3, 5$ and 10 salesmen, MTSP-100 is solved with $m = 3, 5, 10$ and 20, and MTSP-150 is solved with $m = 3, 5, 10, 20$ and 30 salesmen. Table 4 presents detailed results for the HGA and its comparison with baseline algorithms. The following algorithms are used as baselines: ES (Karabulut et al., 2021), HSNR (He and Hao, 2022), and ITSHA (Zheng et al., 2022). ES and ITSHA are run 10 times with a cutoff time of n s to stop the algorithm, where n is the number of cities. HSNR is tested on this set's instances 20 times with two different cutoff times: $(1.36 \times n)/5$ s and $(n/100) \times 4$ min $(2.4 \times n$ s). To provide a fair comparison, the results are compared with the $(1.36 \times n)/5$ s cutoff time of HSNR. In this study, the HGA is implemented on each instance 10 times with a cutoff time of $n/5$ s. The decision to use a cutoff time of $n/5$ s for the HGA was based on the aim of demonstrating its fast convergence capability.

With the exception of He and Hao (2022, 2023), all the studies utilize integer distances when analyzing this set of instances. He and Hao (2022, 2023) used floating point distances. Therefore, we decided

---

Table 4

Results of HGA for *Set III*. Bold values in the 'best' columns are the values equal to BKS, which are the best-known solutions in the literature regardless of the computational time. The shaded values are the new best solutions achieved by HGA.

Algorithm																
				Integer distances							Floating point distances				
Instance	m	BKS	ES	ITSHA		HGA		Gap over		BKS	HSNR		HGA		Gap over	
			best	best	avg	best	avg	best	avg		best	avg	best	avg	best	avg
MTSP-51	3	159	160	**159**	159.0	**159**	159.0	0.00%	0.00%	159.57	**159.57**	159.99	**159.57**	159.57	0.00%	−0.26%
	5	118	**118**	**118**	118.0	**118**	118.0	0.00%	0.00%	118.13	**118.13**	118.13	**118.13**	118.13	0.00%	0.00%
	10	112	**112**	**112**	112.0	**112**	112.0	0.00%	0.00%	112.07	**112.07**	112.07	**112.07**	112.07	0.00%	0.00%
MTSP-100	3	8507	8509	**8507**	8507.0	**8507**	8507.0	0.00%	0.00%	8509.16	**8509.16**	8578.51	**8509.16**	8509.16	0.00%	−0.81%
	5	6766	**6766**	6770	6774.1	6769	6770.6	−0.01%	−0.05%	6765.73	**6765.73**	6774.24	6767.82	6770.50	0.03%	−0.06%
	10	6358	**6358**	**6358**	6358.0	**6358**	6358.0	0.00%	0.00%	6358.49	**6358.49**	6358.49	**6358.49**	6358.49	0.00%	0.00%
	20	6358	**6358**	**6358**	6358.0	**6358**	6358.0	0.00%	0.00%	6358.49	**6358.49**	6358.49	**6358.49**	6358.49	0.00%	0.00%
MTSP-150	3	13039	13151	13084	13236.5	**13039**	13087.7	−0.34%	−1.12%	13038.30	13174.30	13352.37	13038.34	13093.48	−1.03%	−1.94%
	5	8416	8466	8465	8543.6	**8416**	8500.5	−0.58%	−0.50%	8417.02	8479.60	8602.15	**8417.02**	8486.81	−0.74%	−1.34%
	10	5557	**5557**	**5557**	5604.0	5564	5582.9	0.13%	−0.38%	5557.41	5616.71	5668.59	5561.97	5587.37	−0.97%	−1.43%
	20	5246	**5246**	**5246**	5246.0	**5246**	5246.0	0.00%	0.00%	5246.49	**5246.49**	5246.49	**5246.49**	5246.49	0.00%	0.00%
	30	5246	**5246**	**5246**	5246.0	**5246**	5246.0	0.00%	0.00%	5246.49	**5246.49**	5246.49	**5246.49**	5246.49	0.00%	0.00%
Average		5490	5504	5498	5521.9	5491	5503.8	−0.07%	−0.17%	5490.61	5512.10	5548.00	5491.17	5503.92	−0.23%	−0.49%

to solve the instances of this set with both integer and floating point distances. For ES, only the best results are displayed in Table 4. As for ITSHA and HSNR, both the best and average results are reported, and HGA's gaps are compared to these results. According to the table, HGA outperforms the baseline algorithms in terms of best and average solutions. In 10 of 12 instances, HGA achieves results equal to BKS when the distance is an integer. Additionally, it improves the BKS in two instances. When compared with ITSHA's average results, HGA's average results are equal or better in all instances. The HGA achieves results that are equal to the BKS in 9 out of 12 instances when considering floating point distances. The average results of HGA are equivalent or superior to those of HSNR in all instances. It is worthy of note that He and Hao (2023) contributed to finding some of the best-known solutions in the second BKS column. However, since they implemented MA with a cutoff time of $2.4 \times n$ s, their results were not included in the table. Nevertheless, their best result for MTSP-150 is identical to ours for $m = 3$ and almost the same as ours for $m = 5$. In the latter case, MA obtained 13 038.30 and HGA obtained 13 038.34. We can still claim that our results represent the new best solutions with a cutoff time of $n/5$ s, as HGA achieved these solutions 12 times faster.

4.4. Comparative results for Set IV

The set of instances in Table 5 is introduced by Wang et al. (2017) and contains larger instances from TSPLIB. The table follows a similar structure to Table 4 and presents the results of experiments conducted with HGA and baseline algorithms. The baseline algorithms used in this set are MASVND (Wang et al., 2017), ES (Karabulut et al., 2021), HSNR (He and Hao, 2022) and ITSHA (Zheng et al., 2022). Similar to our algorithm, all algorithms stop after $n/5$ s, except for HSNR, which has a cutoff time of $(1.36 \times n)/5$. HSNR and MA were also tested on this set with a cutoff time of $2.4 \times n$ s, and a separate experiment was conducted to compare HGA, HSNR, ITSHA, and MA over a long period of time in Section 4.5. In this instance set, each instance is solved 20 times by all algorithms, including HGA. To the best of our knowledge, the reported BKS column represents the best-known solutions to these problems. Table 5 shows that HGA achieves equal or better results in 22 out of 24 instances compared to the baseline algorithms. Additionally, it improves the best results (within $n/5$ s) for 13 instances in this dataset. On average, HGA outperforms the other algorithms in 18 instances and ties in the remaining instances out of the 24. Based on the current literature reports for a cutoff time of $n/5$ s, ITSHA has the best results on this dataset. However, HGA's best and average results are 0.81% and 1.23% lower, respectively, than ITSHA's best and average results. Moreover, HGA's average performance is either equal to or superior to ITSHA's average in all instances of *Set IV*. Thus, based on the current literature, HGA has achieved the best performance for a cutoff time of $n/5$ s.

4.5. Comparison of the results for a cutoff time of $(n/100) \times 4$ min

In order to compare HGA with MA (He and Hao, 2023), we conduct experiments on *Set III*, *Set IV*, and an additional instance called rand100. The cutoff time used for these experiments is $(n/100) \times 4$ min, as used by MA. The baseline algorithms in this section are HSNR (He and Hao, 2022), ITSHA (Zheng et al., 2022), and MA (He and Hao, 2023). The reported results for ITSHA are provided by He and Hao (2023), and we trust their reported results and present them as they are. It may be unclear to the reader why the results of the att532 instance are not consistent with Table 5. This discrepancy arises because both He and Hao (2022, 2023) use pseudo-Euclidean distances for this instance, as specified in the TSPLIB documentation. Consequently, we perform the same analysis using pseudo-Euclidean distances for HGA as well. Table 6 shows the results of the comparison between HGA and MA. On average, HGA has a slight advantage over MA, with a difference of 0.15%. However, we cannot claim that HGA outperforms MA significantly due to the small difference. Among the 28 instances, HGA finds solutions that are equal to or better than the best-known solutions in 22 instances. Additionally, HGA identifies five new best solutions. In terms of average results across the 28 instances, HGA outperforms MA in 10 cases, loses in 7 cases, and ties in 11 cases.

4.6. Effectiveness of STX and removing intersections

In this section, we conduct an experimental analysis to evaluate the effectiveness of our proposed STX crossover technique, as well as the impact of removing intersections in the context of our problem. A number of instances are selected from various benchmark sets and range in size from 76 to 1173 with different numbers of salesmen. To assess the effectiveness of the STX crossover, we compare its performance with the order-based crossover (OX) method, which is described in Larranaga et al. (1999). Additionally, we examine the impact of removing intersections by solving the instances both with and without this operation. Table 7 presents the results obtained from different combinations of crossover techniques and the presence or absence of intersection removal. To evaluate the effectiveness of STX, we focus on comparing the columns where the status of intersection removal is the same. When intersections are not removed, employing STX results in an average improvement of 1.65% (as observed by comparing the first and second columns). However, when intersections are removed, utilizing STX leads to a greater improvement of 2.31% (as shown in the third and fourth columns). It appears that they may have a synergistic effect. Likewise, we investigate the impact of removing intersections by fixing the crossover function. In this case, we observe an improvement of 1.55% on average by comparing the third and first columns, where OX is used. Furthermore, removing intersections contributes to an average improvement of 1.71% when STX is employed (as seen in the fourth and second columns).

Table 5
Results of HGA for *Set IV*. Bold values in best columns are the values equal or better than BKS, which are the best-known solutions in the literature regardless of the computational time. The shaded values are the new best solutions achieved by HGA.

Algorithm			MASVND	ES	HSNR	ITSHA		HGA		Gap over	
Cutoff time (s)			*n*/5	*n*/5	1.36*n*/5	*n*/5		*n*/5		ITSHA	
Instance	*m*	BKS	best	best	best	best	avg	best	avg	best	avg
ch150	3	2 401.63	2 429.49	2 407.59	2 425.87	2 405.94	2 435.25	**2 401.63**	2 405.79	−0.18%	−1.21%
	5	1 740.63	1 758.08	1 741.61	1 744.26	**1 740.63**	1 765.62	1 741.13	1 741.82	0.03%	−1.35%
	10	1 554.64	**1 554.64**	**1 554.64**	**1 554.64**	1 554.33	1 554.33	**1 554.64**	1 554.64	0.00%	0.00%
	20	1 554.64	**1 554.64**	**1 554.64**	**1 554.64**	1 554.33	1 554.33	**1 554.64**	1 554.64	0.00%	0.00%
kroA200	3	10 691.00	10 831.66	10 768.10	10 801.80	10 760.69	10 892.27	**10 691.03**	10 734.69	−0.65%	−1.45%
	5	7 412.12	7 415.54	7 572.32	7 418.87	7 470.78	7 547.11	7 421.01	7 459.46	−0.67%	−1.16%
	10	6 223.22	**6 223.22**	**6 223.22**	**6 223.22**	**6 223.22**	6 223.22	**6 223.22**	6 223.22	0.00%	0.00%
	20	6 223.22	**6 223.22**	**6 223.22**	**6 223.22**	**6 223.22**	6 223.22	**6 223.22**	6 223.22	0.00%	0.00%
lin318	3	15 663.50	16 206.25	16 273.80	16 094.90	15 918.24	16 237.07	15 698.61	15 806.81	−1.38%	−2.65%
	5	11 276.80	11 752.41	11 604.20	11 458.20	11 548.44	11 811.14	11 289.26	11 372.09	−2.24%	−3.72%
	10	9 731.17	**9 731.17**	**9 731.17**	**9 731.17**	**9 731.17**	9 731.17	**9 731.17**	9 731.17	0.00%	0.00%
	20	9 731.17	**9 731.17**	**9 731.17**	**9 731.17**	**9 731.17**	9 731.17	**9 731.17**	9 731.17	0.00%	0.00%
att532	3	32 223.24	32 403.10	33 597.40	–	32 223.24	32 882.79	**31 651.32**	32 043.34	−1.77%	−2.55%
	5	22 372.68	22 619.66	23 089.70	–	22 372.68	22 867.24	**22 007.07**	22 266.07	−1.63%	−2.63%
	10	18 059.70	18 390.46	18 059.70	–	18 091.15	18 313.77	**18 042.64**	18 155.17	−0.27%	−0.87%
	20	17 641.16	**17 641.16**	17 641.20	–	**17 641.16**	17 700.95	**17 641.16**	17 641.16	0.00%	−0.34%
rat783	3	3 052.41	3 279.16	3 369.40	3 262.52	3 158.34	3 227.53	3 128.69	3 193.79	−0.94%	−1.05%
	5	1 961.12	2 092.77	2 127.99	2 066.38	2 024.27	2 077.57	1 996.33	2 037.13	−1.38%	−1.95%
	10	1 313.01	1 432.34	1 360.89	1 358.06	1 367.98	1 393.77	1 320.35	1 335.60	−3.48%	−4.17%
	20	1 260.88	1 231.69	**1 231.69**	**1 231.69**	**1 231.69**	1 235.00	**1 231.69**	1 231.84	0.00%	−0.26%
pcb1173	3	19 569.50	22 443.22	22 601.70	21 430.10	20 292.61	20 675.14	19 962.93	20 481.85	−1.62%	−0.93%
	5	12 406.60	14 557.30	14 099.50	13 402.30	12 952.97	13 227.20	12 812.20	13 130.61	−1.09%	−0.73%
	10	7 623.59	9 222.92	8 160.25	8 120.45	7 864.11	8 000.58	7 681.41	7 827.65	−2.32%	−2.16%
	20	6 528.86	7 063.23	6 549.14	**6 528.86**	**6 528.86**	6 584.69	**6 528.86**	6 552.97	0.00%	−0.48%
Average			9 909.07	9 886.43	–	9 608.80	9 745.51	9 511.07	9 601.49	−0.81%	−1.23%

Table 6
Results of HGA with a cutoff time of (*n*/100)×4 min. Bold values in best columns are the values equal to or better than BKS, which are the best-known solutions in the literature regardless of the computational time. The shaded values are the new best solutions achieved by HGA.

Instance	*m*	BKS	HSNR		ITSHA		MA		HGA		Gap over MA	
			best	avg	best	avg	best	avg	best	avg	best	avg
rand-100	3	3 031.95	**3 031.95**	3 032.67	**3 031.95**	3 033.65	**3 031.95**	3 031.95	**3 031.95**	3 031.95	0.00	0.00
	5	2 409.63	2 411.68	2 415.00	2 411.68	2 414.65	**2 409.63**	2 409.65	**2 409.63**	2 409.63	0.00	0.00
	10	2 299.16	**2 299.16**	2 299.16	**2 299.16**	2 299.16	**2 299.16**	2 299.16	**2 299.16**	2 299.16	0.00	0.00
	20	2 299.16	**2 299.16**	2 299.16	**2 299.16**	2 299.16	**2 299.16**	2 299.16	**2 299.16**	2 299.16	0.00	0.00
ch150	3	2 401.63	2 407.34	2 435.49	2 407.34	2 416.87	**2 401.63**	2 401.86	**2 401.63**	2 401.92	0.00	0.00
	5	1 741.13	1 741.71	1 743.48	**1 741.13**	1 752.06	**1 741.13**	1 741.13	**1 741.13**	1 741.13	0.00	0.00
	10	1 554.64	**1 554.64**	1 554.64	**1 554.64**	1 554.64	**1 554.64**	1 554.76	**1 554.64**	1 554.64	0.00	−0.01
	20	1 554.64	**1 554.64**	1 554.64	**1 554.64**	1 554.64	**1 554.64**	1 554.64	**1 554.64**	1 554.64	0.00	0.00
kroA200	3	10 691.00	10 748.10	10 987.69	10 700.57	10 819.85	**10 691.00**	10 691.41	**10 691.03**	10 700.06	0.00	0.08
	5	7 412.12	7 418.87	7 494.44	7 449.22	7 513.67	**7 412.12**	7 414.21	7 418.87	7 420.23	0.09	0.08
	10	6 223.22	**6 223.22**	6 223.22	**6 223.22**	6 223.22	**6 223.22**	6 249.10	**6 223.22**	6 223.22	0.00	−0.41
	20	6 223.22	**6 223.22**	6 223.22	**6 223.22**	6 223.22	**6 223.22**	6 223.22	**6 223.22**	6 223.22	0.00	0.00
lin318	3	15 663.50	15 902.50	16 207.05	15 930.04	16 088.56	**15 663.50**	15 699.92	**15 663.54**	15 714.31	0.00	0.09
	5	11 276.80	11 295.20	11 596.35	11 430.65	11 601.67	**11 276.80**	11 291.59	11 288.52	11 297.35	0.10	0.05
	10	9 731.17	**9 731.17**	9 731.17	**9 731.17**	9 731.17	**9 731.17**	9 731.17	**9 731.17**	9 731.17	0.00	0.00
	20	9 731.17	**9 731.17**	9 731.17	**9 731.17**	9 731.17	**9 731.17**	9 731.17	**9 731.17**	9 731.17	0.00	0.00
att532	3	9 966.00	10 231.00	10 565.30	10 158.00	10 344.50	**9 966.00**	10 064.00	9 973.00	10 038.05	0.07	−0.26
	5	6 986.00	7 067.00	7 334.00	7 067.00	7 156.80	6 986.00	7 070.95	**6 950.00**	7 003.80	−0.52	−0.95
	10	5 709.00	**5 709.00**	5 738.90	5 731.00	5 787.50	5 770.00	5 796.75	**5 698.00**	5 716.75	−1.25	−1.38
	20	5 580.00	**5 580.00**	5 580.00	5 583.00	5 601.75	**5 580.00**	5 589.35	**5 580.00**	5 580.00	0.00	−0.17
rat783	3	3 052.41	3 187.90	3 237.29	3 131.99	3 180.79	**3 052.41**	3 083.52	3 086.70	3 106.39	1.12	0.74
	5	1 961.12	2 006.46	2 044.32	2 018.44	2 058.65	**1 961.12**	1 989.68	**1 959.16**	1 981.51	−0.10	−0.41
	10	1 313.01	1 334.76	1 345.88	1 357.65	1 381.69	**1 313.01**	1 325.54	**1 304.70**	1 321.33	−0.63	−0.32
	20	1 231.69	**1 231.69**	1 231.69	**1 231.69**	1 231.84	**1 231.69**	1 235.37	**1 231.69**	1 231.69	0.00	−0.30
pcb1173	3	19 569.50	20 813.80	21 144.92	20 288.75	21 144.92	**19 569.50**	19 858.77	19 724.24	20 016.94	0.79	0.80
	5	12 406.60	13 032.30	13 216.99	12 816.55	13 216.99	**12 406.60**	12 636.49	12 517.65	12 655.64	0.90	0.15
	10	7 623.59	7 758.26	7 897.20	7 801.18	7 910.09	7 623.59	7 745.00	**7 512.43**	7 617.21	−1.46	−1.65
	20	6 528.86	**6 528.86**	6 528.86	**6 528.86**	6 534.75	**6 528.86**	6 548.87	**6 528.86**	6 528.86	0.00	−0.31
Average			6 922.17	7 042.20	7 139.50	7 016.30	7 115.04	6 924.71	6 967.85	6 928.72	6 962.13	−0.03	−0.15

Table 7
Contribution of STX crossover and removing the intersections.

Crossover		OX	STX		OX		STX	
Intersections		Not removed	Not removed		Removed		Removed	
n	m	avg (10 runs)	avg	imprv	avg	imprv	avg	imprv
76	2	283.91	283.28	−0.22%	281.38	−0.89%	281.62	−0.81%
99	3	520.44	520.74	0.06%	517.72	−0.52%	517.81	−0.51%
99	7	438.01	437.88	−0.03%	437.58	−0.10%	437.31	−0.16%
150	3	13 287.01	13 154.68	−1.00%	13 230.87	−0.42%	13 093.48	−1.46%
150	5	8 679.41	8 591.76	−1.01%	8 594.86	−0.97%	8 486.81	−2.22%
150	3	2 468.89	2 440.98	−1.13%	2 423.64	−1.83%	2 405.79	−2.56%
200	3	11 078.12	10 822.11	−2.31%	10 961.14	−1.06%	10 734.69	−3.10%
200	5	7 641.64	7 545.30	−1.26%	7 551.27	−1.18%	7 459.46	−2.38%
318	3	16 510.62	15 981.36	−3.21%	16 276.37	−1.42%	15 806.81	−4.26%
318	5	11 854.43	11 643.22	−1.78%	11 713.21	−1.19%	11 372.09	−4.07%
783	10	1 482.70	1 399.26	−5.63%	1 394.32	−5.96%	1 335.60	−9.92%
783	20	1 255.48	1 245.00	−0.83%	1 239.74	−1.25%	1 231.84	−1.88%
1173	10	8 687.84	8 300.38	−4.46%	8 347.27	−3.92%	7 827.65	−9.90%
1173	20	6 714.53	6 697.22	−0.26%	6 651.21	−0.94%	6 552.97	−2.14%
Average		6 478.63	6 361.66	−1.65%	6 401.47	−1.55%	6 253.14	−3.26%

5. Conclusions and future directions

This paper presented a hybrid genetic algorithm (HGA) for solving the multiple traveling salesman problem (mTSP) with a min–max objective function. The HGA utilized TSP sequences to represent solutions and employed a dynamic programming-based algorithm called SPLIT to determine the optimal mTSP solution for each sequence. To generate tours for the offspring, a novel crossover algorithm called STX was designed, which combines similar tours from the parent solutions. The HGA further enhanced solutions using different neighborhoods as local searches. Moreover, the paper introduced a novel approach to detect and remove intersections between different tours, which proved to be highly effective for mTSPs with min–max objectives. The HGA was evaluated on four different datasets, comprising a total of 89 instances. In 76 out of the 89 problems, HGA's best solution either matched or surpassed the existing best-known solutions. Additionally, HGA improved the best-known solutions in 21 instances. Furthermore, HGA demonstrated equal or superior average performance in 78 out of the 89 problems compared to the baseline algorithm. Overall, the paper presented a hybrid genetic algorithm that leverages TSP sequences, dynamic programming, novel crossover, local search, and intersection detection techniques to solve the mTSP with a min–max objective. The experimental results indicated the effectiveness of the proposed approach in achieving high-quality solutions and outperforming the baseline algorithms in terms of both best and average performance.

In addition to the current findings, there are several promising avenues for future research in the domain of the min–max multiple traveling salesman problem (mTSP). One potential direction is to extend the proposed hybrid genetic algorithm (HGA) to handle the multi-depot mTSP. This would require modifications to the SPLIT algorithm and the crossover functions to accommodate multiple depots. Furthermore, considering the emerging field of drone technology, there is an opportunity to explore the integration of drones as agents in the mTSP. This would involve addressing challenges such as limited flight time.

CRediT authorship contribution statement

Sasan Mahmoudinazlou: Methodology, Software, Formal analysis, Writing – original draft. **Changhyun Kwon:** Conceptualization, Validation, Writing – review & editing, Supervision.

Data availability

We will share the data and code via Github upon publication.

References

Alves, R.M., Lopes, C.R., 2015. Using genetic algorithms to minimize the distance and balance the routes for the multiple traveling salesman problem. In: 2015 IEEE Congress on Evolutionary Computation (CEC). IEEE, pp. 3171–3178.

Applegate, D., Cook, W., Dash, S., Rohe, A., 2002. Solution of a min–max vehicle routing problem. INFORMS J. Comput. 14 (2), 132–143.

Brown, E.C., Ragsdale, C.T., Carter, A.E., 2007. A grouping genetic algorithm for the multiple traveling salesperson problem. Int. J. Inf. Technol. Decis. Mak. 6 (02), 333–347.

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.

Chen, S.-H., Chen, M.-C., 2011. Operators of the two-part encoding genetic algorithm in solving the multiple traveling salesmen problem. In: 2011 International Conference on Technologies and Applications of Artificial Intelligence. IEEE, pp. 331–336.

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

He, P., Hao, J.-K., 2022. Hybrid search with neighborhood reduction for the multiple traveling salesman problem. Comput. Oper. Res. 142, 105726.

He, P., Hao, J.-K., 2023. Memetic search for the min–max multiple traveling salesman problem with single and multiple depots. European J. Oper. Res. 307 (3), 1055–1070.

Helsgaun, K., 2017. An Extension of the Lin-Kernighan-Helsgaun Tsp Solver for Constrained Traveling Salesman and Vehicle Routing Problems, Vol. 12. Roskilde University, Roskilde.

Hu, Y., Yao, Y., Lee, W.S., 2020. A reinforcement learning approach for optimizing multiple traveling salesman problems over graphs. Knowl.-Based Syst. 204, 106244.

Jiang, C., Wan, Z., Peng, Z., 2020. A new efficient hybrid algorithm for large scale multiple traveling salesman problems. Expert Syst. Appl. 139, 112867.

Junjie, P., Dingwei, W., 2006. An ant colony optimization algorithm for multiple travelling salesman problem. In: First International Conference on Innovative Computing, Information and Control-Volume I (ICICIC'06), Vol. 1. IEEE, pp. 210–213.

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.

Kim, M., Park, J., Park, J., 2023. Learning to CROSS exchange to solve min–max vehicle routing problems. In: The Eleventh International Conference on Learning Representations. URL https://openreview.net/forum?id=ZcnzsHC10Y.

Larranaga, P., Kuijpers, C.M.H., Murga, R.H., Inza, I., Dizdarevic, S., 1999. Genetic algorithms for the travelling salesman problem: A review of representations and operators. Artif. Intell. Rev. 13, 129–170.

Li, J., Sun, Q., Zhou, M., Dai, X., 2013. A new multiple traveling salesman problem and its genetic algorithm-based solution. In: 2013 IEEE International Conference on Systems, Man, and Cybernetics. IEEE, pp. 627–632.

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: 2009 4th IEEE Conference on Industrial Electronics and Applications. IEEE, pp. 1533–1537.

Liu, X.-X., Liu, D., Yang, Q., Liu, X.-F., Yu, W.-J., Zhang, J., 2021. Comparative analysis of five local search operators on visiting constrained multiple traveling salesmen problem. In: 2021 IEEE Symposium Series on Computational Intelligence (SSCI). IEEE, pp. 01–08.

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

Mahmoudinazlou, S., Kwon, C., 2023. A hybrid genetic algorithm with type-aware chromosomes for traveling salesman problems with drone. arXiv preprint arXiv: 2303.00614.

Necula, R., Breaban, M., Raschip, M., 2015. Tackling the bi-criteria facet of multiple traveling salesman problem with ant colony systems. In: 2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI). IEEE, pp. 873–880.

Park, Y.-B., 2001. A hybrid genetic algorithm for the vehicle scheduling problem with due times and time deadlines. Int. J. Prod. Econ. 73 (2), 175–188.

Park, J., Kwon, C., Park, J., 2023. Learn to solve the min–max multiple traveling salesmen problem with reinforcement learning. In: Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems. AAMAS '23. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, pp. 878–886.

Perron, L., Furnon, V., 2023. OR-Tools v9.6. URL https://developers.google.com/optimization/.

Prins, C., 2004. A simple and effective evolutionary algorithm for the vehicle routing problem. Comput. Oper. Res. 31 (12), 1985–2002.

Singh, A., Baghel, A.S., 2009. A new grouping genetic algorithm approach to the multiple traveling salesperson problem. Soft Comput. 13, 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.

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

Vidal, T., Crainic, T.G., Gendreau, M., Lahrichi, N., Rei, W., 2012. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. Oper. Res. 60 (3), 611–624.

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

Wang, Z., Fang, X., Li, H., Jin, H., 2020. An improved partheno-genetic algorithm with reproduction mechanism for the multiple traveling salesperson problem. IEEE Access 8, 102607–102615.

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

Zheng, J., Hong, Y., Xu, W., Li, W., Chen, Y., 2022. An effective iterated two-stage heuristic algorithm for the multiple traveling salesmen problem. Comput. Oper. Res. 143, 105772.