Applied Reinforcement Learning methods for the Capacitated Vehicle Routing Problem

Hoang Tran Nhat Minh

Supervisor: **Dr. Pham Quang Dung**

Project 3

Data Science and Artificial Intelligence 01 - 2020

January 2024

Hanoi University of Science and Technology

School of Information and Communication Technology

Abstract

This project reviews reinforcement learning (RL) approaches to optimization problems in general, then dive deeper for the Capacitated Vehicle Routing Problem (CVRP). The problem is formulated as a RL problem, then several RL methods are implemented as an endeavor to solve it, in comparison with the solutions provided by OR-Tools, namely: Deep Q-Network (DQN), Advantage Actor-Critic (A2C), Proximal Policy Optimization (PPO).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Reinforcement learning

Reinforcement Learning (RL) is an area of machine learning, alongside supervised and unsupervised learning. RL aims to maximize the cumulative reward when an intelligent agent takes actions in a dynamic environment (Figure 1.1).

1.2 Reinforcement learning for optimization problems

First, RL is appliable for optimization problems, where Environment is Problem, (Long-term) Reward is Objective, State is Configuration, Agent is Algorithm, and Action is Decision.

RL is well-fit for optimization problems since it is good in sequential decision making, that is, RL can learn a series of sequential decisions to maximize a long-term objective. RL can also balance exploration and exploitation. Moreover, RL can handle very complex problems provided enough resources. For delayed rewards in general optimization problems, RL algorithms can use its experience to optimize the cumulative reward based on immediate rewards and their past experience. And the greatest advantage of RL is its transferability, that is, trained agent can be directly used on the

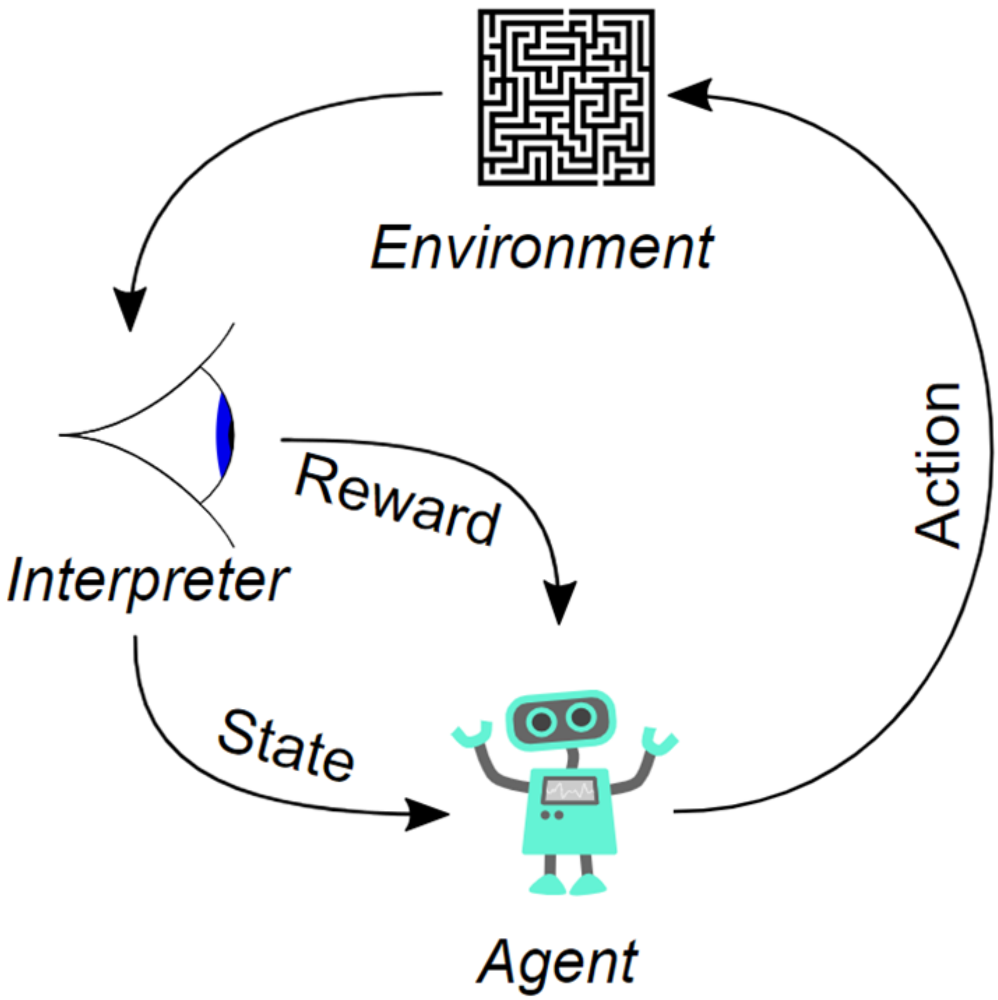

Figure 1.1: General reinforcement learning framework.

same problem with different configurations.

There are many ways to approach optimization problems using RL, the following three are used in this project: Deep Q-Network (DQN), Advantage Actor-Critic (A2C), and Proximal Policy Optimization (PPO).

1.3 Capacitated Vehicle Routing Problem

Also known as the Vehicle Routing Problem (VRP), Capacitated Vehicle Routing Problem (CVRP) is a optimization problem, classified as a combinatorial optimization problem and an integer programming problem. It answers the question: "What is the optimal set of routes for a fleet of vehicles to traverse in order to deliver to a given set of customers?" (Figure 1.2)

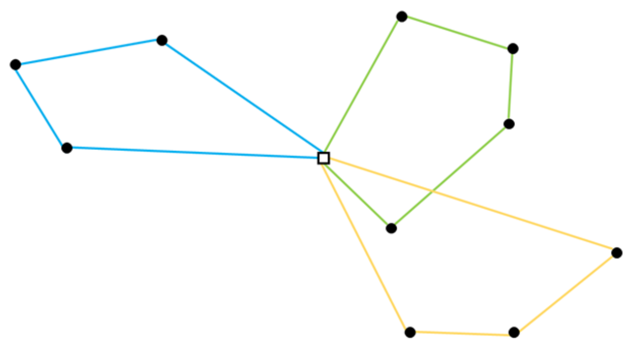

Figure 1.2: Capacitated Vehicle Routing Problem in two-dimensional Euclidean space.

This project formulates the CVRP problem as a RL problem then trains the three algorithms DQN, A2C, and PPO to solve it. A code review of an RL approach for the Travelling Salesman Problem was also conducted in this project. [1]

[1] The code was outdated and non-functioning, therefore the code review is discarded from this report. See more in https://github.com/htnminh/CVRP-RL.

Chapter 2

Reinforcement learning methods

2.1 Two general types of reinforcement learning algorithms

There are two main types of RL algorithms: value-based and policy-based algorithms.

2.1.1 Value-based algorithms

Value-based algorithms try to approximate the optimal value function, which is one of the two following mappings:

$$state \rightarrow cumulative\_reward$$

or

$$(state, action) \rightarrow cumulative\_reward$$

for $\forall action \in action\_space$ and $\forall state \in state\_space$.

If this mapping is optimized, the higher the cumulative reward, the better the state (or (state, action) tuple).

Q-learning is an example of a value-based algorithm, which learns the latter mapping. The *cumulative_reward* in this case is often denoted as Q-value: Q.

Advantages of value-based algorithms are that they are sample efficient and steady.

2.1.2 Policy-based algorithms

Policy-based algorithms try to approximate the optimal policy function, which is the mapping:

$$state \to P(action|state)$$

for $\forall action \in action\_space$ in current $state$.

$P(action|state)$ is often denoted as π.

If this mapping is optimized, the higher the action probability, (probably) the better the action.

REINFORCE is an example of a policy-based algorithm.

Advantages of policy-based algorthms are that they converges faster and they are generally better for continous spaces.

2.2 Q-learning

As stated, Q-learning is a value-based RL algorithm which learns the mapping:

$$(state, action) \to Q$$

for $\forall action \in action\_space$ and $\forall state \in state\_space$.

2.2.1 Pseudocode

```
Q_table = random_values(Q_table.shape);   // Initialize random values
state = initial_state;                    // Initialize state
while true {
        action = choose(Q_table, state, action_space);
                // Choose an action based on a kind of policy
        next_state, reward = execute(state, action);
        Q_table[state, action] = Q_table[state, action] + alpha * (
                reward + gamma * max(Q[next_state, action_space])
                - Q_table[state, action]);  // Update Q-table
        if final(state) {state = initial_state}
                else {state = next_state};  // Continue to the next state
        if stop_training {break};  // Break based on a stopping condition
}
```

2.2.2 Epsilon-greedy "policy"

This "policy" is simple enough for Q-learning to still be considered a policy-free algorithm.

```
action = choose(Q_table, state, action_space);
```

Epsilon-greedy chooses an action at state s in the following manner (Figure 2.1): for a constant $\varepsilon \in [0, 1]$, there is an ε chance that a completely random action is chosen; and the other $1 - \varepsilon$ chance that the best known action is chosen, or the action:

$$a = \arg\max_a Q(s, a)$$

2.2.3 Bellman equation

```
Q_table[state, action] = Q_table[state, action] + alpha * (
        reward + gamma * max(Q[next_state, action_space])
        - Q_table[state, action]);   // Update Q-table
```

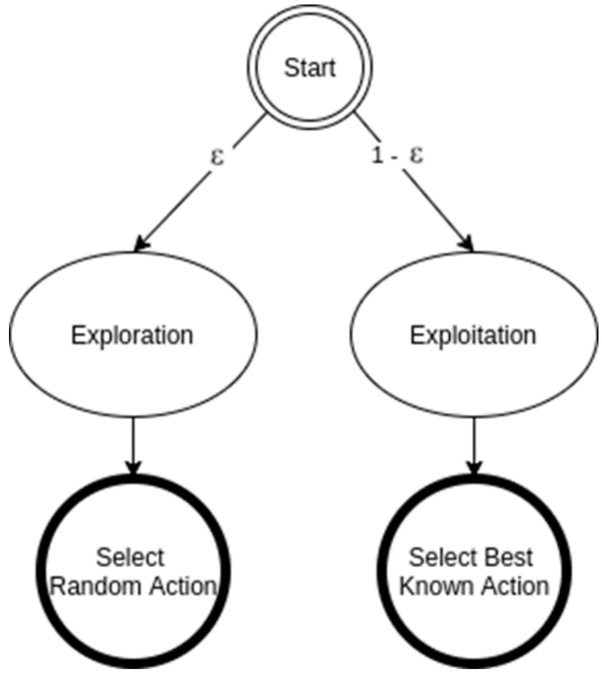

Figure 2.1: Epsilon-greedy.

The Q-table is updated using Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max(Q(s',a')) - Q(s,a))$$

where $Q(s,a)$ is the Q-table value at state s and action a, α is the learning rate, r is the immediate reward (after executing action a on state s), $\gamma \in [0,1]$ is a discount factor, and $\max(Q(s',a'))$ is the maximum Q-value of the next state s' (after executing all actions available in $a' = action\_space$.)

Since $Q(s,a)$ converges to $r + \gamma \max(Q(s',a'))$, the discount factor $\gamma \in [0,1]$ is the importance of future reward. For example, $\gamma = 0$ means $Q(s,a)$ converges to r, so the future reward is not considered; or $\gamma = 1$ means $Q(s,a)$ converges to $r + \max(Q(s',a'))$, so the future reward is as important as the immediate reward. Typically, $0 < \gamma < 1$ to help many algorithms, including Q-learning, to converge properly and faster.

2.3 Deep Q-network

In case the state space and/or the action space grow larger, the size of Q-table grows exponentially with them. Storing the Q-values $Q(s, a)$ as a table has a main drawback that is it is infeasible for almost any non-trivial problems.

In Deep Q-Network (DQN), the table instead will be "stored" as a deep neural network, as shown in Figure 2.2.

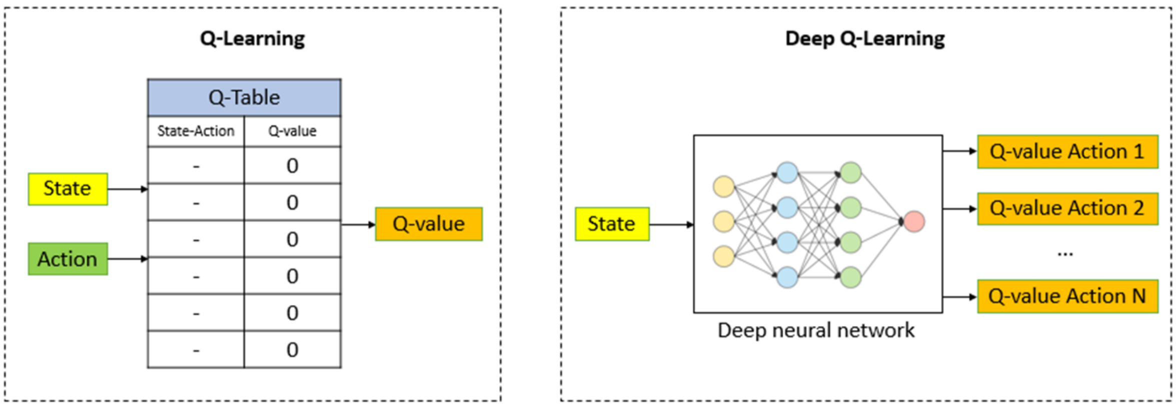

Figure 2.2: Deep Q-network.

Deep Q-learning refers to a Q-learning implementation using a Deep Q-Network. Some other mechanisms will be used to optimize this algorithm: Replay memory and Target network.

For Replay memory, each "experience" will be stored as a tuple (state, action, reward, next state). This dataset of many tuples will be sampled during the training process of networks. The Deep Q-network learns the mapping $state \rightarrow Q(action|state)$ for each action in the action space using the above dataset. Target network is another network to estimate the target Q-values, which is a copy of the main network, but it is updated periodically to prevent overfitting and mitigate the effect of delayed reward.

Everything else is the same as the traditional Q-learning method.

2.4 Actor-Critic

Actor-Critic algorithm combines the two types of value-based and policy-based algorithms. It can be split into two parts: Actor and Critic.

Actor is the policy-based part, which learns the mapping:

$$state \rightarrow \pi$$

While running deterministically, it returns the action with the highest probability, which probably is the best action.

Critic is the value-based part, which learns the mapping:

$$state \rightarrow cumulative\_reward$$

It updates the actor accordingly using policy gradient (Figure 2.3). Policy gradient is an optimization technique that has the same idea as gradient descent. *cumulative_reward* in this case is often denoted as just value V.

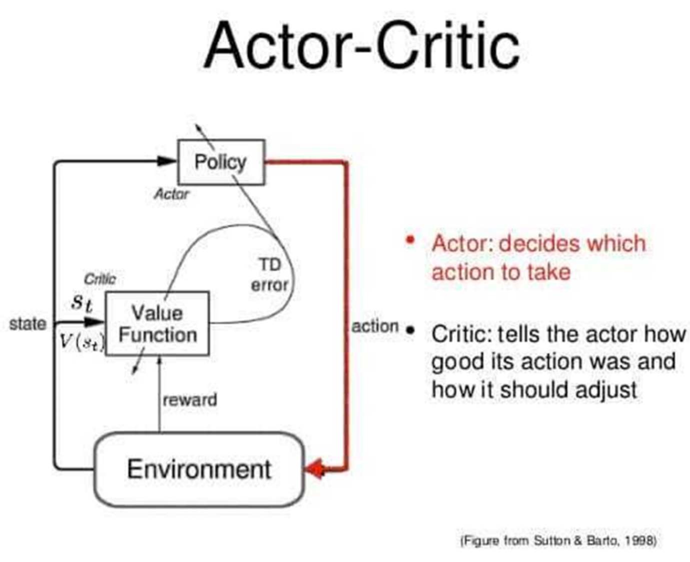

Figure 2.3: Actor-Critic.

2.5 Advantage Actor-Critic

As mentioned, Critic in Actor-Critic learns the mapping:

$$state \to V$$

Advantage Actor-Critic (A2C) (Figure 2.4) splits the Q-value into two parts, state value V and Advantage value $A(s, a)$, based on action a:

$$Q(s, a) = V(s) + A(s, a)$$
$$\implies A(s, a) = Q(s, a) - V(s)$$

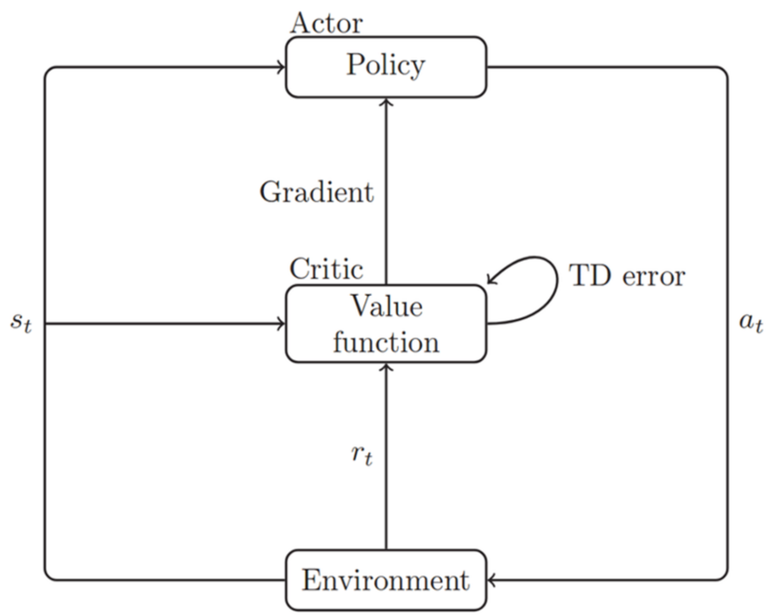

Figure 2.4: Advantage Actor-Critic.

The difference in Critic part between Actor-Critic and A2C is that, in A2C, the Critic learns the Advantage value from the tuple (state, action) instead of the state value from state (Table 2.1). The idea is that the algorithm instead of learning how "good" is a *state*, the critic instead learn how much "advantage" it will gain if the *action* is executed on that *state*.

The A2C algorithm will generalize better than Actor-Critic, especially on complex problems, but obviously at the cost of using more memory.

Table 2.1: Actor-Critic and Advantage Actor-Critic comparison

Component \ Algorithm	**Actor-Critic**	**Advantage Actor-Critic**
Actor	$state \rightarrow \pi$	$state \rightarrow \pi$
Critic	$state \rightarrow V$	$(state, action) \rightarrow A$

2.6 Proximal Policy Optimization

As an improvement on A2C, Proximal Policy Optimization (PPO) improves the learning progress of the Actor in A2C using Trust Region Policy Optimization (TRPO). TRPO is a technique to limit how large can the Actor update its policy π, or clipping, to avoid too large updates. (Figure 2.5)

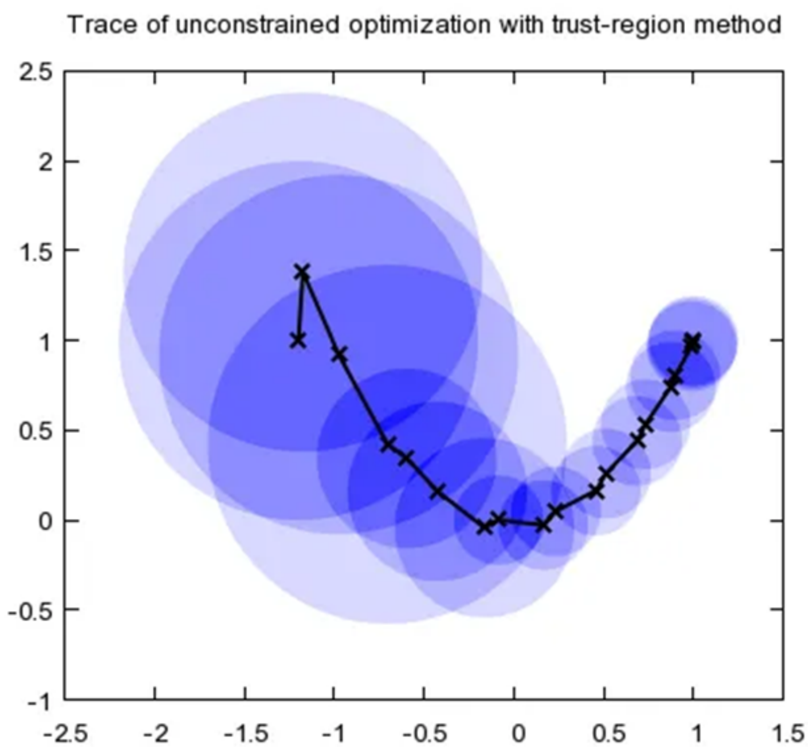

Figure 2.5: An illustration of Trust Region Policy Optimization.

Since many of the mathematical details of TRPO are out of the scope of this project, only some quick explanations about its mathematical concepts are listed. The constraint is expressed in terms of Kullback–Leibler divergence, which is the "distance" between two probability distributions; in this case, the old policy and the new policy. The theoretical TRPO update is impractical, so an approximation is used using the Taylor expansion of the objective function around the parameters of current π. The problem is then solved by Lagrangian duality to give the Natural Policy Gradient. Since the approximation will naturally have an error, sometimes really large, the Natural Policy Gradient is then tweaked by using a backtracking coefficient to mitigate the impact of the approximation error. The backtracking coefficient defines how much the approximation should "relies" on some of its previous ones.

Chapter 3

Reinforcement learning implementation for Capacitated Vehicle Routing Problem

3.1 Environment intialization

3.1.1 Parameters

The data are generated using the following parameters values: `n_stops` is the number of stops (including the depot), `max_demand` is the maximum demand of all cities, `max_vehicle_cap` is the maximum capacity of the vehicle, and `max_env_size` is the maximum coordinate of all cities.

3.1.2 Assumptions

To ensure the feasibility of the problem, assume that `max_demand` $\leq$ `max_vehicle_cap` and the number of vehicles = `n_stops`. Those asssumptions ensure that the problem always has a naïve solution: all vehicles move to all stops then comeback to depot immediately after.

To somewhat simplify the problem; assume that all vehicles have the same capacity;

and the objective is the total distance only, disregard the time, number of vehicles used, profit, or any other factors.

3.1.3 Data generation process

`demands` are the demands of all stops, which is a list with `n_stops` elements. The demand of depot is always 0, and the following `n_stops - 1` elements are random integers in `[1, max_demand]`.

`stop_coords` are the coordinates of stops, which is a matrix of shape `(n_stops, 2)`, the entries are random integers in `[0, max_env_size]`.

Note that the depot is the first stop in those lists (index 0).

`vehicle_cap` is the vehicle capacity, which is a random integer between `max_demand` and `max_vehicle_cap` (both are inclusive).

3.2 Formulation as a reinforcement learning problem

With the above intialization, the problem can be translated into a one vehicle routing problem, and it cannot visit a stop if the load left is not enough for it.

The environment, which remains constant after intialized, contains `demands`, `stop_coords`, and `vehicle_cap`.

The state; which may change over time within one episode; contains `current_stop`, which is the current stop of the vehicle; `visited`, which is a (boolean or binary) list to store the status of if each stop is visited or not; along with some other derived information, including `current_length`, which is the total length moved.

The objective is to minimize `current_length`.

The immediate reward is split into two cases: valid and invalid action. If the action is valid, the immediate reward is (initially) defined as `reward = - segment_length`, in which `segment_length` is the length between the two stops. Otherwise, if the vehicle tries to move to a visited stop (except depot), or to a stop with not enough load, the immediate reward (initially) is `reward = - 2 * n_stops * max_env_size`. The idea behind this invalid-move-reward is that the maximum `segment_length` is $\sqrt{2}\times$ `max_env_size` $\approx 1.41\times$ `max_env_size`, which is still smaller than $2\times$ `max_env_size`, then the agent is punished even more, proportional to `n_stops`.

However, the reward function defined above would be too large, and may cause the algorithms struggle to converge. Therefore, the final immediate reward in both cases are respectively divided by `max_env_size`. That is, if the action on a state is valid, `reward = - segment_length / max_env_size`, else, `reward = - 2 * n_stops`.

Chapter 4

Results

The implementation of the CVRP formulation and the three algorithms are in pure Python with Stable Baselines3 (SB3).

"#Timesteps" in the following result tables are the minimum numbers of environment total timesteps during training. Each result is the minimum objective value found out of 10 episodes.

Table 4.1, Table 4.2, and Table 4.3 are the results of DQN, A2C, and PPO, respectively. Then, the best results are summarized and compared with the solutions provided by OR-Tools (Table 4.4).

Figure 4.1, Figure 4.2, Figure 4.3, and Figure 4.4 are example solutions given by DQN, A2C, PPO, and OR-Tools, respectively.

From the results, one can conclude that DQN is the best RL model. As expected, the RL models get worse when there are more stops, or when the problems are more difficult. However, they will still give a reasonable solution on a live environment without prior knowledge in a practical amount of time.

Table 4.1: Deep Q-Learning results

#Timesteps / #Stops	**250**	**500**	**1000**	**5000**	**10000**	**50000**
5	3958	3602	5902	2855	4164	3730
6	4356	4417	4639	5071	4356	4417
7	3126	3306	2934	3037	3822	3228
8	4916	4930	4699	4916	4643	5011
9	6051	5492	5153	5167	4976	5557
10	6897	5827	5910	5218	6386	5864
12	6097	6856	6915	5345	5973	6550
15	10511	10852	11351	10335	10393	10079
20	14061	12699	12436	14362	12399	11828

Table 4.2: Advantage Actor-Critic results

#Timesteps / #Stops	**250**	**500**	**1000**	**5000**	**10000**	**50000**
5	3602	2988	3730	3602	2988	3602
6	4356	4516	4639	4902	4417	4516
7	3855	3990	2794	2772	3093	3870
8	4817	4817	5187	4904	5259	5212
9	5040	5085	5348	5462	5612	5040
10	5749	7196	6946	5309	6661	6150
12	7902	9392	7345	8812	7951	8097
15	10143	10458	10038	10164	10165	9970
20	13239	11280	11118	12582	10982	14469

Table 4.3: Proximal Policy Optimization results

#Stops \ #Timesteps	**250**	**500**	**1000**	**5000**	**10000**	**50000**
5	2855	2855	3958	3602	3225	3581
6	4356	4356	4356	4516	4803	4417
7	3309	3315	3057	3037	3660	3550
8	4699	4930	5295	5475	5104	5011
9	5229	5240	5076	5282	5282	5368
10	6372	5258	5460	6127	5364	6003
12	7321	6444	6817	7171	7462	7415
15	8354	10118	9039	9035	9325	11043
20	11741	11079	11259	11454	11649	13128

Table 4.4: Best results

#Stops \ Algorithm	**A2C**	**DQN**	**PPO**	**OR-Tools**
5	2988	**2855**	**2855**	2855
6	**4356**	**4356**	**4356**	4356
7	**2772**	2934	3037	2772
8	4817	**4643**	4699	4643
9	5040	**4976**	5076	4648
10	5309	**5218**	5258	3554
12	7345	**5345**	6444	4196
15	9970	10079	**8354**	7325
20	**10982**	11828	11079	7521

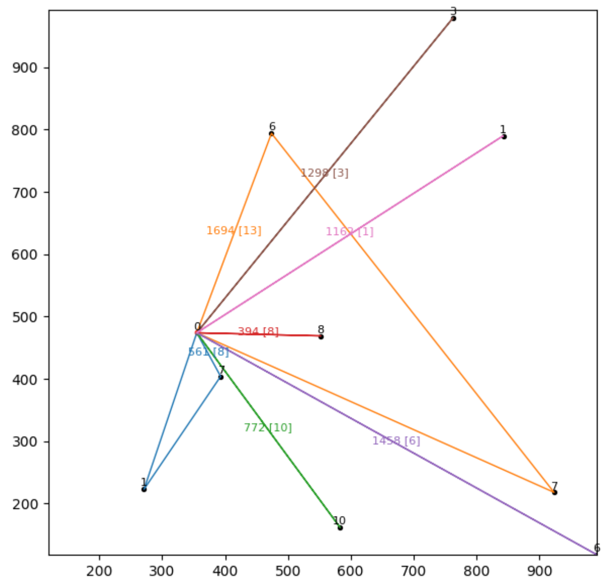

Figure 4.1: A solution of Advantage Actor-Critic.

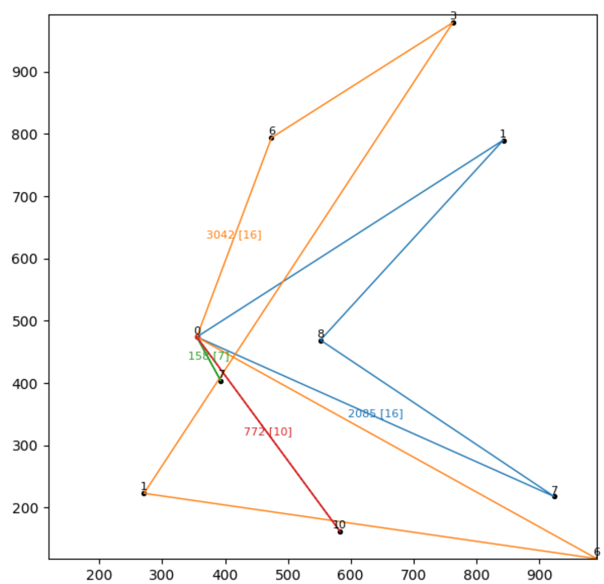

Figure 4.2: A solution of Deep Q-Learning.

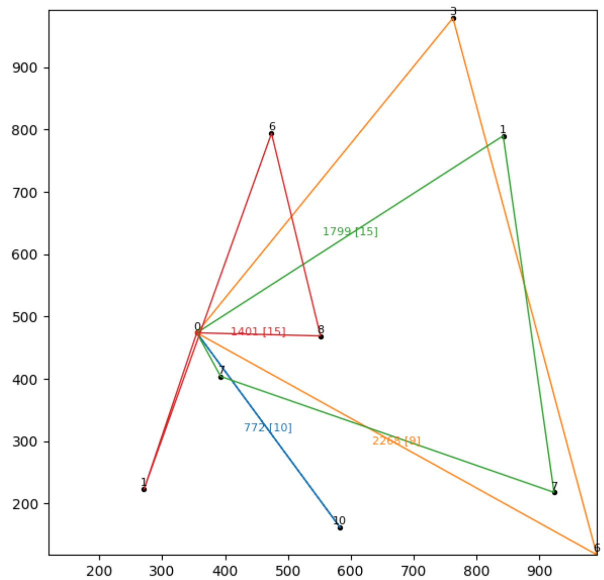

Figure 4.3: A solution of Proximal Policy Optimization.

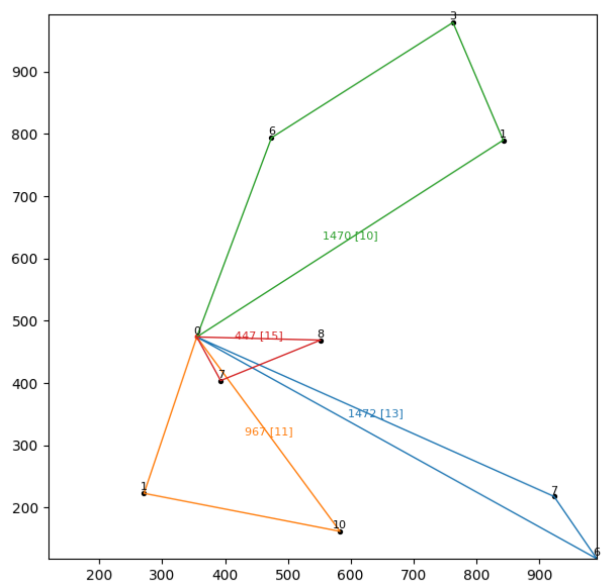

Figure 4.4: A solution of OR-Tools.

Chapter 5

Conclusion and Future works

In conclusion, this project has investigated reinforcement learning (RL) approaches to optimization problems in general, reviewed a Q-learning implementation for the Travelling Salesman Problem, formulated Capacitated Vehicle Routing Problem (CVRP) as a RL problem, implemented the Advantage Actor-Critic, Deep Q-Network, and Proximal Policy Optimization algorithms for CVRP. The project has achieved remarkable results with Deep Q-Network is the best model out of the experimented RL models.

The future works that work up from this project should continue to adjust the implemented algorithms, implement more algorithms, and explore more RL applications for traditional optimization problems.

References

1. Solving the Traveling Salesman Problem with Reinforcement Learning. (2021, November 3). Eki.Lab. https://ekimetrics.github.io/blog/2021/11/03/tsp/#references

2. L. Wang, Z. Pan and J. Wang, "A Review of Reinforcement Learning Based Intelligent Optimization for Manufacturing Scheduling," in Complex System Modeling and Simulation, vol. 1, no. 4, pp. 257-270, December 2021, doi: 10.23919/CSMS.2021.0027.

3. Wikipedia contributors. (2024, January 5). Reinforcement learning. In Wikipedia, The Free Encyclopedia. Retrieved 04:02, January 16, 2024, from https://en.wikipedia.org/w/index.php?title=Reinforcement_learning&oldid=1193685081

4. Gilman, R. (2018, January 9). Intuitive RL: Intro to Advantage-Actor-Critic (A2C). HackerNoon. https://hackernoon.com/intuitive-rl-intro-to-advantage-actor-critic-a2c-4ff545978752

5. Karagiannakos, S. (2018, November 17). The idea behind Actor-Critics and how A2C and A3C improve them — AI Summer. AI Summer. https://theaisummer.com/Actor_critics/

6. Understanding the role of the discount factor in reinforcement learning. (n.d.). Cross Validated. https://stats.stackexchange.com/questions/221402/understanding-the-role-of-the-discount-factor-in-reinforcement-learn

ing

7. Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... & Kavukcuoglu, K. (2016, June). Asynchronous methods for deep reinforcement learning. In International conference on machine learning (pp. 1928-1937). PMLR.

8. Wikipedia contributors. (2024, January 4). Q-learning. In Wikipedia, The Free Encyclopedia. Retrieved 04:08, January 16, 2024, from https://en.wikipedia.org/w/index.php?title=Q-learning&oldid=1193548086

9. A2C — Stable Baselines3 2.3.0a1 documentation. (n.d.). https://stable-baselines3.readthedocs.io/en/master/modules/a2c.html

10. Luu, Q. T., & Luu, Q. T. (2023, May 2). Q-Learning vs. Deep Q-Learning vs. Deep Q-Network — Baeldung on Computer Science. Baeldung on Computer Science. https://www.baeldung.com/cs/q-learning-vs-deep-q-learning-vs-deep-q-network

11. Science, B. O. C., & Science, B. O. C. (2023, March 24). Epsilon-Greedy Q-Learning — Baeldung on Computer Science. Baeldung on Computer Science. https://www.baeldung.com/cs/epsilon-greedy-q-learning

12. GeeksforGeeks. (2021, September 27). Bellman equation. https://www.geeksforgeeks.org/bellman-equation/

13. Wikipedia contributors. (2023, December 29). Bellman equation. In Wikipedia, The Free Encyclopedia. Retrieved 04:12, January 16, 2024, from https://en.wikipedia.org/w/index.php?title=Bellman_equation&oldid=1192511038

14. DQN — Stable Baselines3 2.3.0a1 documentation. (n.d.). https://stable-baselines3.readthedocs.io/en/master/modules/dqn.html

15. Fig. 7. Advantage actor critic. (n.d.). ResearchGate. https://www.researchgate.net/figure/Advantage-Actor-Critic_fig3_334521853

16. Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015, June). Trust region policy optimization. In International conference on machine learning (pp. 1889-1897). PMLR.

17. Proximal Policy Optimization — Spinning Up documentation. (n.d.). https://spinningup.openai.com/en/latest/algorithms/ppo.html

18. Trust Region Policy Optimization — Spinning Up documentation. (n.d.). https://spinningup.openai.com/en/latest/algorithms/trpo.html

19. Karunakaran, D. (2021, December 16). Trust Region Policy Optimisation(TRPO) — a policy-based Reinforcement Learning. Medium. https://medium.com/intro-to-artificial-intelligence/trust-region-policy-optimisation-trpo-a-policy-based-reinforcement-learning-fd38ff9e996e

20. Wikipedia contributors. (2023, December 27). Travelling salesman problem. In Wikipedia, The Free Encyclopedia. Retrieved 04:19, January 16, 2024, from https://en.wikipedia.org/w/index.php?title=Travelling_salesman_problem&oldid=1191996141

21. Wikipedia contributors. (2023, December 1). Vehicle routing problem. In Wikipedia, The Free Encyclopedia. Retrieved 04:18, January 16, 2024, from https://en.wikipedia.org/w/index.php?title=Vehicle_routing_problem&oldid=1187812878

22. Capacitated Vehicle Routing Problem formulation — AIMMS How-To. (2020, August 31). https://how-to.aimms.com/Articles/332/332-Formulation-CVRP.html