Learning-guided iterated local search for the minmax multiple traveling salesman problem

Pengfei He [a], Jin-Kao Hao [b,*] Jinhui Xia [a]

[a] *School of Automation and Control of Complex Systems of Engineering, Ministry of Education, Southeast University, Nanjing 210096, China*

[b] *LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France*

Computers & Operations Research 185: 107255, 2026

Abstract

The minmax multiple traveling salesman problem involves minimizing the costs of a longest tour among a set of tours. The problem is of great practical interest because it can be used to formulate several real-life applications. To solve this computationally challenging problem, we propose a learning-driven iterated local search approach that combines an effective local search procedure to find high-quality local optimal solutions and a multi-armed bandit algorithm to select removal and insertion operators to escape local optimal traps. Extensive experiments on 77 commonly used benchmark instances show that the algorithm achieves excellent results in terms of solution quality and running time. In particular, it achieves 32 new best results (improved upper bounds) and matches the best-known results for 35 other instances. Additional experiments shed light on the understanding of the algorithm's constituent elements. Multi-armed bandit selection can be used advantageously in other multi-operator local search algorithms.

Keywords: Traveling salesman; Minmax; Iterated local search; Multi-armed bandit; Learning-driven search.

1 Introduction

The minmax multiple traveling salesman problem (minmax mTSP) is formulated within a complete, undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_0\} \cup \mathcal{N}$ is the set of vertices, with the vertex v_0 serving as the depot and $\mathcal{N} = \{v_1, v_2, \cdots, v_n\}$ being the set of cities, while $\mathcal{E}$ represents the set of edges. Each edge $e_{ij} \in \mathcal{E}$ is associated with a cost c_{ij} (or distance) and the edge

* Corresponding author.
Email addresses: pengfeihe606@gmail.com (Pengfei He), jin-kao.hao@univ-angers.fr (Jin-Kao Hao), jinhuixia@seu.edu.cn (Jinhui Xia).

costs satisfy the triangle inequality such that $c_{ij} + c_{jk} \geq c_{ik}$ for all distinct $v_i, v_j, v_k \in \mathcal{V}$. Given a set of salesmen $\mathcal{M} = \{1, 2, \cdots, m\}$, the minmax mTSP aims to find m mutually exclusive Hamiltonian tours (also called routes) such that each tour that starts and ends at the depot v_0 must visit at least one city, and each city must be visited exactly once. The objective of the minmax mTSP is to minimize the costs of a longest tour among the m tours.

The minmax mTSP, proposed by França et al. [1], has a number of practical applications, where it is necessary to make a fair and equitable distribution of workloads.Common applications encompass optimizing two-dimensional laser cutting paths [2], elaborating multiple robot spot welding paths [3], and enhancing patrol and delivery services [4]. Despite its relevance, the minmax mTSP has received much less attention than its counterpart, the minsum mTSP, which focuses on minimizing the cumulative travel cost over m tours. It is worth noting that He and Hao [5] showed that transforming the minsum mTSP into the classical traveling salesman problem (TSP) [6] can effectively solve the problem using state-of-the-art TSP algorithms. However, the situation is quite different when it comes to the minmax mTSP. Due to its inherent strong $\mathcal{NP}$-hardness, solving the minmax mTSP remains a computational challenge.

The goal of this work is to improve the state of the art in solving the minmax mTSP by introducing a learning-guided heuristic approach to solving large instances more effectively. Thanks to this algorithm, we report 32 new upper bounds out of 77 minmax mTSP benchmark instances in the literature. The main contributions are summarized as follows.

- The proposed multi-armed bandit guided iterated local search (MILS) consists of several complementary search components. The local search procedure uses the best-improvement strategy to explore multiple neighborhoods to attain high-quality local optimal solutions. To ensure continuous diversification to visit different regions of the search space, MILS uses a probabilistic criterion to accept new local optimal solutions. To escape from the basin of attraction of deep local optima, MILS applies a learning-driven perturbation (based on removal and insertion operators) to decide on the most appropriate operators to perturb the given solution. In addition, MILS benefits from an effective TSP heuristic algorithm for single route optimization.
- We thoroughly evaluate the performance of the proposed algorithm on the 77 widely used benchmark instances including 41 small and medium instances (set $\mathbb{S}$) [7,8,5,9] and 36 large instances (set $\mathbb{L}$) [5,9]. The algorithm finds 32 new upper bounds on challenging instances. This demonstrates its competitiveness and effectiveness in solving the minmax mTSP. Additionally, we perform experiments to gain insight into the role of the algorithm's key components.

The rest of the paper is organized as follows. Section 2 provides a literature review. Section 3 presents the proposed algorithm. Section 4 shows an experimental comparison with state-of-the-art methods in the literature. Section 5 presents additional experiments to analyze the main components of the algorithm. Section 6 concludes with perspectives.

2 Literature review

This section reviews the most representative algorithms for the minmax mTSP, as well as learning assisted techniques for solving routing problems. For a comprehensive presentation of the existing studies, the reader is referred to dedicated surveys [10,11].

Among the most representative recent studies for the minmax mTSP, Karabulut et al. [12] presented an evolution strategy approach for the mTSP with minsum and minmax objectives where they used a self-adaptive Ruin and Recreate heuristic to generate new solutions, which are improved by a local search procedure. This algorithm showed competitive results. He and Hao [5] presented a hybrid search with neighborhood reduction (HSNR) where a dedicated strategy is used to streamline the neighborhood search by eliminating non-promising candidate solutions. Zheng et al. [8] introduced an iterated two-stage heuristic algorithm (ITSHA), characterized by a special random greedy initialization procedure and an adaptive variable neighborhood search. Mahmoudinazlou and Kwon [7] developed a hybrid genetic algorithm (HGA) with a novel crossover operator and a self-adaptive random local search component. The hybrid approach makes a significant contribution to the problem-solving landscape. He and Hao [9] proposed a memetic algorithm (MA) to solve the minmax mTSP with single and multiple depots. This algorithm incorporates a generalized edge assembly crossover, an efficient variable neighborhood descent, and a post-optimization phase. MA demonstrated superior performance compared to state-of-the-art algorithms.

These approaches were successful on a variety of benchmark instances. However, there is room for improvement in their results on large instances. For example, although the MA algorithm of [9] achieved excellent results on large instances, it takes longer to do so. Furthermore, existing algorithms use a local search procedure based on the *first-improvement* strategy instead of the best-improvement strategy, even though the latter performs well for several routing problems [13,14,15]. Therefore, it is motivating to know if a local search procedure with the best-improvement strategy can contribute to effectively solve the minmax mTSP.

Moreover, local search for routing problems usually uses multiple perturbation operators based on removal and insertion to escape local optima traps. When multiple perturbation operators are available, a selection strategy is

needed to decide which perturbation operator to apply next. An example is the adaptive layer of the popular adaptive large neighborhood search (ALNS) framework, which uses the roulette wheel method to select the next operator based on the continuously updated probabilities associated with the candidate operators [16]. More recently, machine learning techniques have also been used to select perturbation operators. For example, Kallestad et al. [17] used deep reinforcement learning within the ALNS framework for this purpose. Unlike the adaptive layer method, which only considers the past performance of the operators for future selection, the learning agent considers additional information from the search process, such as the difference in objective values between iterations, to make better decisions. Unlike most ALNS algorithms, which require to select a single destroy and a single repair operator per iteration, hyper-heuristic (HH) algorithms must determine a sequence of low-level heuristics used in each iteration. To address this challenge, Lagos and Pereira [18] proposed a multi-armed bandit (MAB) framework to guide the selection process. The approach integrates Thompson sampling and exponential weights for exploration and exploitation, enabling online parameter learning. Empirical results demonstrate that the MAB-based strategy significantly enhances solution quality for the studied routing problem.

In this work, we present an effective iterated local search with a multi-armed bandit technique for better solving the minmax mTSP. The algorithm uses the multi-armed bandit method to select perturbation operators based on information collected during the search process. This is the first application of MAB to the minmax mTSP. Additionally, MILS employs an effective local search procedure based on the best-improvement strategy to explore a set of neighborhoods. Experimental results on benchmark instances allow us to demonstrate the significance and relevance of the learning-driven operator selection technique and the best-improvement local search strategy.

3 Multi-armed bandit-driven iterated local search

The proposed MILS algorithm for the minmax mTSP follows the general approach of iterated local search [19], and iterates three main phases: local optima exploration (Section 3.2), probabilistic solution acceptance (Section 3.3), and local optima escaping (Section 3.4).

As illustrated in Algorithm 1, the algorithm starts with an initial solution built with a greedy randomized heuristic (Section 3.1). The solution is then improved to attain a local optimal solution during the local optima exploration phase, which uses an efficient best-improvement local search procedure with multiple neighborhoods (Section 3.2.1). The local optima exploration phase also includes a single tour improvement procedure, which is triggered under certain conditions and is based on a state-of-the-art TSP heuristic (Section 3.2.2). The improved local optimal solution is then submitted to the proba-

bilistic solution acceptance criterion to decide whether to accept it as the new current solution. This is followed by the local optima escaping phase to get rid of the current local optimum trap and lead the search to a new search region. For this purpose, the local optima exploration phase perturbs the current solution by a learning-driven removal-and-insertion procedure with MAB to generate a new solution for the next round of the search. Finally, if the search is considered to have sufficiently explored the current search region (indicated by the condition $Iter > I_{max}$ where $Iter$ is the iteration number of the algorithm and I_{max} is a parameter), the algorithm starts its next search round by creating a fresh initial solution using the greedy randomized heuristic. The algorithm stops and returns the global best solution φ^* when the predetermined stopping condition is met, such as reaching a cutoff time limit or a specified number of iterations.

3.1 Initial solution

MILS uses the following greedy randomized heuristic to build an initial solution which consists of m tours.

(1) Initiate m tours, where each tour starts at the depot v_0 and includes a randomly chosen city;
(2) Identify the shortest tour r among the m tours under construction; let v be the last city of tour r. If there is more than one shortest tour, select one at random;
(3) Identify the city u among the α nearest cities of v (see Section 3.2.1) such that city u causes the least increase in length;
(4) Insert the city u into the tour r after v;
(5) Repeat Steps (2) to (4) until all cities are assigned to routes.

With this greedy randomized initialization procedure, we can obtain an initial solution. The time complexity of the algorithm is bounded by $\mathcal{O}(n \times \alpha)$.

3.2 Local optima exploration

The local optima exploration phase is a key search component within the MILS algorithm. Specifically, it includes a local search with multiple neighborhoods, which are exploited using the best-improvement strategy to explore local optimal solutions, and a single tour improvement procedure using the state-of-the-art TSP heuristic EAX-TSP [20] to improve each individual tour. In particular, due to the time-consuming nature of EAX-TSP, the single tour improvement procedure is applied only to an elite solution φ when it updates the global best solution φ^* (i.e., $f(\varphi) < f(\varphi^*)$).

Algorithm 1: Main framework of the MILS algorithm

Input: Instance I, parameter I_{max};
Output: The best solution φ^* found;

begin
  $\varphi = GreedyRandom(I)$; /* φ represents the current solution, Section 3.1 */
  $\varphi' \leftarrow \varphi$; /* φ' is the local optimal solution of the current round of the search */
  $\varphi^* \leftarrow \varphi$; /* φ^* is the global best solution found so far */
  $Iter \leftarrow 0$; /* Iteration counter */
  while *Stopping condition is not met* **do**
    /* Local optima exploration */
    $\varphi \leftarrow LocalSearch(\varphi)$; /* Improve the current solution, Section 3.2.1 */
    if $f(\varphi) < f(\varphi^*)$ **then**
      $\varphi \leftarrow SingleTourImprove(\varphi)$; /* Further improve each elite solution, Section 3.2.2 */
    end
    /* Probabilistic solution acceptance */
    $<\varphi', \varphi^*> \leftarrow AcceptStrategy(\varphi, \varphi', \varphi^*)$; /* Section 3.3 */
    /* Local optima escaping */
    $D \leftarrow MAB(\mathcal{D})$;$R \leftarrow MAB(\mathcal{R})$; /* Select removal and insertion operators with multi-armed bandit, Section 3.4.2 */
    $\varphi \leftarrow Perturb(\varphi', D, R)$; /* Perturb the current solution, Section 3.4.1 */
    /* Restarting */
    if $Iter > I_{max}$ **then**
      $\varphi = GreedyRandom(I)$; /* Create a new initial solution */
      $\varphi' \leftarrow \varphi$; $Iter \leftarrow 0$;
    else
      $Iter \leftarrow Iter + 1$;
    end
  end
  return φ^*;
end

3.2.1 Local search

The local search procedure uses the best-improvement strategy to exploit multiple neighborhoods generated by ten neighborhood operators (also called move operators), which are explored by variable neighborhood descent [21]. Note that these move operators have been widely used in various routing problems.

Let vertex $v \in \mathcal{V}$ be an α-nearest neighbor of city $u \in \mathcal{N}$, where α ($\alpha < |\mathcal{N}|$) is the granularity threshold that restricts the search to nearby vertices. Let

$r(u)$ and $r(v)$ denote two tours which visit vertices u and v, respectively. Additionally, let x and y represent the successors of u in $r(u)$ and v in $r(v)$, respectively. Then the ten neighborhood operators are defined as follows.

- M1: city u is removed from $r(u)$ and inserted into $r(v)$ after vertex v.
- M2: Two consecutive cities u and x are removed from $r(u)$ and inserted into $r(v)$ after vertex v.
- M3: Two consecutive cities u, x are removed from $r(u)$ and placed before vertex v in the reverse ordre x, u.
- M4: Interchange the position of city x and city v.
- M5: Interchange two consecutive cities u, x and city v.
- M6: Interchange two consecutive cities u, x and two consecutive cities v, y.
- M7: Two consecutive cities u, x are selected. The reversed cities x, u are swapped with two consecutive cities v, y.
- M8: This is the 2-opt operator, which replaces edges (u, x) and (v, y) by (u, v) and (x, y) if $r(u) = r(v)$.
- M9: This is the 2-opt* operator where $r(u) \neq r(v)$. As shown in Fig. 1 (left), two edges (u, x) and (v, y) are replaced by two new edges (u, y) and (v, x).
- M10: This is the other 2-opt* operator where $r(u) \neq r(v)$. As shown in Fig. 1 (right), two edges (u, x) and (v, y) are removed and two new edges (u, v) and (x, y) are added to form two new routes.

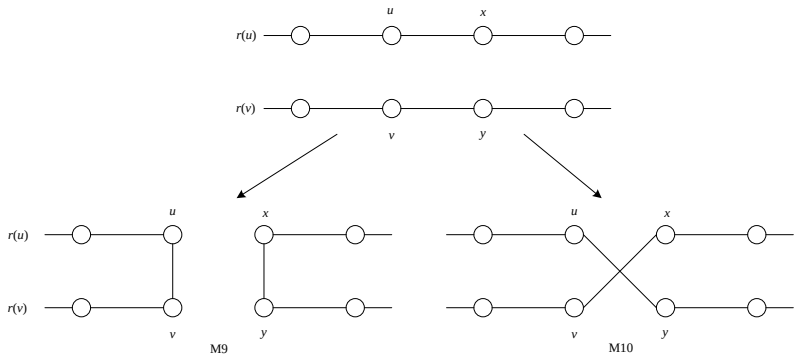


Fig. 1. Illustration of 2-opt*.

Due to the granularity threshold rule, the search is restricted to the nearest α vertices. Thus, the computational complexity of each of the ten operators is bounded by $\mathcal{O}(n \times \alpha)$. Unlike the two state-of-the-art algorithms, HSNR [5] and MA [9], the cross-exchange operator is excluded from MILS duo to its relatively high computational cost compared to the ten move operators we used. Furthermore, inspired by the concept of *don't look bits* [22], we have incorporated this acceleration technique to speed up our neighborhood searches by filtering out unpromising neighborhood solutions.

The local search procedure explores local optimal solutions with the best-improvement strategy by considering these neighborhoods in the listed order. In Section 5.1, we also examine the first-improvement strategy, which is an-

other popular neighborhood exploitation strategy largely used in the context of minimizing the total travel cost (the minsum objective) for various routing problems [23,13,14,15]. Indeed, although no significant differences have been observed between these two strategies for the minsum objective, little is known regarding the behavior of these two strategies for minmax problems like the minmax mTSP. In Section 5.1, we partially fill this gap by reporting experimental evidence to show that the best-improvement strategy dominates the first-improvement strategy for the minmax objective.

3.2.2 Single tour improvement

In addition to the aforementioned local search procedure, the local optima exploration also includes a single tour improvement procedure to further improve each new elite solution. For this, the single tour improvement procedure uses the edge assembly crossover designed for the TSP (EAX-TSP) [1] to minimize each individual tour of the solution. Specifically, for each tour, its cities are submitted to EAX-TSP, which then optimizes the order of these cities and returns a new tour. After EAX-TSP is finished, its final solution is given to the local search procedure, which can possibly lead to further improvement by exchanging cities between tours.

Finally, since EAX-TSP is time consuming, we apply it only to high-quality elite solutions, which are typically reached when the algorithm has searched long enough. Specifically, the single tour improvement procedure is only triggered when the underlying solution (φ) updates the global best solution φ^* ($f(\varphi) < f(\varphi^*)$) and the number of iterations reaches a threshold $I_{threshold}$ (empirically set to $I_{threshold} = 1000$).

3.3 Probabilistic acceptance and stopping criterion

Let φ be the solution from the local optima exploration phase, and let φ' be the current local optimal solution. We use a probabilistic acceptance criterion to decide whether φ is accepted to update φ'. The decision to incorporate φ into the subsequent trajectory of the search process follows the acceptance criterion of simulated annealing [23,24]. As shown in Algorithm 2, within each iteration of MILS, if the solution φ is better than the global best solution φ^* ($f(\varphi) < f(\varphi^*)$), φ is accepted. Consequently, both φ^* and the current local optimal solution φ' undergo corresponding updates. However, if $f(\varphi^*) < f(\varphi) < f(\varphi')$, only the current local optimal solution φ' receives an update. In the event that $f(\varphi) > f(\varphi')$, the acceptance of the solution φ depends on a probability determined by $e^{-\Delta f/T}$, where $\Delta f = f(\varphi) - f(\varphi')$ and T is the temperature parameter, which is tuned according to Eqs. (1) and (2). The initial temperature T_0 is determined by Eq. (3), where φ_{init} is the given initial

[1] The code of EAX-TSP is available at: https://github.com/sugia/GA-for-TSP

solution, w is a parameter set to 0.35 following [16] and p_{accept} is a probability parameter to accept candidate solutions. The final temperature T_f over I_{max} iterations is set to 0.0001 empirically.

Algorithm 2: Acceptance criterion

Input: Solution φ from the local optima exploration procedure, current local optimal solution φ', global best solution φ^*;
Output: $<\varphi', \varphi^*>$;

```
begin
    Δf ← f(φ) − f(φ′);
    if f(φ) < f(φ*) then
        φ* ← φ; φ′ ← φ;
    else if f(φ) < f(φ′) then
        φ′ ← φ;
    else if e^{−Δf/T} > random(0,1) then
        φ′ ← φ; /* Accept φ as new current local optimal solution,
         otherwise refuse φ.                                        */
    return <φ′,φ*>;
end
```

$$T_{Iter+1} = cT_{Iter} \tag{1}$$

$$c = (\frac{T_f}{T_0})^{I_{max}} \tag{2}$$

$$e^{-(w \cdot f(\varphi_{init}))/T_0} = p_{accept} \tag{3}$$

3.4 Local optima escaping

When the local optima exploration procedure stagnates, the search is trapped in a deep local optimum. To escape the current basin of attraction and explore alternative neighboring search regions, the local optima escaping procedure is triggered to perturb the current solution φ'. To this end, we use a set of removal operators ($\mathcal{D}$) and a set of insertion operators ($\mathcal{R}$) to modify the solution φ' by selectively removing and then reinserting some cities. Specifically, a pair of removal and insertion operators is chosen from the sets $\mathcal{D}$ and $\mathcal{R}$ during each iteration. The selection can be performed randomly or using the roulette-wheel strategy as in [16]. Here, we adopt a multi-armed bandit algorithm to intelligently identify the appropriate removal and insertion operators.

3.4.1 Removal and insertion operators

We use five removal operators: *Shaw removal*, *Random removal*, *Cross removal*, *Worst removal* and *Information removal*. Each removal operator removes $\lfloor l \times$

$n\rfloor$ cities and keeps them in a set $\mathcal{S}$, where l $(0 < l < 1)$ is a parameter called the length of perturbation.

(1) *Shaw removal.* The operator removes $\lfloor l \times n \rfloor$ cities based on a given criterion, such as geographic location [25]. Specifically, a random city v_i is selected and $\mathcal{S} = \{v_i\}$. Then the distance between v_i and each remaining city in the set $\mathcal{N} \setminus \mathcal{S}$ is computed. The distances are then used to sort the remaining cities in ascending order in the list $\mathcal{L}$. To introduce randomness into the selection of cities to remove, a parameter denoted as γ (where $\gamma \geq 1$) is used. Then, given a random number y from $random(0, 1)$, $\lfloor y^{\gamma} * |\mathcal{L}| \rfloor$th city (u_i) in the list $\mathcal{L}$ is selected and removed from the solution and added in the set $\mathcal{S}$ $(\mathcal{S} \leftarrow \mathcal{S} \cup \{u_i\})$. The newly selected city u_i is used to select the next city according to the above procedure. The operation continues until $\lfloor l \times n \rfloor$ cities are successfully identified and selected for removal.

(2) *Random removal.* The operator randomly selects $\lfloor l \times n \rfloor$ cities, removes them from the solution $\varphi^{'}$, and keeps them in set $\mathcal{S}$.

(3) *Cross removal.* The operator focuses on removing cities that are close to each other and visited by different tours. In fact, cities with close coordinates, assigned to different tours, have an increased probability of being reassigned. The reassignment serves as a means of perturbing the solution, making it easier for the algorithm to escape local optima. Specifically, for each city v_i, the operator incrementally counts the number of neighboring cities in its vicinity, denoted by α, that are visited by alternative tours. Consequently, if a majority of a city's neighbors are actually served by different tours, it becomes more likely that this city will be moved to increase the chances of escaping local optima trap. Thus, cities are stored and sorted in $\mathcal{L}$ according to the number of neighbors visited by alternative tours. Similar to *Shaw removal*, we also use a parameter γ to introduce randomness into the selection of cities.

(4) *Worst removal.* The objective of the operator is to identify and remove cities that contribute to a significant increase in the total travel cost along the tour. The operator first calculates for each city the potential reduction in travel costs that would result from removing it from the solution. These computed values are then used to sort the cities in descending order $\mathcal{L}$, based on the magnitude of these reductions. Similar to *Shaw removal*, the selection of cities to remove involves an element of randomization, moderated by a deterministic parameter γ. The parameter controls the degree of randomization in the selection of the 'worst' city to remove from the solution.

(5) *Information Removal.* It is a well-established observation that high-quality solutions to routing problems often exhibit structural proximity to the optimal solution, sharing a significant number of common edges with them [14]. In particular, these solutions tend to exhibit recurring sequences of consecutive visits, commonly referred to as *patterns*, which are typ-

ically not actively explored during the standard local search procedure [26]. Consequently, if some cities are frequently involved in neighborhood operators, this tendency may be advantageous in allowing MILS to escape local optima. To take advantage of this insight, a statistical analysis is performed during the local search procedure to track the frequency with which each city is involved in the neighborhood operators. Then, all cities are ordered in $\mathcal{L}$ in descending order based on their involvement frequency. Like *Worst removal*, the selection of cities for removal also uses the parameter γ to control the randomness of the selection. The procedure terminates when a specified number of $\lfloor l \times n \rfloor$ cities have been successfully removed.

After the application of the removal operators, the removed cities (retained in the set $\mathcal{S}$) are reinserted into the solution by using three insertion operators: *Greedy insert*, *Greedy insert with blink*, and *Regret insert*.

(1) *Greedy insertion.* For each city $v_i \in \mathcal{S}$, a city $u_j \in \mathcal{N} \backslash \mathcal{S}$, which comes from its α-nearest neighborhood and minimizes the travel cost of the tour after insertion, is chosen. Then, city v_i is inserted after city u_j. The process stops when all cities in $\mathcal{S}$ are successfully inserted into the solution.
(2) *Greedy insertion with blink.* The operator, originally proposed in [27] for the vehicle routing problem, introduces a controlled element of randomness into the insertion process. For each city $v_i \in \mathcal{S}$, a city $u_j \in \mathcal{N} \backslash \mathcal{S}$ from its α-nearest neighborhood is selected for insertion with a probability of $1 - \beta$ (β is a parameter, set to $\beta = 0.01$ following [27]). If the probability is not met, the insertion position is skipped. The process continues until all cities in the set $\mathcal{S}$ have been successfully inserted into the solution.
(3) *Regret insertion.* With this operator, the selection process at each iteration focuses on identifying the removed cities that would cause the most regret if they were not inserted at their optimal position during the current iteration. For each city $v_i \in \mathcal{S}$, we first compute the cost, denoted as Δf_i^1, associated with inserting v_i at its best possible position within the incumbent solution. Additionally, we compute Δf_i^q for $q \in \{2, \cdots, k\}$, representing the cost of inserting v_i at its q^{th} best position. Next, for each city $v_i \in \mathcal{S}$, we compute a regret value $reg_i = \sum_{q=2}^{k} (\Delta f_i^q - \Delta f_i^1)$. Finally, we select a position with the highest regret value, and insert the city v_i at that location. Following [16], the parameter k is set to 3.

3.4.2 Multi-armed bandit

Building on the concept in [18], where a multi-armed bandit algorithm was used to determine the sequence of lower-level heuristics within the hyper-heuristic framework, we explore applying MAB to select removal and insertion operators for exploring neighborhood solutions. Unlike sequencing heuristic operators, operator selection is a more tractable problem.

For the removal operators, each operator $D_i \in \mathcal{D}$ is associated with three critical attributes: a weight ω_{D_i}, a score π_{D_i}, and a counter θ_{D_i} that records the number of times the operator D_i has been applied. Initially, all operators are given the same weight, with the constraint that the sum of these weights over all operators in $\mathcal{D}$ equals 1. We divide the algorithm execution into segments of fixed duration (100 iterations per segment in our case), with constant operator weights maintained throughout each segment. Before starting a new segment, the scores (π_{D_i}) of all operators are reset to zero. During each iteration within a segment, the score of the selected operator D_i is updated by $\pi_{D_i} \leftarrow \pi_{D_i} + \delta_u$, where δ_u ($u \in \{1, 2, 3\}$) contains three different cases that govern the score change π_{D_i}. Specifically, the score π_{D_i} receives an increase of δ_1 when the global best solution φ^* undergoes an update ($f(\varphi) < f(\varphi^*)$). Similarly, an increase of δ_2 is received when the local optimum solution φ' is improved ($f(\varphi) < f(\varphi')$), and a score increment of δ_3 is applied when solution φ is accepted according to the probabilistic solution acceptance criterion, even if it does not represent an overall improvement. In our case, we used $\delta_u = 3, 5, 10$, for $u = 1, 2, 3$. At the end of each segment, we compute new weights for the operators based on the accumulated scores according to Eqs. (4) and (5), where $\omega_{D_i,j}$ is the weight of operator D_i in last segment j and λ is the reaction factor ($\lambda = 0.1$), which controls how quickly the weight adjustment reacts to changes.

$$\pi_{D_i} = \frac{\pi_{D_i}}{\theta_{D_i}} \tag{4}$$

$$\omega_{D_i,j+1} = (1 - \lambda)\omega_{D_i,j} + \lambda\pi_{D_i} \tag{5}$$

For each iteration of MILS, one removal and one insertion operator must be selected from $\mathcal{D}$ and $\mathcal{R}$, respectively. Unlike the approach in [18], which uses Thompson sampling and exponential weights for exploration and exploitation algorithms in the MAB framework to determine the sequence of low-level heuristics, we adopt the ϵ-greedy algorithm. The choice of ϵ-greedy is motivated by the simplified decision-making scenario, where only two operators need to be selected per generation without requiring sorting heuristic operators. The ϵ-greedy algorithm operates as follows: Given a random number $r \sim$ Uniform(0, 1) and a fixed exploration rate ϵ, the algorithm selects the operator associated with the highest weight if $r < \epsilon$ (exploitation); otherwise, it randomly selects an operator (exploration).

4 Experimental Evaluation and Comparisons

The purpose of this section is to evaluate the performance of MILS through experiments and compare it with state-of-the-art algorithms.

Table 1
Parameter tuning results.

Parameter	Section	Description	Considered values	Final values
I_{max}	3.3	maximum iterations per restart	$\{10000, 20000, 30000, 40000, 50000\}$	40000
p_{accept}	3.3	accept probability	$\{0.5, 0.6, 0.7, 0.8, 0.9\}$	0.7
α	3.2.1	granularity threshold	$\{5, 10, 15, 20, 25, 30\}$	10
l	3.4	perturbation length	$\{0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3\}$	0.15
ϵ	3.4	ϵ-greedy parameter	$\{0.005, 0.01, 0.015, 0.02, 0.025\}$	0.01

4.1 Benchmark instances

We conducted experiments using two benchmark sets that are commonly tested in the literature. The instances (77 in total) and the best solutions obtained by MILS are available online [2].

- Set $\mathbb{S}$: The set contains 41 small and medium instances with 51 to 1173 cities and 3 to 30 tours. These instances have been widely used to evaluate minmax mTSP algorithms [5,9,28,29,8,7]. Of these 41 instances, the optimal solution is known for 17 of them, all of which have a longest tour composed of only one city. Given that the instances of this set have been studied intensively, it is challenging to improve the best-known results for the instances with unknown optima.
- Set $\mathbb{L}$: The set has 36 large instances with 1357 to 5915 cities and 3 to 20 tours, which was introduced in [5] and further tested in [9]. Among these 36 instances, optimal solutions are known for 5 instances due to the fact that their longest tour is composed of only one city. The $\mathbb{L}$ instances are known to be more challenging than those of the set $\mathbb{S}$.

4.2 Experimental protocol and reference algorithms

Parameter setting. The MILS algorithm has five parameters: the maximum number of iterations per restart I_{max}, the accept probability p_{accept}, the nearest neighbor granularity threshold α, the perturbation length l, the ϵ-greedy parameter ϵ. In order to calibrate these parameters, the automatic parameter tuning package Irace [30] was used. The tuning was performed on 8 instances with 150-2392 cities among the 77 benchmark instances. The tuning budget was set to be 2000 runs. The candidate and final values for the parameters are shown in Table 1. The values recommended by Irace were used throughout our experiments and can be considered as the default parameter setting of the MILS algorithm.

Reference algorithms. For our comparative experiments, we used the following four recent state-of-the-art algorithms, which hold most of the current best-known solutions for the minmax mTSP benchmark instances.

[2] https://github.com/pengfeihe-angers/mils.git

- BKS. This indicates the best-known solutions (current best upper bounds) that are compiled from the best results reported by the state-of-the-art algorithms [5,8,7,9,12] and the best results we obtained by running the available codes of the algorithms in [5,8,9].
- HSNR (2022) [5]: This hybrid search algorithm with neighborhood reduction was coded in C++ and executed on a computer with a Xeon E5-2670 CPU at 2.5 GHz and 6 GB RAM.
- ITSHA (2022) [8]: This iterated two-stage heuristic algorithm was implemented in C++, and we ran the source code on our computer (Xeon E5-2670 CPU at 2.5 GHz and 6 GB RAM).
- HGA (2023) [7]: This hybrid genetic algorithm was coded in Julia and executed on a computer with an Apple M1 CPU and 16 GB RAM. According to Passmark [3], the M1 CPU is significantly faster than the Xeon E5-2670 processor used in this study, in terms of the single thread performance. Therefore, given the same running time, the HGA algorithm has a more favorable computational budget.
- MA (2023) [9]: This memetic algorithm was implemented in C++ and executed on a computer with a 2.5 GHz Xeon E5-2670 processor and 8 GB of RAM.

Experimental setting and stopping criterion. We implemented the MILS algorithm in C++ and compiled the program using the g++ compiler with the -O3 option [4]. All experiments were conducted on a 2.5 GHz Intel Xeon E-2670 processor with 6 GB RAM running Linux with a single thread. Given the stochastic nature of the algorithm, MILS was run 20 times on each instance with different random seeds. The algorithm terminates when it reaches the cutoff limit of $(n/100) \times 4$ minutes (this is the same stopping condition used in the reference algorithms [9,5,8]). The reference algorithms are run under the same cutoff limits as MILS, except HGA, whose code is unavailable. For HGA, we report its results given in [7].

4.3 Computational results and comparisons

We provide a brief summary of the results comparing the MILS algorithm with the reference algorithms in Table 2 and the detailed results in the appendix. In Table 2, we show the number of instances where our MILS algorithm obtains a better (#Wins), equal (#Ties), or worse (#Losses) result compared to each reference, including the BKS. To verify whether there is a statistically significant difference between MILS and each reference algorithm, the Wilcoxon signed-rank test is used at a 0.05 confidence level, where a *p-value* less than 0.05 indicates a statistically significant difference.

[3] https://www.cpubenchmark.net/singleThread.html#

[4] The code of MILS will be available at the link of footnote: https://github.com/pengfeihe-angers/mils.git

Table 2
Summary of comparative results between MILS and the reference algorithms on the 77 benchmark instances.

Pair algorithms	Groups	Best				Avg.			
		#Wins	#Ties	#Losses	*p-value*	#Wins	#Ties	#Losses	*p-value*
MILS vs. BKS	m=3	10	7	2	2.10E-02	-	-	-	-
	m=5	11	5	3	5.25E-03	-	-	-	-
	m=10	7	7	5	1.10E-01	-	-	-	-
	m=20	4	14	0	1.25E-01	-	-	-	-
	m=30	0	2	0	0.00E+00	-	-	-	-
MILS vs. HSNR	m=3	16	3	0	4.38E-04	19	0	0	1.32E-04
	m=5	16	1	2	6.29E-04	17	1	1	7.38E-04
	m=10	10	7	2	9.28E-03	10	6	4	3.40E-02
	m=20	4	14	0	1.25E-01	4	14	0	1.25E-01
	m=30	0	2	0	0.00E+00	0	2	0	0.00E+00
MILS vs. ITSHA	m=3	16	3	0	4.38E-04	17	2	0	2.93E-04
	m=5	17	2	0	2.93E-04	16	1	2	1.18E-03
	m=10	11	7	1	3.42E-03	12	3	5	1.48E-02
	m=20	7	11	0	1.56E-02	13	5	0	2.44E-04
	m=30	0	2	0	0.00E+00	0	2	0	0.00E+00
MILS vs. HGA	m=3	6	4	0	3.13E-02	6	2	2	1.09E-01
	m=5	7	3	0	1.56E-02	6	2	2	7.81E-02
	m=10	1	6	3	6.25E-01	1	3	6	2.19E-01
	m=20	0	9	0	0.00E+00	2	7	0	5.00E-01
	m=30	0	2	0	0.00E+00	0	2	0	0.00E+00
MILS vs. MA	m=3	10	7	2	2.10E-02	11	3	5	4.94E-02
	m=5	11	5	3	5.25E-03	10	2	7	1.02E-01
	m=10	7	7	5	6.40E-02	9	4	7	8.79E-02
	m=20	7	11	0	1.56E-02	11	7	0	9.77E-04
	m=30	0	2	0	0.00E+00	0	2	0	0.00E+00

First, regarding the BKS, Table 2 shows that MILS updates 32 current best-known results (new upper bounds) out of the 77 instances (41.6%) and matches 35 other BKS values (45.5%). Of these 33 new best results, 7 are for instances in $\mathbb{S}$ and 25 for instances in $\mathbb{L}$. These results are notable because the instances in the set $\mathbb{S}$ have been extensively studied in the literature, and the instances in the set $\mathbb{L}$ are known to be difficult for existing algorithms.

Now, if we examine the results in terms of the number of tours, we can make the following comments. For $m \in \{3, 5\}$, MILS performs remarkably well compared to the BKS values, and the *p-values* (<0.05) clearly indicate that the differences are statistically significant. For $m \in \{10, 20\}$, MILS competes well with the BKS results too, as it still yields several new bounds. However, the *p-values* (> 0.05) indicate that the differences are marginal. As shown in Tables A.1 and A.2 of the appendix, for small instances ($n < 532$), MILS consistently gives excellent results when $m \in \{3, 5\}$, without losing a single case. For medium and large instances, our algorithm also improves many BKS results (improvement gap up to 9.03%), indicating that there is still room for improvement for these large instances.

Compared to the reference algorithms, MILS significantly outperforms HSNR [5] and ITSHA [8] (*p-values* < 0.05) on the instances with $m \in \{3, 5, 10\}$. Furthermore, MILS achieves a remarkable performance compared to the two

Table 3
Summary information of MILS on different groups.

Index	m=3	m=5	m=10	m=20
Average gap (%) to BKS	-0.26	-0.87	-1.39	-1.28
Number of improved instances	10	11	7	4

Table 4
Summary of comparative results between MILS and $MILS_0$.

Pair algorithms	Best				Avg.			
	#Wins	#Ties	#Losses	*p-value*	#Wins	#Ties	#Losses	*p-value*
MILS vs $MILS_0$	49	26	2	1.08E-08	60	14	3	2.18E-09

latest algorithms HGA [7] and MA [9], losing only in a few of cases. To further verify the performance of MILS on instances with different tours, Table 3 shows the average gap of MILS to the BKS results. We can observe that MILS can improve more instances with $m \in \{3, 5\}$, but the improvement is marginal. On the contrary, although MILS can improve fewer instances with $m \in \{10, 20\}$, the improvements are larger. In conclusion, we can say that MILS is particularly competitive when solving instances with relatively few tours. Note that HSNR [5] and MA [9] can achieve remarkable results on instances with $m \in \{10\}$, making MILS a suitable algorithm for challenging instances with $m \in \{3, 5, 20\}$.

5 Additional experiments

This section presents experiments to gain additional insights into the influences of the components of the MILS algorithm: the local search procedure, the MAB algorithm. All experiments are based on the two sets of 77 instances. In addition, we also investigate the long-term convergence behavior of the algorithm under a relaxed stopping condition. To ensure a fair comparison, all variants of MILS were run on the same machine with the same parameters as MILS, and each variant was run 20 times per instance.

5.1 Rational behind the local search procedure

The local search procedure of MILS uses the best-improvement strategy to explore each of the used neighborhoods (see Section 3.2.1). One question is how MILS performs when the best-improvement strategy is replaced by the first-improvement strategy, which is very popular for routing problems with the minsum objective [13,14]. To perform a rigorous evaluation of these two strategies, we compared a variant of MILS (called $MILS_0$) where the first-improvement strategy is used in the local search procedure. The results of the comparison are summarized in Fig. 2 and Table 4.

According to Table 4, the best-improvement strategy dominates the first-improvement strategy. Replacing the best-improvement strategy with the first-improvement strategy significantly worsens the performance of the local search

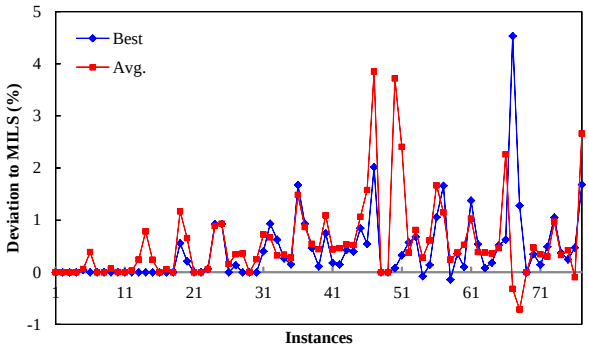


Fig. 2. Comparative results of MILS with variant MILS_0 on the 77 instances.

Table 5
Summary of average results between two improvement strategies.

Instances	Deviation from initial solutions (%)	
	Best	First
att532-3	18.228	16.406
pcb1173-3	15.565	14.467
pr2392-5	15.416	14.331
fl3795-3	8.575	8.480

procedure in terms of both the best and average results. Moreover, as shown in Fig. 2, the differences become even more significant as the size of the instances increases. These results demonstrate the critical contributions of the best-improvement strategy.

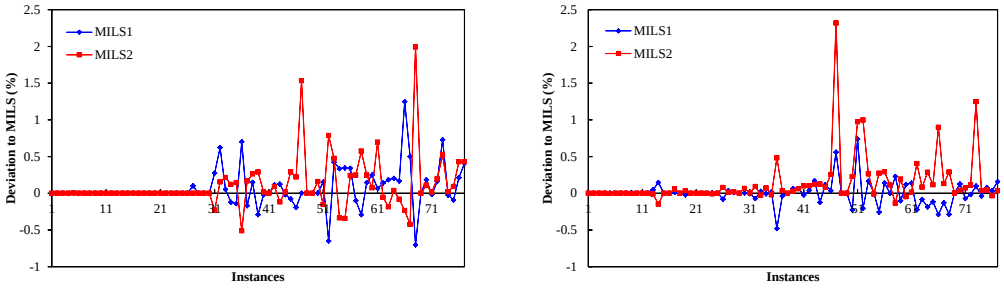
To gain a deeper understanding of why the best-improvement strategy outperforms the first-improvement strategy in solving the minmax mTSP, we study both strategies and determine the average deviation from the initial solutions when reaching the local optimal solutions, starting from the same initial solution. For this study, we used four instances (att532-3, pcb1173-3, pr2392-5, and fl3795-3) with different number of vertices, routes, and edge weight types. We ran the local search procedure with both strategies for each instance exactly 10,000 times using a different initial solution for each run. Then, the 1,000 best local optimal solutions of each algorithm are retained for analysis to avoid the influence of random features of local search procedures. As shown in Table 5, for these instances, the average deviation from initial solutions with the best-improvement strategy is larger than with the first-improvement strategy, which means that the best-improvement strategy allows the algorithm to go further in its search to attain high-quality solutions that cannot be reached with the first-improvement strategy. These results underscore the effectiveness of the best-improvement strategy in achieving superior results, making it the first choice for solving min-max objective routing problems.

5.2 *Rationale behind the MAB algorithm*

To evaluate the benefit of the MAB algorithm for escaping local optima, two MILS variants (denoted by MILS_1 and MILS_2) were created where the MAB algorithm is replaced by the roulette-wheel strategy and the random strategy, like in [16], respectively. Table 6 summarizes the comparative results of the three algorithms, and Fig. 3 plots the best/average gap between these variants and MILS on all instances. The X-axis is the instance label, while the Y-axis is the deviation from the MILS results (%).

Table 6
Summary of comparative results of MILS with two variants.

Pair algorithms	Best				Avg.			
	#Wins	#Ties	#Losses	*p-value*	#Wins	#Ties	#Losses	*p-value*
MILS vs MILS_1	29	32	16	8.64E-03	32	19	26	6.15E-01
MILS vs MILS_2	33	33	11	5.48E-03	47	19	11	1.38E-06



(a) Best (b) Average

Fig. 3. Comparative results of MILS with two variants MILS_1 (with roulette-wheel strategy) and MILS_2 (with random strategy) on the 77 instances.

Table 6 clearly shows that MILS outperforms MILS_1 in terms of best and average results, which is confirmed by *p-values* $\ll 0.05$. As shown in Fig. 3, there are no significant differences between MILS and MILS_1 when solving small and medium instances ($n < 532$). On the contrary, MILS_1 becomes worse when solving large instances, in terms of best and average results. As to the MILS_2 variant, Table 6 shows that it performs the worst. Thus, we can conclude that the MAB algorithm is a better strategy for operator selection than the roulette-wheel strategy, and the roulette strategy is better than the random strategy.

5.3 *Convergence analysis of the MILS algorithm*

In Section 4.3, the stopping condition of MILS was set to the maximum running time of $(n/100) \times 4$ minutes in accordance with the reference algorithms. To investigate the convergence behavior of the MILS algorithm for a long run (denoted by MILS_L), the stopping condition is set to a relaxed running time of $(n/100) \times 8$ minutes.

Table 7
Summary of comparative results between MILS and MILS$_L$.

Pair algorithms	Best				Avg.			
	#Wins	#Ties	#Losses	*p-value*	#Wins	#Ties	#Losses	*p-value*
MILS$_L$ vs MILS	23	54	0	2.95E-04	56	21	0	7.55E-11

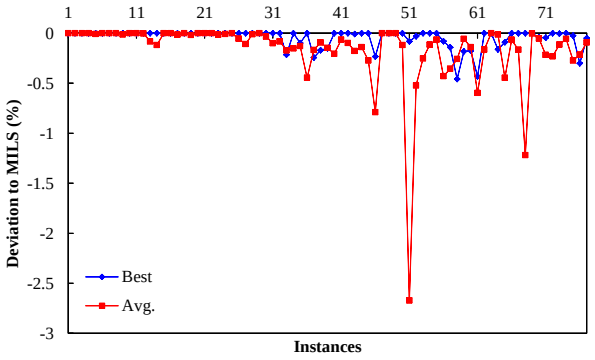


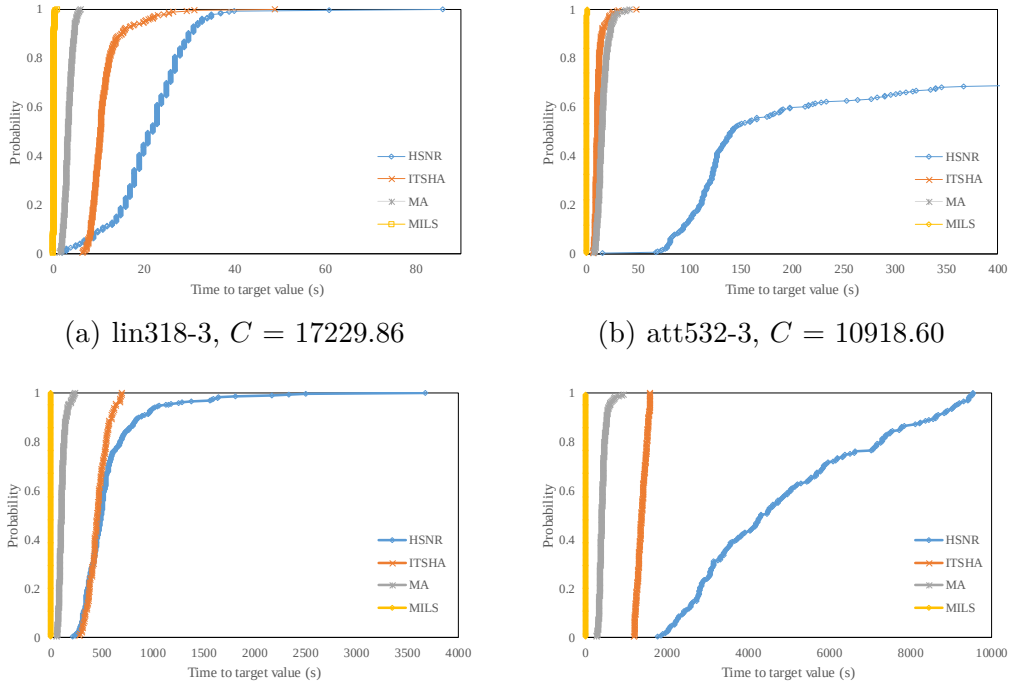
Fig. 4. Comparative results of MILS with variant MILS$_L$ on the 77 instances.

From Table 7, one observes that with a large runtime budget, MILS is able to further significantly improve its results obtained with the default cutoff time ($n/100\times$ 4 minutes) both in terms of best and average results (*p-values* $\ll 0.05$). Specifically, the best results can be even improved as much as 0.46%, while the average results can be improved by 2.67%. As shown in Fig. 4, the improvements only concern medium and large instances ($n \geq 532$), which probably indicates that the results of MILS with the default cutoff time ($n/100\times$ 4 minutes) for the small instances, if not optimal, could be very close to the optimal results. This experiment shows that the MILS algorithm has the highly desirable ability to find even better results with a longer cutoff limit for large and difficult instances.

5.4 A time-to-target analysis of the compared algorithms

To further demonstrate the computational efficiency of our MILS algorithm compared to the state-of-the-art algorithms, we perform a time-to-target (T2T) analysis by comparing the time needed for an algorithm to reach a given objective target. For this experiment, we compare MILS with HSNR [5], ITHSA [8], and MA [9], while excluding the HGA algorithm [7] because its source code is unavailable. Four instances were used: lin318-3, att532-3, nrw1379-5, and pcb3038-5. They are from different benchmark sets and have different features, such as the number of vertices, routes, and edge weight types. To ensure statistical robustness, each algorithm solves each instance 300 times with distinct random seeds. To guarantee that all compared algorithms can consistently attain the same target objective value, the target value was set 10% above the best known objective value. The experimental results are illustrated in Fig. 5, where the X-axis represents the runtime required to reach the predefined target value, and the Y-axis shows the probability that an algorithm

will finds a solution of the required quality.



(a) lin318-3, $C = 17229.86$ (b) att532-3, $C = 10918.60$

(c) nrw1379-5, $C = 12902.67$ (d) pcb3038-5, $C = 31373.43$

Fig. 5. Probability distributions of the running time (in seconds) needed to reach a given target objective value by each algorithm on 4 instances with various features.

Fig. 5 shows that the MILS algorithm consistently outperforms other algorithms in terms of convergence speed to the target value. Notably, MILS achieves a success probability of nearly 100% within the first 10 seconds. In contrast, MA, ITHSA, and HSNR require substantially longer durations to attain comparable results: over 50 seconds for nrw1379-5 and over 200 seconds for pcb3038-5. For small instances, the computational times remain relatively similar across algorithms, except for HSNR, which performs more slowly. However, as instance sizes increase, the reference algorithms become significantly less efficient, requiring more time to reach target values. In contrast, MILS demonstrates robust computational efficiency regardless of the problem's scale, indicating superior scalability. In summary, this experiment confirms that MILS is significantly more time-efficient than the reference algorithms, especially for large instances.

6 Conclusions

We introduced the first multi-armed bandit algorithm driven iterated local search for solving the minmax mTSP. The algorithm integrates an efficient local search procedure associated with the best-improvement strategy to find high-quality local optimal solutions, a probabilistic criterion to continuously

diversify the search and explore new search regions, and a multi-armed bandit algorithm to dynamically select tailored perturbation operators to escape deep local optima. It also takes advantage of an effective TSP heuristic for individual tour optimization.

Computational experiments on two sets of 77 commonly used benchmark instances (with up to 5915 cities and $m = 3, 5, 10, 20, 30$ routes) show that the proposed algorithm can effectively solve a wide range of instances in a short running time, leading to new best-known results for 32 challenging instances and matching 35 other best-known results. The algorithm is shown to perform particularly well on instances with 3, 5 and 20 tours, making it an attractive complementary approach to existing algorithms that perform well on instances with 10 tours. These new results can be valuable for future research on the minmax mTSP. In addition, we investigate the contributions of the multi-armed bandit algorithm and the best-improvement strategy for the local search to the performance of the algorithm in the context of the minmax mTSP.

Since the minmax mTSP is a relevant model for real-world problems, our algorithm can be used to solve some of these practical applications more effectively.

There are several possible directions for future work. First, since the local search component is the most time consuming part of the algorithm, it would be interesting to explore techniques, such as dynamic radius search [31] to improve its computational efficiency. Second, this work demonstrates the effectiveness of the multi-armed bandit algorithm for the minmax mTSP. This learning technique could contribute to designing effective algorithms using multiple operators for other routing problems [24,27,32]. Third, a promising way to improve the algorithm further is to design more discriminating evaluation functions that can differentiate solutions of the same objective value, like in [33,34]. Fourth, it would be interesting to adapt MILS to solve the generalized minmax mTSP with multi-depots. Additionally, ideas from this work, such as the learning-based operator selection method, could be applied to other algorithms for the multi-depot case. Finally, there is no effective exact method for the minmax mTSP. Therefore, it would be useful to develop new mathematical formulations that can be efficiently solved by general solvers, such as CPLEX and Gurobi, as well as dedicated exact algorithms that can solve reasonable-sized instances optimally.

Declaration of competing interest

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

Acknowledgments

We would like to thank the authors of [8], especially Dr. Jiongzhi Zheng, for sharing the code of their algorithm. We are grateful to the reviewers for their insightful comments. This work was supported by National Natural Science Foundation of China (Grant number: 62403123), The Fundamental Research Funds for the Central Universities (MCCSE2024B03).

References

[1] P. M. França, M. Gendreau, G. Laporte, F. M. Müller, The m-traveling salesman problem with minmax objective, Transportation Science 29 (3) (1995) 267–275.

[2] R. Dewil, P. Vansteenwegen, D. Cattrysse, A review of cutting path algorithms for laser cutters, The International Journal of Advanced Manufacturing Technology 87 (2016) 1865–1884.

[3] B. Zhou, R. Zhou, Y. Gan, F. Fang, Y. Mao, Multi-robot multi-station cooperative spot welding task allocation based on stepwise optimization: An industrial case study, Robotics and Computer-Integrated Manufacturing 73 (2022) 102197.

[4] X. Zhang, L. Duan, Fast deployment of uav networks for optimal wireless coverage, IEEE Transactions on Mobile Computing 18 (3) (2018) 588–601.

[5] P. He, J.-K. Hao, Hybrid search with neighborhood reduction for the multiple traveling salesman problem, Computers & Operations Research 142 (2022) 105726.

[6] M. Rao, A note on the multiple traveling salesmen problem, Operations Research 28 (3-part-i) (1980) 628–632.

[7] S. Mahmoudinazlou, C. Kwon, A hybrid genetic algorithm for the min–max multiple traveling salesman problem, Computers & Operations Research 162 (2024) 106455.

[8] J. Zheng, Y. Hong, W. Xu, W. Li, Y. Chen, An effective iterated two-stage heuristic algorithm for the multiple traveling salesmen problem, Computers & Operations Research 143 (2022) 105772.

[9] P. He, J.-K. Hao, Memetic search for the minmax multiple traveling salesman problem with single and multiple depots, European Journal of Operational Research 307 (3) (2023) 1055–1070.

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

[11] O. Cheikhrouhou, I. Khoufi, A comprehensive survey on the multiple traveling salesman problem: applications, approaches and taxonomy, Computer Science Review 40 (2021) 100369.

[12] K. Karabulut, H. Öztop, L. Kandiller, M. F. Tasgetiren, Modeling and optimization of multiple traveling salesmen problems: An evolution strategy approach, Computers & Operations Research 129 (2021) 105192.

[13] T. Vidal, Hybrid genetic search for the CVRP: Open-source implementation and swap* neighborhood, Computers & Operations Research 140 (2022) 105643.

[14] P. He, J.-K. Hao, General edge assembly crossover-driven memetic search for split delivery vehicle routing, Transportation Science 57 (2) (2023) 482–511.

[15] P. He, J.-K. Hao, Q. Wu, Hybrid genetic algorithm for undirected traveling salesman problems with profits, Networks: An International Journal 82 (3) (2023) 189–221.

[16] S. Ropke, D. Pisinger, An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows, Transportation Science 40 (4) (2006) 455–472.

[17] J. Kallestad, R. Hasibi, A. Hemmati, K. Sörensen, A general deep reinforcement learning hyperheuristic framework for solving combinatorial optimization problems, European Journal of Operational Research 309 (1) (2023) 446–468.

[18] F. Lagos, J. Pereira, Multi-armed bandit-based hyper-heuristics for combinatorial optimization problems, European Journal of Operational Research 312 (1) (2024) 70–91.

[19] H. R. Lourenço, O. C. Martin, T. Stützle, Iterated local search, in: Handbook of metaheuristics, Springer, 2003, pp. 320–353.

[20] Y. Nagata, S. Kobayashi, A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem, INFORMS Journal on Computing 25 (2) (2013) 346–363.

[21] N. Mladenović, P. Hansen, Variable neighborhood search, Computers & Operations Research 24 (11) (1997) 1097–1100.

[22] J. J. Bentley, Fast algorithms for geometric traveling salesman problems, ORSA Journal on Computing 4 (4) (1992) 387–411.

[23] L. Accorsi, D. Vigo, A fast and scalable heuristic for the solution of large-scale capacitated vehicle routing problems, Transportation Science 55 (4) (2021) 832–856.

[24] V. R. Máximo, M. C. Nascimento, A hybrid adaptive iterated local search with diversification control to the capacitated vehicle routing problem, European Journal of Operational Research 294 (3) (2021) 1108–1119.

[25] P. Shaw, Using constraint programming and local search methods to solve vehicle routing problems, in: International Conference on Principles and Practice of Constraint Programming, Springer, 1998, pp. 417–431.

[26] F. Arnold, Í. Santana, K. Sörensen, T. Vidal, Pils: exploring high-order neighborhoods by pattern mining and injection, Pattern Recognition 116 (2021) 107957.

[27] J. Christiaens, G. Vanden Berghe, Slack induction by string removals for vehicle routing problems, Transportation Science 54 (2) (2020) 417–433.

[28] V. Pandiri, A. Singh, Two metaheuristic approaches for the multiple traveling salesperson problem, Applied Soft Computing 26 (2015) 74–89.

[29] Y. Wang, Y. Chen, Y. Lin, Memetic algorithm based on sequential variable neighborhood descent for the minmax multiple traveling salesman problem, Computers & Industrial Engineering 106 (2017) 105–122.

[30] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, T. Stützle, The irace package: Iterated racing for automatic algorithm configuration, Operations Research Perspectives 3 (2016) 43–58.

[31] J. B. Gauthier, S. Irnich, Inter-depot moves and dynamic-radius search for multi-depot vehicle routing problems, Gutenberg School of Management and Economics (2022) (2020).

[32] Z. Zheng, S. Yao, G. Li, L. Han, Z. Wang, Pareto improver: Learning improvement heuristics for multi-objective route planning, IEEE Transactions on Intelligent Transportation Systems 25 (1) (2024) 1033–1043.

[33] D. C. Porumbel, J.-K. Hao, P. Kuntz, A study of evaluation functions for the graph k-coloring problem, in: N. Monmarché, E. Talbi, P. Collet, M. Schoenauer, E. Lutton (Eds.), 8th International Conference on Artificial Evolution, Tours, France, October 29-31, 2007, Revised Selected Papers, Vol. 4926 of Lecture Notes in Computer Science, Springer, 2007, pp. 124–135.

[34] E. Rodriguez-Tello, J.-K. Hao, J. Torres-Jimenez, An improved simulated annealing algorithm for bandwidth minimization, European Journal of Operational Research 185 (3) (2008) 1319–1335.

Appendix

A Computational results

This section presents the detailed computational results of the proposed MILS algorithm together with the results of the three state-of-the-art algorithms: HSNR (2022) [5], ITSHA (2022) [8], and HGA (2023) [7]. In Tables A.1 and A.2, column 'Instance' indicates the name of each instance (Instances marked with an asterisk * have known optimal solutions; column 'Time(s)' shows the running times of our MILS algorithm (note that the timing information is unavailable for the other algorithms); column 'BKS' are the best-known results summarized from the literature; 'Best' and 'Avg. ' are the best and average results over 20 independent runs, respectively, obtained by the corresponding algorithm in the column header; '$\delta(\%)$' is calculated as $\delta = 100 \times (f_{best} - BKS)/BKS$, where f_{best} is the best objective value of MILS. The *Average* row is the average value of a performance indicator over the instances of a

benchmark set. Improved best results (new upper bounds) are indicated by negative $\delta(\%)$ values highlighted in bold. In all tables, the dark gray color indicates that the corresponding algorithm obtains the best result among the compared algorithms on the corresponding instance; the medium gray color displays the second best results, and so on.

Table A.1. Results for the minmax mTSP on the 41 small and medium instances of set $\mathbb{S}$.

Instances	Time(s)	BKS	Best						Avg.				
			HSNR[5]	ITSHA [8]	HGA [7]	MA [9]	MILS	δ(%)	HSNR [5]	ITSHA [8]	HGA [7]	MA [9]	MILS
mtsp51-3	122.40	159.57	159.57	159.57	159.57	159.57	159.57	0.00	159.85	159.57	159.57	159.57	159.57
mtsp51-5	122.40	118.13	118.13	118.13	118.13	118.13	118.13	0.00	118.13	118.13	118.13	118.13	118.13
mtsp51-10	122.40	112.07*	112.07	112.07	112.07	112.07	112.07	0.00	112.07	112.07	112.07	112.07	112.07
mtsp100-3	240.00	8509.16	8509.16	8509.16	8509.16	8509.16	8509.16	0.00	8513.75	8509.16	8509.16	8509.16	8509.16
mtsp100-5	240.00	6765.73	6765.73	6767.82	6767.82	6765.73	6766.73	0.01	6770.67	6772.95	6770.50	6766.78	6768.16
mtsp100-10	240.00	6358.49*	6358.49	6358.49	6358.49	6358.49	6358.49	0.00	6358.49	6358.49	6358.49	6358.49	6358.49
mtsp100-20	240.00	6358.49*	6358.49	6358.49	6358.49	6358.49	6358.49	0.00	6358.49	6358.49	6358.49	6358.49	6358.49
rand100-3	240.00	3031.95	3031.95	3031.95	3031.95	3031.95	3031.95	0.00	3032.67	3033.65	3031.95	3031.95	3031.95
rand100-5	240.00	2409.63	2411.68	2412.35	2412.35	2409.63	2409.63	0.00	2415.00	2414.65	2409.63	2409.64	2409.96
rand100-10	240.00	2299.16*	2299.16	2299.16	2299.16	2299.16	2299.16	0.00	2299.16	2299.16	2299.16	2299.16	2299.16
rand100-20	240.00	2299.16*	2299.16	2299.16	2299.16	2299.16	2299.16	0.00	2299.16	2299.16	2299.16	2299.16	2299.16
mtsp150-3	360.00	13038.30	13075.80	13088.74	13038.34	13038.30	13038.30	0.00	13169.37	13210.69	13093.48	13079.74	13038.46
mtsp150-5	360.00	8417.02	8477.96	8492.97	8417.02	8417.02	8417.02	0.00	8538.83	8572.77	8486.81	8453.15	8430.47
mtsp150-10	360.00	5557.41	5590.64	5593.56	5561.97	5557.41	5590.19	0.59	5604.92	5608.50	5587.37	5588.86	5607.72
mtsp150-20	360.00	5246.49*	5246.49	5246.49	5246.49	5246.49	5246.49	0.00	5246.49	5246.49	5246.49	5246.49	5246.49
mtsp150-30	360.00	5246.49*	5246.49	5246.49	-	5246.49	5246.49	0.00	5246.49	5246.49	-	5246.49	5246.49
gtsp150-3	360.00	2401.63	2407.34	2407.34	2401.63	2401.63	2401.63	0.00	2435.49	2416.87	2401.63	2401.86	2402.19
gtsp150-5	360.00	1741.13	1741.71	1741.13	1741.13	1741.13	1741.13	0.00	1743.48	1752.06	1741.13	1741.13	1741.13
gtsp150-10	360.00	1554.64*	1554.64	1554.64	1554.64	1554.64	1554.64	0.00	1554.64	1554.64	1554.64	1554.76	1557.42
gtsp150-20	360.00	1554.64*	1554.64	1554.64	1554.64	1554.64	1554.64	0.00	1554.64	1554.64	1554.64	1554.64	1554.64
gtsp150-30	360.00	1554.64*	1554.64	1554.64	-	1554.64	1554.64	0.00	1554.64	1554.64	-	1554.64	1554.64
kroA200-3	480.00	10691.00	10748.10	10700.57	10691.03	10691.00	10691.00	0.00	10987.69	10819.85	10700.06	10691.41	10691.00
kroA200-5	480.00	7412.12	7418.87	7449.22	7421.12	7412.12	7413.80	0.02	7494.44	7513.67	7420.23	7414.21	7418.57
kroA200-10	480.00	6223.22*	6223.22	6223.22	6223.22	6223.22	6223.22	0.00	6223.22	6223.22	6223.22	6249.10	6226.12
kroA200-20	480.00	6223.22*	6223.22	6223.22	6223.22	6223.22	6223.22	0.00	6223.22	6223.22	6223.22	6223.22	6223.22
lin318-3	763.20	15663.50	15902.50	15930.04	15698.61	15663.50	15663.50	0.00	16207.05	16088.56	15714.31	15699.92	15751.70
lin318-5	763.20	11276.80	11295.20	11430.65	11289.26	11276.80	11276.80	0.00	11596.35	11601.67	11297.35	11291.59	11301.19
lin318-10	763.20	9731.17*	9731.17	9731.17	9731.17	9731.17	9731.17	0.00	9731.17	9731.17	9731.17	9731.17	9731.29
lin318-20	763.20	9731.17*	9731.17	9731.17	9731.17	9731.17	9731.17	0.00	9731.17	9731.17	9731.17	9731.17	9731.17
att532-3	1276.80	9966.00	10231.00	10158.00	9973.00	9966.00	9926.00	**-0.40**	10565.30	10344.50	10038.03	10064.00	9938.25
att532-5	1276.80	6950.00	7067.00	7067.00	6950.00	6986.00	6942.00	**-0.12**	7334.00	7156.80	7003.80	7070.95	6976.35
att532-10	1276.80	5698.00	5709.00	5731.00	5698.00	5770.00	5783.00	1.49	5738.90	5787.50	5716.75	5796.75	5846.30
att532-20	1276.80	5580*	5580.00	5583.00	5580.00	5580.00	5580.00	0.00	5580.00	5601.75	5580.00	5589.35	5580.00
rat783-3	1879.2	3052.41	3187.90	3131.99	3086.70	3052.41	3040.75	**-0.38**	3237.29	3180.79	3106.39	3083.52	3052.02
rat783-5	1879.2	1959.16	2006.46	2018.44	1959.16	1961.12	1941.90	**-0.88**	2044.32	2058.65	1981.51	1989.68	1948.48
rat783-10	1879.2	1304.70	1334.76	1357.65	1304.70	1313.01	1318.77	1.08	1345.88	1381.69	1321.33	1325.54	1341.19
rat783-20	1879.2	1231.69*	1231.69	1231.69	1231.69	1231.69	1231.69	0.00	1231.69	1231.84	1231.69	1235.37	1231.69
pcb1173-3	2815.5	19569.50	20813.80	20288.75	19724.24	19569.50	19412.40	**-0.80**	21144.92	20473.45	20016.94	19858.77	19572.01
pcb1173-5	2815.5	12406.60	13032.30	12816.55	12517.65	12406.60	12224.60	**-1.47**	13216.99	13045.14	12655.64	12639.49	12266.15
pcb1173-10	2815.5	7512.43	7758.26	7801.18	7512.43	7623.59	7476.78	**-0.47**	7897.20	7910.09	7617.21	7745.00	7511.91
pcb1173-20	2815.5	6528.86*	6528.86	6528.86	6528.86	6528.86	6528.86	0.00	6528.86	6534.75	6528.86	6548.87	6528.86
Average	-	6900.50	6015.33	6000.98	6077.37	5943.29	5929.96	-	6076.73	6043.73	6100.80	5971.30	5943.21

Table A.2. Results for the minmax mTSP on the 36 large instances of set $\mathbb{L}$.

Instances	Time(s)	BKS	Best					Avg.			
			HSNR [5]	ITSHA [8]	MA [9]	MILS	δ(%)	HSNR [5]	ITSHA [8]	MA [9]	MILS
nrw1379-3	3309.60	19222.10	20495.90	19871.21	19222.10	19126.40	**-0.50**	20765.70	20085.29	19472.39	19202.46
nrw1379-5	3309.60	11913.40	12416.50	12218.09	11913.40	11729.70	**-1.54**	12652.56	12493.99	12203.76	11769.61
nrw1379-10	3309.60	6846.27	7114.71	7008.61	6846.27	6576.46	**-3.94**	7212.24	7136.56	6987.31	6626.48
nrw1379-20	3309.60	5370.82*	5370.82	5370.82	5381.59	5370.82	0.00	5371.08	5402.42	5388.99	5371.08
fl1400-3	3360.00	7854.97	9192.38	8310.34	7854.97	7894.73	0.51	9621.59	8482.77	7989.25	8137.51
fl1400-5	3360.00	6116.28	6268.25	6360.45	6116.28	6280.38	2.68	6783.62	6572.53	6185.40	6701.72
fl1400-10	3360.00	5763.26*	5763.26	5763.26	5763.26	5763.26	0.00	5763.26	5785.69	5763.26	5763.26
fl1400-20	3360.00	5763.26*	5763.26	5763.26	5763.26	5763.26	0.00	5763.26	5763.26	5763.26	5763.26
d1655-3	3972.00	23921.90	25229.30	25403.26	23921.90	23853.70	**-0.29**	25635.98	25804.06	24149.72	24179.13
d1655-5	3972.00	16512.20	17181.20	17502.88	16512.20	16366.40	**-0.88**	17454.32	17824.61	16754.81	17025.55
d1655-10	3972.00	11320.10	11660.00	11814.34	11320.10	11430.20	0.97	11816.04	11971.60	11528.33	11761.92
d1655-20	3972.00	9598.94	9598.94	9910.12	9627.28	9598.94	0.00	9607.73	10172.24	9669.97	9602.37
u2152-3	5164.80	22127.80	24207.40	23295.73	22127.80	22010.50	**-0.53**	24747.01	23746.41	22629.25	22120.16
u2152-5	5164.80	14094.00	15055.10	14778.56	14094.00	13776.60	**-2.25**	15394.85	15236.58	14442.21	13843.31
u2152-10	5164.80	8332.12	8624.61	8763.71	8332.12	7981.80	**-4.20**	8780.91	9018.30	8499.64	8101.72
u2152-20	5164.80	6171.89	6171.89	6605.48	6253.35	6171.89	0.00	6225.82	6676.91	6339.15	6171.89
pr2392-3	5740.80	130015.00	141627.00	135763.02	130015.00	129213.00	**-0.62**	143703.00	137589.12	132228.70	129943.70
pr2392-5	5740.80	82408.50	88083.20	87465.60	82408.50	80098.90	**-2.80**	89582.83	88179.42	84448.64	80581.39
pr2392-10	5740.80	49033.60	51085.30	50514.84	49033.60	46131.80	**-5.92**	52100.80	50929.73	49985.84	46454.63
pr2392-20	5740.80	35325.30	35325.30	35999.41	35455.50	34746.30	**-1.64**	35709.02	36546.00	36107.77	35285.25
pcb3038-3	7291.20	46994.60	51049.90	48351.41	46994.60	46511.70	**-1.03**	51582.38	49081.79	47686.85	46853.08
pcb3038-5	7291.20	29223.00	31140.20	30089.85	29223.00	28521.30	**-2.40**	31495.59	30603.78	29864.61	28678.01
pcb3038-10	7291.20	16031.70	16949.90	16878.69	16031.70	15462.10	**-3.55**	17450.44	16645.10	16509.95	15545.04
pcb3038-20	7291.20	10769.60	10835.00	10827.78	10769.60	10375.70	**-3.66**	11004.40	11196.47	10961.26	10530.69
fl3795-3	9108.00	10927.40	11971.00	12290.18	10927.40	11106.00	1.63	12815.54	13022.12	11321.19	11889.81
fl3795-5	9108.00	7715.36	7923.71	8151.43	7715.36	7615.50	**-1.29**	8610.84	8657.18	8014.97	8750.10
fl3795-10	9108.00	5763.26*	5763.26	5824.14	5764.85	5805.59	0.73	5823.89	5990.03	5810.07	6416.12
fl3795-20	9108.00	5763.26*	5763.26	5763.26	5763.26	5763.26	0.00	5763.26	5766.51	5763.74	5763.26
fnl4461-3	10706.40	62016.70	66903.70	64140.70	62016.70	61643.60	**-0.60**	67971.34	65268.41	62894.65	61955.88
fnl4461-5	10706.40	38265.50	40721.20	39839.40	38265.50	37382.40	**-2.31**	41777.11	39839.40	38935.22	37698.19
fnl4461-10	10706.40	20671.60	22041.50	22145.65	20671.60	19499.40	**-5.67**	22891.45	22954.31	20996.25	19660.79
fnl4461-20	10706.40	12347.80	12630.10	12789.54	12347.80	11275.80	**-8.68**	13046.38	12987.65	12542.14	11444.20
rl5915-3	14196.00	193879.00	213864.00	210056.49	193879.00	190121.00	**-1.94**	226819.75	215466.06	198796.00	191662.90
rl5915-5	14196.00	120418.00	133457.00	125537.65	120418.00	116331.00	**-3.39**	145173.07	132524.74	124718.05	116977.65
rl5915-10	14196.00	66329.40	76585.20	70853.30	66329.40	61375.00	**-7.47**	84459.02	71353.13	67961.45	62271.92
rl5915-20	14196.00	43121.00	48958.50	44716.69	43121.00	39227.50	**-9.03**	60306.22	46080.96	44373.90	39739.66
Average	-	36758.87	35077.55	34076.09	32450.03	31608.39	-	36713.40	34801.53	33158.00	31951.21