AI Capstone Final Project
Using Different Methods to Solve TSP

110550088 李杰穎、110652019 林楷傑、110950011 廖家淯

June 16, 2023

1 Background Overview

The Traveling Salesman Problem (TSP) is a well-known optimization problem in the field of computer science and mathematics. It belongs to the category of NP-hard problems, which means that finding an optimal solution becomes increasingly difficult as the problem size increases.

The problem is defined as follows: given a list of cities and the distances between each pair of cities, the task is to find the shortest possible route that visits each city exactly once and returns to the starting city. The objective is to minimize the total distance traveled.

The TSP has been extensively studied since its introduction in the 1930s and has applications in various fields such as logistics, transportation planning, circuit board drilling, and DNA sequencing. Its name was inspired by the analogy to a traveling salesman who needs to find the shortest route to visit a set of cities and return to the starting point.

One of the reasons why the TSP is of great interest to researchers is its computational complexity. As the number of cities increases, the number of possible routes grows factorially, resulting in an exponential increase in computational effort required to find the optimal solution. For large problem instances, it becomes practically impossible to explore all possible solutions.

In summary, the Traveling Salesman Problem is a classic optimization problem that seeks to find the shortest route visiting a set of cities and returning to the starting point. Its computational complexity has driven the development of various algorithms and approaches, making it a fundamental problem in the fields of computer science and mathematics.

2 Selected Paper

In this project, we used three methods to solve TSP, including ant colony optimization (ACO), deep Q-learning (DQN) and firefly algorithm. In this section, we will explain these three methods and their correspond papers respectively.

2.1 ACO

For ACO, we mainly referenced [1], this paper is a overview of using ACO to solve TSP. It introduced three methods of ACO, we will introduce these three methods.

2.1.1 Introduction

Ant Colony Optimization (ACO) is widely used to solve NP-hard optimizing problem, in particular, TSP problem. In the paper, the authors introduced three different ant system methods, including ant system (AS), ant colony system (ACS) and MAX-MIN ant system (MMAS). I will breifly explain how these methods work in the next few sections.

However, because of the complexity of implementation and the fact that MMAS is a improved version of AS, I will only implement MMAS, and only MMAS will be compared with DQN and firefly algorithm.

Before explaining the two methods, I will first introduce the concept of ant system. Ant system is inspired by the behavior of real ant colony. When ants are finding the shortest path from nest to food, they would spread pheromone along the path, so that other ants can follow the path to get the food. Because pheromone can evaporate, after time, longer path will get less pheromone, thus less attractive to other ants. Therefore, after period of time, the ant colony will find the shortest path eventually.

Ant system used artificial pheromone and ants to simulate the behavior of real ant. The pheromone some how reflect their experience while solving a particular problem.

The below notations will be used in the two methods.

- $\tau_{ij}(t)$ is the pheromone strength of arc (i,j) in t iteration
- d_{ij} is the distance from city i to j
- $\eta_{ij}(t)$ is the priori available heuristic value of arc (i,j). Usually set to $1/d_{ij}(t)$

Noted that in this project all the TSP problems are symmetric, i.e. $d_{ij}(t) = d_{ji}(t)$. Therefore, $\tau_{ij}(t)$ is also symmetric, i.e. $\tau_{ij}(t) = \tau_{ji}(t)$.

2.1.2 AS

The are mainly three variants of AS, ant-density, ant-quantiy and ant-cycle. The first two methods update pheromone right after a move from a city to adjacent one. The ant-cycle AS update pheromone after a full tour is completed. Because ant-cycle AS perform much better than the other two variants, the paper used ant-cycle variants.

In AS, each m artificial ants construct a solution to TSP. Noted that local search doesn't apply to original AS.

Tour Construction Initially, each m artificial ants put randoml chosen city. At each construction step, ant k choose the next traverse city by the probability describe as follow:

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}(t)]^\alpha \cdot [\eta_{il}]^\beta} \quad \text{if } j \in \mathcal{N}_i^k \tag{1}$$

We can notice that the arc with more phermone or shorter distance will have more chance to be traversed, and $\mathcal{N}_i^k$ is the is the feasible neighborhood of ant k, that is, the set of cities which ant k has not yet visited. We can also noticed that if $\alpha = 0$, the closest cities are more likely to be selected. And if $\beta = 0$ only pheromone amplification is at work: this method will lead to the rapid emergence of a stagnation situation with the corresponding generation of tours which, in general, are strongly suboptimal.

Pheromone update After all ants have constructed their tours, the pheromone trails are updated. This is done by first lowering the pheromone strength on all arcs by a constant factor and then allowing each ant to add pheromone on the arcs it has visited:

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^k(t) \tag{2}$$

, where $0 < \rho \leq 1$ is the pheromone trail evaporation rate. With evaporation, the algorithm can "forget" previous done bad decisions. And the $\Delta\tau_{ij}^k(t)$ here is defined as:

$$\Delta\tau_{ij}^k(t) = \begin{cases} 1/L^k(t) & \text{if arc}(i,j) \text{ is used by ant } k \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$L^k(t)$ is the path distance of ant k in t iteration.

By **Tour Construction** and **Pheromone Update**, the AS algorithm can approach optimal value after iterations. However, AS often fall into local optimal value, thus can't achieve good solution. To overcome this situation, the later researcher used ACS instead.

2.1.3 MMAS

MIN-MAX ant system (MMAS) is also a improved AS. Not like ACS, the implementation of MMAS is very similar to AS. The solutions in MMAS are constructed in exactly the same way as in AS, that is, the selection probabilities are calculated as in Equation 1.

The main modification of MMAS is that it adopted the lower and upper bound for pheromone level, the lower and upper bound is denoted as τ_{min} and τ_{max} respectively. After pheromone update, any value that is smaller than τ_{min} will be increased to τ_{min} and any value that is greater than τ_{max} will be decreased to τ_{max}. In this way, we can avoid that ants fall into local optimal, with each cities has relatively higher chance to be traversed compared with original AS.

Below are the actual steps of MMAS.

Update of pheromone trails. Like in AS, after all the ants constructed a path, the pheromone is updated as follow:

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{best}(t) \tag{4}$$

, where $\tau_{ij}^{best}(t) = \frac{1}{L^{best}}$. The L^{best} here is either global best solution T^{gb} or iteration best solutions T^{ib}. In the experiments, we will set L^{best} to one of these two solution by a schedule. By experiments, the authors of [1] mentioned that gradually increased the frequency of choosing T^{gb} yields a better solution. The detail of scheduler will be explained in the experiment section.

2.1.4 Local Search of Ants

To better improved the performance of MMAS, the authors suggests that we can apply local search algorithm to a constructed path. This will make the solution better.

A popular local search algorithm is `2-opt`. `2-opt` enumerates all possible pair of cities, and swap the order of these two cities in the path, and check if the distance of the swapped path decreased for a given percentage. After applying `2-opt` the solution of MMAS is far better, and the convergence speed is also increased.

2.2 Firefly Algorithm

2.2.1 Introduction

Firefly Algorithm got the idea from how fireflies synchronize their lights. In this paper, the author use Firefly Algorithm to solve the TSP problem and could converge to the optimal solution faster than PCO for some specific problems. This algorithm is introduced in [2].

2.2.2 Firefly

Each firefly represent a solution to the TSP problem, the solution is represented by the order of the city to travel.

For example, if there are 5 cities labeled with number 0 to 4, than the firefly (0, 2, 4, 1, 3) means the salesman travels to city 0, 2, 4, 1, 3 and goes back to city 0.

2.2.3 Distance

The distance of two firefly is defined by the total number of different arcs.

For example, the distance between fireflyA (0, 1, 2, 3, 4) and fireflyB (0, 1, 3, 2, 4) is 2, since arc 1->2 and 2->3 in fireflyA is not in fireflyB.

2.2.4 Attractiveness

The attractiveness of fireflyA to fireflyB is defined by the following formula, where r is the distance between fireflyA and fireflyB.

$$\beta(r) = \beta_0 e^{-\gamma r^2} \tag{5}$$

γ is called light absorption factor and can be used to control how much a firefly can affect others. if γ is set to 0, all firefly have the same attractiveness regardless near or far, and if γ is set to ∞ than all firefly will have no affect on others.

2.2.5 Movement

In each iteration in firefly algorithm, each firefly will " move" toward the most attractive firefly. In this paper, since a firefly means a solution of the TSP problem, so the firefly is moving in the solution space.

The movement of fireflyA to another fireflyB is determined by

$$\beta(r) = \beta_0 e^{-\gamma r^2} \tag{6}$$

where x_A is the length of movement, and r_{AB} is the distance between fireflyA and fireflyB and scale into the range of [0, 10].

Than we randomly select a index in fireflyA's solution and perfrom Inverse Permutation with length x_A.

2.3 DQN

2.3.1 Introduction

DQN, or Deep Q-Network, is a reinforcement learning algorithm that combines deep learning and Q-learning to solve complex decision-making problems. It first appears in [3]. At its core, DQN leverages a deep neural network to estimate the Q-values of different actions in a given state. The Q-values represent the expected future rewards for taking a particular action in a specific state. By learning these values, the agent can make informed decisions on which actions to take to maximize its cumulative reward.

The DQN algorithm uses a technique called experience replay, which stores the agent's experiences (state, action, reward, next state) in a replay memory. During training, the agent samples mini-batches of experiences from this memory to decorrelate the data and improve learning efficiency.

The learning process of DQN can be summarized using the following equation:

$$Q(s,a) \leftarrow (1-\alpha)\cdot Q(s,a) + \alpha\cdot(r+\gamma\cdot\max_{a'} Q(s',a')) \tag{7}$$

Here, $Q(s,a)$ represents the estimated Q-value of action a in state s. The algorithm updates this estimate using a weighted average of the current estimate $(1-\alpha)\cdot Q(s,a)$ and a new estimate $\alpha\cdot(r+\gamma\cdot\max_{a'} Q(s',a'))$. In this equation, r represents the immediate reward obtained after taking action a in state s, s' represents the resulting state, α is the learning rate, and γ is the discount factor that determines the importance of future rewards.

By repeatedly updating the Q-values based on this equation, DQN gradually learns to approximate the optimal Q-function, which leads to better decision-making capabilities for the agent.

2.3.2 RL Environment

For the graph $G(V,E)$, $N = |V|$.

- State representation
  State representation is Nx5 vector, where for each node we store whether this node is in the sequence, whether it is first or last, and its (x,y) coordinates. The coordinates of each nodes have normalize to one, for the sake of convenience of training neural net.
- Action
  The action is the integer from 1 to N. The env will mask the nodes that have been visited. The solution list is a list contained the nodes that have been visited.
- Reward definition
  The reward is calculated based on the total distance traveled by the solution compared to the current solution. `reward = -(total_dist(next_solution) - total_dist(solution))`

2.3.3 Deep Neual Network Design

The network consists of five fully connected (dense) layers (hidden size is 1600) with ReLU activation applied to the output of each layer except for the last one. The input dimension is defined by shape of state tensor, the number of actions is defined by total number of nodes, and the size of the hidden layers is set to `hidden_layer_size`. The network takes input states and produces corresponding Q-values, which are used to estimate the value of each action in a reinforcement learning context.

3 Experiments and Findings

3.1 Problem Instances

In this project, we used the TSP from TSPLIB [4]. These problems already has optimal solutions. Thus, it's easier to compare with the solution generated by the three methods.

We pick five problem instances, the name and optimal solution can be found in Table 1. Noted that this five TSP is relatively small problem size, because we afraid that large TSP will make the running of algorithm increased.

Table 1: The five chosen TSP instances from TSPLIB.

Problem	Number of Cities	Optimal Solution
berlin52	52	7542
rat99	99	1211
bier127	127	118282
ch130	130	6110
a280	280	2579

3.2 Parameters in Experiments

In this section, we list the parameters used in the experiments.

3.2.1 ACO

The method we used in ACO is MMAS, the parameters of MMAS is as follow:

- Number of ants = 100
- Max iterations = 500
- $\alpha = 1$
- $\beta = 2$
- $\rho = 0.2$
- Every ants runs `2-opt` right before update pheromone
- $\tau_{max} = \frac{1}{\rho T^{gb}}$, $\tau_{min} = \frac{\tau_{max}}{2n}$
-

The scheduler used in Equation 4 works as follow, when $0 < t \leq 25$, use only iteration best T^{ib}. When $25 < t \leq 75$, for every 5 iterations, use T^{gb} instead. When $75 < t \leq 125$, for every 3 iterations, use T^{gb}. When $125 < t \leq 250$, for every 2 iterations, use T^{gb}. Finally, when $250 < t$, use T^{ib} and T^{gb} alternatively.

We also apply a technique to increased exploration rate in the late stage, when most of arc has low pheromone level, i.e. $\approx \tau_{min}$, reset the pheromone level of all arc to τ_{max}. In this way, when the algorithm converges in local optimal solution, it has the chance to escape.

Also noted that, if our algorithm has already found the true optimal value, the algorithm stop immediately.

3.2.2 Firefly Algorithm

- $\beta_0 = 1$
- $\gamma = 1$
- Number of iterations: 2000

3.2.3 DQN

- Episode: 2000
- Exploration rate: 0.03 (Lower means less exploration)
- Learning rate: 0.005,
- Learning rate decay rate: 0.95
- Gamma: 0.997
- Batch size: 32
- Replay buffer capacity: 10000
- Dense layer hidden size: 1600

3.3 Experiments Results

As we can observer in Table 2, the solutions given by MMAS is far better than other two methods. We think the reason is that MMAS utilize the `2-opt` and other technique to escape from local optimal value.

The following graphs is the visualized solutions given by MMAS and firefly algorithm, we can also observe the quality of two methods visually.

3.3.1 Solutions of MMAS

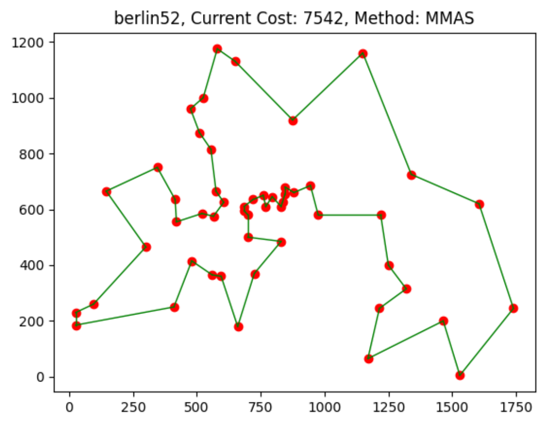

Figure 1: The optimal solution of berlin52 given by MMAS

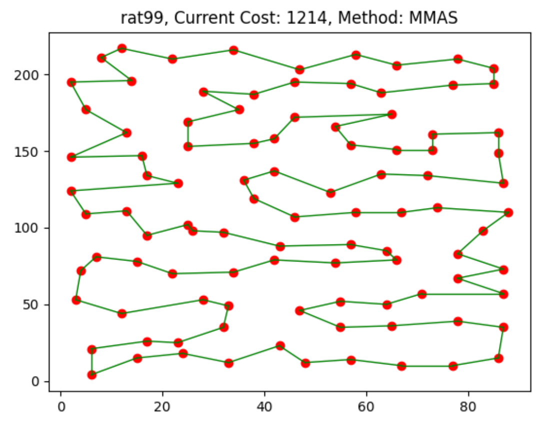

Figure 2: The optimal solution of rat99 given by MMAS

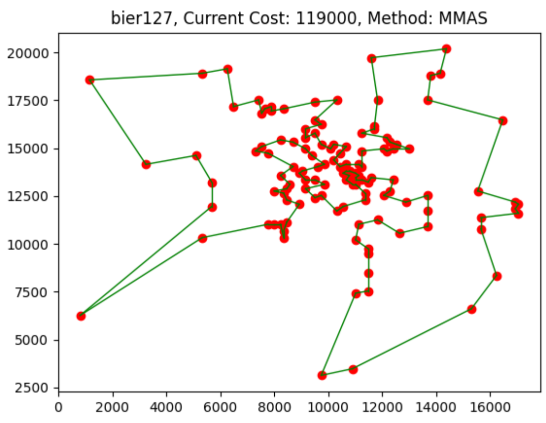

Figure 3: The optimal solution of bier127 given by MMAS

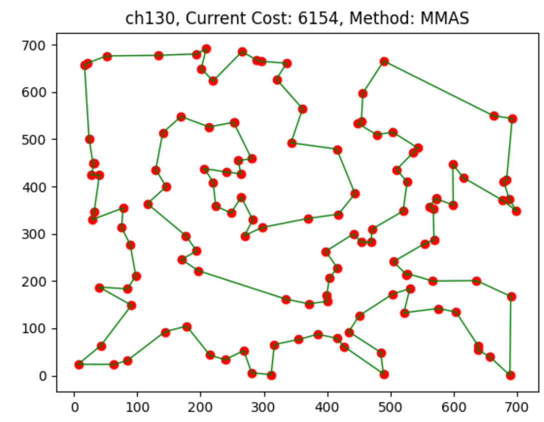

Figure 4: The optimal solution of ch130 given by MMAS

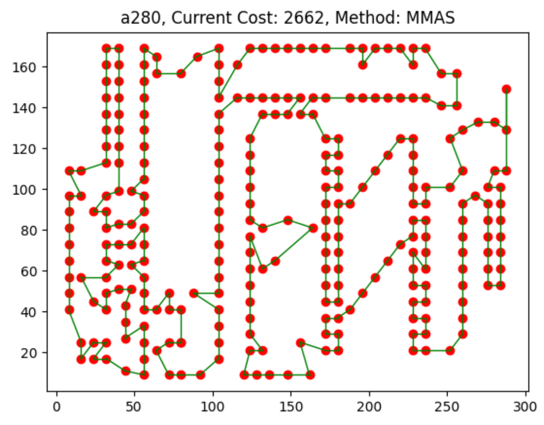

Figure 5: The optimal solution of a280 given by MMAS

3.3.2 Solutions of firefly

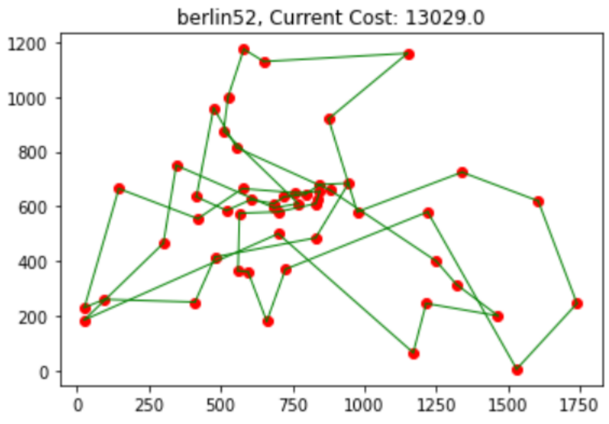

Figure 6: The optimal solution of berlin52 given by firefly algorithms

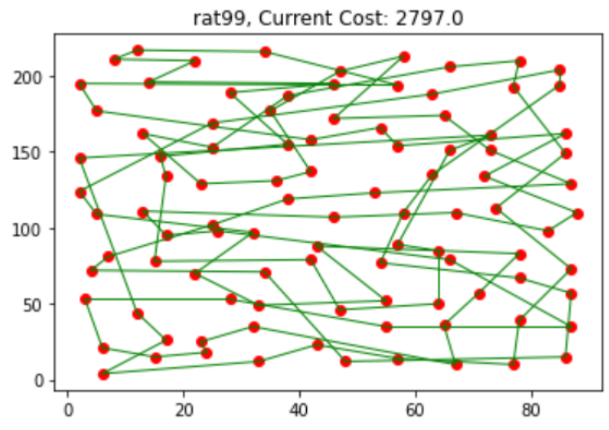

Figure 7: The optimal solution of rat99 given by firefly algorithms

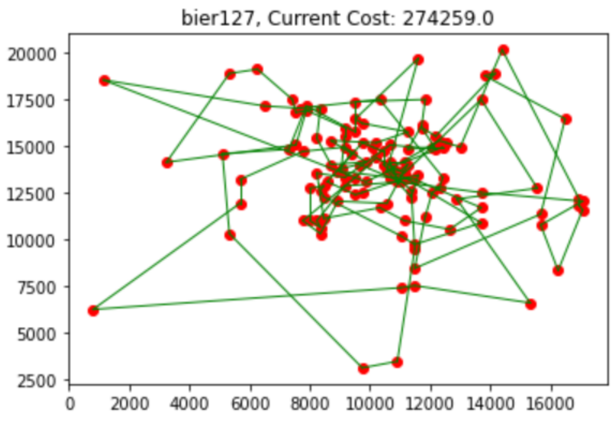

Figure 8: The optimal solution of bier127 given by firefly algorithms

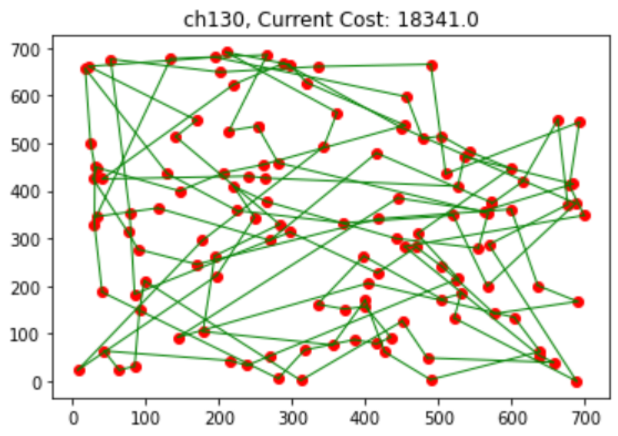

Figure 9: The optimal solution of ch130 given by firefly algorithms

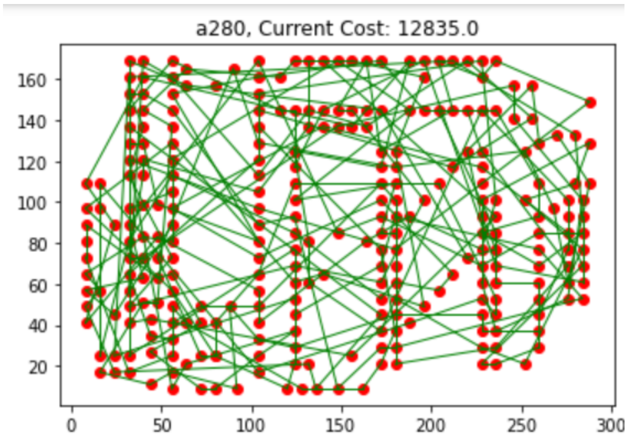

Figure 10: The optimal solution of a280 given by firefly algorithms

3.3.3 Solutions of DQN

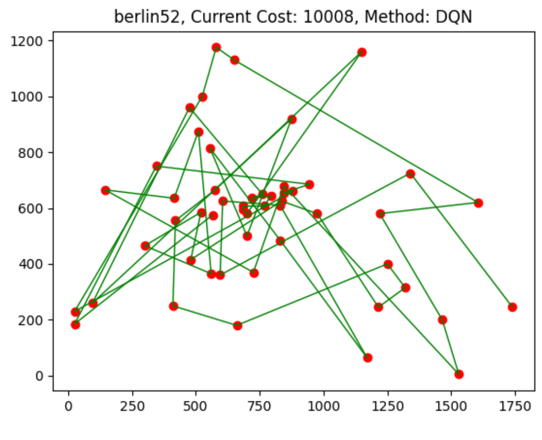

Figure 11: The optimal solution of berlin52 given by DQN

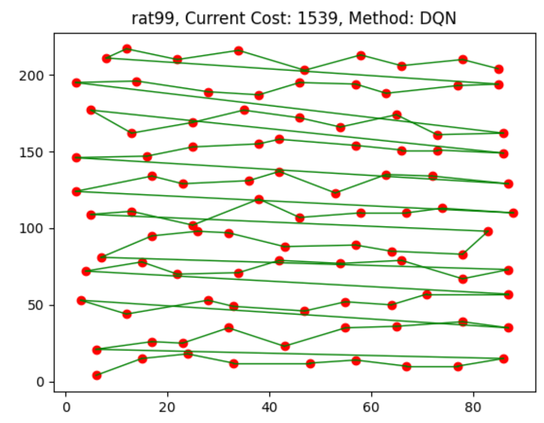

Figure 12: The optimal solution of rat99 given by DQN

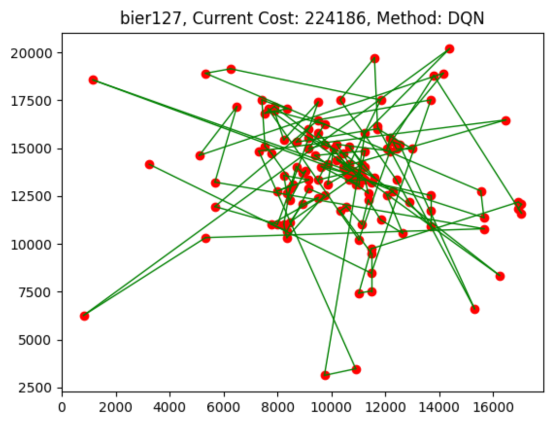

Figure 13: The optimal solution of bier127 given by DQN

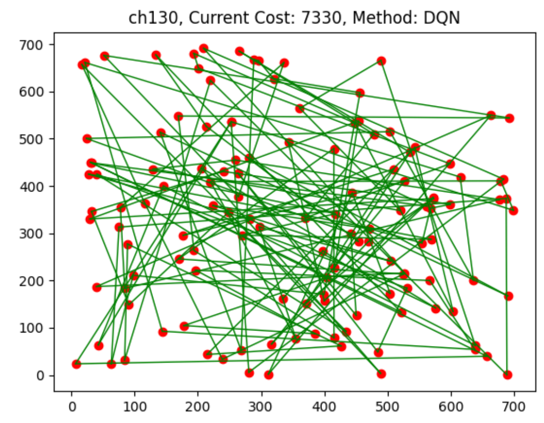

Figure 14: The optimal solution of ch130 given by DQN

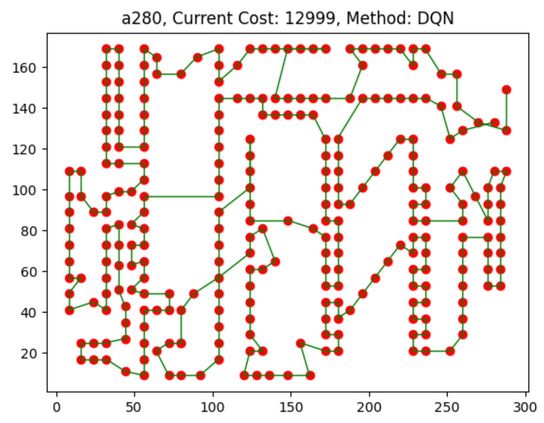

Figure 15: The optimal solution of a280 given by DQN

4 Discussion

4.1 The running time of MMAS

As in Table 1, the average running time of MMAS is significant higher than other two methods, the reason is that the algorithm always run for 500 iterations if it didn't found optimal solution. Actually, it may converge to the results in the early iterations. Therefore, the actual running time when can be lower. For example, MMAS can find the optimal solution of berlin52 before tenth iterations.

In conclusion, when we used MMAS, we can stop the algorithm if we observe that the given solution has already converged.

4.2 The solution quality of MMAS

MMAS introduced many techniques to find the balance between exploitation and exploration, and it also utilized the use of local search and heuristic information, all of these make the solution quality of MMAS is significant better than other two methods, with only 5% maximum relative difference to optimal solution.

4.3 The solution quality of firefly algorithm

The performance of Firefly Algorithm is not ideal, most of the time it well stuck at local optimal and could not get the optimal solution at all 5 problems.The problem is that the design of the algorithm does not have any mechanism that incentivize fireflies from getting better solutions besides natural selection, so it is basically a genetic algorithm.

Some possible ways to improve Firefly Algorithm:

- Define how fireflies move differently: In the original paper the movement of a firefly is defined by doing inverse permutation on the solution, while it could preserver the order of the cities, but there is no "direction" in this definition. Thus to fully unleash the power of Firefly Algorithm, we may need to redefine how a firefly moves.
- Clustering: Since in every iteration it would only select the best fireflies, it is likely that the fireflies remains are almost the same. So maybe cluster the fireflies into different species and in each iterations fireflies in different cluster have a chance to exchange genome could prevent the algorithm from stucking in local optimal.

4.4 The solution quality of DQN

In the experiments, we can found that DQN did not performed well in the TSP taks. There are following reasons why DQN can not solve this problem.

- High dimensionality: TSP has a high-dimensional search space, making it challenging for DQN to explore and learn an optimal policy effectively.
- Lack of generalization: In the TSP, the problem size increases exponentially with the number of cities, making it impractical to train DQN on all possible problem instances. Consequently, DQN may struggle to generalize its learned policy to unseen TSP instances.

In conclusion, through DQN has the ability to leverage dynamic progamming method and it can solve the problem optimaly in theory, it can not perform well. In the future, we may design a new neural network with more sophisticated architecture to solve the problems above.

5 Conclusion

In this project, we implement three different methods, MMAS [1], firefly algorithm[2] and DQN[3]. In these three methods, the performance of MMAS is better than two methods, and MMAS is also able to speed up by parallel simulate the ants.

DQN methods can be

References

[1] T. Stützle, M. Dorigo, *et al.*, "Aco algorithms for the traveling salesman problem," *Evolutionary algorithms in engineering and computer science*, vol. 4, pp. 163–183, 1999.

[2] G. K. Jati and Suyanto, "Evolutionary discrete firefly algorithm for travelling salesman problem," in *International conference on adaptive and intelligent systems*, Springer, 2011, pp. 393–403.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[4] G. Reinelt, "Tsplib—a traveling salesman problem library," *ORSA journal on computing*, vol. 3, no. 4, pp. 376–384, 1991.

Table 2: The solutions of five TSP instances given by MMAS, DQN and Firefly.

Problem	**Method**	**Avg. Distance**	**Std. Distance**	**Avg. Time (sec.)**	**Best Distance**	**Worst Distance**	**Relative Diff. (%)**	**Optimal Distance**
	DQN	9841.4	212.3	0.34	9075	10074	30.49	
berlin52	MMAS	**7542**	0	6.90	7542	7542	**0**	7542
	Firefly	11541.2	1046.2	33.01	9520	12321	53.03	
	DQN	1578.6	70.34	0.16	1465	1697	30.36	
rat99	MMAS	**1226.2**	7.43	428.65	1214	1234	**1.26**	1211
	Firefly	2915.6	278.44	59.72	2604	3440	140.76	
	DQN	221276.6	3125.08	0.2096	216331	224186	87.08	
bier127	MMAS	**119981.8**	277.35	632.72	119700	120365	**1.44**	118282
	Firefly	272727.6	8591.6	85.98	261526	286210	130.57	
	DQN	7989.6	385.75	0.214	7330	8529	30.76	
ch130	MMAS	**6211**	34.34	657.71	6154	6240	**1.65**	6110
	Firefly	18001.8	847.49	68.81	16940	19055	194.63	
	DQN	12828.2	181.05	0.8	12459	13049	397.41	
a280	MMAS	**2685.6**	20.26	2553.39	2662	2718	**4.13**	2579
	Firefly	12791.4	366.47	134.7	12189	12189	395.98	