Reinforcement Learning for the Traveling Salesman Problem: Performance Comparison of Three Algorithms

Jiaying Wang, Chenglong Xiao, Shanshan Wang, and Yaqi Ruan

***Abstract*—TSP is one of the most famous problems in graph theory, as well as one of the typical NP-hard problems in combinatorial optimization. Its applications range from how to plan the most reasonable and efficient road traffic to how to better set up nodes in the Internet environment to facilitate information flow, among others. Reinforcement learning has been widely regarded as an effective tool for solving combinatorial optimization problems. This paper attempts to solve the TSP problem using different reinforcement learning algorithms and evaluated the performance of three RL algorithms (Q-learning, Sarsa, and Double Q-Learning) under different reward functions, ε-greedy decay strategies, and running times. The results show that the Double Q-Learning algorithm is the best algorithm, as it could produce results closest to the optimal solutions, and by analyzing the results, better reward strategies and epsilon-greedy decay strategies are obtained.**

***Index Terms*—Travelling Salesman Problem, Reinforcement Learning, Q-learning algorithm, Sarsa algorithm, Double Q-Learning algorithm, the reward function.**

I. Introduction

Combinatorial optimization is a fundamental problem in computer science that aims to find the best combination of variables [1], [11]. The Travelling Salesman Problem (TSP) is a classic example of a combinatorial optimization problem that holds great importance in operations research and theoretical computer science. TSP is of practical importance, having originated in transportation applications such as aircraft routing, mail delivery, courier services, and designing school bus routes. Today, TSP and its variants have wide-ranging applications in various fields such as mathematics, computer science, operations research, genetics, engineering, and electronics. For instance, it is used for machine scheduling problems [2], robot path planning [10], [14], vehicle routing [8], production planning [7], service time optimization [9], hardware device design, and computer networks.

Several methods exist for solving the traveling salesman problem. Some of the most commonly used ones include Simulated Annealing (SA) [12], Tabu Search (TS) [3], Genetic Algorithm (GA) [4], and Ant Colony Optimization (ACO) [5]. In recent years, reinforcement learning has also emerged as a promising approach for tackling TSP problems.

Reinforcement learning is a machine learning paradigm that relies on trial-and-error learning based on the Markov decision-making process. It is used to teach agents how to optimize their behavior and achieve specific goals while interacting with their environment [6]. This paper investigates how different reward functions and ε-greed decay strategies impact the performance of reinforcement learning algorithms. And it also compares the performance of three popular reinforcement learning algorithms, namely q-learning, Double Q-Learning, and Sarsa, in solving the traveling salesman problem (TSP). The main objective of the experiment is to investigate the effects of these factors on algorithm performance, rather than to find the optimal solution to the TSP.

The rest of this paper is organized as follows: Section II describes related work on the application of reinforcement learning to the TSP problem. Section III then presents an overview of three reinforcement learning algorithms for TSP. Section IV compares the experimental results and provides analysis of the results. Section V gives the conclusion and a future outlook.

II. Related Works

When initially tackling combinatorial optimization problems, researchers apply various heuristic algorithms to solve them. In 2019, Hanif Halim summarized the performance of various heuristic algorithms on TSP problems [12].Six heuristic algorithms are included: Tabu Search, Genetic Algorithm, Simulated Annealing, Nearest Neighbor, Tree Physiology Optimization and Ant Colony Optimization. The computational time, accuracy and convergence of these algorithms are compared.

Recently, with the development and application of new algorithms in the field of RL, researchers have begun to study the ability of RL to solve combinatorial optimization problems. One popular approach is Q-learning, which has been used in several TSP studies, such as [15] and [16] . Q-learning is a model-free RL algorithm that learns the optimal Q-value function through trial and error. It has been shown to achieve good results in TSP with small-scale problems, but its performance can degrade as the problem size increases.Another common RL algorithm for TSP is the Sarsa algorithm, which is proposed in [13]. The Sarsa algorithm is also a model-free algorithm that updates Q-values based on the current state-action-reward-next state transition. Sarsa has been shown to be effective in small-scale TSP problems, but it can suffer from slow convergence and poor performance on larger problems [30].

Afterwards, reinforcement learning is used in combination with other methods to solve tsp problems, and Bello et al. from Google Brain's team proposed to use neural networks and RL to solve combinatorial optimization problems, using TSP as

an example [17].Experiments have shown that neural combinatorial optimisation achieves near-optimal results. Zheng et al. [34] combined Q-learning, Sarsa and Monte Carlo with LKH and proposed a new algorithm called Variable Strategy Reinforced Lin–Kernighan–Helsgaun (VSR-LKH) algorithm.

In the study of reinforcement learning, one of the most critical issues remains parameter estimation. In research on this issue, André L. C. Ottoni et al. have carried out a lot of work on the effect of hyperparameters on reinforcement learning algorithms. Although RL is stochastic and TSP is combinatorial, they are unlikely at first glance to determine any relationship between parameters and results. However, Ottoni's experiments show that response surface models are usually able to determine such relationships. The response surface method is first used to reflect the influence of the learning rate α and the discount coefficient gamma on the reinforcement learning algorithm [18]. They then conducted experiments using response surface models to estimate better hyperparameters [19].

Despite the fact that several reinforcement learning algorithms have been used to solve the TSP problem, there are no comparisons between these popular reinforcement learning methods in the available literature. As a result, the major goal of this work is to adapt the Q-learning, SARSA, and Double Q-Learning algorithms to solve the TSP problem and evaluate the three methods with different learning specifications in terms of runtime and result quality.

III. Three Reinforcement Learning Algorithms for TSP Problem

A. Problem Definition

The traveling salesman problem (TSP) is a fundamental problem in computer science that aims to find the shortest possible path that visits every city (C_1,C_2, $\cdots$,C_n) exactly once and returns to the starting city [20]. This problem can be modeled using an undirected weighted graph where the cities are represented as vertices and the roads as edges, with the distance of each road being the weight of the corresponding edge. The objective is to minimize the total weight of the edges in the path while ensuring that each vertex is visited exactly once.

TSP problems can be classified into two types: symmetric TSP problems and asymmetric TSP problems [21].

In the case of the symmetric TSP problem, the distance between each pair of cities is equal, resulting in an undirected graph. The goal of the problem is to find a Hamiltonian cycle with the minimum weight in a complete weighted undirected graph.

On the other hand, in the asymmetric TSP problem, the distances between pairs of cities are not equal or there are one-way paths. Thus, the undirected graph transforms into a directed graph. Due to its asymmetry, ATSP is more challenging to solve. In reality, the majority of TSP problems are asymmetric, and therefore, ATSP problems have more practical applications.

Algorithm 1: Q-learning Algorithm

```
Set the parameters: α, γ and ε
Initialize the matrix Q(s, a)
Observe the state s
repeat
    Take action a using the ε-greedy method
    Receive immediate reward r(s, a),Observe the new state s'
    Update Q(s, a) with Eq. (1)
    s = s'
until the stopping criterion is satisfied;
```

B. Reinforcement Learning

When applying the reinforcement learning algorithm to the TSP problem, the agent represents the traveler, the environment represents the cities to visit, the state represents the cities that have been visited, the action represents the next city to visit, and the reward represents the distance between the two cities. The ε-greedy strategy is commonly used for action selection [23].

Next, it will briefly introduce the three reinforcement learning algorithms adapted to TSP problems.

1) Q-Learning: The Q-Learning algorithm is first proposed by Watkins in [24]. It updates the Q matrix using the following formula:

$$Q_{t+1} = Q_t(s,a) + \alpha[r(s,a) + \gamma max_{a'} Q(s',a') - Q_t(s,a)] \tag{1}$$

The algorithm stores action values Q(s, a) by creating a table based on the reward r earned by the agent for taking an action in a particular state [25]. This table is known as the Q-table. Each cell in the table corresponds to a state S and an action taken at time step t. Initially, all values in the table are set to 0. Each time the agent accesses a state and takes an action, the Q table is updated using the Bellman equation described in Equation (1). Algorithm 1 shows the Q-learning process.

2) Sarsa: The Sarsa algorithm is similar to the Q-Learning algorithm, but the main difference is the way the Q value is updated [29]. The Sarasa matrix is updated as follows (2):

$$Q_{t+1} = Q_t(s,a) + \alpha[r(s,a) + \gamma Q(s',a') - Q_t(s,a)] \tag{2}$$

The difference between Equation (1) and Equation (2) is that the former uses the maximum value of the next state-action, while the latter uses the current state-action. In other words, the Q value is updated using the Q value of the next state-action for Q-Learning, while for Sarsa, the Q value is updated using the Q value of the current state-action [26]. The pseudo code of Sarsa algorithm is presented in Algorithm 2.

3) Double Q-Learning: The Double Q-Learning algorithm is an improvement over the Q-Learning algorithm. Q-Learning sometimes has the problem of overestimation, and the Double Q-Learning algorithm can solve this problem [27], [32]. Equation (3)(4) is an update function of the Double Q-Learning algorithm:

Algorithm 2: Sarsa

Set the parameters: α, γ and ε
Initialize the matrix $Q(s, a)$
Observe the state s
Chooses the action a using the ε-greedy method
repeat
  Take the action a
  Receive immediate reward $r(s, a)$,Observe the new state s'
  Choose the new action a using ε-greedy method
  Update $Q(s, a)$ with Eq. (2)
  $s = s', a = a'$
until *the stopping criterion is satisfied*;

Algorithm 3: Double Q-Learning

Set the parameters: α, γ and ε
Initialize the matrix $Q^A(s, a) = 0$, $Q^B(s, a) = 0$
Observe the state s
repeat
  Take action a(e.g.ε-greedy) based on Q^A, Q^B
  Receive immediate reward $r(s, a)$,Observe the new state s'
  Choose (e.g.random)either UPDATE(A) or UPDATE(B)
  if *UPDATE(A)* **then**
    Update Q^A with Eq. (3)
  if *UPDATE(B)* **then**
    Update Q^B with Eq. (4)
  $s = s'$
until *end*;

$$Q^A_{t+1} = Q^A_t(s,a) + \alpha[r(s,a) + \gamma max_{a'} Q^B(s',a') - Q^A_t(s,a)] \tag{3}$$

$$Q^B_{t+1} = Q^B_t(s,a) + \alpha[r(s,a) + \gamma max_{a'} Q^A(s',a') - Q^B_t(s,a)] \tag{4}$$

There are two functions in Double Q-Learning, $Q^A$and Q^B (Double Q-Learning), and each function updates the next state with the value of the other Q function. When selecting an action, it is determined based on the average of $Q^A$and Q^B. The pseudo code of Double Q-Learning algorithm is presented in Algorihtm 3.

4) ε-greedy: During the process of reinforcement learning, it is important to balance between exploitation and exploration. This means that the algorithm cannot rely solely on the values in the Q table and must continue to explore new actions, rather than solely relying on past experience. The ε-greedy strategy is used to determine when to exploit the Q table and when to explore randomly [28].

The ε-greedy strategy selects the action with the highest Q value with a probability of 1-ε, and a random action with a probability of ε. The value of ε is initially set to a high value to encourage exploration, but it is gradually decreased as the learning progresses to encourage exploitation of the Q table. The value of ε can be adjusted depending on the specific problem being solved.

This strategy is represented mathematically in Equation (5):

$$\begin{cases} max_a Q(s,a), 1-\varepsilon \\ randomly, \varepsilon \end{cases} \tag{5}$$

As the number of cycles increases, the value of ε gradually decreases. Initially, route selection is completely random, but as reinforcement learning progresses, the value of ε decreases, and the selection becomes less random and more biased towards selecting the maximum value. Finally, reinforcement learning relies entirely on the maximum value to choose the next route.

C. Reinforcement Learning

When applying reinforcement learning to TSP problems, it is necessary to define a model that includes states, actions, and rewards. In this paper, The model is defined as follows:

- State: The state of the agent is defined as the current city the agent is visiting. In other words, the state of the agent is the city that has been visited by the agent so far.
- Action: The action of the agent is defined as the next city to visit [18]. In other words, the agent chooses an action by selecting the next city to visit from the set of unvisited cities.
- Reward: In TSP, the reward is related to the distance traveled between cities. The goal of the agent is to minimize the total distance traveled, so the reward function is defined as the negative distance between the current city and the next city to visit. The negative distance is used to align the goal of the agent with the reward signal, which is to maximize the reward.Three reward function used in this paper are as follows:

$$R_1 = 1/d_{ij} \tag{6}$$

$$R_2 = -d_{ij} \tag{7}$$

$$R_3 = -(d_{ij})^2 \tag{8}$$

d_{ij} is the length between two city nodes. Equation(6) takes the reciprocal of the distance [23], [30], Equation(7) takes the negative number of the distance [30], and Equation(8) takes the negative number of the square of the distance [30].

IV. Experimental Results and Analysis

A. Experiment Setup

All the algorithms were implemented in Python. The experiments were conducted on an Intel Core i7 2.0GHz CPU with 16 Gbytes main memory.

1) Experimental Data: TSPLIB [22] is a database of instances of TSP and related problems from various sources and of various types, such as TSP and ATSP. The library provides many instances of different complexities and gives their optimal solutions. The TSP instances selected for this experiment are all from TSPLIB, including five TSP problems and three ATSP problems.

TABLE I
TSP INSTANCES

Type	Instance	n	Optimal Solution
TSP	eil51	51	426
	berlin52	52	7542
	st70	70	675
	kroA100	100	21282
	tsp225	225	3919
ATSP	br17	17	39
	ftv64	64	1839
	ftv170	170	2755

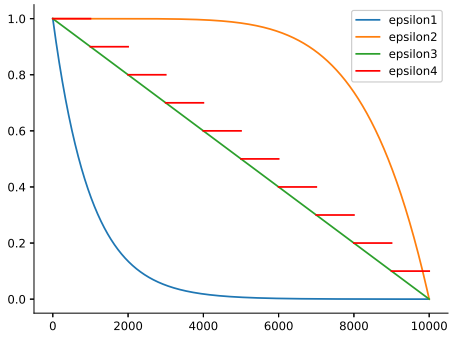

Fig. 1. The corresponding curves of four ε-greedy strategies

The selected TSP instances are shown in Table I. The instances range from smaller ones, such as br17, to larger ones, such as ftv170 and tsp225, which contain a greater number of cities.

2) Algorithm Specifications: The following is a comparison and analysis of the performance of reinforcement learning algorithms under three different conditions:

- Different algorithms: Q-Learning, Sarsa, and Double Q-Learning are three widely used reinforcement learning algorithms. Q-Learning and Double Q-Learning are both off-policy algorithms, while Sarsa is an on-policy algorithm;
- Different reward functions: The reward function plays an important role in guiding the agent towards finding an optimal solution. R1, R2, and R3 are three different reward functions used in this experiment;
- Different ε-greedy: The ε-greedy strategy is a popular exploration-exploitation method used in reinforcement learning. In this experiment, four different dynamic decay methods are used to adjust the value of ε during the learning process:
  1) $\varepsilon = 1 - i/N$. Decrease the ε linearly;
  2) $\varepsilon = 0.999^i$. ε is a concave function [28];
  3) $\varepsilon = -(i/N)^6 + 1$. ε is a convex function;
  4) The ε is initially 1 and decreases by 0.1 after every thousand pieces of training.

The current number of training is denoted by i, and N represents the total number of training sessions. The advantage of using dynamic decay to set the value of ε is that it increases the likelihood of exploration at the beginning of the run, while gradually favoring the maximum value selection as the number of runs increases and the Q table becomes more refined.

The experiment consisted of 36 combinations $(3{*}3{*}4)$, with different combinations of reinforcement learning algorithms, reward functions, and ε-greedy decay methods. The learning rate (α) is set to 0.01, and the discount coefficient (γ) is varied based on the problem instance: (0.01, 0.15, 0.3, 0.45, 0.6, 0.75, 0.9, 0.99) [18] for eil51, berlin52 TSP instances, and br17, ftv64 ATSP instances, and (0.01, 0.15, 0.3, 0.45) for st70, kroA100, tsp225 TSP instances, and ftv170 ATSP instance.

The number of simulations varied based on the number of cities in the problem instance. For instances with a small number of cities, each combination is simulated 10 times, while for larger instances, each combination is simulated 5 times. The number of training sessions per simulation is set to 10,000, and the result of each simulation is the distance traveled by the agent completing a lap back to the starting point. The number of simulations is reduced for larger instances to reduce the time cost.

The optimal values, averages, and running times of all simulation results were compared to determine the best combination of reinforcement learning algorithms, reward functions, and ε-greedy decay methods for each problem instance.

B. Results and Analysis

In this section, the experimental results are compared and analyzed in terms of the following indicators:

1) Optimal operating results;
2) Average operating results;
3) Running time.

1) Comparison of the best results: Table II shows the shortest distances obtained by the three algorithms for each problem instance. Table III shows the combination of the optimal reward function and ε-greedy when the result is achieved. The data in the table indicates that in most cases (berlin52, st70, kroA100, tsp225), the Double Q-Learning and Q-Learning algorithms outperform Sarsa, particularly when the number of cities is larger. Furthermore, in most instances, the optimal value is obtained with (R_1), with the br17 instance showing the most significant difference in results. For this instance, all three algorithms achieve the optimal solution (R_1), while the optimal solution cannot be obtained using the other two reward functions, even with the Q-learning algorithm using any ε-greedy decay strategy.

Figure 2 shows the range of best values achieved by the three algorithms for different TSP problems with 12 different reward functions and ε-greedy combinations. As can be seen from the figures, Double Q-Learning is a very unstable algorithm, which can achieve a better solution while at the same time may fetch a very poor solution. In contrast, the Q-Learning and Sarsa algorithms are relatively much more stable, with the difference between the optimal and worst solutions being relatively small.

2) Comparison of the average results: Table IV is the average result of the three algorithms in 12 different combinations.Table V is the combination of reward function and decay strategy for optimal value.For the analysis of the average

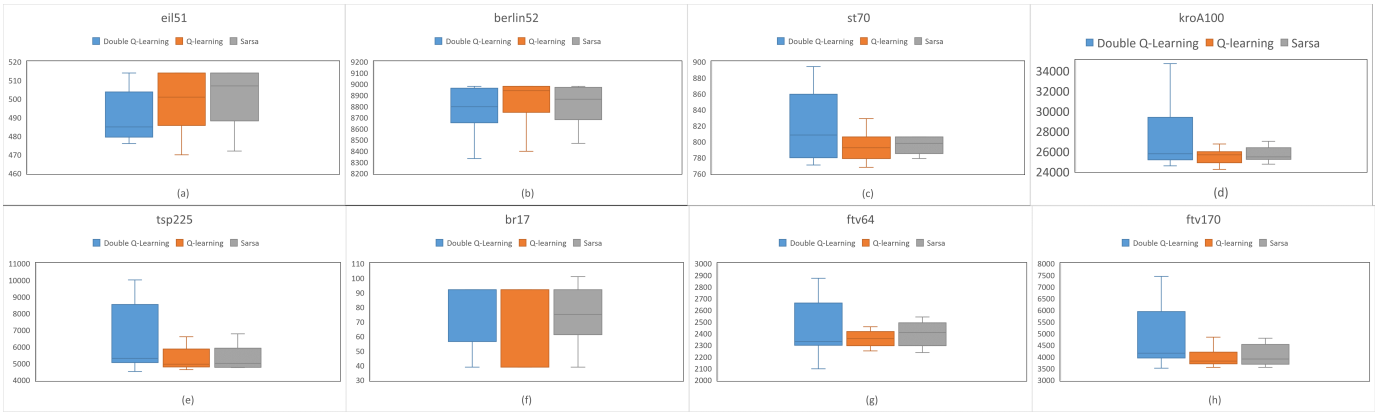

Fig. 2. (a)-(h) The range of best values achieved by the three algorithms for different TSP problems with 12 different reward functions and ε-greedy combinations

TABLE II
THE OPTIMAL RESULT OF THE ALGORITHM

Algorithm	eil51	berlin52	st70	kroA100	tsp225	br17	ftv64	ftv170
Double Q-Learning	476	8141	771	24547	4519	39	2098	3512
Q-Learning	470	8215	768	24187	4634	39	2251	3553
Sarsa	472	8470	779	24715	4760	39	2237	3552

TABLE III
THE COMBINATION OF REWARD FUNCTION AND DECAY STRATEGY WHEN THE OPTIMAL VALUE IS OBTAINED

Algorithm	eil51	berlin52	st70	kroA100	tsp225	br17	ftv64	ftv170
Double Q-Learning	R_1/ε_1	R_3/ε_2	R_1/ε_4	R_1/ε_1	R_1/ε_4	R_1/ε_2	R_1/ε_4	R_1/ε_3
Q-Learning	R_1/ε_1	R_1/ε_2	R_2/ε_2	R_1/ε_1	R_3/ε_4	R_1/all	R_1/ε_2	R_1/ε_1
Sarsa	$R_3/\varepsilon_1, \varepsilon_3$	R_1/ε_2	R_3/ε_3	R_1/ε_3	R_3/ε_2	R_1/ε_2	R_1/ε_1	R_1/ε_3

results, it is necessary to divide them according to the TSP and ATSP problems.

Firstly, for the TSP problem, Table IV shows that although the optimal value can be obtained using (R_1) for the first two algorithms, the optimal average result is usually achieved when using (R_3). By observing Table V,preliminary analysis suggests that although R_1 can yield better results, it is not stable and requires adjustments to the learning rate and discount coefficient to find the optimal value. On the other hand, R_3 is more stable and can produce similar results for different learning rates and discount coefficients. For the Sarsa algorithm, it is apparent that the best results are obtained when using ε_1 in combination with ε_3. Therefore, for the Sarsa algorithm, R_1 is the optimal reward function.

Secondly, for the ATSP problem, the results for the instance with a small number of cities are similar to those for the TSP problem, where better averages can be obtained with R_3. However, as the number of cities increases, the optimal average value can be obtained in the ftv170 problem under the same conditions of R1 and ε_3. Therefore, it can be concluded that for the ATSP problems with a large number of cities, the optimal reward function for the three algorithms is R_1.

3) Comparison of running time: The average running time of the three algorithms on kroA100 and ATSP's ftv64 is shown in Figure 3 and Figure 4 respectively.The running time results show that the Q-Learning algorithm has the shortest running time among the three algorithms, while Sarsa had the

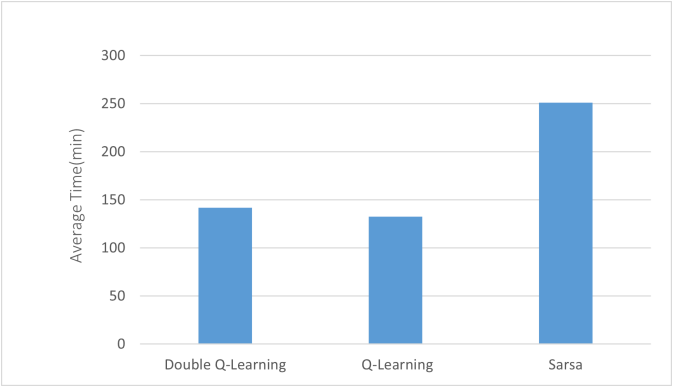

Fig. 3. The average running time of the three algorithms on instance kroA100

longest running time. When comparing the running time of different reward functions and ε-greedy strategies for the same algorithm, it can observe that there is no significant difference between changing only the reward function. However, the second ε-greedy strategy required the least running time, while the third ε-greedy took the most running time, with the other two in between. The running times of different combinations on the benchmarks kroA100 and ATSP's ftv64 are shown in the table VI and table VII respectively.

Sarsa algorithm may require longer running time in some cases because it needs to perform policy evaluation at every

TABLE IV
OPTIMAL AVERAGE RESULTS

Algorithm	eil51	berlin52	st70	kroA100	tsp225	br17	ftv64	ftv170
Double Q-Learning	512	8981	823	26858	5494	92	2609	4065
Q-Learning	506	8956	806	27137	5049	92	2540	4116
Sarsa	533	9932	849	28633	5270	107	2687	3908

TABLE V
THE COMBINATION OF REWARD FUNCTION AND EPSILON-GREEDY STRATEGY WHEN THE OPTIMAL AVERAGE VALUE IS OBTAINED

Algorithm	eil51	berlin52	st70	kroA100	tsp225	br17	ftv64	ftv170
Double Q-Learning	R_3/ε_1	R_3/ε_2	R_3/ε_2	R_3/ε_1	R_1/ε_3	$R_2, R_3/all$	R_3/ε_2	R_1/ε_3
Q-Learning	R_1/ε_1	R_3/ε_3	$R_3/\varepsilon_1, \varepsilon_2, \varepsilon_4$	R_3/ε_3	R_3/ε_4	$R_2, R_3/all$	R_3/ε_2	R_1/ε_3
Sarsa	R_1/ε_3	R_1/ε_3	R_1/ε_3	R_1/ε_3	R_1/ε_3	R_2/ε_4	R_1/ε_3	R_1/ε_3

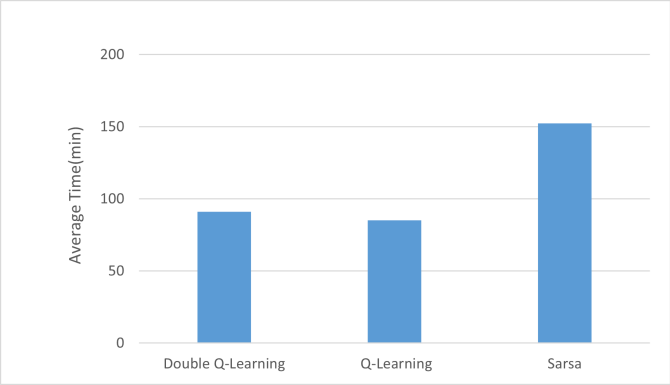

Fig. 4. The average running time of the three algorithms on instance ftv64

TABLE VI
RUNNING TIME OF DIFFERENT COMBINATIONS ON THE BENCHMARK KROA100(IN MIN)

Algo.	R	ε	Time
Q-learning	R_1	ε_1	134.4
		ε_2	55.3
		ε_3	211.8
		ε_4	132.9
	R_2	ε_1	141.4
		ε_2	51.5
		ε_3	209.8
		ε_4	130.2
	R_3	ε_1	144.4
		ε_2	55.1
		ε_3	198.5
		ε_4	125.0
Sarsa	R_1	ε_1	272.3
		ε_2	98.8
		ε_3	379.0
		ε_4	252.2
	R_2	ε_1	274.7
		ε_2	98.1
		ε_3	378.6
		ε_4	234.0
	R_3	ε_1	273.7
		ε_2	99.6
		ε_3	413.1
		ε_4	239.9
Double Q-Learning	R_1	ε_1	151.0
		ε_2	63.0
		ε_3	200.0
		ε_4	130.4
	R_2	ε_1	143.1
		ε_2	67.4
		ε_3	200.0
		ε_4	130.0
	R_3	ε_1	141.1
		ε_2	64.5
		ε_3	200.5
		ε_4	132.7

step, and it is an online algorithm that requires interaction with the environment at every step. On the other hand, Q-learning algorithm relies more on offline policy evaluation and policy improvement, so it may require fewer interaction steps in some cases. However, this also depends on the specific implementation and parameters used in the algorithm, so in practical applications, different algorithms need to be compared and evaluated based on the specific circumstances.

After conducting the above analyses, it can be concluded that both the Double Q-Learning algorithm and the Q-Learning algorithm outperform the Sarsa algorithm on the Traveling Salesman Problem, both in terms of quality of results and running time.For the first two algorithms, the Double Q-Learning algorithm is easier to obtain better solutions. Regarding the reward functions and decay strategies, it is clear that R_1 provides a greater chance of finding a better solution for any decay strategy, but only by a small margin. While the different decay strategies have very little difference in results, with ε_1, we can achieve a slightly better results. However, different decay strategies can result in significant different running time. Overall, ε_2 leads to the shortest running time.

V. CONCLUSION

This paper investigates the application of reinforcement learning to the Traveling Salesman Problem (TSP) and compares the performance of three algorithms under various learning specifications through statistical experiments. The study analyzes the algorithms' performance and identifies the most suitable algorithm and its parameters for TSP.

The findings indicate that reinforcement learning algorithms perform better when using a reward function of $R_1 = 1/d_{ij}$. However, there is no significant difference in performance when using different ε-greedy decay strategies, as long as the value of ε decreases gradually. The Sarsa algorithm's performance on TSP is inferior to the other two reinforcement learning algorithms, and the other two algorithms' performance is not significantly different. While finding relatively better results, the running time is not significantly different.

Future studies could explore the reasons for the lack of significant differences among ε-greedy decay strategies. In

TABLE VII
RUNNING TIME OF DIFFERENT COMBINATIONS ON THE BENCHMARK FTV64

Algo.	R	ε	Time
Q-learning	R_1	ε_1	86.4
		ε_2	40.8
		ε_3	124.5
		ε_4	78.6
	R_2	ε_1	86.6
		ε_2	41.0
		ε_3	130.3
		ε_4	84.9
	R_3	ε_1	91.4
		ε_2	40.5
		ε_3	133.4
		ε_4	84.5
Sarsa	R_1	ε_1	149.6
		ε_2	61.9
		ε_3	255.1
		ε_4	158.3
	R_2	ε_1	158.5
		ε_2	65.1
		ε_3	229.1
		ε_4	143.7
	R_3	ε_1	166.7
		ε_2	65.3
		ε_3	228.3
		ε_4	144.7
Double Q-Learning	R_1	ε_1	91.8
		ε_2	50.3
		ε_3	139.0
		ε_4	97.7
	R_2	ε_1	95.2
		ε_2	47.0
		ε_3	130.3
		ε_4	83.2
	R_3	ε_1	92.3
		ε_2	46.0
		ε_3	130.4
		ε_4	88.5

addition, current algorithms still have some limitations in solving large-scale problems, so more efficient algorithms can be explored, such as combining reinforcement learning with neural networks and deep learning.

REFERENCES

[1] Luke, Sean. Essentials of metaheuristics. Vol. 2. Raleigh: Lulu, 2013.

[2] Cunha, Bruno, et al. "Deep reinforcement learning as a job shop scheduling solver: A literature review." Hybrid Intelligent Systems: 18th International Conference on Hybrid Intelligent Systems (HIS 2018) Held in Porto, Portugal, December 13-15, 2018 18. Springer International Publishing, 2020.

[3] Osaba, Eneko, et al. "Hybrid quantum computing-tabu search algorithm for partitioning problems: preliminary study on the traveling salesman problem." 2021 IEEE Congress on Evolutionary Computation (CEC). IEEE, 2021.

[4] Gharib, Abdelhakim, Jamal Benhra, and Mohsine Chaouqi. "A performance comparison of PSO and GA applied to TSP." International Journal of Computer Applications 130.15 (2015): 34-39.

[5] Haroun, Sabry Ahmed, and Benhra Jamal. "A performance comparison of GA and ACO applied to TSP." International Journal of Computer Applications 117.20 (2015).

[6] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

[7] Crama, Yves, Joris Van De Klundert, and Frits CR Spieksma. "Production planning problems in printed circuit board assembly." Discrete Applied Mathematics 123.1-3 (2002): 339-361.

[8] Bertsimas, Dimitris J., and David Simchi-Levi. "A new generation of vehicle routing research: robust algorithms, addressing uncertainty." Operations research 44.2 (1996): 286-304.

[9] Chang, Tsung-Sheng, Yat-wah Wan, and Wei Tsang Ooi. "A stochastic dynamic traveling salesman problem with hard time windows." European Journal of Operational Research 198.3 (2009): 748-759.

[10] Sheng, Weihua, et al. "Robot path planning for dimensional measurement in automotive manufacturing." J. Manuf. Sci. Eng. 127.2 (2005): 420-428.

[11] Davendra, Donald, ed. Traveling salesman problem: Theory and applications. BoD–Books on Demand, 2010.

[12] Halim, A. Hanif, and IJAoCMiE Ismail. "Combinatorial optimization: comparison of heuristic algorithms in travelling salesman problem." Archives of Computational Methods in Engineering 26 (2019): 367-380.

[13] Fedorov, Eugene, and Olga Nechyporenko. "Methods for Solving the Traveling Salesman Problem Based on Reinforcement Learning and Metaheuristics." (2022).

[14] Le, Anh Vu, et al. "Coverage path planning using reinforcement learning-based TSP for hTetran—a polyabolo-inspired self-reconfigurable tiling robot." Sensors 21.8 (2021): 2577.

[15] Gambardella, Luca M., and Marco Dorigo. "Ant-Q: A reinforcement learning approach to the traveling salesman problem." Machine learning proceedings 1995. Morgan Kaufmann, 1995. 252-260.

[16] Liu, Fei, and Guangzhou Zeng. "Study of genetic algorithm with reinforcement learning to solve the TSP." Expert Systems with Applications 36.3 (2009): 6995-7001.

[17] Bello, Irwan, et al. "Neural combinatorial optimization with reinforcement learning." arXiv preprint arXiv:1611.09940 (2016).

[18] Ottoni, André Luiz Carvalho, et al. "Análise da influência da taxa de aprendizado e do fator de desconto sobre o desempenho dos algoritmos Q-learning e SARSA: aplicação do aprendizado por reforço na navegação autônoma." Revista Brasileira de Computação Aplicada 8.2 (2016): 44-59.

[19] Ottoni, André LC, Erivelton G. Nepomuceno, and Marcos S. de Oliveira. "A response surface model approach to parameter estimation of reinforcement learning for the travelling salesman problem." Journal of Control, Automation and Electrical Systems 29 (2018): 350-359.

[20] Dorigo, Marco, and Luca Maria Gambardella. "Ant colony system: a cooperative learning approach to the traveling salesman problem." IEEE Transactions on evolutionary computation 1.1 (1997): 53-66.

[21] Matai, Rajesh, Surya Prakash Singh, and Murari Lal Mittal. "Traveling salesman problem: an overview of applications, formulations, and solution approaches." Traveling salesman problem, theory and applications 1 (2010).

[22] Reinelt, Gerhard. "TSPLIB—A traveling salesman problem library." ORSA journal on computing 3.4 (1991): 376-384.

[23] Bianchi, Reinaldo AC, Carlos HC Ribeiro, and Anna HR Costa. "On the relation between ant colony optimization and heuristically accelerated reinforcement learning." 1st international workshop on hybrid control of autonomous system. Palo Alto, CA: AAAI, 2009.

[24] Watkins, Christopher John Cornish Hellaby. "Learning from delayed rewards." (1989).

[25] WATKINS, CJCH. "Q-Learning, in Reinforcement Learning." Technical Note (1993): 55-68.

[26] Ma, Jia, et al. "Neurodynamic programming: a case study of the traveling salesman problem." Neural Computing and Applications 17 (2008): 347-355.

[27] Hasselt, Hado. "Double Q-Learning." Advances in neural information processing systems 23 (2010).

[28] Benford, Samuel Levente. Solving the Binary Knapsack Problem Using Tabular and Deep Reinforcement Learning Algorithms. Diss. Northeastern University, 2021.

[29] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

[30] Ottoni, André LC, et al. "Tuning of reinforcement learning parameters applied to sop using the scott–knott method." Soft Computing 24.6 (2020): 4441-4453.

[31] Matai, Rajesh, Surya Prakash Singh, and Murari Lal Mittal. "Traveling salesman problem: an overview of applications, formulations, and solution approaches." Traveling salesman problem, theory and applications 1 (2010).

[32] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with Double Q-Learning." Proceedings of the AAAI conference on artificial intelligence. Vol. 30. No. 1. 2016.

[33] Li, Yuxi. "Deep reinforcement learning: An overview." arXiv preprint arXiv:1701.07274 (2017).

[34] Zheng, Jiongzhi, et al. "Reinforced Lin–Kernighan–Helsgaun algorithms for the traveling salesman problems." Knowledge-Based Systems 260 (2023): 110144.