Article

A Discrete JAYA Algorithm Based on Reinforcement Learning and Simulated Annealing for the Traveling Salesman Problem

Jun Xu [1], Wei Hu [1], Wenjuan Gu [2,*] and Yongguang Yu [3]

[1] School of Systems Science, Beijing Jiaotong University, Beijing 100044, China; 21120754@bjtu.edu.cn (J.X.); huwei@bjtu.edu.cn (W.H.)
[2] School of Modern Post, Beijing University of Posts and Telecommunications, Beijing 100876, China
[3] School of Mathematics and Statistics, Beijing Jiaotong University, Beijing 100044, China; ygyu@bjtu.edu.cn
* Correspondence: wenjuan.gu@bupt.edu.cn

Abstract: The JAYA algorithm is a population-based meta-heuristic algorithm proposed in recent years which has been proved to be suitable for solving global optimization and engineering optimization problems because of its simplicity, easy implementation, and guiding characteristic of striving for the best and avoiding the worst. In this study, an improved discrete JAYA algorithm based on reinforcement learning and simulated annealing (QSA-DJAYA) is proposed to solve the well-known traveling salesman problem in combinatorial optimization. More specially, firstly, the basic Q-learning algorithm in reinforcement learning is embedded into the proposed algorithm such that it can choose the most promising transformation operator for the current state to update the solution. Secondly, in order to balance the exploration and exploitation capabilities of the QSA-DJAYA algorithm, the Metropolis acceptance criterion of the simulated annealing algorithm is introduced to determine whether to accept candidate solutions. Thirdly, 3-opt is applied to the best solution of the current iteration at a certain frequency to improve the efficiency of the algorithm. Finally, to evaluate the performance of the QSA-DJAYA algorithm, it has been tested on 21 benchmark datasets taken from TSPLIB and compared with other competitive algorithms in two groups of comparative experiments. The experimental and the statistical significance test results show that the QSA-DJAYA algorithm achieves significantly better results in most instances.

Keywords: JAYA algorithm; traveling salesman problem; population-based meta-heuristics; reinforcement learning; metropolis acceptance criterion

MSC: 90-08; 68R01; 68W50; 90C06; 90C10; 90C27

check for updates

Citation: Xu, J.; Hu, W.; Gu, W.; Yu, Y. A Discrete JAYA Algorithm Based on Reinforcement Learning and Simulated Annealing for the Traveling Salesman Problem. *Mathematics* **2023**, *11*, 3221. https://doi.org/10.3390/math11143221

Academic Editors: Takfarinas Saber and Aman Singh

Received: 13 June 2023
Revised: 19 July 2023
Accepted: 19 July 2023
Published: 22 July 2023

1. Introduction

In the real world, the essence of a large number of complex problems are combinatorial optimization problems. The traveling salesman problem (TSP) is considered to be one of the most common problems in the field of combinatorial optimization, especially in logistics transportation and distribution. In this problem, the traveler starts from one city, passes through all the cities, and finally returns to the starting city. Since Dantzig and Ramser [1] proposed this problem in 1959, it has attracted increasing attention. However, the TSP is an NP-hard problem [2,3], that is, there is no polynomial time algorithm to solve this problem. Solving such problems is still full of challenge. It should be noted that other combinatorial optimization problems, such as knapsack problems, assignment problems, job-shop scheduling problems, and so on, are also NP-hard problems similar to the TSP. If the TSP can be solved efficiently, it will also provide promising solutions for other similar problems.

Generally speaking, most exact approaches to the TSP are based on the linear programming algorithm, the branch-and-bound algorithm, and the dynamic programming algorithm [4]. The basic idea of the exact algorithms is to find the optimal solution by traversing the entire solution space, which determines the high time complexity of the above exact algorithms. Therefore, this kind of exact algorithms can only effectively solve small-scale TSPs, but it is difficult to solve the medium and large-scale TSPs. The explosion of the solution space caused by the increase in the scale is the most severe problem in the process of solving the TSP. Traditional exact algorithms have prominent disadvantages in the face of this difficulty, making it difficult to effectively solve this optimization problem [4]. To overcome this difficulty, researchers have designed many meta-heuristic algorithms inspired by the laws of physical change and biological systems in nature, such as Genetic Algorithm (GA) [5], Cuckoo Search (CS) [6], Particle Swarm Optimization (PSO) [7–9], Bat Algorithm (BA) [2], Simulated Annealing (SA) [10], Ant Colony Optimization algorithm (ACO) [8,11–14], Frog-Leaping Algorithm (FLA) [15], and Artificial Bee Colony (ABC) [16–20]. Those kinds of meta-heuristic algorithms are simple in principle, are flexible in mechanism, and easily find the approximate optimal solution in a short time.

The JAYA algorithm is a meta-heuristic algorithm, proposed by Rao in 2016, which is used to solve constrained and unconstrained continuous optimization problems [21]. It has been applied to solve different kinds of problems, such as flexible job-shop scheduling [22], text clustering [23], solid oxide fuel cell parameter optimization [24], feature selection [25], hydropower reservoir operation optimization [26], etc. Mesut Gunduz and Murat Aslan [27] discretized the JAYA algorithm by modifying the encoding mode and updating mechanism, and applied it to the TSP for the first time. Although the results on the selected cases are good in terms of convergence speed and solution quality, it is still inferior to the Discrete Tree-Seed Algorithm (DTSA) [28] used for comparison. Fortunately, the traditional JAYA algorithm can be improved by combining it with other meta-heuristic algorithms and introducing some advanced ideas such as reinforcement learning to mitigate the problems of instability and easily falling into local optima.

In this paper, an improved discrete JAYA algorithm based on reinforcement learning and SA, viz., the QSA-DJAYA algorithm, is proposed to solve the TSP. In the QSA-DJAYA algorithm, six transformation operators are used for producing the candidate solutions. Unlike the DJAYA algorithm of [27], which used roulette wheel selection to select transformation operators, the proposed QSA-DJAYA algorithm utilizes the Q-learning algorithm in reinforcement learning to choose the most promising transformation operator. Then, inspired by SA, we introduce the Metropolis acceptance criterion to determine whether to accept the candidate solution. In addition, 3-opt is applied to the best solution of the current iteration at a certain frequency to further improve the efficiency of this algorithm. To evaluate the performance of the proposed approach, it was compared with four representative algorithms developed by ourselves and eight efficient methods the from literature on 21 instances from TSPLIB [29]. The experimental results and the statistical significance test show the effectiveness of the proposed QSA-DJAYA algorithm. The main contributions are as follows:

- A novel improved discrete JAYA algorithm for the TSP is designed.
- The Q-learning algorithm in reinforcement learning is embedded to adaptively select the promising transformation operator.
- The Metropolis acceptance criterion of SA is introduced to help jump out of the local optima.
- Compared with four typical algorithms developed by ourselves and eight efficient methods from the literature, the proposed algorithm displays great superiority in solving the TSP.

The remainder of this paper is organized as follows: Section 2 describes the related work. In Section 3, the proposed QSA-DJAYA algorithm is introduced. Section 4 discusses the experimental results of the proposed algorithm and other advanced competing algorithms on the TSP benchmark datasets. Section 5 draws conclusions and proposes future work.

2. Related Work on TSP and JAYA Algorithm

2.1. Research on the Meta-Heuristic Algorithms of TSP

2.1.1. The Traveling Salesman Problem

The aim of the TSP is to look for a trip that ensures the salesman visits each city exactly once and returns to the starting with the minimum total travel distance. The TSP can be defined as a complete weighted graph $G(V,E)$, where $V = \{1, 2, \cdots, n\}$ is the set of vertices and E is the set of edges. Mathematically, the model of TSP can be formulated as follows:

$$\text{Min } C = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}, \tag{1}$$

subject to:

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i \in V, \tag{2}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j \in V, \tag{3}$$

$$\sum_{i \in S} \sum_{j \in \bar{S}} x_{ij} \geqslant 1 \qquad \forall S \subset V, S \neq \varnothing, \tag{4}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V, \tag{5}$$

where n is the number of cities to be visited, and d_{ij} is the distance between city i and city j. Equation (1) is the objective function that minimizes the total distance traveled. Equations (2) and (3) ensure that each city is visited exactly once. Equation (4) is the subtour elimination constraint. And, in Equation (5), x_{ij} is a binary variable, and it denotes whether the arc from i to j is selected by the salesman.

2.1.2. Related Work on TSP

For the TSP, as the size of the problem increases, the number of solutions increases exponentially. And, the exact algorithm cannot give the optimal or approximate optimal solution within a reasonable time. Inspired by natural phenomena, especially the collective behavior of social animals, researchers begin to study various meta-heuristic algorithms to solve the TSP efficiently. Some meta-heuristic algorithms are proposed earlier and studied intensively, such as GA, ACO, PSO, ABC, and so on. Those kinds of meta-heuristic algorithms are further adapted to improve their performance. Ebadinezhad [11] proposed an adaptive ACO that can dynamically adjust the parameters in order to overcome the disadvantages of low convergence speed and easily falling into local optima. The mechanism selected the starting point based on clustering to realize the shortest path. Zhong et al. [16] proposed a hybrid discrete ABC with threshold acceptance criteria, in which employed and onlooker bees decided whether to accept the newly generated solutions according to the threshold acceptance criteria. This non-greedy acceptance strategy maintained the population diversity. Choong et al. [17] used the modified selection function (MCF) to automatically adjust the selection of neighborhood search heuristics adopted by the employed bees and the onlooker bees and improved the model performance by combining the Lin–Kernighan local search strategy. Khan and Maiti [18] improved the ABC and adopted multiple updating rules and the K-opt operation to solve the TSP. Karaboga and Gorkemli [19] also proposed two new versions of the ABC. One was a combined version of the standard ABC, and the other is a further refinement of the combined version.

In addition to the abovementioned meta-heuristic algorithms, a large number of other meta-heuristic algorithms have been proposed in recent years. For example, Hatamlou [30] studied the application of the Black Hole algorithm (BH) in solving the TSP, and the experiments showed that the algorithm can find a solution with better quality in a shorter time than the classical GA, ACO, and PSO. Similarly, Zhang and Han [31] proposed a discrete sparrow search algorithm (DSSA) with a global perturbation strategy to solve the TSP. Zheng et al. [32] solved a variant of the TSP, namely the multiple traveling salesmen problem (mTSP). For this problem, they proposed an iterated two-stage heuristic algorithm called ITSHA, whose first stage was an initialization phase aimed at generating high quality and diverse initial solutions and the second stage was an improvement phase mainly using novel Variable Neighborhood Search (VNS) methods they developed to optimize the initial solutions. Liu et al. [33] studied an evolutionary algorithm to solve the multimodal multiobjective traveling salesman problem (MMTSP), which had the potential to solve many real-world problems. The proposed algorithm, where two new edge assembly crossover operators were embedded, used a new environmental selection operator to maintain a balance between the objective space diversity and decision space diversity. Tsai et al. [34] developed a novel method to improve biogeography-based optimization (BBO) for solving the TSP. The proposed method combined a greedy randomized adaptive search procedure and the 2-opt algorithm. However, it was only tested on three datasets and the largest size of the instances was 100.

However, when solving the TSP, a single meta-heuristic algorithm is more likely to fall into the local optimal, which degrades the performance of the algorithm. Therefore, many scholars have proposed hybrid meta-heuristic algorithms, which fuse two or more algorithms together to make full use of the advantages of different algorithms. This balances the exploration and exploitation capabilities of the algorithms and helps algorithms to better solve complex problems. Baraglia et al. [35] solved the classical TSP by combining GA with Lin–Kernighan local search. Similarly, Yang and Pei [7] proposed a hybrid method based on ABC and PSO. Mahi et al. [8] proposed a hybrid algorithm by combining PSO, ACO, and the 3-opt algorithm. Gulcu et al. [12] proposed a parallel cooperative hybrid algorithm combining ACO and the 3-opt algorithm. Saji and Barkatou [2] combined the random walk of Lévy flight with bat movement to improve the traditional BA to solve the classic TSP. In order to improve the diversity and convergence of the population, a uniform crossing operator in GA was embedded in the proposed algorithm. Yang et al. [9] proposed a new method to solve the TSP with an arbitrary neighborhood. In the hybrid algorithm, the outer loop used linear decreasing inertial weight PSO to search the continuous access location, while the inner loop used GA to optimize the discrete visiting sequence. The experiments showed that this hybrid algorithm can significantly reduce the search space without lowering the quality of solutions and can find high-quality solutions in a reasonable time. It can be seen that scientifically combining two or more meta-heuristic algorithms and redesigning the algorithm by utilizing their advantages and discarding their disadvantages to solve the TSP plays an important role in promoting the quality and efficiency of the solution algorithm.

2.2. Research on JAYA Algorithm

2.2.1. The Basic JAYA Algorithm

JAYA is interpreted as victory in Sanskrit. The algorithm strives to achieve victory by obtaining the optimal solution; hence, the algorithm was named the JAYA algorithm. Based on the principle of continuous improvement, the JAYA algorithm approaches excellent individuals while constantly moving away from poor ones, thus improving the quality of solutions [21]. Unlike other evolutionary algorithms that require many parameters, the JAYA algorithm only needs to adjust parameters of the iterative process for specific problems such as random numbers, which can avoid the problem that too many parameters need to be adjusted during the implementation of the algorithm. Therefore, compared with other meta-heuristic algorithms, the JAYA algorithm has a unique orientation characteristic of striving for the best and avoiding the worst. It has the advantages of few control parameters, simple structure, flexible mechanism, and easy understanding and implementing, which make it be suitable for solving diverse optimization problems.

In the basic JAYA algorithm, each individual in the population iteratively evolves to obtain a new solution based on Equation (6), as follows:

$$x'_{k,i} = x_{k,i} + r_1\left(x_{best,i} - \left|x_{k,i}\right|\right) - r_2\left(x_{worst,i} - \left|x_{k,i}\right|\right), \tag{6}$$

where $x_{k,i}$ is the i-th dimensional variable of the k-th individual. $x_{best,i}$ and $x_{worst,i}$ are the i-th dimension variable of the individual with the best and the worst fitness values in the current iteration, respectively. Both r_1 and r_2 are random numbers in the range [0,1]. $x'_{k,i}$ is the updated value of the i-th dimension variable of the k-th individual. It can be seen from Equation (6) that $r_1(x_{best,i} - |x_{k,i}|)$ shows the evolution trend of the current solution to the current best one and $r_2(x_{worst,i} - |x_{k,i}|)$ shows the evolution trend of the current solution away from the worst one. Therefore, the core of JAYA algorithm is to approach the optimal solution while staying away from the worst solution. The flowchart of the basic JAYA algorithm is shown in Figure 1.

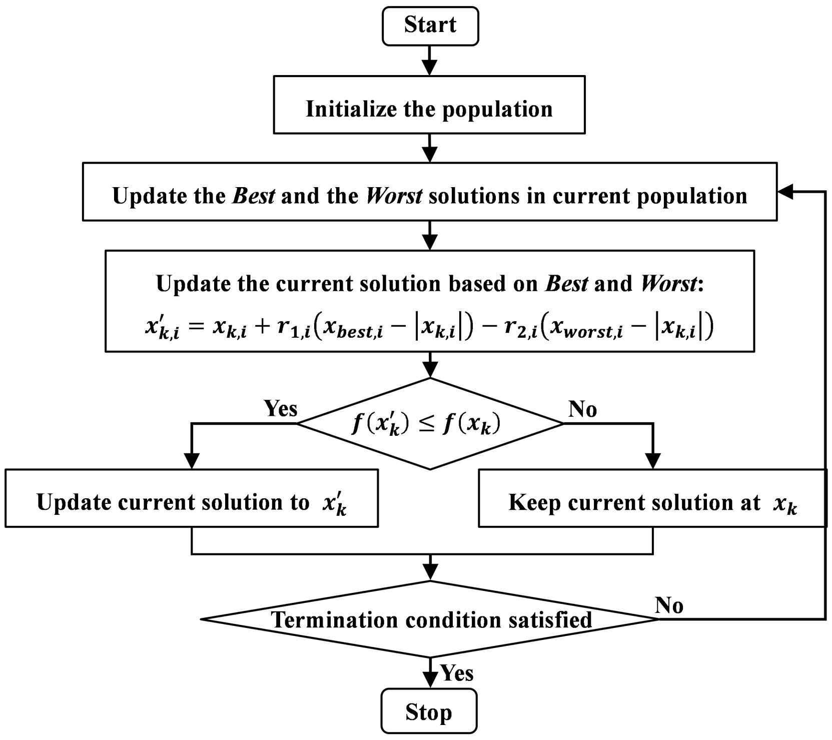

Start
Initialize the population
Update the *Best* and the *Worst* solutions in current population
Update the current solution based on *Best* and *Worst*:
$x'_{k,i} = x_{k,i} + r_{1,i}(x_{best,i} - |x_{k,i}|) - r_{2,i}(x_{worst,i} - |x_{k,i}|)$
$f(x'_k) \le f(x_k)$
Yes
No
Update current solution to x'_k
Keep current solution at x_k
Termination condition satisfied
No
Yes
Stop

Figure 1. Flowchart of the basic JAYA algorithm.

2.2.2. Related Work on JAYA Algorithm

The traditional JAYA algorithm is a powerful meta-heuristic algorithm proposed by Rao [21] in 2016. Rao proved its excellent performance by solving 30 unconstrained benchmark problems. According to the related literature, the JAYA algorithm has a unique orientation characteristic of striving for the best and avoiding the worst. It has the advantages of few control parameters, a simple structure, and a flexible mechanism, which make it be suitable for solving diverse optimization problems. And, the JAYA algorithm has been successfully applied in many fields such as in society and industry and has a wide range of application scenarios. Aslan et al. [36] proposed a binary optimization algorithm based on the JAYA algorithm, which replaced the updating rules of the traditional JAYA algorithm with newly designed transformation operators for binary optimization. Rao and More [37] proposed an adaptive JAYA algorithm to solve the design optimization and analysis of selected thermal devices. Pradhan and Bhende [38] introduced linear inertia weights and nonlinear inertia weights based on fuzzy logic, respectively, which effectively improved the ability of the JAYA algorithm to solve complex problems. Wang et al. [39] proposed a parallel JAYA algorithm based on graphics processors (GPUs) to estimate the model parameters of lithium-ion batteries, which can not only accurately estimate the model parameters of batteries but also greatly shorten the runtime. Xiong et al. [24] combined the JAYA algorithm and differential evolution to propose a hybrid meta-heuristic optimization algorithm, which has been successfully applied to the parameter optimization of solid oxide fuel cells. Thirumoorthy and Muneeswaran [23] applied the hybrid JAYA optimization algorithm to text document clustering, which achieved the highest quality clustering in all selected benchmark instances. Gunduz and Aslan [27] improved upon the algorithm, leading to the discrete JAYA algorithm,and applied it for the first time to solve the TSP. They proved that the proposed DJAYA algorithm was a highly competitive and robust optimizer for the TSP. Li et al. [22] once again verified the effectiveness of the JAYA algorithm by solving the flexible job-shop scheduling problem with an improved JAYA algorithm.

3. The Proposed QSA-DJAYA Algorithm for TSP

As mentioned above, the JAYA algorithm was originally proposed by Rao to solve continuous optimization problems [21]. By means of discretization, the JAYA algorithm has been applied to solve flexible job-shop scheduling [22], text clustering [23], solid oxide fuel cell parameter optimization [24], feature selection [25], the hydropower reservoir operation optimization problem [26], the TSP [27], etc.

In the proposed algorithm, the initialization method is the same as that in [27], that is, the first individual is constructed using the nearest neighbor algorithm and the rest individuals in the population are generated via random permutation. Following the core of the basic JAYA algorithm, two search trend parameters $ST1$ and $ST2$ are used to control the selection of an individual among the best, worst, and current for solution updating. And, this strategy is also taken from [27]. The selection of updating operators takes into account the basic Q-learning algorithm in reinforcement learning, which will be introduced in detail in Section 3.1. In addition, the characteristic of SA will be introduced in the acceptance criteria of solutions, which is described in Section 3.2. Below, the proposed algorithm is called the QSA-DJAYA algorithm.

3.1. Strategy Selection Based on Q-Learning Algorithm

Q-learning is an approximate value-based algorithm in reinforcement learning. When an agent implements an action, the environment will give a corresponding reward R according to the action. Q is $Q(s,a)$, that is, the expected income obtained by taking action a $(a \in \mathcal{A})$ in a certain state s $(s \in \mathcal{S})$. Therefore, the main idea of the algorithm is to build a Q-table based on state and action to store Q-values and, then, to select the actions that can obtain the maximum benefits according to the Q-values. Learning is a dynamic process, and the action-value function $Q(s,a)$ is iteratively updated by learning from the collected experiences of the current policy [40]. The equation for updates uses the following method of time difference:

$$Q(s_{t+1}, a_t) = Q(s_t, a_t) + \alpha[R_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t)], \tag{7}$$

where α is the learning rate and γ is the rewarding decay coefficient. Usually, in the Q-learning algorithm, the way to select an action is via the ε-greedy strategy, that is, the action with the best Q-value for s_{t+1} is selected in most cases, and an action is randomly selected in a few cases. According to this strategy, the action that should be implemented in each step can be selected through the iteratively updated Q-values. The actions obtained according to this method can maximize the reward, so as to achieve the optimal policy. But, in order to improve the efficiency, the strategy of selecting the optimal action using a combination of the roulette and greedy strategy is adopted in the proposed QSA-DJAYA algorithm. In order to promote the action with a large Q-value to have a high probability of being selected, the calculation of the specified probability follows Equation (8). The pseudo-code of Q-learning is shown in Algorithm 1.

$$prob(a_k) = \frac{\exp\left(Q(s, a_k)\right)}{\sum_k \exp\left(Q(s, a_k)\right)}, \tag{8}$$

where a_k represents the k-th action. The exponential function for Q-values is to avoid the denominator being 0.

Algorithm 1 Pseudo-code of Q-learning.

```
1:  Initialize the Q-table
2:  Initialize the initial state $s_0$
3:  repeat
4:      if $rand < \varepsilon$ then
5:          Select an action $a_t$ among the set of all actions $\mathcal{A}$ at random
6:      else
7:          Select an action $a_t$ that satisfies $\max_{a_t \in \mathcal{A}} Q(s_{t+1}, a)$
8:      end if
9:      Take action $a_t$, observe the reward $R_t$ and state $S_t$
10:     Update Q-values according to Equation (7)
11: until Termination condition satisfied
```

The QSA-DJAYA algorithm integrates the abovementioned basic idea of the Q-learning algorithm into the discrete JAYA algorithm, obtaining a discrete JAYA algorithm based on Q-learning. Specifically, the six operators—swap, shift, symmetry, insertion, reversion, and 2-opt—are used as each action and each state, and the most appropriate updating operator is selected through the Q-learning algorithm in the updating process of each solution. A Q-table of the QSA-DJAYA algorithm is shown in Table 1.

Table 1. Q-table design of QSA-DJAYA algorithm.

	Action 1	**Action 2**	**Action 3**	**Action 4**	**Action 5**	**Action 6**
State 1	Q(1,1)	Q(1,2)	Q(1,3)	Q(1,4)	Q(1,5)	Q(1,6)
State 2	Q(2,1)	Q(2,2)	Q(2,3)	Q(2,4)	Q(2,5)	Q(2,6)
State 3	Q(3,1)	Q(3,2)	Q(3,3)	Q(3,4)	Q(3,5)	Q(3,6)
State 4	Q(4,1)	Q(4,2)	Q(4,3)	Q(4,4)	Q(4,5)	Q(4,6)
State 5	Q(5,1)	Q(5,2)	Q(5,3)	Q(5,4)	Q(5,5)	Q(5,6)
State 6	Q(6,1)	Q(6,2)	Q(6,3)	Q(6,4)	Q(6,5)	Q(6,6)

In terms of operator selection, although the 3-opt operator has high efficiency, the time cost of applying this operator is too high for the large-scale TSP. So, the 3-opt updating operator is only applied to the current global optimal solution at a certain frequency. The six operators used in the current link are swap, insertion, reversion, shift, symmetry, and 2-opt. Brief descriptions of these operators are described below.

Swap: Two positions are randomly selected from the current route A (marked in green), and then, the elements at these two positions are exchanged. The transformed route is B. An example of a swap operation is shown diagrammatically in Figure 2.

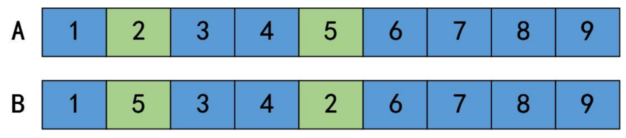


Figure 2. An example of swap transformation for TSP.

Insertion: Two positions are randomly selected from the current route A (marked in green), and the element at the first position is inserted after the second element. An example of insertion operation is shown diagrammatically in Figure 3.

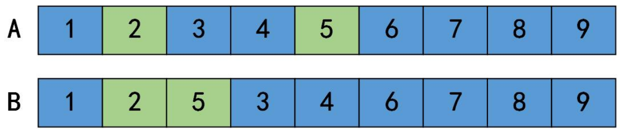


Figure 3. An example of insertion transformation for TSP.

Reversion: Two positions are randomly selected from the current route A (marked in green), and then, the elements between these two positions are arranged in reverse order. An example of reversion operation is shown diagrammatically in Figure 4.

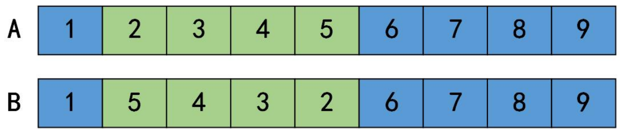


Figure 4. An example of reversion transformation for TSP.

Shift: Two positions are randomly selected from the current route A, namely pos_1 and pos_2. Then, the element at pos_1 (marked in green) is stored, and the first element on the right side of the element at pos_1 to the element at pos_2 (marked in orange) is moved one position to the left in the original order. Finally, the element at pos_1 is placed at pos_2. An example of shift operation is shown diagrammatically in Figure 5.

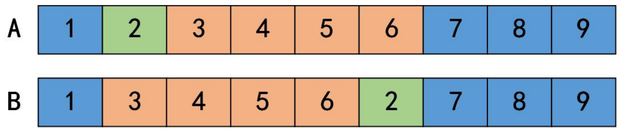


Figure 5. An example of shift transformation for TSP.

Symmetry: According to the length of the current route A, a reasonable length of single transformation segment L is randomly selected, and then, the starting point of transformation is randomly selected on the premise that the route is reasonable. Two consecutive segments of length L are randomly selected on A (marked in green and orange, respectively), and the position of the two segments is switched first, and then, the sequence reversal is carried out on the two segments after the exchange. An example of symmetry operation is shown diagrammatically in Figure 6.

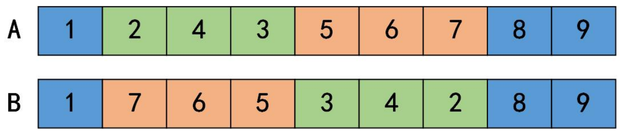


Figure 6. An example of symmetry transformation for TSP.

2-opt: 2-opt is a local search algorithm that basically operates by randomly selecting two arcs from a route and swapping them if this results in a new shorter route. All possible 2-opt transformations for a route are tried, and then, the best 2-opt transformation is chosen. An example of a 2-opt operation is shown diagrammatically in Figure 7 , the segment that need to be reversed are marked in green.

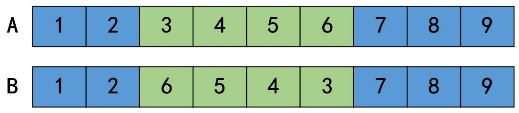


Figure 7. An example of 2-opt transformation for TSP.

After updating the operation with the selected operator, the corresponding reward is set according to the degree of improvement in the solution's fitness. The reward is calculated as Equation (9).

$$reward = \max\,(1 - \frac{fitness_{new}}{fitness_{old}}, 0), \tag{9}$$

where $fitness_{new}$ represents the corresponding fitness of the new solution obtained after updating the solution with the operator bound to the action. $fitness_{old}$ represents the original fitness of the solution before the implementation of the action. According to the above equation, the reward must be a number not less than 0. And, the better the improvement effect of the updating operator, the greater the reward.

Therefore, unlike [27], the QSA-DJAYA algorithm is inspired by the advanced idea of reinforcement learning in terms of solution updating and uses the Q-learning algorithm to dynamically select operators with superior performance to update the solution, making the updating mechanism more reasonable.

3.2. Acceptance Strategy Based on SA

The SA is a stochastic optimization algorithm based on the Monte Carlo iterative strategy. It is inspired by the annealing process of the solid matter in physics. SA is actually a greedy algorithm, but its search process uses the Metropolis acceptance criterion. That is, it accepts a solution worse than the current solution with a certain probability. Therefore, it is possible to jump out of the local optimal solution to find the global optimal solution.

After evaluating the fitness of the new solution, the QSA-DJAYA algorithm also introduces the Metropolis acceptance criterion to judge whether to accept the new solution, and the acceptance probability is calculated as Equation (10).

$$p = \begin{cases} 1, & \text{if } f(x_{new}) < f(x_{old}), \\ \exp\,(-\frac{f(x_{new}) - f(x_{old})}{T}), & \text{if } f(x_{new}) \geq f(x_{old}), \end{cases} \tag{10}$$

where $f(x_{new})$ denotes the fitness of the new solution and $f(x_{old})$ denotes the fitness of the current solution. T denotes the current temperature, $T = rate * T_0$, $rate \in (0, 1)$, T_0 is the initial temperature. Therefore, according to the Metropolis criterion, if $f(x_{new})$ is less than $f(x_{old})$, the current solution is updated as a new solution with probability 1, that is, if the new solution is found to be better, the new solution is accepted. If $f(x_{new})$ is greater than or equal to $f(x_{old})$, then the probability p needs to be calculated according to Equation (10). When the random number $r \in (0, 1)$ is less than p, the current solution will be updated to the new solution.

3.3. The Proposed QSA-DJAYA Algorithm

As shown in Figure 8, the QSA-DJAYA algorithm starts by generating the initial population containing N-1 individuals constructed via random permutation and 1 individual constructed via the nearest neighbor algorithm. At each iteration, the best and worst solutions in the current population, *Best* and *Worst*, are updated. According to the guidance of $ST1$ and $ST2$, a solution is selected as the current solution for the updating operation from *Best*, *Worst* and x_k. Then, the ε-greedy strategy is followed, and either the transformation operator is randomly chosen or the most promising transformation operator is chosen. When the new solution is generated and its fitness value is evaluated, the acceptance of the new solution is judged according to the Metropolis acceptance criterion. When the reward R is calculated and the Q-table is updated, the current best solution is further improved by 3-opt at a certain frequency. Finally, it stops when the termination condition is satisfied.

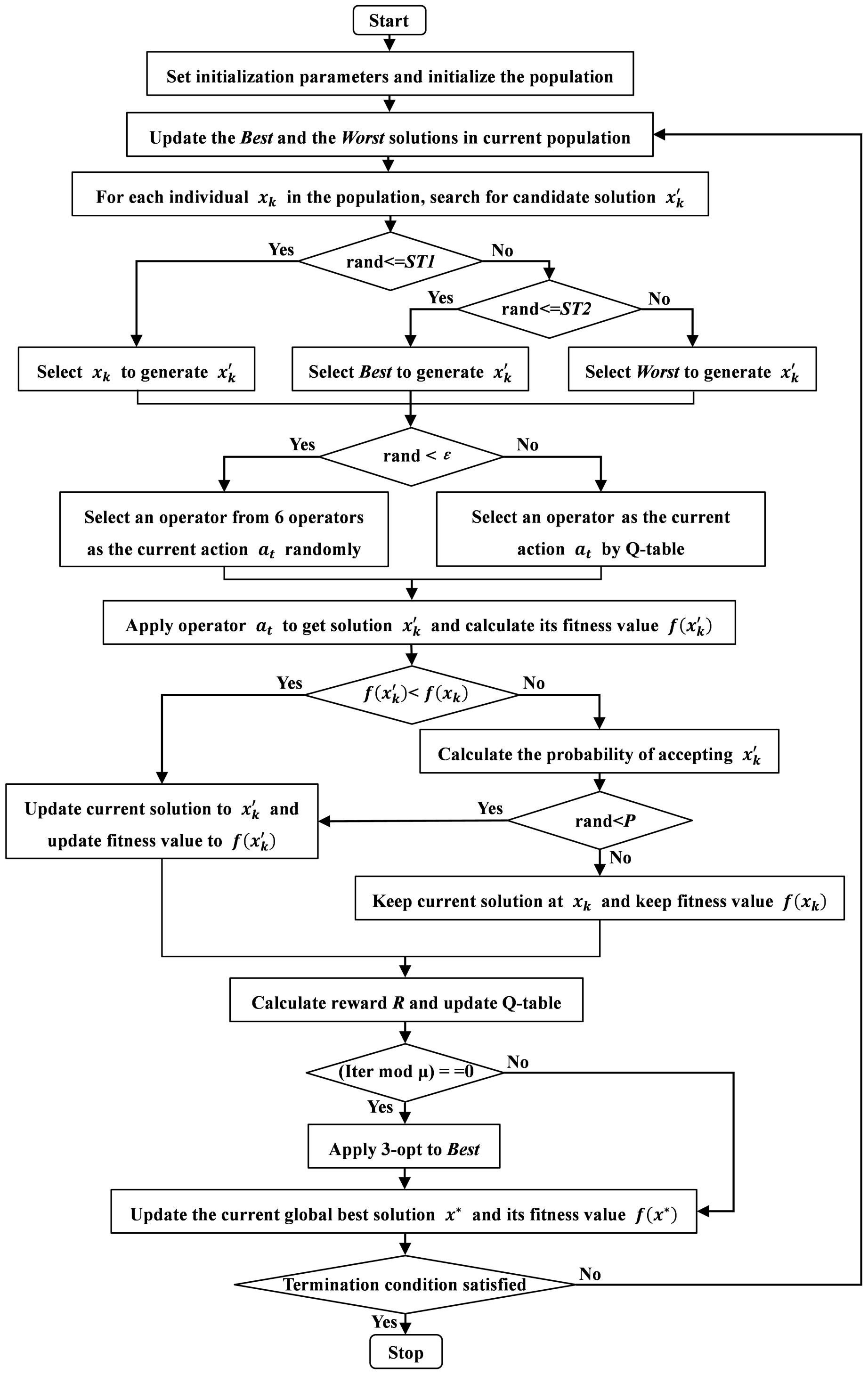

Start
Set initialization parameters and initialize the population
Update the *Best* and the *Worst* solutions in current population
For each individual x_k in the population, search for candidate solution x'_k
rand<=*ST1*
Yes
No
rand<=*ST2*
Yes
No
Select x_k to generate x'_k
Select *Best* to generate x'_k
Select *Worst* to generate x'_k
rand < ε
Yes
No
Select an operator from 6 operators as the current action a_t randomly
Select an operator as the current action a_t by Q-table
Apply operator a_t to get solution x'_k and calculate its fitness value $f(x'_k)$
$f(x'_k)< f(x_k)$
Yes
No
Calculate the probability of accepting x'_k
Update current solution to x'_k and update fitness value to $f(x'_k)$
rand<*P*
Yes
No
Keep current solution at x_k and keep fitness value $f(x_k)$
Calculate reward *R* and update Q-table
(Iter mod μ) = =0
No
Yes
Apply 3-opt to *Best*
Update the current global best solution x^* and its fitness value $f(x^*)$
Termination condition satisfied
No
Yes
Stop

Figure 8. Flowchart of the QSA-DJAYA algorithm.

4. Experimental Results

In order to evaluate the performance of the designed QSA-DJAYA algorithm, we conducted two groups of comparative experiments. In experiment 1, we compared the proposed algorithm with four other representative algorithms, viz., DJAYA, GA, ACO, and SA, developed by ourselves to verify the excellence of the QSA-DJAYA algorithm's framework. In addition, we conducted a comparison with eight efficient methods from the literature, which include ACO, PSO, GA, BH, ABC, the Hierarchic Approach (HA), DTSA, and DJAYA in experiment 2. This section details the experimental settings, experimental results, and comparative studies.

4.1. Experimental Settings

All experiments with the algorithms developed by us were run on an Intel(R)Core(TM)i9-10900K 3.70GHz desktop with 64.0GB of memory. All codes were implemented in MATLAB. The 21 test instances we selected were all the benchmark TSP datasets obtained from TSPLIB [29]. The instances are listed in Table 2. In Table 2, the name of the instance, the scale of the problem, and the best known solution are listed.

Table 2. List of experimental instances.

Number	Name	N	BKS
1	gr17	17	2085
2	bayg29	29	1610
3	bays29	29	2020
4	oliver30	30	420
5	swiss42	42	1273
6	eil51	51	426
7	berlin52	52	7542
8	st70	70	675
9	pr76	76	108,159
10	eil76	76	538
11	rat99	99	1211
12	kroA100	100	21,282
13	kroB100	100	22,141
14	kroC100	100	20,749
15	kroD100	100	21,294
16	kroE100	100	22,068
17	eil101	101	629
18	lin105	105	14,379
19	pr124	124	59,030
20	ch150	150	6528
21	tsp225	225	3919

In the comparative experiment 1, the algorithm was run 30 times for each instance. While in experiment 2, the number of runs was consistent with the experimental settings of the literature compared for a fair comparison. Over these test replicates, the shortest tour length obtained from each run and the computational time to obtain the shortest tour length were recorded. For each algorithm, we sorted out the best value, the worst value, the average value, the standard deviation, and the Gap value obtained in these replicates of each instance through the experimental results. The Gap value is a percentage and is calculated as shown in Equation (11).

$$Gap(\%) = \frac{Average - BKS}{BKS} \times 100, \tag{11}$$

where $Average$ is the average of the shortest tour length, and BKS is the currently best known solution of the instance. So, the optimization performance of each algorithm can be evaluated using this index.

4.2. Parameter Tuning

The values of $ST1$ and $ST2$ were the same as those in [27], viz., $ST1 = ST2 = 0.5$. And, although 3-opt is an efficient operator, for computation time reasons, the frequency of applying 3-opt μ was set to 100 in all experiments. Except for these two parameters, there are five parameters in the QSA-DJAYA algorithm that needed to be adjusted. In order to determine these parameters, the instance kroC100 was chosen as a test instance to carry out parameter tuning experiments. For each parameter tuning experiment, the algorithm stopped when the maximum number of function evaluations (MaxF) reached 300,000.

The first parameter tuning experiment was performed for the population size $popsize$. The $popsize$ was set to 10∼100, the learning factor in reinforcement learning was $\alpha = 0.9$, the discount factor in reinforcement learning was $\gamma = 0.8$, the probability of escaping from the local optimal in reinforcement learning was $\varepsilon = 0.1$, and the initial temperature in SA was $T_0 = 0.045$. The row corresponding to $popsize$ in Table 3 shows the experimental results of the QSA-DJAYA algorithm on the instance kroC100 under different $popsize$. It can be seen that when the $popsize$ was set to 10, the average value of the shortest tour length obtained over the 10 runs was the smallest, so $popsize$ was determined to be 10 in subsequent experiments.

Table 3. Results of parameter tuning.

Parameter	Value	Average	Std	Gap (%)
$popsize$	10	**21,012.30**	138.55	1.27
	20	21,066.90	175.61	1.53
	30	21,198.10	39.09	2.16
	40	21,161.20	65.40	1.99
	50	21,187.60	13.80	2.11
	60	21,183.00	0.00	2.09
	70	21,183.00	0.00	2.09
	80	21,183.00	0.00	2.09
	90	21,183.00	0.00	2.09
	100	21,183.00	0.00	2.09
α	0.1	20,933.00	114.30	0.89
	0.2	21,033.90	128.93	1.37
	0.3	21,008.50	89.24	1.25
	0.4	20,921.70	163.11	0.83
	0.5	20,971.50	159.76	1.07
	0.6	20,999.10	173.70	1.21
	0.7	20,926.90	150.25	0.86
	0.8	**20,902.70**	118.09	0.74
	0.9	21,012.30	138.55	1.27
γ	0.1	21,036.20	140.78	1.38
	0.2	21,000.00	164.05	1.21
	0.3	20,960.80	154.05	1.02
	0.4	20,910.50	141.26	0.78
	0.5	21,006.80	180.07	1.24
	0.6	21,036.50	131.14	1.39
	0.7	20,984.80	224.06	1.14
	0.8	**20,902.70**	118.09	0.74
	0.9	21,060.60	170.80	1.50
ε	0.1	**20,902.70**	118.09	0.74
	0.2	21,095.50	157.97	1.67
	0.3	20,998.60	164.42	1.20
	0.4	20,950.20	195.06	0.97
	0.5	20,940.00	181.61	0.92
T_0	0.01	21,179.30	22.03	2.07
	0.02	20,995.40	129.59	1.19
	0.03	21,073.80	217.49	1.57
	0.04	21,028.60	227.02	1.35
	0.05	**20,925.10**	98.58	0.85

The second parameter tuning experiment was carried out for α, and α was set to 0.1∼0.9. According to the first parameter tuning experiment, the $popsize$ in this experiment was set to 10. The row corresponding to α in Table 3 shows the experimental results of the

QSA-DJAYA algorithm on the instance kroC100 under different α. The results show that the result was best when α was set to 0.8.

The third parameter tuning experiment was carried out for γ, and γ was set to 0.1$\sim$0,9. According to the first and second experiments, the $popsize$ in this experiment was set to 10 and the α was set to 0.8. The row corresponding to γ in Table 3 shows the experimental results of the QSA-DJAYA algorithm on the instance kroC100 under different γ. The results show that γ should be set to 0.8.

The fourth is carried out for ε, and ε was set to 0.1$\sim$0.5. According to the previous three parameter tuning experiments, the $popsize$ in this experiment was set to 10, the α was set to 0.8, and the γ was set to 0.8. The row corresponding to ε in Table 3 shows the experimental results of the QSA-DJAYA algorithm on the instance kroC100 under different ε. The results show that ε should be set to 0.1.

The last parameter tuning experiment was carried out for the T_0, and T_0 was set to 0.01$\sim$0.05. According to the previous parameter tuning experiments, the $popsize$ in this experiment was set to 10, the α was set to 0.8, the γ was set to 0.8, and the ε was set to 0.1. The row corresponding to T_0 in Table 3 shows the experimental results of the QSA-DJAYA algorithm on the instance kroC100 under different T_0. According to the experimental results, T_0 should be set to 0.05 to optimize the performance of the QSA-DJAYA algorithm.

In summary, the parameter in the following experiments were set as population size $popsize = 10$, learning factor in reinforcement learning was $\alpha = 0.8$, discount factor in reinforcement learning was $\gamma = 0.8$, escape local optimal probability in reinforcement learning was $\varepsilon = 0.1$, initial temperature in SA was $T_0 = 0.05$.

4.3. Experimental Results and Statistical Analysis

In this section, we first show the results of experiments 1 and 2. Then, the comparative analysis carried out according to these experimental results and the significance of the obtained results verified using the non-parametric Friedman statistical test and non-parametric Mann–Whitney test are presented. It should be noted that the DJAYA algorithm in experiment 1 is different from that in [27] because it is embedded with the same six operators as those in the QSA-DJAYA algorithm in order to avoid the loss of fairness caused by different operator efficiencies.

4.3.1. Results of Experiment 1

According to the experimental scheme determined in Section 4.1 and the parameter settings determined in Section 4.2, we conducted experiment 1 on a set of TSP benchmark instances containing a total of 20 instances. On the basis of the size of the instance, the instances were divided into two sets. The first set of small-scale instances included six instances with the number of cities ranging from 17 to 52. The second set of large-scale instances included fourteen examples with the number of cities ranging from 70 to 225. Due to the difference in scale, the stopping criterion was the MaxF, which was set to $N \times 500$. N is the number of cities involved in the instance. The basic experimental results of small-scale and large-scale instances are shown in Tables 4 and 5, respectively.

Table 4. Results of five representative compared algorithms on small-scale instances.

Name	Algorithm	Best	Worst	Median	Average	Std	Gap (%)	Time (s)
gr17	GA	2238	2489	2376.0	2377.53	60.52	14.03	0.24
	ACO	2085	2149	2115.0	2114.67	23.30	1.42	0.77
	SA	2085	2090	2087.5	2087.50	2.50	0.12	0.09
	DJAYA	2085	2088	**2085.0**	2085.70	1.27	0.03	0.80
	QSA-DJAYA	2085	2085	**2085.0**	**2085.00**	0.00	0.00	0.50

Table 4. *Cont.*

Name	Algorithm	Best	Worst	Median	Average	Std	Gap (%)	Time (s)
bayg29	GA	2378	2698	2531.0	2539.03	70.30	57.70	0.50
	ACO	1638	1672	1661.5	1657.83	6.58	2.97	3.90
	SA	1610	1683	1622.0	1626.47	19.64	1.02	0.15
	DJAYA	1615	1674	1634.0	1637.73	15.69	1.72	2.68
	QSA-DJAYA	1610	1646	**1610.0**	**1618.83**	12.32	0.55	2.08
bays29	GA	2858	3360	3213.5	3175.40	134.26	57.20	0.49
	ACO	2020	2062	2020.0	2024.60	9.44	0.23	3.79
	SA	2020	2082	2033.0	2038.43	19.58	0.91	0.15
	DJAYA	2026	2035	**2026.0**	2028.70	3.57	0.43	2.69
	QSA-DJAYA	2020	2034	**2026.0**	**2028.40**	3.68	0.42	2.09
swiss42	GA	2380	2710	2593.0	2587.63	84.98	103.27	0.87
	ACO	1287	1303	1299.0	1297.93	3.53	1.96	6.67
	SA	1273	1391	1335.5	1329.87	35.51	4.47	0.23
	DJAYA	1273	1274	**1273.0**	1273.23	0.42	0.02	6.93
	QSA-DJAYA	1273	1273	**1273.0**	**1273.00**	0.00	0.00	6.12
eil51	GA	881	984	921.0	921.70	22.17	116.36	1.22
	ACO	437	457	447.0	447.37	4.98	5.02	11.26
	SA	428	454	441.0	440.97	6.36	3.51	0.29
	DJAYA	428	438	**432.0**	**432.30**	3.68	1.48	12.26
	QSA-DJAYA	426	434	434.0	432.73	2.38	1.58	10.92
berlin52	GA	13705	16367	15563.0	15550.57	505.02	106.19	1.27
	ACO	7662	7767	7679.0	7683.53	24.27	1.88	11.84
	SA	7542	8317	7988.5	7954.83	190.78	5.47	0.29
	DJAYA	7542	7711	7657.0	7641.27	41.95	1.32	12.68
	QSA-DJAYA	7542	7798	**7542.0**	**7557.33**	57.19	0.20	11.74

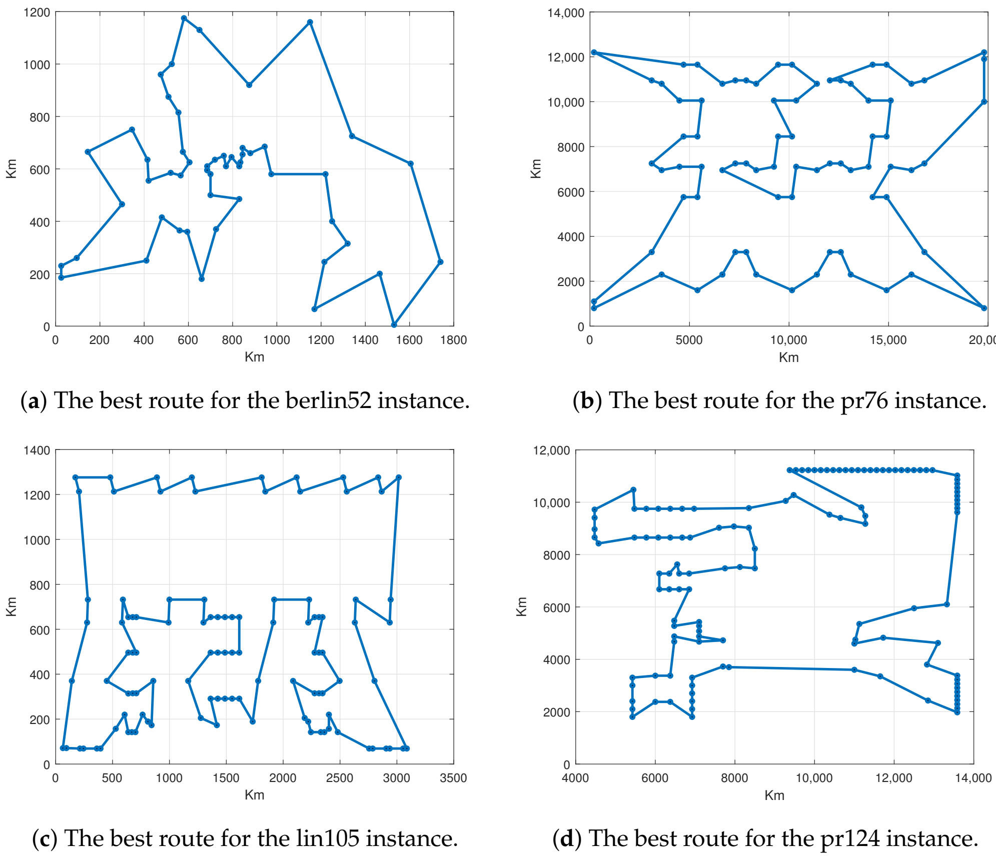
Table 5. Results of five representative compared algorithms on large-scale instances.

Name	Algorithm	Best	Worst	Median	Average	Std	Gap (%)	Time (s)
st70	GA	1888	2093	2005.5	1994.07	51.53	195.42	2.01
	ACO	708	734	718.0	718.83	6.37	6.49	25.68
	SA	684	744	706.5	709.00	15.55	5.04	0.41
	DJAYA	687	714	703.0	701.90	8.87	3.99	39.36
	QSA-DJAYA	684	703	**684.0**	**686.80**	4.85	1.75	29.46
pr76	GA	306,770	329,482	322,696.5	320,291.60	6367.43	196.13	2.36
	ACO	115,846	121,443	118,745.5	118,676.93	1172.13	9.72	30.59
	SA	109,872	120,095	113,747.5	114,336.93	2762.89	5.71	0.46
	DJAYA	109,190	110,684	110,684.0	110,564.07	383.17	2.22	57.08
	QSA-DJAYA	108,159	110,858	**109,653.0**	**109,530.73**	907.21	1.27	38.73
eil76	GA	1335	1498	1425.0	1420.47	37.08	164.03	2.32
	ACO	558	568	565.0	564.57	2.08	4.94	32.23
	SA	553	585	567.0	567.43	8.53	5.47	0.47
	DJAYA	553	563	558.0	558.90	2.94	3.88	56.18
	QSA-DJAYA	540	553	**551.0**	**550.60**	2.23	2.34	38.34
rat99	GA	4120	4633	4474.0	4467.73	113.72	268.93	3.74
	ACO	1287	1337	1313.0	1312.17	11.30	8.35	61.79
	SA	1257	1349	1307.5	1307.33	25.28	7.95	0.65
	DJAYA	1253	1257	1256.0	1255.83	0.69	3.70	182.15
	QSA-DJAYA	1230	1256	**1253.0**	**1253.20**	4.52	3.48	95.56

Table 5. *Cont.*

Name	Algorithm	Best	Worst	Median	Average	Std	Gap (%)	Time (s)
kroA100	GA	83,919	93,133	90,348.5	89,856.23	2208.86	322.22	3.74
	ACO	22,428	23,318	22,748.5	22,760.13	238.53	6.95	67.86
	SA	21,829	23,595	22,495.0	22,610.57	513.41	6.24	0.65
	DJAYA	21,319	21,578	21,514.0	21,498.97	43.24	1.02	191.85
	QSA-DJAYA	21,292	21389	**21,292.0**	**21,295.37**	17.40	0.06	97.56
kroB100	GA	83,257	92,138	89,154.5	88,831.47	2269.08	301.21	3.81
	ACO	22,901	23,446	23,256.5	23,254.00	114.49	5.03	67.49
	SA	22,647	24,310	23,691.5	23,670.10	402.66	6.91	0.65
	DJAYA	22,258	23,162	22,762.0	22,739.27	154.60	2.70	190.28
	QSA-DJAYA	22,220	22,724	**22,708.0**	**22,635.70**	130.59	2.23	96.98
kroC100	GA	83,948	94,564	88,964.0	89,056.27	2414.13	329.21	3.84
	ACO	21,511	21,775	21,680.0	21,661.67	68.92	4.40	67.93
	SA	21,449	24,221	22,067.5	22,282.13	593.60	7.39	0.66
	DJAYA	21,185	21,309	21,206.0	21,212.37	29.33	2.23	188.90
	QSA-DJAYA	20,965	21,331	**21,183.0**	**21,180.80**	48.06	2.08	97.18
kroD100	GA	81680	89459	86842.0	86690.60	1873.05	307.11	3.80
	ACO	22,572	23,360	23,020.0	23,007.33	159.53	8.05	68.44
	SA	21,822	24,076	22,701.0	22,741.30	523.69	6.80	0.67
	DJAYA	21,620	22,863	22,001.0	22,060.23	299.54	3.60	187.00
	QSA-DJAYA	21495	21,896	**21,575.0**	**21,583.57**	79.89	1.36	97.08
kroE100	GA	85,061	93,912	90,908.5	90,557.90	1932.14	310.36	3.83
	ACO	23,196	23,877	23,667.0	23,661.23	142.12	7.22	68.02
	SA	22,712	24,138	23,354.0	23,364.07	360.65	5.87	0.65
	DJAYA	22,509	22,679	22,547.0	22,562.10	49.78	2.24	187.80
	QSA-DJAYA	22,130	22,475	**22,466.0**	**22,429.53**	74.21	1.64	94.22
eil101	GA	1872	2021	1963.0	1958.70	42.85	211.40	3.88
	ACO	677	705	693.0	693.70	6.24	10.29	66.95
	SA	647	689	667.0	666.73	10.50	6.00	0.67
	DJAYA	642	664	650.0	650.27	5.52	3.38	190.78
	QSA-DJAYA	630	646	**635.0**	**635.27**	3.02	1.00	99.91
lin105	GA	58,391	67,943	63,848.0	63,722.03	1998.94	343.16	4.18
	ACO	14,902	15,150	15,054.0	15,050.50	71.47	4.67	87.30
	SA	14,767	16,061	15,484.5	15,461.77	347.25	7.53	0.71
	DJAYA	14,576	15,071	14,877.5	14,856.93	118.66	3.32	233.07
	QSA-DJAYA	14379	14,660	**14,438.0**	**14,451.43**	84.12	0.50	115.69
pr124	GA	339,211	373,718	362,675.0	360,680.93	8077.14	511.01	5.63
	ACO	60,590	63,297	61,714.5	61,795.57	570.39	4.69	114.72
	SA	60,220	69,852	62,575.0	62,844.47	2128.64	6.46	0.87
	DJAYA	59,246	59,792	**59,246.0**	**59,350.17**	194.25	0.54	510.69
	QSA-DJAYA	59,030	59,792	59,548.0	59,454.43	270.90	0.72	207.90
ch150	GA	30,083	32,150	31,169.5	31,084.30	479.35	376.17	8.46
	ACO	6758	6850	6824.0	6824.60	19.99	4.54	210.20
	SA	6862	7533	7241.5	7223.60	180.03	10.66	1.16
	DJAYA	6598	6633	6629.0	6625.07	9.38	1.49	1259.00
	QSA-DJAYA	6566	6624	**6598.0**	**6596.73**	10.44	1.05	372.08
tsp225	GA	22,763	24,184	23,645.0	23,618.03	368.21	502.65	19.53
	ACO	4225	4380	4293.5	4291.07	37.70	9.49	658.87
	SA	4313	4542	4388.0	4412.00	58.10	12.58	2.10
	DJAYA	4038	4069	4056.0	4054.80	8.11	3.47	11,326.09
	QSA-DJAYA	3994	4059	**4012.0**	**4013.77**	11.85	2.42	1715.49

In Tables 4 and 5, the Best, Worst, Median, Average, and Std columns represent the best value, worst value, median value, average value, and standard deviation of the shortest tour length over 30 independent runs on each instance, respectively. For the Best, Worst, Median and Average columns, the smaller the value is, the stronger the search ability of the algorithm is. For the Std column, the smaller the value is, the higher the reliability and stability of the algorithm is. The calculation of each value in the Gap column follows Equation (11). Since the currently best known solution of most instances is already the optimal solution of the instance, a small value for the Gap value indicates that the efficiency of the algorithm is high. The best shortest tour length for the same instance in the Average and Median columns is shown in bold. In Table 4, except for the instance eil51, the QSA-DJAYA algorithm shows the best performance for the median and average. The DJAYA algorithm performs optimally on the instances eil51, which confirms the effectiveness of the compared algorithms. In Table 5, in terms of the median and average, the proposed algorithm also performs best on all large-scale instances except pr124. For instance pr124, the DJAYA algorithm obtains the best median and average value but does not find the best known solution, while the QSA-DJAYA algorithm finds it. Although the QSA-DJAYA algorithm does not achieve the minimum standard deviation on all instances, the standard deviation of it on seven instances is all minimal. Thus, the stability is better than the compared algorithms. In addition, it can be noted that the calculation time of both the QSA-DJAYA algorithm and the DJAYA algorithm is longer than that of the other algorithms due to the time-consuming calculation of the operators embedded in both algorithms. However, under the premise of the same operator, it can be observed that the calculation time of the QSA-DJAYA algorithm is much less than that of the DJAYA algorithm. Moreover, it can be seen that under the premise of fair comparison, the QSA-DJAYA algorithm obtains the currently best known solution on nine instances. For one small-scale numerical instance and three large-scale numerical instances, the graphical presentations of tours corresponding to the best solutions are presented in Figure 9.



(**a**) The best route for the berlin52 instance. (**b**) The best route for the pr76 instance.

(**c**) The best route for the lin105 instance. (**d**) The best route for the pr124 instance.

Figure 9. Tours correspondingto the best solutions found by QSA-DJAYA.

In summary, Tables 4 and 5 show that the QSA-DJAYA algorithm outperforms all the compared algorithms. However, it is worth mentioning that some compared algorithms such as GA and SA consumed less computational time under the same MaxF, although they achieved poor-quality solutions. The comprehensive consideration of solution quality and computational time is important to evaluate the performance of algorithms. Therefore, additional subexperiments of experiment 1 were performed with the same execution time. Detailed execution times and experimental results are provided in Table 6. From Table 6, it is clearly seen that QSA-DJAYA can achieve higher-quality solutions with the same execution time. This comparative experiment once again confirms the superiority of the proposed QSA-DJAYA in terms of solution quality and efficiency.

Table 6. Results of five representative compared algorithms on large-scale instances.

Name	Algorithm	Best	Worst	Median	Average	Std	Gap (%)	Time (s)
swiss42	GA	2350	2632	2515.0	2514.80	74.42	97.55	2.00
	ACO	1287	1303	1299.0	1298.07	3.53	1.97	2.00
	SA	1273	1398	1332.5	1332.07	34.51	4.64	2.00
	DJAYA	1273	1293	1281.0	1281.07	6.79	0.63	2.00
	QSA-DJAYA	1273	1274	**1273.0**	**1273.03**	0.18	0.00	2.00
berlin52	GA	13,727	15,744	15,078.5	15,033.30	385.67	99.33	10.00
	ACO	7547	7791	7679.0	7676.30	44.08	1.78	10.00
	SA	7542	8534	7970.0	7970.93	226.49	5.69	10.00
	DJAYA	7542	7657	7657.0	7633.30	38.79	1.21	10.00
	QSA-DJAYA	7542	7798	**7542.0**	**7563.60**	61.42	0.29	10.00
pr76	GA	276,734	314,038	305,579.5	303,113.67	7994.43	180.25	60.00
	ACO	114,953	120,653	117,885.5	117,736.00	1530.89	8.85	60.00
	SA	109,265	120,582	113,844.5	113,985.27	2870.60	5.39	60.00
	DJAYA	109,190	111,336	110,684.0	110,489.73	530.23	2.15	60.00
	QSA-DJAYA	108,159	111,464	**109,190.0**	**109,429.10**	1015.60	1.17	60.00
kroA100	GA	81,263	88,952	87,087.5	86,860.57	1506.34	308.14	120.00
	ACO	22,317	23,110	22,592.5	22,647.50	206.69	6.42	120.00
	SA	21,438	23,949	22,136.0	22,359.70	648.97	5.06	120.00
	DJAYA	21,292	21,389	21,389.0	21,385.77	17.41	0.49	120.00
	QSA-DJAYA	21,292	21,711	**21,292.0**	**21,333.70**	106.19	0.24	120.00
lin105	GA	57,147	63,181	60,680.5	60,620.53	1229.44	321.59	120.00
	ACO	14,844	15,203	15,042.5	15,037.40	79.55	4.58	120.00
	SA	14,988	16,029	15,296.0	15,323.40	265.98	6.57	120.00
	DJAYA	14,438	14,849	14,743.0	14,698.77	112.30	2.22	120.00
	QSA-DJAYA	14,379	14,706	**14,463.5**	**14,473.60**	96.34	0.66	120.00
pr124	GA	331,771	356,698	346,274.0	346,512.67	6497.63	487.01	120.00
	ACO	60,726	62,930	61,742.5	61,829.87	571.21	4.74	120.00
	SA	59,354	64,857	61,227.0	61,409.57	1364.10	4.03	120.00
	DJAYA	59,246	59,792	**59,246.0**	59,431.53	252.66	0.68	120.00
	QSA-DJAYA	59,076	59,792	**59,246.0**	**59,396.73**	246.66	0.62	120.00
ch150	GA	28,719	30,887	30,384.0	30,316.60	421.04	364.41	120.00
	ACO	6792	6889	6825.0	6828.03	21.07	4.60	120.00
	SA	6697	7147	6974.5	6961.63	124.89	6.64	120.00
	DJAYA	6624	6705	6692.5	6689.07	19.00	2.47	120.00
	QSA-DJAYA	6574	6625	**6605.0**	**6601.83**	10.05	1.13	120.00
tsp225	GA	22,872	23,777	23,375.5	23,342.80	257.03	495.63	120.00
	ACO	4261	4420	4330.5	4329.63	30.21	10.48	120.00
	SA	4060	4312	4183.0	4178.93	63.93	6.63	120.00
	DJAYA	4178	4196	4190.0	4189.13	3.12	6.89	120.00
	QSA-DJAYA	4037	4146	**4093.0**	**4090.37**	18.31	4.37	120.00

Further, to observe the difference in the results obtained by the above five algorithms on the same problem more intuitively, the boxplots were drawn, as shown in Figure 10. The selected comparison data are the Gap values obtained via each algorithm on each instance , and the red plus signs in Figure 10 represent several outliers. This is beneficial to compare the stability of the search abilities of each algorithm. It should be noted that the experimental results of GA are very poor compared with those of the other four algorithms, so only the other four algorithms are compared here. Through the distribution and range of Gap values presented in the box graphs, it is proved that QSA-DJAYA is better than GA, ACO, SA, and DJAYA.

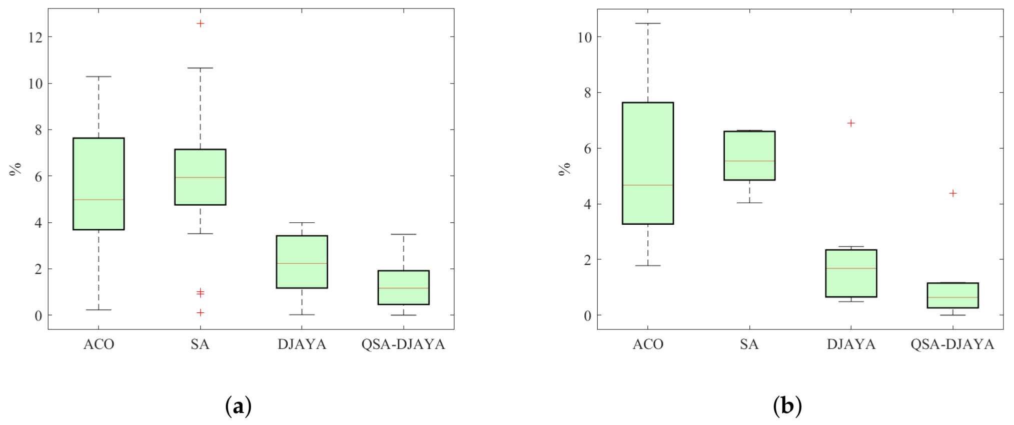


Figure 10. Boxplots of Gap values for the four algorithms. (**a**) Boxplot for the first subexperiment of experiment 1. (**b**) Boxplot for the second subexperiment of experiment 1.

4.3.2. Results of Experiment 2

In experiment 2, a comparison of our proposed method with eight efficient methods from literature was performed. Specifically, the performance of QSA-DJAYA was compared with those of ACO, PSO, GA, and BH in the first subexperiment. The experimental results of ACO, PSO, GA, and BH on seven instances were taken from [30]. In the second subexperiment, we compared QSA-DJAYA with the methods proposed in some studies [27,28,41]. The experimental results of ACO, ABC, and HA were taken from [41], those of DTSA were taken from [28], and those of DJAYA were taken from [27]. For providing a fair comparison, the experimental scheme of experiment 2 was the same as that of the compared methods. Compared results of the first and the second subexperiment are depicted in Tables 7 and 8.

Table 7. The compared results of QSA-DJAYA with the ACO, PSO, GA, and BH algorithms.

Name	Algorithm	Best	Worst	Average	Std
bays29	ACO	9239.1973	11,014.4483	9823.20	722.42
	PSO	9120.3388	9498.1711	9195.91	168.97
	GA	9751.4255	10,513.9142	10,015.23	319.88
	BH	9396.475	9507.1701	9463.25	60.96
	QSA-DJAYA	2020	2026	**2024.80**	2.40
bayg29	ACO	9447.4929	11,033.5484	9882.22	675.83
	PSO	9329.251	11,332.7224	9947.03	799.41
	GA	9579.1234	10,411.1991	9771.95	127.11
	BH	9375.4418	9375.4418	9375.44	0.00
	QSA-DJAYA	1610	1626	**1616.40**	7.84
eil51	ACO	454.3895	469.0531	461.02	6.30
	PSO	469.1551	737.5258	574.80	107.24
	GA	448.8397	462.1142	453.48	9.42
	BH	437.893	526.8977	458.93	38.64
	QSA-DJAYA	427	434	**432.60**	2.80

Table 7. *Cont.*

Name	Algorithm	Best	Worst	Average	Std
berlin52	ACO	7757.0263	10,541.1228	8522.90	1152.20
	PSO	9218.4682	14,279.4331	11,089.53	2067.93
	GA	8779.7559	9565.3744	9288.45	1301.21
	BH	8188.0714	9356.7483	8455.83	508.99
	QSA-DJAYA	7542	7596	**7552.80**	21.60
st70	ACO	711.6515	855.2032	757.75	59.61
	PSO	1030.8484	1756.1227	1321.81	269.28
	GA	1112.3078	1242.2011	1158.85	52.17
	BH	723.2691	1081.1087	797.57	125.23
	QSA-DJAYA	683	697	**686.60**	5.24
eil76	ACO	574.2404	665.9995	594.14	40.22
	PSO	804.2667	1195.9021	975.64	152.41
	GA	619.2262	679.7864	652.06	122.10
	BH	566.243	925.8417	659.10	152.18
	QSA-DJAYA	551	551	**551.00**	0.00
eil101	ACO	725.0996	868.2047	763.92	59.97
	PSO	1158.704	1973.8192	1499.99	319.75
	GA	828.8806	854.4381	838.83	9.96
	BH	720.3838	1249.8684	897.38	210.14
	QSA-DJAYA	634	638	**636.00**	1.67

Table 8. The compared results of QSA-DJAYA with the ACO, ABC, HA, DTSA, and DJAYA algorithms.

Name	Algorithm	Average	Std	Gap (%)
oliver30	ACO	424.68	1.41	0.22
	ABC	462.55	12.47	9.16
	HA	423.74	0.00	0.00
	DTSA	428.50	4.21	1.12
	DJAYA	426.88	2.74	0.74
	QSA-DJAYA	**423.65**	2.94	0.87
eil51	ACO	457.86	4.07	6.76
	ABC	590.49	15.79	37.69
	HA	443.39	5.25	3.39
	DTSA	443.93	4.04	3.51
	DJAYA	440.18	4.95	2.64
	QSA-DJAYA	**432.30**	2.69	1.48
berlin52	ACO	7659.31	38.70	1.52
	ABC	10,390.26	439.69	37.72
	HA	7544.37	0.00	0.00
	DTSA	7545.83	21.00	0.02
	DJAYA	7580.30	80.60	0.48
	QSA-DJAYA	**7552.20**	43.33	0.14
st70	ACO	709.16	8.27	4.73
	ABC	1230.49	41.79	81.73
	HA	700.58	7.51	3.47
	DTSA	708.65	6.77	4.66
	DJAYA	702.30	9.56	3.72
	QSA-DJAYA	**686.70**	3.95	1.73

Table 8. *Cont.*

Name	Algorithm	Average	Std	Gap (%)
eil76	ACO	561.98	3.50	3.04
	ABC	931.44	24.86	70.78
	HA	557.98	4.10	2.31
	DTSA	578.58	3.93	6.09
	DJAYA	573.17	6.33	5.10
	QSA-DJAYA	**550.40**	2.65	2.30
pr76	ACO	116,321.22	885.79	7.55
	ABC	205,119.61	7379.16	89.65
	HA	115,072.29	742.90	6.39
	DTSA	14,930.03	1545.64	6.26
	DJAYA	113,258.29	1711.93	4.71
	QSA-DJAYA	**109,417.35**	944.67	1.16
kroA100	ACO	22,880.12	235.18	7.49
	ABC	53,840.03	2198.36	152.94
	HA	22,435.31	231.34	5.40
	DTSA	21,728.40	358.13	2.08
	DJAYA	21,735.31	331.33	2.13
	QSA-DJAYA	**21,297.05**	21.11	0.07
eil101	ACO	693.42	6.80	7.96
	ABC	1315.95	35.28	104.88
	HA	683.39	6.56	6.40
	DTSA	689.91	4.47	7.41
	DJAYA	677.37	4.87	5.46
	QSA-DJAYA	**635.50**	3.22	1.03
ch150	ACO	6702.87	20.73	2.61
	ABC	21,617.48	453.71	230.93
	HA	6677.12	19.30	2.22
	DTSA	6748.99	32.63	3.32
	DJAYA	6638.63	52.79	1.63
	QSA-DJAYA	**6596.30**	9.97	1.05
tsp225	ACO	4176.08	28.34	8.22
	ABC	17,955.12	387.35	365.28
	HA	4157.85	26.27	7.74
	DTSA	4230.45	58.76	9.93
	DJAYA	4095.02	42.54	6.12
	QSA-DJAYA	**4011.10**	8.61	2.35

As seen from Table 7, QSA-DJAYA performed optimally on all seven instances as far as the average is concerned. Additionally, Table 8 reveals that QSA-DJAYA also achieved a shorter route length in all instances, shown in bold. The obtained results from QSA-DJAYA for all numerical instances are satisfactory in terms of the best, worst, and average values. However, we cannot compare the median values of each algorithm because the relevant results are not provided in the literature.

4.3.3. Results of Statistical Tests

In order to compare the algorithms' performance of experiments 1 and 2 statistically, the non-parametric Friedman statistical test is applied. In experiment 1, for the average tour length obtained using each algorithm over 30 runs, the p values of the first and the second subexperiments are 1.9810×10^{-14} and 6.7936×10^{-6}, respectively. In experiment 2, the p value of the first subexperiment is 9.2960×10^{-4}, and the p value of the second subexperiment is 6.0690×10^{-7}. Therefore, the statistical test results returned by the Friedman statistical test confirm that there are significant differences between the experimental results obtained using the competing algorithms. Meanwhile, statistical significance testing was also performed via the non-parametric Mann–Whitney U test. The null hypothesis was that there was no significant difference between the *Average* values of the two algorithms on the instances participating in the comparison with a 95% confidence level. The results of the Mann–Whitney U test of all comparative experiments based on the *Average* values with a 95% confidence level are summarized in Table 9. Although the QSA-DJAYA is not significantly better than some algorithms at the 95% confidence level, it outperforms all competing methods derived from the literature in terms of solution quality.

Table 9. Results returned by the Mann–Whitney U test in two groups of comparative experiments.

Experiments	Algorithm	U_{min}	U_{α}	Decision
Experiment 1-1	QSA-DJAYA vs. GA	65	127	Negate the null hypothesis.
	QSA-DJAYA vs. ACO	101	127	Negate the null hypothesis.
	QSA-DJAYA vs. SA	100	127	Negate the null hypothesis.
	QSA-DJAYA vs. DJAYA	100	127	Negate the null hypothesis.
Experiment 1-2	QSA-DJAYA vs. GA	21	13	Retain the null hypothesis.
	QSA-DJAYA vs. ACO	14	13	Retain the null hypothesis.
	QSA-DJAYA vs. SA	15	13	Retain the null hypothesis.
	QSA-DJAYA vs. DJAYA	14	13	Retain the null hypothesis.
Experiment 1-2	QSA-DJAYA vs. GA	21	13	Retain the null hypothesis.
	QSA-DJAYA vs. ACO	14	13	Retain the null hypothesis.
	QSA-DJAYA vs. SA	15	13	Retain the null hypothesis.
	QSA-DJAYA vs. DJAYA	14	13	Retain the null hypothesis.
Experiment 2-1	QSA-DJAYA vs. ACO	-2	8	Negate the null hypothesis.
	QSA-DJAYA vs. PSO	8	8	Retain the null hypothesis.
	QSA-DJAYA vs. GA	-2	8	Negate the null hypothesis.
	QSA-DJAYA vs. BH	-2	8	Negate the null hypothesis.
Experiment 2-2	QSA-DJAYA vs. ACO	27	23	Retain the null hypothesis.
	QSA-DJAYA vs. ABC	19	23	Negate the null hypothesis.
	QSA-DJAYA vs. HA	27	23	Retain the null hypothesis.
	QSA-DJAYA vs. DTSA	26	23	Retain the null hypothesis.
	QSA-DJAYA vs. DJAYA	27	23	Retain the null hypothesis.

5. Conclusions and Future Work

In this paper, we proposed an improved discrete JAYA algorithm based on reinforcement learning and SA (QSA-DJAYA) to solve the TSP. The QSA-DJAYA algorithm has been mainly modified in two aspects. On the one hand, the basic Q-learning algorithm in reinforcement learning was introduced to choose the transformation operator when the solution needs to be updated. On the other hand, the SA was introduced in the solution acceptance criterion. The core was to accept poor solutions with a certain probability to balance the exploration and exploitation capabilities of the algorithm. The performance of the QSA-DJAYA algorithm was tested on 21 widely used benchmark instances in the TSPLIB. The comparison results show that the QSA-DJAYA algorithm has significant competitiveness.

Our possible future work is to analyze the applicability of the QSA-DJAYA algorithm to other routing problems, especially variants of the TSP. We may improve the QSA-DJAYA algorithm by combining advanced ideas in reinforcement learning and transfer learning to solve more complex vehicle routing problems in practical applications.

Author Contributions: Conceptualization, J.X., W.H., W.G. and Y.Y.; Methodology, J.X., W.H., W.G. and Y.Y.; Validation, J.X., W.H., W.G. and Y.Y.; Investigation, J.X., W.H., W.G. and Y.Y.; Writing—review & editing, J.X., W.H., W.G. and Y.Y.; Supervision, W.H., W.G. and Y.Y.; Funding acquisition, W.H., W.G. and Y.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Fundamental Research Funds for the Central Universities under Grant 2023JBMC042, the Double First-class Talent Introduction Project of China under Grant No. 505022102 (505021149) and the National Natural Science Foundation of China under Grants 62173027 and 72288101.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Dantzig, G.B.; Ramser, J.H. The Truck Dispatching Problem. *Manag. Sci.* **1959**, *6*, 80–91. [CrossRef]
2. Saji, Y.; Barkatou, M. A discrete bat algorithm based on Lévy flights for Euclidean traveling salesman problem. *Expert Syst. Appl.* **2021**, *172*, 114639. [CrossRef]
3. Arora, S. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *J. ACM* **1998**, *45*, 753–782. [CrossRef]
4. Laporte, G. The traveling salesman problem: An overview of exact and approximate algorithms. *Eur. J. Oper. Res.* **1992**, *59*, 231–247. [CrossRef]
5. Potvin, J.Y. Genetic algorithms for the traveling salesman problem. *Ann. Oper. Res.* **1996**, *63*, 337–370. [CrossRef]
6. Zhang, Z.; Yang, J. A discrete cuckoo search algorithm for traveling salesman problem and its application in cutting path optimization. *Comput. Ind. Eng.* **2022**, *169*, 108157. [CrossRef]
7. Yang, W.; Pei, Z. Hybrid ABC/PSO to solve travelling salesman problem. *Int. J. Comput. Sci. Math.* **2013**, *4*, 214–221. [CrossRef]
8. Mahi, M.; Baykan, Ö.K.; Kodaz, H. A new hybrid method based on Particle Swarm Optimization, Ant Colony Optimization and 3-Opt algorithms for Traveling Salesman Problem. *Appl. Soft Comput.* **2015**, *30*, 484–490. [CrossRef]
9. Yang, Z.; Xiao, M.Q.; Ge, Y.W.; Feng, D.L.; Zhang, L.; Song, H.F.; Tang, X.L. A double-loop hybrid algorithm for the traveling salesman problem with arbitrary neighbourhoods. *Eur. J. Oper. Res.* **2018**, *265*, 65–80. [CrossRef]
10. Geng, X.; Chen, Z.; Yang, W.; Shi, D.; Zhao, K. Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search. *Appl. Soft Comput.* **2011**, *11*, 3680–3689. [CrossRef]
11. Ebadinezhad, S. DEACO: Adopting dynamic evaporation strategy to enhance ACO algorithm for the traveling salesman problem. *Eng. Appl. Artif. Intell.* **2020**, *92*, 103649. [CrossRef]
12. Gülcü, Ş.; Mahi, M.; Baykan, Ö.K.; Kodaz, H. A parallel cooperative hybrid method based on ant colony optimization and 3-Opt algorithm for solving traveling salesman problem. *Soft Comput.* **2018**, *22*, 1669–1685. [CrossRef]
13. Zhang, Z.; Xu, Z.; Luan, S.; Li, X.; Sun, Y. Opposition-based ant colony optimization algorithm for the traveling salesman problem. *Mathematics* **2020**, *8*, 1650. [CrossRef]
14. Shahadat, A.S.B.; Akhand, M.; Kamal, M.A.S. Visibility Adaptation in Ant Colony Optimization for Solving Traveling Salesman Problem. *Mathematics* **2022**, *10*, 2448. [CrossRef]
15. Dong, Y.; Wu, Q.; Wen, J. An improved shuffled frog-leaping algorithm for the minmax multiple traveling salesman problem. *Neural Comput. Appl.* **2021**, *33*, 17057–17069. [CrossRef]
16. Zhong, Y.; Lin, J.; Wang, L.; Zhang, H. Hybrid discrete artificial bee colony algorithm with threshold acceptance criterion for traveling salesman problem. *Inf. Sci.* **2017**, *421*, 70–84. [CrossRef]
17. Choong, S.S.; Wong, L.P.; Lim, C.P. An artificial bee colony algorithm with a Modified Choice Function for the traveling salesman problem. *Swarm Evol. Comput.* **2019**, *44*, 622–635. [CrossRef]
18. Khan, I.; Maiti, M.K. A swap sequence based Artificial Bee Colony algorithm for Traveling Salesman Problem. *Swarm Evol. Comput.* **2019**, *44*, 428–438. [CrossRef]
19. Karaboga, D.; Gorkemli, B. Solving Traveling Salesman Problem by Using Combinatorial Artificial Bee Colony Algorithms. *Int. J. Artif. Intell. Tools* **2019**, *28*, 1950004. [CrossRef]
20. Pandiri, V.; Singh, A. A hyper-heuristic based artificial bee colony algorithm for k-Interconnected multi-depot multi-traveling salesman problem. *Inf. Sci.* **2018**, *463-464*, 261–281. [CrossRef]
21. Venkata Rao, R. Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems. *Int. J. Ind. Eng. Comput.* **2016**, *7*, 19–34. [CrossRef]
22. Li, J.Q.; Deng, J.W.; Li, C.Y.; Han, Y.Y.; Tian, J.; Zhang, B.; Wang, C.G. An improved Jaya algorithm for solving the flexible job shop scheduling problem with transportation and setup times. *Knowl.-Based Syst.* **2020**, *200*, 106032. [CrossRef]
23. Thirumoorthy, K.; Muneeswaran, K. A hybrid approach for text document clustering using Jaya optimization algorithm. *Expert Syst. Appl.* **2021**, *178*, 115040. [CrossRef]
24. Xiong, G.; Zhang, J.; Shi, D.; Zhu, L.; Yuan, X. Optimal identification of solid oxide fuel cell parameters using a competitive hybrid differential evolution and Jaya algorithm. *Int. J. Hydrogen Energy* **2021**, *46*, 6720–6733. [CrossRef]
25. Chaudhuri, A.; Sahu, T.P. A hybrid feature selection method based on Binary Jaya algorithm for micro-array data classification. *Comput. Electr. Eng.* **2021**, *90*, 106963. [CrossRef]

26. Chong, K.L.; Lai, S.H.; Ahmed, A.N.; Wan Jaafar, W.Z.; El-Shafie, A. Optimization of hydropower reservoir operation based on hedging policy using Jaya algorithm. *Appl. Soft Comput.* **2021**, *106*, 107325. [CrossRef]
27. Gunduz, M.; Aslan, M. DJAYA: A discrete Jaya algorithm for solving traveling salesman problem. *Appl. Soft Comput.* **2021**, *105*, 107275. [CrossRef]
28. Cinar, A.C.; Korkmaz, S.; Kiran, M.S. A discrete tree-seed algorithm for solving symmetric traveling salesman problem. *Eng. Sci. Technol. Int. J.* **2020**, *23*, 879–890. [CrossRef]
29. Reinelt, G. TSPLIB—A Traveling Salesman Problem Library. *ORSA J. Comput.* **1991**, *3*, 376–384. [CrossRef]
30. Hatamlou, A. Solving travelling salesman problem using black hole algorithm. *Soft Comput.* **2018**, *22*, 8167–8175. [CrossRef]
31. Zhang, Z.; Han, Y. Discrete sparrow search algorithm for symmetric traveling salesman problem. *Appl. Soft Comput.* **2022**, *118*, 108469. [CrossRef]
32. Zheng, J.; Hong, Y.; Xu, W.; Li, W.; Chen, Y. An effective iterated two-stage heuristic algorithm for the multiple Traveling Salesmen Problem. *Comput. Oper. Res.* **2022**, *143*, 105772. [CrossRef]
33. Liu, Y.; Xu, L.; Han, Y.; Zeng, X.; Yen, G.G.; Ishibuchi, H. Evolutionary Multimodal Multiobjective Optimization for Traveling Salesman Problems. *IEEE Trans. Evol. Comput.* **2023**. [CrossRef]
34. Tsai, C.H.; Lin, Y.D.; Yang, C.H.; Wang, C.K.; Chiang, L.C.; Chiang, P.J. A Biogeography-Based Optimization with a Greedy Randomized Adaptive Search Procedure and the 2-Opt Algorithm for the Traveling Salesman Problem. *Sustainability* **2023**, *15*, 5111. [CrossRef]
35. Baraglia, R.; Hidalgo, J.; Perego, R. A hybrid heuristic for the traveling salesman problem. *IEEE Trans. Evol. Comput.* **2001**, *5*, 613–622. [CrossRef]
36. Aslan, M.; Gunduz, M.; Kiran, M.S. JayaX: Jaya algorithm with xor operator for binary optimization. *Appl. Soft Comput.* **2019**, *82*, 105576. [CrossRef]
37. Rao, R.; More, K. Design optimization and analysis of selected thermal devices using self-adaptive Jaya algorithm. *Energy Convers. Manag.* **2017**, *140*, 24–35. [CrossRef]
38. Pradhan, C.; Bhende, C.N. Online load frequency control in wind integrated power systems using modified Jaya optimization. *Eng. Appl. Artif. Intell.* **2019**, *77*, 212–228. [CrossRef]
39. Wang, L.; Zhang, Z.; Huang, C.; Tsui, K.L. A GPU-accelerated parallel Jaya algorithm for efficiently estimating Li-ion battery model parameters. *Appl. Soft Comput.* **2018**, *65*, 12–20. [CrossRef]
40. Mazyavkina, N.; Sviridov, S.; Ivanov, S.; Burnaev, E. Reinforcement learning for combinatorial optimization: A survey. *Comput. Oper. Res.* **2021**, *134*, 105400. [CrossRef]
41. Gündüz, M.; Kiran, M.S.; Özceylan, E. A hierarchic approach based on swarm intelligence to solve the traveling salesman problem. *Turk. J. Electr. Eng. Comput. Sci.* **2015**, *23*, 103–117. [CrossRef]