Reinforced Hybrid Genetic Algorithm for the Traveling Salesman Problem

Jiongzhi Zheng\textsuperscript{a,b}, Jialun Zhong\textsuperscript{a}, Menglei Chen\textsuperscript{a} and Kun He\textsuperscript{a,*}

\textsuperscript{a}School of Computer Science, Huazhong University of Science and Technology, China

\textbf{ARTICLE INFO}

\textbf{Keywords:}
- Combinatorial optimization
- Traveling salesman problem
- Hybrid genetic algorithm
- Local search
- Reinforcement learning

\textbf{ABSTRACT}

In this paper, we propose a new method called the Reinforced Hybrid Genetic Algorithm (RHGA) for solving the famous NP-hard Traveling Salesman Problem (TSP). Specifically, we combine reinforcement learning with the well-known Edge Assembly Crossover genetic algorithm (EAX-GA) and the Lin-Kernighan-Helsgaun (LKH) local search heuristic. In the hybrid algorithm, LKH can help EAX-GA improve the population by its effective local search, and EAX-GA can help LKH escape from local optima by providing high-quality and diverse initial solutions. We restrict that there is only one special individual among the population in EAX-GA that can be improved by LKH. Such a mechanism can prevent the population diversity, efficiency, and algorithm performance from declining due to the redundant calling of LKH upon the population. As a result, our proposed hybrid mechanism can help EAX-GA and LKH boost each other’s performance without reducing the convergence rate of the population. The reinforcement learning technique based on Q-learning further promotes the hybrid genetic algorithm. Experimental results on 138 well-known and widely used TSP benchmarks with the number of cities ranging from 1,000 to 85,900 demonstrate the excellent performance of RHGA.

\textbf{1. Introduction}

Given a complete, undirected graph \( G = (V, E) \), where \( V = \{1, 2, ..., n\} \) denotes the set of \( n \) cities and \( E = \{(i, j) | i, j \in V \} \) denotes the set of all pairwise edges. \( d(i, j) \) represents the distance (cost) of edge \((i, j)\), i.e., the distance of traveling from city \( i \) to city \( j \). The Traveling Salesman Problem (TSP) aims to find a Hamiltonian cycle represented by a permutation \((s_1, s_2, ..., s_n)\) of cities \(\{1, 2, ..., n\}\) that minimizes the total distance, i.e., \(d(s_1, s_2) + d(s_2, s_3) + ... + d(s_{n-1}, s_n) + d(s_n, s_1)\). The TSP is one of the most famous and well-studied NP-hard combinatorial optimization problems, which is very easy to understand but very difficult to solve to the optimality. Over the years, the TSP has become a touchstone in the field of the combinatorial optimization.

Typical methods for solving the TSP can be categorized into exact algorithms, approximation algorithms, and heuristics. The exact algorithms may be prohibitive for large instances, and the approximation algorithms may suffer from weak optimal guarantees or empirical performance [1]. Heuristics are known to be the most efficient and effective approaches for solving the TSP. Two of the state-of-the-art heuristics are the Lin-Kernighan-Helsgaun (LKH) local search algorithm [2] and the Edge Assembly Crossover genetic algorithm (EAX-GA) [3]. Both of them provide the best-known solutions on many TSP benchmark instances.

As two representative heuristic algorithms, both LKH and EAX-GA have advantages and disadvantages. For example, EAX-GA is very efficient and powerful in solving TSP instances with tens to hundreds thousands of cities, providing the best-known solutions of the six famous instances with 100,000 to 200,000 cities in the Art TSP benchmarks\textsuperscript{1}. But EAX-GA is hard to scale to super large instances, such as TSP instances with millions of cities, since the convergence of the population is too time-consuming. As an efficient local search algorithm, LKH can yield near-optimal solutions faster than EAX-GA does. It is also suitable for TSP instances with various scales, especially for super large instances, providing the best-known solution of the famous World TSP instance with 1,904,711 cities\textsuperscript{2}. However, LKH is not as good as EAX-GA in solving the TSPs with 10,000 to 200,000 cities, since the population can help EAX-GA explore the solution space better than LKH does for instances with such large scales. Based on these characteristics, a straightforward idea is proposed spontaneously. That is, whether there is a reasonable way to combine EAX-GA with LKH and make use of their complementary, so as to help them boost each other.

There have been related studies trying to combine EAX-GA with LKH or its predecessor, the LK heuristic [4]. For example, Tsai et al. [5] propose to combine the earliest version of EAX-GA [6] with LK. Their proposed algorithm HeSEA reports better results than EAX-GA, LK, and LKH in solving TSP instances with at most 15,112 cities. However, HeSEA follows the similar hybrid mechanism of many other hybrid algorithms [7, 8, 9, 10, 11, 12, 13, 14] that combine genetic algorithms with LK-based algorithms (or other local search methods like 2-opt). That is, applying the local search methods to optimize every individual in the current population or every surviving offspring generated. Such a mechanism has two disadvantages: 1) the population diversity will be broken because the local optimal solutions (of different tours) calculated by the same local search method are similar. 2) It is very time-consuming to frequently apply local search methods to calculate the local optimal solutions, as HeSEA [5] shows worse efficiency and

\textsuperscript{1}http://www.math.uwaterloo.ca/tpsp/data/art/index.html

\textsuperscript{2}http://www.math.uwaterloo.ca/tpsp/world/index.html

\textsuperscript{*}Corresponding author. Email: brooklet60@hust.edu.cn

\textsuperscript{ORCID iD} (a): 0000-0002-9059-5054

\textsuperscript{1}Corresponding author. Email: jzheng@hust.edu.cn

\textsuperscript{2}http://www.math.uwaterloo.ca/tpsp/world/index.html

\textsuperscript{3}http://www.math.uwaterloo.ca/tpsp/world/index.html
reinforcement learning in our RHGA and EAX-GA. In this way, both LKH and EAX-GA can be improved by the local search algorithm of LKH, and help them boost each other. As a result, the hybrid mechanism in our proposed algorithm can fix the aforementioned issues of the hybrid mechanisms in the existing hybrid genetic algorithms [7, 8, 9, 5, 10, 11, 12, 13, 14] and hybrid solver [15] for the TSP and fits well with EAX-GA and LKH.

Moreover, we apply reinforcement learning [16] to further improve the performance of the proposed hybrid genetic algorithm. We apply the technique proposed by Zheng et al. [19] that employs reinforcement learning to learn an adaptive Q-value as a metric for evaluating the quality of the edges. We use the adaptive Q-value learned by the Q-learning algorithm [16] to replace the important evaluation metrics of the edges used in the key steps of both EAX-GA and LKH. In this way, both LKH and EAX-GA can be enhanced by the reinforcement learning in our RHGA algorithm. Related studies of (reinforcement) learning based methods for the TSP are referred to Section 2.1, where we also describe the advantages of our reinforcement learning method over them.

The main contributions of this work are as follows:

- We propose a creative and distinctive hybrid mechanism to combine two of the state-of-the-art TSP heuristic algorithms, EAX-GA and LKH, through a special individual. In the proposed RHGA algorithm, EAX-GA and LKH can boost each other with the bridge of the special individual.

- We propose to combine reinforcement learning with the key steps of both EAX-GA and LKH to further improve the performance of the hybrid genetic algorithm. The adaptive Q-value learned by the Q-learning algorithm significantly outperforms the metrics used in EAX-GA and LKH for evaluating the quality of the edges.

- Our proposed techniques, including the hybrid mechanism of combining genetic algorithm with local search method and the method of combining reinforcement learning with the key search steps of heuristics, can be applied to solve various combinatorial optimization problems, such as variant problems of TSP, the vehicle routing problems and the graph coloring problems.

- Experimental results on 138 well-known and widely used TSP benchmarks with the number of cities ranging from 1,000 to 85,900 demonstrate the promising performance of our proposed algorithm.

2. Related Works

For related works, we first introduce (reinforcement) learning based algorithms for solving the TSP, then briefly introduce the main ideas and approaches in the two state-of-the-art heuristic algorithms for solving the TSP, EAX-GA and LKH, which will also be incorporated into our proposed algorithm. For details of these two algorithms, we refer to [3] and [2].

2.1. Learning Based Algorithms for the TSP

(Reinforcement) learning based methods for the TSP can be divided into two categories. The first category is end-to-end methods [20], which are usually based on deep neural networks. When receiving an input TSP instance, they use the trained learning model to generate a solution directly. For example, Bello et al. [21] address TSP by using an actor-critic method to train a pointer network [22]. The S2V-DQN algorithm [1] applies reinforcement learning to train a graph neural network so as to solve several combinatorial optimization problems, including minimum vertex cover, maximum cut, and TSP. Goh et al. [23] use an encoder based on a standard multi-headed transformer architecture and a Softmax or Sinkhorn [24, 25] decoder to directly solve the TSP. These methods provide good innovations in the field of applying machine learning to solve combinatorial optimization problems. As for the performance, they can yield near-optimal or optimal solutions for the TSP instances with less than hundreds of cities. However, they are usually hard to scale to large instances (with more than thousands of cities) due to the complexity of deep neural networks.

Methods belonging to the second category combine (reinforcement) learning methods with traditional algorithms. Some of them use traditional algorithms as the core and frequently call the learning models to help explore the solution space or guide the search direction. For example, Liu and Zeng [26] employ reinforcement learning to construct mutation individuals in the previous version of EAX-GA [27] and report better results than EAX-GA and LKH on instances with up to 2,392 cities. But the efficiency of their proposed algorithm is not as good as that of LKH.
Costa et al. [28] and Sui et al. [29] use deep reinforcement learning to guide 2-opt and 3-opt local search operators, and report results on instances with no more than 500 cities. Other methods separate the learning models and traditional algorithms. They first apply (reinforcement) learning methods to yield initial solutions [30] or some configuration information [31], and then use traditional algorithms to find high-quality solutions followed the obtained initial solutions or information. Among them, the NeuroLKH algorithm [31] is one of the state-of-the-art, which uses a Sparse Graph Network with supervised learning to generate the candidate edges for LKH. It reports better or similar results compared with LKH in instances with less than 6,000 cities.

In summary, (reinforcement) learning based methods with deep neural networks for the TSP may suffer from the bottleneck of hardly solving large scale instances, and the combination of traditional reinforcement learning methods (training tables, not deep neural networks) with existing (heuristic) algorithms may reduce the efficiency of the algorithm. The reinforcement learning method in our proposed RHGA algorithm can avoid these issues. On the one hand, instead of using deep neural networks, our reinforcement learning method uses the traditional Q-learning algorithm [16] to train a table. Therefore, our algorithm can solve very large instances, as we tested RHGA on instances with at most 85,900 cities. On the other hand, we combine reinforcement learning with the core search steps of LKH and EAX-GA in a reasonable way, which prevents the reduction of the efficiency. The experimental results show that RHGA significantly outperforms the newest versions of LKH and EAX-GA within similar calculation time.

2.2. Edge Assembly Crossover Genetic Algorithm

The EAX-GA algorithm [3] generates offspring solutions by combining edges from the two parent solutions and adding relatively few new short edges determined by a simple search procedure that is similar to the 2-opt local search. The core of EAX-GA is its edge assembly crossover (EAX) operation. Let \( p_A \) and \( p_B \) be two parent solutions, EAX-GA uses the EAX operation to generate \( N_{ch} \) (30 by default) offsprings of \( p_A \) and \( p_B \), and replaces \( p_A \) with the best individual among the \( N_{ch} \) offsprings and \( p_A \) according to an evaluation function based on the edge entropy measure [32]. Applying the edge entropy measure rather than the straightforward tour length measure can significantly improve the diversity of the population. Let \( E_A \subset E \) and \( E_B \subset E \) be the sets of edges corresponding to \( p_A \) and \( p_B \), the EAX operation generates offsprings through the following six steps.

- **Step 1:** Construct an undirected multigraph \( G_{AB} = (V, E_A \cup E_B) \) by combining all the edges of \( E_A \) and \( E_B \). The edges belonging to either \( E_A \) or \( E_B \) in \( G_{AB} \) are labeled.

- **Step 2:** Randomly partition all edges of \( G_{AB} \) into \( AB \)-cycles, where an \( AB \)-cycle consists of alternately linked edges of \( E_A \) and \( E_B \).

- **Step 3:** Construct an \( E \)-set by selecting \( AB \)-cycles according to a given selection strategy, where an \( E \)-set is defined as the union of \( AB \)-cycles.

- **Step 4:** Generate an intermediate solution from \( p_A \) by removing the edges of \( E_A \) and adding the edges of \( E_B \) in the \( E \)-set. Let \( E_C = (E_A \setminus (E_B \cap E_A)) \cup (E_B \cap E_B) \) be the set of edges in the intermediate solution. An intermediate solution consists of one or more sub-tours and may not be a feasible solution for TSP.

- **Step 5:** Connect all sub-tours into a tour to generate a valid offspring. This step merges the smallest sub-tour (the sub-tour with the least number of edges) with other sub-tours each time. Let \( U \) be the set of edges in the smallest sub-tour, the goal is to find 4-tuples of edges \( \{e^*, e^{**}, e'^*, e''^*\} = \arg \min_{e^* \in U, e'^* \in E_B \setminus U} \{ -d(e) - d(e') + d(e^*) + d(e^{**}) \} \), where \( e \) and \( e' \) denote two edges to be removed, and \( e^*, e^{**}, e'^*, e''^* \) denote two edges to be added to connect the breakpoints. Then the sub-tours are connected by \( E_C \leftarrow (E_C \setminus \{e^*, e^{**}\}) \cup \{e'^*, e''^*\} \). In particular, EAX-GA restricts the search to promising pairs of \( e \) and \( e' \) to reduce the search scope and improve the efficiency. For each \( e \in U \), the candidates of \( e' \) are restricted to a set of edges that satisfy the following condition: at least one end of \( e' \) is among the \( N_{near} \) (10 by default) closest to either end of \( e \).

- **Step 6:** Loop steps 3-5 until \( N_{ch} \) offsprings are generated. Then terminate the procedure.

Note that the metric for determining the candidates of \( e' \) in Step 5 is the distance. This metric is very important since it determines the new edges that can be added to the population. In the proposed RHGA algorithm, we replace the distance metric used here with the Q-value learned by the Q-learning algorithm to improve the performance.

The EAX-GA algorithm consists of two stages. It terminates stage I when no improvement in the best solution is found over a period of generations, and then switches to stage II. Specifically, let \( Gen \) be the number of generations at which no improvement in the best solution is found over the recent 1500/\( N_{ch} \) generations. The value of \( Gen \) has already been determined and the best solution does not improve over the last \( G_{max} = Gen/10 \) generations, EAX-GA terminates stage I and proceeds to stage II. Stage II is also terminated by the same condition (both \( Gen \) and \( G_{max} \) should be recalculated in this stage).

The only difference between the two stages is the selection strategy of the \( E \)-set (Step 3) during the EAX crossover process. In stage I, a single \( AB \)-cycle is selected randomly as the \( E \)-set without overlapping with the previous selections. Such a strategy is very simple and fast, thus can help the population converge quickly. In stage II, the block2 strategy [3] is applied, which is effective in solving large TSP instances. Its basic idea is to construct an \( E \)-set by selecting \( AB \)-cycles so that the resulting intermediate solution consists of relatively few sub-tours and the resulting offspring consists of more...
edges of $p_B$. The intermediate solution with few sub-tours corresponds to an offspring that inherits its parents well, and making the offspring inherit more edges of $p_B$ can prevent the algorithm from falling into the local optima easily.

### 2.3. Lin-Kernighan-Helsgaun Algorithm

LKH uses the $k$-opt heuristic [33] as the optimization method to find high-quality solutions. The $k$-opt in LKH replaces at most $k_{max}$ (5 by default) edges in the current tour with the same number of new edges, and restricts that the edges to be added must be selected from the candidate sets, so as to reduce the search scope and improve the efficiency. This subsection introduces two important parts of LKH, i.e., the method of creating the candidate sets and the $k$-opt process.

#### 2.3.1. Candidate Sets in LKH

In LKH, each city has its candidate set that records several candidate cities. Let $CS'_{i}$ be the candidate set of city $i$ ($i \in V$), LKH restricts that the edges to be added in the $k$-opt process must be selected from the set $\{(i, j) \in E | j \in CS'_{i} \vee i \in CS'_{j}\}$. LKH proposes an $α$-value to evaluate the quality of the edges, and applies the $α$-value as the metric for selecting and sorting candidate cities. The $α$-value is defined from the structure of 1-tree [34]. A 1-tree is a lower bound of the optimal TSP solution. The equation for calculating the $α$-value of an edge ($i, j$) is as follows:

$$a(i, j) = L(T^+(i, j)) - L(T),$$

(1)

where $L(T)$ is the length of the minimum 1-tree of the graph $G$, and $L(T^+(i, j))$ is the length of the minimum 1-tree required to contain edge ($i, j$). The candidate set of each city in LKH records five (default value) other cities with the smallest $α$-value to this city in ascending order. The advantage of the candidate set is further enhanced by adding penalties to the cities. Details about the penalties are referred to [2].

#### 2.3.2. $k$-opt in LKH

The $k$-opt process is actually a partial depth-first search process, that the maximum depth of the search tree is restricted to $k_{max}$. The $k$-opt process starts from a starting city $p_1$ (i.e., root of the search tree), then alternatively selects an edge to be removed, i.e., edge $(p_{2k-1}, p_{2k})$, and an edge to be added, i.e., edge $(p_{2k}, p_{2k+1})$, until the maximum search depth is reached or a $k$-opt move that can improve the current tour is found. Note that these edges are connected, thus selecting the involved edges in $k$-opt can be regarded as selecting a sequence (cycle) of cities. The selection of the cities $p_{2k}$ and $p_{2k+1}$ should satisfy the following constraints:

- **C-I:** for $k \geq 2$, connecting $p_{2k}$ back to $p_1$ should result in a feasible TSP tour.

**Algorithm 1:** $k$-opt($x_{in}, p_1, \ p, k, k_{max}$)

| Input: | Output: |
|-------|---------|
| input solution: $x_{in}$, starting city: $p_1$, sequence of the corresponding cities: $p$, current search depth: $k$, the maximum search depth: $k_{max}$ | output solution $x_{out}$, sequence of the corresponding cities $p$ |

1. for $i \leftarrow 1 : 2$
   2. $p_{2k} \leftarrow p_{2k-1}$
   3. if $p_{2k}$ does not satisfy the constraint C-I then continue;
   4. if $k \geq 2 \land \sum_{j=1}^{k} d(p_{2j-1}, p_{2j}) < \sum_{j=1}^{k-1} d(p_{2j}, p_{2j+1}) + d(p_{2k}, p_1)$ then
   5. $x_{out} \leftarrow x_{in}$
   6. for $j \leftarrow 1 : k$
      7. remove edge $(p_{2j-1}, p_{2j})$ from $x_{out}$;
   8. for $j \leftarrow 1 : k - 1$
      9. add edge $(p_{2j}, p_{2j+1})$ into $x_{out}$;
   10. Add edge $(p_{2k}, p_1)$ into $x_{out}$;
   11. return ($x_{out}, p$);
   12. if $k = k_{max}$ then return ($x_{in}, \emptyset$);
   13. for $j \leftarrow 1 : 5$
      14. $p_{2k+1} \leftarrow$ the $j$-th city in $CS_{p_{2k}}$
      15. if $p_{2k+1}$ doesn’t satisfy constraint C-II then continue;
      16. ($x_{temp}, p'$) $\leftarrow k$-opt($x_{in}, p_1, p \cup \{p_{2k}, p_{2k+1}\}, k + 1, k_{max}$);
      17. if $l(x_{temp}) < l(x_{in})$ then return ($x_{temp}, p'$);
   18. $p_{2k+1}$ is always chosen so that \[ \sum_{j=1}^{k} (d(p_{2j-1}, p_{2k}) - d(p_{2j}, p_{2k+1})) > 0. \]

Let $t_1$ be a city randomly picked from the two cities connected with city $t$ in the current TSP tour, $t_2$ be the other, $l(x)$ be the length of solution $x$. The procedure of the $k$-opt process is presented in Algorithm 1. As shown in Algorithm 1, the $k$-opt process tries to improve the current solution by traversing the partial depth-first search tree from the root $p_1$. When selecting the edge to be removed, i.e., edge $(p_{2k-1}, p_{2k})$ (the same as selecting $p_{2k}$ from $p_{2k-1}$), the algorithm traverses the two cities connected with city $p_{2k-1}$ in the current TSP tour (lines 1-2). When selecting the edge to be added, i.e., edge $(p_{2k}, p_{2k+1})$ (the same as selecting $p_{2k+1}$ from $p_{2k}$), the algorithm traverses the candidate set of city $p_{2k}$ (lines 14-15), and the constraint C-II is applied as a smart pruning strategy to improve the efficiency (lines 16-17). Once a $k$-opt move that can improve the current solution is found, the algorithm performs this move on $x_{in}$ and outputs the resulting solution $x_{out}$ (lines 5-12).

Zheng et al.: Preprint submitted to Elsevier
3. The Proposed Algorithm

In the proposed reinforced hybrid genetic algorithm (RHGA), we design a novel hybrid mechanism with a special individual as the core to combine EAX-GA with the LKH local search. The EAX-GA and LKH can boost each other with the help of the special individual. The reinforcement learning technique [19] is combined with the key steps of both LKH and EAX-GA to further improve the hybrid genetic algorithm, by replacing the evaluation metrics for the edges used in LKH (α-value) and EAX-GA (distance) with the learned adaptive Q-value.

This section first introduces how Q-value (i.e., reinforcement learning) is used in RHGA, then introduces the reinforced LKH local search method (Q-LKH) in RHGA, and describes the main process of RHGA that contains the description of the proposed hybrid mechanism, and finally concludes the advantages of RHGA.

3.1. Q-value in RHGA

The Q-value in RHGA actually determines the candidate edges in both LKH and EAX-GA. Note that the larger the Q-value of an edge, the higher-quality of the edge. The candidate set of each city in RHGA records the Q-value, the higher-quality of the edge. The Q-value in RHGA actually determines the candidate edges in both LKH and EAX-GA. Note that the larger the Q-value of an edge, the higher-quality of the edge. The candidate set of each city in RHGA records the Q-value, the higher-quality of the edge. The Q-value in RHGA actually determines the candidate edges in both LKH and EAX-GA. Note that the larger the Q-value of an edge, the higher-quality of the edge.

RHGA designs an initial Q-value for each edge to generate the initial candidate sets. Before calculating the initial Q-value, the algorithm needs to calculate the lower bound of the optimal TSP solution \( L(T) \) (see Eq. 1) and the \( α \)-values corresponding to \( L(T) \), by the method in LKH (see Section 2.3.1). Then the initial Q-value for edge \((i, j)\) can be calculated by:

\[
Q(i, j) = \frac{L(T)}{α(i, j) + d(i, j)}.
\] (2)

The initial Q-value combines the metrics of evaluating the quality of edges in both EAX-GA and LKH, i.e., the distance and \( α \)-value. The \( L(T) \) is applied to adaptively adjust the magnitude of the initial Q-value for different instances.

The Q-value can be updated by the Q-learning algorithm during the Q-LKH component of RHGA (see details in the next subsection). Note that the EAX-GA component only uses the Q-value but does not update it. After each Q-LKH process, the candidate set of each city in RHGA will be sorted according to the updated Q-value. Therefore, the order or the elements of the top five or top ten cities in the candidate set of each city might be changed by our reinforcement learning method. In this way, our reinforcement learning method can provide better candidate edges for both the EAX-GA and LKH components of RHGA and help the algorithm learn to select appropriate edges to be added during the \( k \)-opt process and the sub-tour merging process.

3.2. The Q-LKH Local Search Algorithm

We apply the method proposed by Zheng et al. [19] to combine Q-learning [16] with LKH to learn the Q-value. The reinforced LKH algorithm (by Q-learning) is denoted as Q-LKH.

In Q-LKH, the reinforcement learning is combined with the core \( k \)-opt search process. A \( k \)-opt process corresponds to an episode in reinforcement learning, where the states and actions are the two endpoints of the selected edges to be added during the \( k \)-opt process. Specifically, for an episode \((x', p)\) \( k \)-opt \((x, p_1, \{p\}_1, 1, k_{\text{max}}\), the states are the cities that are going to select the edges to be added from their candidate sets, i.e., cities \( p_2k, k \in \{1, 2, ..., |p| - 1\}, \) and the actions correspond to the selection of the candidate cities, i.e., cities \( p_2k+1, k \in \{1, 2, ..., |p| - 1\} \). The reward of the state-action pair \((p_2k, p_{2k+1})\) is defined as \( r_k = d(p_{2k-1}, p_{2k}) - d(p_{2k}, p_{2k+1}) \), since the \( k \)-opt move replaces edge \((p_{2k-1}, p_{2k})\) with edge \((p_{2k}, p_{2k+1})\).

The Q-LKH applies the Q-learning algorithm to update the Q-value of each state-action pair in each episode (\( k \)-opt process). For an episode \((x', p)\) \( k \)-opt \((x, p_1, \{p\}_1, 1, k_{\text{max}}\), the Q-value of each state-action pair \((p_{2k}, p_{2k+1})\) is updated as follows:

\[
Q(p_{2k}, p_{2k+1}) = (1 - \lambda) \cdot Q(p_{2k}, p_{2k+1}) + \lambda \cdot \left[ r_k + \gamma \max_{a' \in CS} Q(p_{2k+2}, a') \right].
\] (3)

---

Algorithm 2: Q-LKH(\(x_{in}, k_{\text{max}}, \lambda, \gamma\))

**Input:** input solution \(x_{in}\), the maximum search depth: \(k_{\text{max}}\), learning rate: \(\lambda\), reward discount factor: \(\gamma\)

**Output:** output solution: \(x_{out}\)

1. Initialize the set of the cities that have not been selected as the starting city of the \(k\)-opt:
   \(A \leftarrow \{1, 2, ..., n\}\); \(x_{out} \leftarrow x_{in}\)
2. while TRUE do
   3. if \(A = \emptyset\) then break;
   4. \(p_1 \leftarrow\) a random city in \(A\); \(A \leftarrow A \backslash \{p_1\}\);
   5. \((x_{\text{temp}}, p) \leftarrow k\text{-opt}(x_{out}, p_1, \{p_1\}_1, 1, k_{\text{max}})\);
   6. Update the Q-value of each state-action pair in \(p\) according to Eq. 3;
   7. if \(l(x_{\text{temp}}) < l(x_{out})\) then
     8. \(x_{out} \leftarrow x_{\text{temp}}\); for \(j \leftarrow 1 : |p|\) do \(A \leftarrow A \cup \{p_j\}\);
   9. Sort the candidate sets of each city in descending order of the Q-value;
10. return \(x_{out}\);
where $\lambda$ is the learning rate, and $\gamma$ is the reward discount factor.

The procedure of the Q-LKH local search is presented in Algorithm 2. Q-LKH algorithm uses the k-opt heuristic (Algorithm 1) to improve the current solution $x_{out}$ until the local optimum is reached (lines 2-9), i.e., $x_{out}$ cannot be improved by the k-opt heuristic starting from any starting city $p_1 \in \{1, 2, ..., n\}$ (line 3). Once the current solution is improved by a k-opt move (lines 7-8), each involved city can be selected as the root $p_l$ again (line 9). Q-LKH updates the Q-value after each k-opt process (line 6), and reorders the candidate sets of each city at the end of the algorithm (line 10).

3.3. Main Process of RHGA

The main flow of RHGA is presented in Algorithm 3. In the initialization phase of RHGA (lines 1-3), the initial candidate set of each city is generated according to the initial Q-value calculated by Eq. 2, and the initial population (300 by default) individuals is generated by the Generate_Initial_Pop() function, which is a greedy 2-opt local search method used in EAX-GA [3]. Note that the candidate sets and the Q-value are regarded as the global information in the entire RHGA algorithm.

In the improvement phase of RHGA (lines 4-25), the Q-LKH local search algorithm and the EAX genetic algorithm are used to improve the population alternatively. In order to prevent the reduction of population diversity and algorithm efficiency, there is only one special individual (i.e., $x_1$) that can be improved by the Q-LKH local search algorithm. Specifically, before the procedure of the genetic algorithm (lines 21-25) at each generation, RHGA tries to improve the special individual $x_1$ by the Q-LKH local search algorithm in the following three cases:

- **Case 1**: (lines 8-10) When $x_1$ is just initialized or $x_1$ was improved by EAX-GA at the last generation, i.e., when $x_1$ may not be a local optimal solution for the Q-LKH local search algorithm. In this case, the Q-LKH algorithm will try to improve the special individual $x_1$.

- **Case 2**: (lines 11-15) When the tour length of the best individual $x_{best}$ in the population other than $x_1$ is shorter than that of $x_1$, and $x_{best}$ has not been calculated by Q-LKH. In this case, the Q-LKH algorithm will try to improve $x_{best}$. If $x_{best}$ can be improved, replace $x_1$ with the improved solution, and $x_{best}$ will not change.

- **Case 3**: (lines 16-20) When $x_1$ has not been improved for $M_{gen}$ generations. Note that the counter num will always be initialized to zero (no matter whether $x_1$ can be improved). In this case, RHGA randomly selects an individual $x_r$ in the population ($x_r \neq x_1$). If Q-LKH can improve $x_r$, the improved tour is better than $x_1$, the improved tour will replace $x_1$, and $x_r$ will not change.

The design of applying the Q-LKH to improve the special individual in the above three cases is reasonable and effective. Firstly, in the first two cases, the Q-LKH is prohibited from performing on its local optimal solutions to improve the efficiency, since Q-LKH can hardly improve the local optimal solution calculated by itself. Secondly, in Case 2, the individual $x_{best}$ with a shorter length than $x_1$ is a very high-quality initial solution for the Q-LKH. Because $x_{best}$ is better than the local optimal solution of $x_1$, and it may not be a local optimum for Q-LKH. Thus performing Q-LKH on $x_{best}$ in Case 2 is necessary, and may obtain the near-optimal or even the optimal solution. Thirdly, in Case

---

**Algorithm 3: RHGA($N_{pop}, N_{ch}, k_{max}, \lambda, \gamma, M_{gen}, OPT$)**

**Input**: population size: $N_{pop}$; number of offsprings produced by a pair of parents: $N_{ch}$; the maximum search depth: $k_{max}$; learning rate: $\lambda$, reward decay factor: $\gamma$; number of generations to perform Case 3: $M_{gen}$; length of the optimal solution: $OPT$

**Output**: output solution: $x_{out}$

1. Generate the initial candidate sets according to the initial Q-value (Eq. 2);
2. $\{x_1, x_2, ..., x_{N_{pop}}\} \leftarrow \text{Generate}_\text{Initial}_\text{Pop}()$;
3. Initialize $l_{old}^1 \leftarrow +\infty$, $l_{best} \leftarrow +\infty$, num $\leftarrow 0$;
4. while a termination condition is not satisfied do
   5.     $x_{best} \leftarrow \arg\min_{x \in \{x_1, ..., x_{N_{pop}}\}} l(x)$;
   6.     if $l(x_1) = OPT \lor l(x_{best}) = OPT$ then break;
   7.     num $\leftarrow$ num $+$ 1;
   8.     if $l(x_1) < l_{old}^1$ then
       9.         $x_1 \leftarrow \text{Q-LKH}(x_1, k_{max}, \lambda, \gamma)$;
      10.        $l_{old}^1 \leftarrow l(x_1)$, num $\leftarrow 0$;
   11.    if $l(x_{best}) < l(x_1) \land l(x_{best}) < l_{old}^1$ then
       12.        $l_{best} \leftarrow l(x_{best})$;
      13.        $x_{temp} \leftarrow \text{Q-LKH}(x_{best}, k_{max}, \lambda, \gamma)$;
   14.    if $l(x_{temp}) < l(x_{best})$ then
       15.        $x_1 \leftarrow x_{temp}$, $l_{old}^1 \leftarrow l(x_1)$, num $\leftarrow 0$;
   16.    if num $\geq M_{gen}$ then
       17.        $x_r \leftarrow$ a random individual in $\{x_2, ..., x_{N_{pop}}\}$;
      18.        $x_{temp} \leftarrow \text{Q-LKH}(x_r, k_{max}, \lambda, \gamma)$, num $\leftarrow 0$;
   19.    if $l(x_{temp}) < l(x_1)$ then
       20.        $x_1 \leftarrow x_{temp}$, $l_{old}^1 \leftarrow l(x_1)$;
      21.    $r(p) \leftarrow$ a random permutation of $\{1, 2, ..., N_{pop}\}$;
   22.    for $l \leftarrow 1 : N_{pop}$ do
      23.        $P_A \leftarrow x_{r(p(l))}$, $P_B \leftarrow x_{r(p(l)+1))}$;
      24.        $\{c_1, c_2, ..., c_{N_{ch}}\} \leftarrow \text{EAX}(P_A, P_B)$;
      25.        $x_{r(p(l))} \leftarrow \text{Select}_\text{Survive}(c_1, ..., c_{N_{ch}}, P_A)$;
   26.    if $l(x_1) < l(x_{best})$ then return $x_1$;
   27.    else return $x_{best}$;

---

Zheng et al.: Preprint submitted to Elsevier
3, various individuals can provide high-quality and diverse initial solutions for Q-LKH to escape from the local optima.

After the local search process in each generation we have the EAX genetic process (lines 21-25). During this process, each individual in the population is selected once as parent $p_A$ and once as parent $p_B$, in a random order (lines 21-23). The algorithm applies the methods described in Section 2.2 to use the EAX crossover operation represented by function $EAX()$ to parents $p_A$ and $p_B$ to produce $N_{ch}$ offsprings (line 24), and then selects the surviving individual among the offsprings and $p_A$ (line 25).

The RHGA algorithm also consists of two stages as EAX-GA does. The termination conditions of the two stages in RHGA are the same as those in EAX-GA (see Section 2.2). Moreover, if the definite optimal solution of the TSP instance is known, the input parameter $OPT$ is set to the length of the optimal solution, otherwise zero. RHGA also terminates when the definite optimal solution is found (line 6).

3.4. Advantages of RHGA

This subsection illustrates the advantages of the proposed RHGA algorithm, i.e., why the RHGA is effective and better than the baseline algorithms (EAX-GA and LKH)? The advantages of RHGA over the baselines include the mechanism of the hybrid genetic algorithm and the impact of reinforcement learning.

3.4.1. Mechanism of the Hybrid Genetic Algorithm

The combination of EAX-GA and LKH by the proposed mechanism can boost the performance of each other. For the EAX-GA, the special individual $x_1$ can spread good genes (the candidate edges in LKH) to the population, and lead the population to converge to better solutions. For the LKH, the population can help the special individual $x_1$ escape from the local optima of LKH, and provide higher-quality and more diverse initial tours than the initial tours generated by the heuristic in LKH [2, 35]. The hybrid mechanism in RHGA can improve the baseline algorithms without reducing the population diversity and algorithm efficiency, since there is only one special individual in the population. The experimental results also demonstrate that setting only one special individual is reasonable and efficient.

Moreover, the combination of EAX-GA and LKH can combine their advantages and overcome their disadvantages (their pros and cons are described in Section 1). That is, RHGA can solve the TSP instances with tens to hundreds of cities as well as or better than EAX-GA does, and can obtain solutions of acceptable quality within reasonable calculation time when solving the TSP instances with various scales like LKH does.

3.4.2. Impact of the Reinforcement Learning

As indicated by the results in [2], the $α$-value outperforms the distance in determining the candidate cities or evaluating the quality of the edges. As indicated by the results in [19], the Q-value is a better choice than the $α$-value. So why not replace the $α$-value metric used in LKH and the distance metric used in the sub-tours merging process in EAX-GA with our learned adaptive Q-value?

In the RHGA algorithm, the reinforcement learning is incorporated into both the local search process and the population optimization process in RHGA, by learning an adaptive $Q$-value to select and sort the candidate edges in both LKH and EAX-GA. Note that the initial candidate edges determined by the initial $Q$-value (Eq. 2) are better than the candidate edges determined by distance metric or $α$-value (see experimental results in Section 4). The reinforcement learning can further improve the quality of the candidate edges by updating the Q-value and adjusting the candidate sets. In particular, the experimental results demonstrate that the order of the performance of EAX-GA with different metrics with decaying quality is: adaptive Q-value (updated by Eq. 3), initial Q-value (Eq. 2), $α$-value, and finally the distance.

4. Experimental Results

This section presents the computational results and comparisons of RHGA, EAX-GA, LKH, and NeuroLKH [31]. The results show that RHGA significantly outperforms the other three algorithms. We first introduce the experimental setup, the benchmark instances and the baseline algorithms, then present the experimental results.

4.1. Experimental Setup

The experiments of RHGA were implemented in C++ and compiled by g++ with -O3 option. All the algorithms in the experiments were run on a server using an Intel® Xeon® E5-2650 v3 2.30 GHz 10-core CPU and 256 GB RAM, running Ubuntu 16.04 Linux operation system. The algorithms were all run on a single core. The parameters related to genetic algorithm in RHGA are set to be the same as the default settings in EAX-GA [3], i.e., $N_{pop} = 300$, $N_{pb} = 30$. Other parameters are set as follows: $λ = 0.1$, $γ = 0.9$, $M_{gen} = 10(\log_{10} n - 1)$ (i.e., $M_{gen} = 20/30/40$ when $n = 10^{3}/10^{4}/10^{5}$). To reduce the variance in the results, we run each algorithm in the experiments 10 times on each TSP instance.

4.2. Benchmark Instances

The RHGA algorithm was tested on all the TSP instances with the number of cities ranging from 1,000 to 85,900 cities, with a total of 138, in the well-known and widely used benchmark sets for the TSP: TSPLIB3, National TSP benchmarks4, and VLSI TSP benchmarks5. Note that the number in each instance’s name indicates the number of cities in that instance.

In order to make a clear comparison, we divide the 138 instances into small and large according to the instance scale. That is, an instance with less than 20,000 cities is considered to be small, otherwise large. There are a total

3http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95
4http://www.math.uwaterloo.ca/csp/world/countries.html
5http://www.math.uwaterloo.ca/csp/vlsi/index.html
of 111 small instances and 27 large instances among all the 138 tested instances.

Moreover, we further divide the 138 instances into the following three categories according to their difficulty:

- **Easy**: An instance is easy when both RHGA and EAX-GA (with the default settings) can obtain the best-known solution of this instance in each of the 10 runs (i.e., the worst solutions of RHGA and EAX-GA in 10 runs are all equal to the best-known solution when solving this instance). There are a total of 60 easy instances among all the 138 tested instances.

- **Medium**: An instance is medium when it satisfies the following two conditions: 1) the best solutions of RHGA and EAX-GA in 10 runs are all equal to the best-known solution of this instance. 2) At least one of the worst solutions of RHGA and EAX-GA is not equal to the best-known solution of this instance. There are a total of 62 medium instances among all the 138 tested instances.

- **Hard**: An instance is hard if at least one of the best solutions of RHGA and EAX-GA is not equal to the best-known solution of this instance. There are a total of 16 hard instances among all the 138 tested instances.

### 4.3. Baseline Algorithms

For baseline algorithms in the comparison, we choose two state-of-the-art heuristic algorithms, EAX-GA and LKH, as well as one of the state-of-the-art (deep) learning based algorithms, NeuroLKH [31].

For EAX-GA, we generate two baseline algorithms, one with the default parameters (i.e., $N_{pop} = 300$, $N_{ch} = 30$), so-called EAX-300, the other with a larger population size (i.e., $N_{pop} = 400$, $N_{ch} = 30$), so-called EAX-400. Note that the termination condition of RHGA, EAX-300, and EAX-400 are the same (see Section 2.2). We compare RHGA with EAX-400 since their run times for each tested instance are close and a little longer than the run time of EAX-300. Specifically, the average run time for the 138 tested instances of RHGA/EAX-400 is about 37.41%/38.44% longer than that of EAX-300. In order to compare the results of EAX-GA and RHGA within the same computation time, it is reasonable to increase the population size of EAX-GA, rather than the cut-off time, since the individuals in the population can hardly be improved after the population converges.

For the LKH algorithm, we use its newest version\(^6\) as the baseline algorithm. LKH terminates when the number of iterations reaches $n$ (the default termination condition in LKH) or the calculation time reaches the cut-off time. The cut-off time for LKH is set to be $n/2$ seconds for the instances with less than 70,000 cities, and $n$ seconds for the two super large instances ch71009 and pla85900.

For the NeuroLKH algorithm, we do the comparison with NeuroLKH$_R$, which is trained on instances with uniformly distributed nodes, and NeuroLKH$_M$, which is trained on a mixture of instances with uniformly distributed nodes, clustered nodes, half uniform and half clustered nodes. The resources required by NeuroLKH are numerous for large scale instances. The performance of NeuroLKH for large instances is also limited due to the small scale of the supervised training instances. Therefore, we only compare RHGA with NeuroLKH on instances with the number of cities ranging from 1,000 to 10,000. Note that in [31], they only reported results on instances with less than 6,000 cities. NeuroLKH terminates when the number of iterations reaches $n$ or the calculation time reaches $n/2$ seconds.

The results of the baseline algorithms are all obtained by running their source codes. All the algorithms will terminate their current run when they obtain the known optimum.

### 4.4. Comparing RHGA with NeuroLKH

We first compare RHGA with NeuroLKH$_R$ and NeuroLKH$_M$, in solving all the instances with the number of cities ranging from 1,000 to 10,000 and two-dimensional Euclidean distance (EUC$_{2D}$) metric (NeuroLKH only supports the EUC$_{2D}$ metric), a total of 92. We extract the instances that RHGA, NeuroLKH$_R$, and NeuroLKH$_M$ can always yield the optimal solution in each of the 10 runs. The results of the remaining 77 instances are shown in Table 1. We compare the best and average solutions in 10 runs obtained by the algorithms. Column $BKS$ indicates the best-known solution of the corresponding instance, and $Time$ is the average calculation time (in seconds) of the algorithms. The values in the brackets beside the results equal to the gap of the results to the best-known solutions multiplied by 100. We also provide the average gap of the best and average solutions to the best-known solutions.

From the results in Table 1, we can observe that:

1) RHGA significantly outperforms NeuroLKH$_R$ and NeuroLKH$_M$. RHGA can yield all the best-known solutions in 10 runs. The best solutions of RHGA are better than those of NeuroLKH$_R$ (NeuroLKH$_M$) in 42 (29) instances. The average solutions of RHGA are better than those of NeuroLKH$_R$ (NeuroLKH$_M$) on 72 (60) instances. The average gaps of the best solutions and average solutions of RHGA are much smaller than those of NeuroLKH$_R$ and NeuroLKH$_M$. The average calculation time of RHGA is also much smaller than that of NeuroLKH$_R$ and NeuroLKH$_M$.

2) The performance of NeuroLKH$_M$ is better than that of NeuroLKH$_R$, indicating that the performance of NeuroLKH relies on the structure of the training instances. Generating reasonable training instances that help NeuroLKH work well on instances with diverse structures is challenging. Moreover, both NeuroLKH$_R$ and NeuroLKH$_M$ are not good at solving large instances, indicating that the bottlenecks in large scale instances still limit algorithms based on deep neural networks.

\(^6\)http://akira.ruc.dk/\~{}keld/research/LKH/
4.5. Comparing RHGA with EAX-GA and LKH

| Instance | BKS | RHGA | Average | RHGA | Time | Best | Average | Time | Best | Average | Time |
|----------|-----|------|---------|------|------|------|---------|------|------|---------|------|
| 1060     | 22408 22408 | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) |
| 1061     | 22408 22408 | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) |
| 1062     | 22408 22408 | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) |
| 1063     | 22408 22408 | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) |
| 1064     | 22408 22408 | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) | 22408.0 (0.000) |

Reinforced Hybrid GA for the TSP

Zheng et al.:
Table 2: Comparison of RHGA and the baseline algorithms, EAX-300, EAX-400, and LKH, on 60 easy instances. Best results appear in bold.

| Instance | BKS | RHGA | Average (gap%) | Time | EAX-300 | Average (gap%) | Time | LKH | Average (gap%) | Time |
|----------|-----|------|----------------|------|---------|----------------|------|-----|----------------|------|
| pr1002   |    |      |                |      |         |                |      |     |                |      |
| pr1002   |    |      |                |      |         |                |      |     |                |      |
| u1060    |    |      |                |      |         |                |      |     |                |      |
| u1060    |    |      |                |      |         |                |      |     |                |      |
| v1084    |    |      |                |      |         |                |      |     |                |      |
| v1084    |    |      |                |      |         |                |      |     |                |      |
| pr1002   |    |      |                |      |         |                |      |     |                |      |
| pr1002   |    |      |                |      |         |                |      |     |                |      |
| v1084    |    |      |                |      |         |                |      |     |                |      |
| v1084    |    |      |                |      |         |                |      |     |                |      |
| pr1002   |    |      |                |      |         |                |      |     |                |      |
| pr1002   |    |      |                |      |         |                |      |     |                |      |
| v1084    |    |      |                |      |         |                |      |     |                |      |
| v1084    |    |      |                |      |         |                |      |     |                |      |
| pr1002   |    |      |                |      |         |                |      |     |                |      |
| pr1002   |    |      |                |      |         |                |      |     |                |      |
| v1084    |    |      |                |      |         |                |      |     |                |      |
| v1084    |    |      |                |      |         |                |      |     |                |      |
| pr1002   |    |      |                |      |         |                |      |     |                |      |
| pr1002   |    |      |                |      |         |                |      |     |                |      |
| v1084    |    |      |                |      |         |                |      |     |                |      |
| v1084    |    |      |                |      |         |                |      |     |                |      |
| pr1002   |    |      |                |      |         |                |      |     |                |      |
| pr1002   |    |      |                |      |         |                |      |     |                |      |
| v1084    |    |      |                |      |         |                |      |     |                |      |
| v1084    |    |      |                |      |         |                |      |     |                |      |
| pr1002   |    |      |                |      |         |                |      |     |                |      |
| pr1002   |    |      |                |      |         |                |      |     |                |      |
| v1084    |    |      |                |      |         |                |      |     |                |      |
| v1084    |    |      |                |      |         |                |      |     |                |      |
| pr1002   |    |      |                |      |         |                |      |     |                |      |
| pr1002   |    |      |                |      |         |                |      |     |                |      |
| v1084    |    |      |                |      |         |                |      |     |                |      |
| v1084    |    |      |                |      |         |                |      |     |                |      |

**Average**

| BKS | RHGA | Average (gap%) | Time | EAX-300 | Average (gap%) | Time | LKH | Average (gap%) | Time |
|-----|------|----------------|------|---------|----------------|------|-----|----------------|------|
|     |      |                |      |         |                |      |     |                |      |

**Average**
| Instance | BKS | RHGA | EAX-300 | EAX-400 | LKH |
|----------|-----|------|----------|----------|-----|
| 92650.1  | 0.0000 | 8.1   | 92650    | 0.0000   | 3.7 |
| 57209.1  | 0.0142 | 51.0  | 57216    | 0.0262   | 67.5 |
| 6197     | 0.0000 | 26.0  | 6764     | 0.0000   | 34.4 |
| 8258     | 0.0000 | 157.1 | 9772     | 0.0000   | 171.3 |
| 9272     | 0.0000 | 176.3 | 9272.5   | 0.0054   | 167.4 |
| 12503    | 0.0000 | 187.2 | 10959.3  | 0.0027   | 185.4 |
| 556072.3 | 0.0049  | 476.1 | 556045   | 0.0000   | 67.5 |
| 394721.6 | 0.0009  | 572.0 | 394721.4 | 0.0009   | 829.8 |
| 837479   | 0.0000 | 1285.9 | 837554.2 | 0.0090  | 1696.5 |
| 19537.1  | 0.0107  | 1283.5 | 114884.0 | 0.0252  | 2081.2 |
| 28387    | 0.0000  | 988.6  | 28387.3  | 0.0011  | 951.5 |
| 22338    | 0.0000  | 2242.3 | 22338    | 0.0000   | 3174.6 |
| 300900.8 | 0.0006  | 1362.7 | 300900.4 | 0.0005  | 1396.9 |
| 1061881.5| 0.0000  | 1958.3 | 1061881  | 0.0000   | 1958.3 |
| 2578     | 0.0000  | 988.6  | 2578     | 0.0000   | 988.6 |
| 427377   | 0.0000  | 4121.8 | 427382   | 0.0012  | 427399.5 |
| 557319.3 | 0.0008  | 3518.9 | 557317.6 | 0.0005  | 5965.3 |
| 66527    | 0.0000  | 0.0000 | 66527    | 0.0000   | 0.0000 |
| 69294.1  | 0.0001  | 3540.6 | 69335.7  | 0.0010  | 3990.1 |
| 78562    | 0.0000  | 8673.2 | 79623.2  | 0.0015  | 5828.6 |
| 99159.7  | 0.0007  | 14413.7| 99159    | 0.0000   | 0.0000 |
| 125183.0 | 0.0000  | 0.0000 | 125183   | 0.0000   | 0.0000 |

**Average**
Table 4: Comparison of RHGA and the baseline algorithms, EAX-300, EAX-400, and LKH, on 16 hard instances. Best results appear in bold.

| Instance | RHGA | EAX-300 | EAX-400 | LKH |
|----------|------|---------|---------|-----|
| u2319    | 234256 (0.0000) | 234256 (0.0000) | 234318.8 (0.0268) | 136.9 |
| fl3795   | 28777.6 (0.0195) | 28815 (0.1495) | 28824.0 (0.1807) | 140.6 |
| pm98079  | 28772 (0.0000) | 28777.6 (0.0195) | 428.3 | 28815 (0.1495) |
| ar9152   | 28779 (0.0243) | 28821.5 (0.1720) | 202.9 | 28906.5 (0.4675) |
| pm8079   | 28779 (0.0243) | 28821.5 (0.1720) | 202.9 | 28906.5 (0.4675) |
| pla7397  | 23260728 | 569288 (0.0000) | 23260728 | 569288 (0.0000) |
| fl1577   | 23260728 | 569288 (0.0000) | 23260728 | 569288 (0.0000) |
| pla33810 | 66050069 (0.0017) | 66050888.1 (0.0029) | 13383.5 | 66054829.0 (0.0089) |
| u38478   | 108319.8 (0.0017) | 108321.1 (0.0029) | 10023.5 | 108337.0 (0.0175) |
| ics39603 | 106820 (0.0009) | 106822.2 (0.0030) | 10214.3 | 106834.2 (0.0142) |
| fl47608  | 125105 (0.0008) | 125108.5 (0.0036) | 20059.4 | 125128 (0.0192) |
| fna52057 | 147790 (0.0007) | 147792.9 (0.0026) | 22594.7 | 147800 (0.0074) |
| bna56769 | 158080.4 (0.0015) | 158080.6 (0.0016) | 25839.3 | 158080.6 (0.0016) |
| dan59296 | 165372 (0.0006) | 165373.3 (0.0014) | 41479.0 | 165373 (0.0012) |
| ch71009  | 4566508 (0.0000) | 4566508.7 (0.0001) | 205363.6 | 4566508 (0.0000) |
| pla85900 | 142384855 (0.0016) | 142386004.9 (0.0024) | 70301.9 | 142398897.0 (0.0114) |

Reinforced Hybrid GA for the TSP

(1) On all the 60 easy instances, RHGA, EAX-300 and EAX-400 can easily yield the optimal solution in almost each of the 10 runs.

(2) On all the 62 medium instances, RHGA exhibits better stability and robustness than EAX-300 and EAX-400, such as in solving the instances u1432, fl1577, pm98079, and ar9152. The average gap of the average solutions of RHGA is 84.5% (85.1%) less than that of EAX-300 (EAX-400).

(3) On all the 16 hard instances, RHGA greatly outperforms EAX-300 and EAX-400. Specifically, the best solutions, the average solutions, and the worst solutions of RHGA are all better than those of EAX-300. The best solutions of RHGA are better than those of EAX-400 in 9 hard instances, and worse than those of EAX-400 in 2 hard instances. The average gap of the best solutions of RHGA is 96.4% (84.6%) less than that of EAX-300 (EAX-400), and the average gap of the average solutions of RHGA is 84.0% (82.4%) less than that of EAX-300 (EAX-400), indicating a significant improvement.

(4) The calculating time of RHGA and EAX-400 is close, indicating that RHGA can yield better performance than EAX-GA within the same parameters (compared to EAX-300) or similar calculation time (compared to EAX-400).

(5) The LKH is weaker than the other three algorithms in solving most of the tested instances. However, in solving the instances such as u2319 and pla7397, LKH shows significantly better performance. Thus EAX-GA and LKH are complementary in solving different TSP instances, and our combination can make full use of their advantages and boost their performance.

To make a clearer comparison of RHGA and EAX-GA, we apply the cumulative metrics including cumulative gap on the solution quality and cumulative run time to compare RHGA with EAX-300 and EAX-400 in solving all the 51 small but not easy instances and all the 27 large instances. Let \( gap(j) = \frac{1}{10} \sum_{i=1}^{10} \frac{A_j - BKS_j}{BKS_j} \) be the average gap of calculating the \( j \)-th instance by an algorithm in 10 runs, where \( A_j \) is the result of the \( i \)-th calculation and \( BKS_j \) is the best-known solution of the \( j \)-th instance. The smaller the average gap, the closer the average solution is to the best-known solution. For an algorithm, \( C_{gap}(j) = \sum_{i=1}^{10} gap(i) \) is the cumulative gap. The cumulative run time can be calculated similarly. The comparison results are shown in Figure 1.

The results indicate again that the robustness and stability of EAX-GA are not good. It shows much worse performance than RHGA in solving some instances, such as fl1577, u2319, fl3795, pla33810, and pla85900. The increase of population size from 300 to 400 can not help EAX-GA escape from the local optima when solving these instances, while the proposed methods including the hybrid mechanism and reinforcement learning could. We can also observe that the calculation time of RHGA is close to EAX-400 and a little bit longer than that of EAX-300. The results clearly show again that RHGA significantly outperforms EAX-GA within the same parameters or similar calculation time.
5. Further Analysis

This section provides insight on why and how the proposed RHGA is effective, suggesting that the creative combination of EAX-GA and LKH can boost the performance mutually. The results further indicate that the adaptive Q-value learned by reinforcement learning is a better metric for determining the candidate cities and evaluating the edge quality than the $\alpha$-value in LKH as well as the distance in EAX-GA. We first introduce various variant algorithms involved in the experiments, then present and analyze the results.

5.1. Various Variants of the Algorithm

We first present various variant algorithms of RHGA and EAX-GA for comparison and analysis. The variant algorithms include the following:

- **Alpha-EAX**: A variant of EAX-300 using the $\alpha$-value (Eq. 1) to replace the distance metric used in the process of merging sub-tours in EAX-300 (see Step 5 in Section 2.2).

- **FixQ-EAX**: A variant of EAX-300 using the initial Q-value (Eq. 2) to replace the distance metric used in the process of merging sub-tours in EAX-300.

- **Q-EAX**: A variant of EAX-300 using the adaptive Q-value learned by the Q-learning method (Eq. 3) to replace the distance metric used in the process of merging sub-tours in EAX-300. The population size in Q-EAX is set to be 301. The extra individual is the special individual used to learn the Q-value and provide the adaptive Q-value for the algorithm. The special individual in Q-EAX can be improved by the Q-LKH local search and the genetic algorithm, but other individuals cannot be improved by crossing with the special individual (i.e., the special individual in Q-EAX can only be $p_A$, not $p_B$). In addition, the result of Q-EAX is the best individual in the population except for the special individual, and Q-EAX will not terminate if the special individual is the optimum solution (if known).
• **Q-EAX+Special**: A variant of Q-EAX that the output is the best individual in the population includes the special individual. Q-EAX+Special terminates when the special individual is the optimum solution. In a word, the difference between Q-EAX+Special and Q-EAX includes: 1) whether the special individual can be the output solution. 2) Whether the algorithm terminates when the special individual is the optimum solution.

• **EAX-LKH**: A variant of RHGA that combines EAX-GA with the LKH algorithm as RHGA does but with no reinforcement learning. The metrics in the EAX-GA and LKH local search are not changed.

• **Alpha-EAX-LKH**: A variant of EAX-LKH using the $\alpha$-value (Eq. 1) to replace the distance metric in EAX-GA.

• **FixQ-EAX-LKH**: A variant of EAX-LKH using the initial Q-value (Eq. 2) to replace the distance metric in EAX-GA and the $\alpha$-value metric in LKH.

• **RHGA-k**: A variant of RHGA that the number of the special individuals is $k$ (we tested $k = 3, 5, 20, 50$ in experiments). Each special individual $x_i$, $i \leq k$, can be improved by Q-LKH when: 1) it is just initialized, or it was improved by EAX-GA at the last generation. In this case, Q-LKH will try to improve $x_i$. 2) It has not been improved for $M_{gen}$ generations, and the algorithm randomly selects an individual $x_i$ ($x_r$ is not a special individual). If $x_i$ can be improved by Q-LKH and the improved tour is better than $x_r$, then replace $x_r$ with the improved tour. Moreover, when the best individual $x_{best}$ in the population besides the $k$ special individuals is better than the best special individual and $x_{best}$ has not been calculated by Q-LKH, Q-LKH will try to improve $x_{best}$. The improved tour will replace $x_1$ (do not replace $x_1$ if $x_{best}$ cannot be improved by Q-LKH).

The termination conditions of all the above algorithms are the same as in RHGA. Since the calculation time of RHGA is a little longer than EAX-300 and close to EAX-400 when solving the tested benchmarks, the calculation times of the variant algorithms, including Alpha-EAX, FixQ-EAX, Q-EAX, Q-EAX+Special, EAX-LKH, Alpha-EAX-LKH, FixQ-EAX-LKH, are also roughly between that of EAX-300 and EAX-400.

We further introduce three hybrid algorithms that combine EAX-GA with LKH in a straightforward manner:

• **Min[EAX, LKH]**: Given an input instance, this algorithm first uses EAX-GA and LKH to calculate the instance 10 times respectively, and then selects the one with the better average solution as the output.

• **LKH+EAX**: A hybrid algorithm that uses LKH to generate the initial population, and then runs EAX-GA starting from this population.

• **EAX+LKH**: A hybrid algorithm that first uses EAX-GA to calculate the input instance until the termination conditions of EAX-GA are reached (or the known optimum is obtained), then runs LKH with its initial solution equal to the best individual. The LKH algorithm in EAX+LKH terminates as EAX-GA does. That is, let $Gen$ be the number of iterations at which no improvement in the best solution is found by LKH over the recent 1500/$N_{ch}$ iterations. If $Gen$ has been determined and the best solution does not improve over the last $G_{max} = Gen/10$ iterations, EAX+LKH terminates.

5.2. **Analysis on the Combination Mechanism of RHGA**

In order to demonstrate that our proposed combination mechanism is reasonable and effective, we first compare EAX-LKH with the hybrid algorithms Min[EAX, LKH], LKH+EAX, and EAX+LKH, as well as the baselines EAX-300 and EAX-400 on all the 51 small but not easy instances in Figure 2(a). We also present the results without LKH+EAX in Figure 2(b) for a clearer comparison.

As shown in Figure 2, LKH+EAX is much worse than EAX-300 and EAX-400. Note that the method to generate the initial population in EAX-GA is a simple greedy 2-opt local search [3]. Why does LKH+EAX use the effective LKH local search to replace the simple 2-opt results in much worse performance? The reason is that the initial population of LKH+EAX contains too many candidate edges provided by LKH. Thus the population diversity is broken and it is easy for the algorithm to get stuck in local optima. In this case, if LKH can provide good genes, i.e., the edges in the optimal solution are contained in the candidate edges, LKH+EAX can obtain better results than EAX-GA. Otherwise, its performance is poor. The results in Figure 2 can demonstrate this comment, as the performance of LKH+EAX mainly depends on LKH. For the instances that LKH works well (see detailed results of LKH in Tables 2, 3, and 4), such as $u1432$ and $u2119$, LKH+EAX shows better performance than EAX-300 and EAX-400. For the instances that LKH can not work well, such as $fl3795$, $eg7146$, and $hol4472$, LKH+EAX shows much worse performance than EAX-300 and EAX-400. This result can also demonstrate that our mechanism that only one special individual can be improved by LKH is reasonable.

Moreover, algorithms Min[EAX, LKH] and EAX+LKH can surely obtain results no worse than EAX-GA, because they can obtain at least the same solution as EAX-GA does. However, the straightforward combinations of EAX-GA and LKH can not make full use of their complementary, as the EAX-LKH algorithm with our combination mechanism shows much better performance than Min[EAX, LKH] and EAX+LKH. Note that Min[EAX, LKH] can be regarded as the hybrid solver proposed by Kerschke et al. [15] consists of EAX-GA and LKH with a perfect prediction model. Therefore, the combination mechanism in RHGA is much more effective than the straightforward hybrid mechanisms we designed.
Reinforced Hybrid GA for the TSP

Figure 2: Analysis on the combination mechanism on 51 small but not easy instances.

Figure 3: Analysis on different metrics and the special individual on 51 small but not easy instances.

5.3. Analysis on Different Metrics and the Special Individual

We then compare RHGA with the variant algorithms to evaluate the performance of different metrics in determining the candidate edges, and evaluate the effectiveness of the special individual in RHGA. Figure 3(a) compares the results of 10 different algorithms, including RHGA, Alpha-EAX, FixQ-EAX, Q-EAX, Q-EAX+Special, EAX-LKH, Alpha-EAX-LKH, FixQ-EAX-LKH, EAX-300, and EAX-400, on the same 51 instances in Figure 2. We also present the results without EAX-300 and EAX-400 in Figure 3(b) for a clearer comparison.

The results of EAX-LKH, Alpha-EAX-LKH, FixQ-EAX-LKH, and RHGA in Figure 3 indicate that, the order with decaying quality of the performance of different metrics is: adaptive Q-value (updated by reinforcement learning according to Eq. 3), initial Q-value (Eq. 2), α-value and distance. The results of EAX-300, Alpha-EAX, FixQ-EAX, and Q-EAX can draw the same conclusion. As a result, the EAX-GA algorithm can be improved simply by replacing the distance metric with the α-value or initial Q-value, and reinforcement learning can further improve the performance of the initial Q-value. Moreover, the combination of EAX-GA and LKH by our proposed mechanism is always effective, since EAX-LKH/Alpha-EAX-LKH/FixQ-EAX-LKH/RHGA outperforms EAX-300/Alpha-EAX/FixQ-EAX/Q-EAX significantly.

The results of Q-EAX, Q-EAX+Special, and RHGA in Figure 3 can further lead to the following comments. First, Q-EAX+Special outperforms Q-EAX, indicating that the special individual can provide better solutions than the other
individuals. Second, RHGA outperforms Q-EAX+Special, indicating that the special individual can also improve the population by spreading its genes to other individuals. In summary, the benefits of the special individual are as follows: 1) it can provide the adaptive Q-value for the EAX-GA to improve the performance. 2) it can obtain good solutions since it can be improved by both Q-LKH and EAX-GA (when \( x_1 \) is parent \( p_A \)). 3) it can improve the population by providing its high-quality genes for other individuals (when \( x_1 \) is parent \( p_B \)).

5.4. Comparison with Multiple Special Individuals

We then compare RHGA with the RHGA-\( k \) algorithms, including RHGA-3, RHGA-5, RHGA-20, and RHGA-50, on the same 51 instances in Figure 2, to analyze the influence of the number of special individuals on the performance. The comparison results are shown in Figure 4.

From the results we can observe that:

1) RHGA and RHGA-3 show similar performance in solving the 51 instances. The algorithms with a larger number of special individuals show better performance in solving the instances with relatively small scales. For example, the RHGA-5 is good at solving the instances with less than 3,000 cities, and the RHGA-20 and RHGA-50 are good at solving the instances with less than 6,000 cities. However, neither of the three algorithms with a larger number of special individuals is good at solving the instances with larger than 14,000 cities, indicating that for solving relatively large and complex instances, an excessive proportion of special individuals in the population may reduce the algorithm performance.

2) The calculation time of the RHGA-\( k \) algorithm increases rapidly as \( k \) increases. Specifically, the calculation time of RHGA-3/5/20/50 is about 24/51/256/687% more than that of RHGA.

In summary, setting multiple special individuals in the population is not reasonable and time-consuming. Thus it is suitable and effective to set only one special individual in the population, which can help the algorithm obtain a good performance without reducing the efficiency.

6. Conclusion

In this paper, we address the famous NP-hard traveling salesman problem and propose a reinforced hybrid genetic algorithm (RHGA) that combines reinforcement learning with two state-of-the-art heuristic algorithms, the EAX-GA genetic algorithm and the LKH local search heuristic, in a more interactive form. The EAX-GA and LKH are integrated with the help of a special and unique individual, which can be improved by both the genetic algorithm and the local search algorithm. In our proposed hybrid mechanism, the population provides diverse and high-quality initial solutions for the LKH local search algorithm, and the local search algorithm leads the population to converge to better results. In a word, the two state-of-the-art TSP heuristics, EAX-GA and LKH, can boost each other in our proposed hybrid mechanism. Moreover, the Q-learning algorithm is applied to learn an adaptive Q-value to replace the distance metric used in the process of merging sub-tours in EAX-GA and the \( \sigma \)-value used in LKH for determining the candidate cities. As a result, our reinforcement learning method can improve both the EAX-GA and the LKH algorithms by providing better candidate edges.

Extensive experimental results demonstrate that RHGA outperforms the powerful EAX-GA and LKH algorithms, as well as one of the state-of-the-art (deep) learning based algorithms, NeuroLKH, for solving the TSP. Further and extensive ablation studies are adopted to show the effectiveness of the proposed hybrid mechanism and the reinforcement learning method, and to demonstrate that the proposed hybrid mechanism can make full use of the complementary of EAX-GA and LKH, and the setting of only one special individual is reasonable and efficient.

In future work, the proposed mechanism of combining genetic algorithms with local search could be applied to solve various combinatorial optimization problems, and the method of combining reinforcement learning with the core process of heuristics would also be applied to improve other heuristic algorithms.
References

[1] Elias B. Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dimitkova, and Le Song. Learning combinatorial optimization algorithms over graphs. In Annual Conference on Neural Information Processing Systems 2017, NIPS 2017, pages 6348–6358, 2017.

[2] Keld Helsgaun. An effective implementation of the lin-kernighan traveling salesman heuristic. European Journal of Operational Research, 126(1):106–130, 2000.

[3] Yuichi Nagata and Shigenobu Kobayashi. A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. INFORMS Journal on Computing, 25(2):346–363, 2013.

[4] Shen Lin and Brian W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. Operations Research, 21(2):498–516, 1973.

[5] Huai-Kuang Tsai, Jinn-Moon Yang, Yuan-Fang Tsai, and Cheng-Yan Kao. An evolutionary algorithm for large traveling salesmen problems. IEEE Transactions on Systems, Man, and Cybernetics, Part B, 34(4):1718–1729, 2004.

[6] Yuichi Nagata and Shigenobu Kobayashi. Edge assembly crossover: A high-power genetic algorithm for the travelling salesman problem. In The Seventh International Conference on Genetic Algorithms, ICGA 1997, pages 450–457, 1997.

[7] Neco LJ Ulider, Emilie HL Aarts, Hans-Jürgen Bandelt, Peter JM Van Laarhoven, and Erwin Pesch. Genetic local search algorithms for the traveling salesman problem. In International Conference on Parallel Problem Solving from Nature, PPSN 1990, volume 496, pages 109–116, 1990.

[8] Bernd Freisleben and Peter Merz. New genetic local search operators for the traveling salesman problem. In Parallel Problem Solving from Nature - PPSN IV, International Conference on Evolutionary Computation. The Fourth International Conference on Parallel Problem Solving from Nature, PPSN 1996, volume 1141, pages 890–899, 1996.

[9] Peter Merz and Bernd Freisleben. Genetic local search for the tsp: new results. In IEEE International Conference on Evolutionary Computation, ICEC 1997, pages 159–164, 1997.

[10] Hung Dinh Nguyen, Ikao Yoshihara, Kunihito Yamamori, and Moritoshi Yasunaga. Implementation of an effective hybrid GA for large-scale traveling salesman problems. IEEE Transactions on Systems, Man, and Cybernetics, Part B, 37(1):92–99, 2007.

[11] L. Darrell Whitley, Doug Hains, and Adele E. Howe. A hybrid genetic algorithm for the travelling salesman problem using generalized partition crossover. In Parallel Problem Solving from Nature - PPSN XI, Eleventh International Conference, Part I, PPSN I 2010, volume 6238, pages 566–575, 2010.

[12] Yong Wang. The hybrid genetic algorithm with two local optimization strategies for traveling salesman problem. Computers & Industrial Engineering, 70:124–133, 2014.

[13] Mohammad Harun Rashid and Miguel A. Mosteiro. A greedy-genetic local-search heuristic for the traveling salesman problem. In 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications, ISPA/UCC 2017, pages 868–872, 2017.

[14] Vladimir Ilin, Dragan Simic, Svetislav D. Simic, and Svetlana Simic. Hybrid genetic algorithms and tour construction and improvement algorithms used for optimizing the traveling salesman problem. In Fifteenth International Conference on Soft Computing Models in Industrial and Environmental Applications, SOCO 2020, volume 1268, pages 530–539, 2020.

[15] Pascal Kerschke, Lars Kotthoff, Jakob Bossek, Holger H. Hoos, and Heike Trautmann. Leveraging TSP solver complementarity through machine learning. Evolutionary Computation, 26(4), 2018.

[16] Richard S. Sutton and Andrew G. Barto. Reinforcement learning - an introduction. Adaptive computation and machine learning. MIT Press, 1998.

[17] Yujiao Hu, Yuan Yao, and Wee Sun Lee. A reinforcement learning approach for optimizing multiple traveling salesmen problems over graphs. Knowledge-Based Systems, 204:106244, 2020.

[18] Ajani S. Oladayo and Rammohan Mallipeddi. Adaptive evolution strategy with ensemble of mutations for reinforcement learning. Knowledge-Based Systems, 245:108624, 2022.

[19] Jiongzheng Zheng, Kun He, Jianrong Zhou, Yan Jin, and Chu-min Li. Combining reinforcement learning with lin-kernighan-helsgaun algorithm for the traveling salesman problem. In Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, pages 12445–12452, 2021.

[20] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d’horizon. European Journal of Operational Research, 290(2):405–421, 2021.

[21] Irwan Bello, Hiru Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In Fifth International Conference on Learning Representations, Workshop Track Proceedings, ICLR (Workshop) 2017, 2017.

[22] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In Annual Conference on Neural Information Processing Systems 2015, NIPS 2015, pages 2692–2700, 2015.

[23] Yong Liang Goh, Wee Sun Lee, Xavier Bresson, Thomas Laurent, and Nicholas Lim. Combining reinforcement learning and optimal transport for the traveling salesman problem. CoRR, abs/2203.00093, 2022.

[24] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In Annual Conference on Neural Information Processing Systems 2013., pages 2292–2300, 2013.

[25] Patrick Emami and Sanjay Ranka. Learning permutations with sinkhorn policy gradient. CoRR, abs/1805.07010, 2018.

[26] Fei Liu and Guangzhou Zeng. Study of genetic algorithm with reinforcement learning to solve the TSP. Expert Systems with Applications, 36(3):6995–7001, 2009.

[27] Yuichi Nagata. New EAX crossover for large TSP instances. In Parallel Problem Solving from Nature - PPSN IX, Ninth International Conference, PPSN 2006, volume 4193, pages 372–381, 2006.

[28] Paulo R. de O. da Costa, Jason Rhuggenaath, Yingqian Zhang, and Alk Alyak. Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. In The Twelfth Asian Conference on Machine Learning, ACML 2020, volume 129, pages 465–480, 2020.

[29] Jingyan Sui, Shizhe Ding, Ruizhi Liu, Liming Xu, and Dongbo Bu. Learning 3-opt heuristics for traveling salesman problem via deep reinforcement learning. In The Thirteenth Asian Conference on Machine Learning, ACML 2021, volume 157, pages 1301–1316, 2021.

[30] Jiuxia Zhao, Minjia Mao, Xi Zhao, and Jianhua Zou. A hybrid of deep reinforcement learning and local search for the vehicle routing problems. IEEE Transactions on Intelligent Transportation Systems, 22(11):7208–7218, 2021.

[31] Liangxing Goh, Wee Sun Lee, Xavier Bresson, Thomas Laurent, and Nicholas Lim. Combining reinforcement learning and optimal transport for the traveling salesman problem. CoRR, abs/2203.00093, 2022.

[32] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In Annual Conference on Neural Information Processing Systems 2013., pages 2292–2300, 2013.

[33] Patrick Emami and Sanjay Ranka. Learning permutations with sinkhorn policy gradient. CoRR, abs/1805.07010, 2018.

[34] Fei Liu and Guangzhou Zeng. Study of genetic algorithm with reinforcement learning to solve the TSP. Expert Systems with Applications, 36(3):6995–7001, 2009.

[35] Yuichi Nagata. New EAX crossover for large TSP instances. In Parallel Problem Solving from Nature - PPSN IX, Ninth International Conference, PPSN 2006, volume 4193, pages 372–381, 2006.

[36] Paulo R. de O. da Costa, Jason Rhuggenaath, Yingqian Zhang, and Alk Alyak. Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. In The Twelfth Asian Conference on Machine Learning, ACML 2020, volume 129, pages 465–480, 2020.

[37] Jingyan Sui, Shizhe Ding, Ruizhi Liu, Liming Xu, and Dongbo Bu. Learning 3-opt heuristics for traveling salesman problem via deep reinforcement learning. In The Thirteenth Asian Conference on Machine Learning, ACML 2021, volume 157, pages 1301–1316, 2021.

[38] Jiuxia Zhao, Minjia Mao, Xi Zhao, and Jianhua Zou. A hybrid of deep reinforcement learning and local search for the vehicle routing problems. IEEE Transactions on Intelligent Transportation Systems, 22(11):7208–7218, 2021.

[39] Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Neurolkh: Combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem. In Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, pages 7472–7483, 2021.

[40] Keiji Maekawa, Naoki Mori, Hisashi Tamaki, Hajime Kita, and Yoshikazu Nishikawa. A genetic solution for the traveling salesman problem by means of a thermodynamical selection rule. In IEEE International Conference on Evolutionary Computation, ICEC 1996, pages 529–534, 1996.

[41] Shen Lin. Computer solutions of the traveling salesman problem. Bell System Technical Journal, 44(10):2245–2269, 1965.

[42] Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees. Operations Research, 18(6):1138–1162, 1970.

[43] Keld Helsgaun. General 4-opt submoves for the lin-kernighan TSP heuristic. Mathematical Programming Computation, 1(2-3):119–163, 2009.