LONDON'S GLOBAL UNIVERSITY

UCL

Machine Learning Solution on Postal Service Industry

Chun Kin Cheung[1]

BSc Computer Science

Dr Carlo Ciliberto

Submission date: May 10, 2024

[1]**Disclaimer:** This report is submitted as part requirement for the BSc Computer Science at UCL. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Abstract

The rapid expansion of E-commence as the aftermath of the pandemic has become one of the most significant industry to drive the world's economy away from depression. The increased demand for online shopping service has led to a sudden rise in demand for postal services. However, the postal industry often suffers from shortages of staff, or lost from traffic incidents which delayed or stopped postal vans from delivering their packages on time. Unsuccessful postage leads to terrible user experience and users tend to ask the delivery person to simply leave the postage outside their doorways, exposed to bad weather condition and criminals which may cost lost in assets. Improvement have to be done to increase the precision of postal companies estimating their delivery time, to both reduce their running cost, and improve user experience.

One way to achieve this is to have better knowledge of the traffics within a postal area and design a optimal route for delivery to maximize efficiency. This is the famous Traveling Salesman Problem. Although many solutions exist in tackling the Traveling Salesman Problem, none of them can solve the problem optimally without any significant drawbacks, which includes high computational time, or extreme inconsistency. The fast development of recent artificial intelligence technology improves many techniques that can be utilised in solving these 'unsolvable' problems. Machine learning especially sparks the interest of this project to study and implement a machine learning solution as an alternative algorithm for solving the Traveling Salesman Problem.

This paper detailed the study of machine learning and an attempt to apply reinforcement learning model on solving a custom Traveling Salesman environment. The implementation is proved to be successful with reasonable performance, while its strengths and weaknesses are vastly evaluated.

Contents

Chapter 1

Introduction

This chapter introduced the point of interest of this project, outlined the objectives to be achieved and explained the structure of this report.

1.1 Problem Outline

At the time of the information age, the invention of smartphones and the fifth generation of mobile network allow internet users to access products and services online unrestricted by their geographic locations. Many retailers seek this opportunity to promote their products via popular online retailing platform and social media, and find their success in reaching a much larger customer base compared to traditional physical shop retailing.

With the ever-growing E-commence industry, the demand for postal service grows drastically over the past few years, alongside the pandemic acting as the major accelerant. However, as the demand for postal services rises, many problems have risen up to the surface. The relatively low-paid, long-working-hour nature of the postal industry often meets a shortage of staff members to meet the high demand. Shortage of staff such as postal truck drivers leads to lower number of trucks to service one postal area thus the reduced chances of customer's postage arriving on time. The increasingly long distance one single truck is required to travel also increases the uncertainty on estimated arrival time due to the higher chance on encountering live traffic incidence. Due to these circumstances, postal service companies often produce vague estimation on their arrival time and many customers found staying home waiting for their parcel to arrive throughout the entire day unacceptable. This leads to many incidences of missing delivery, or posted packages being stolen only because the customers are not present to retrieve the delivered postal.

One way to improve the estimation of post arrival time, is to improve the route taken by the postal truck during deliveries. Reducing the route distance and time needed for one truck to complete not only help reducing the uncertainty of predicting arriving time, but also increases the effectiveness and supply of postal service in general. Solving the problem of optimizing delivery route is in fact, is the Traveling Salesman Problem (TSP).

1.2 Project Objective

While there were many existing attempts, there was never a generic optimized solution discovered to solve TSP linearly. In fact, TSP is a classic nondeterministic polynomial hard (NP-hard) problem that can never be reduced into linear time by existing theory. This leads to the introduction of machine learning algorithm to tackle TSP. Since machine learning model, if learnt successfully, can be re-fit to similar problem with unseen data to provide similar high accuracy solution achievable by other algorithms. The objective of this project is to focus mainly on solving TSP with machine learning algorithm and compare the result with other existing algorithms.

By the end of the project, we hope to implement multiple machine learning models to compare each of their effectiveness on TSP problem, as well as applying them onto TSP of difference sizes to study their versatility with increasing problem dimension. The final aim of this project is to study whether a pretrained model can be applied to new, unseen TSP data to achieve machine learning algorithms' most significant advantage.

1.3 Report Structure

This report consists of 7 chapters. Chapter 1 and 2 explains the project background and interest, including the introduction of Traveling salesman problem and some of the common existing algorithms designed to tackle the problem. Chapter 3 detailed the development of Machine Learning as well as the three classifications of learning models. Chapter 4 explains the methodology and system design used for this project while Chapter 5 detailed the implementation of the methods in a technical style. Chapter 6 studies the obtained results and carry out evaluation. Chapter 7 concludes the project findings and justify for the project goal suggested in Chapter 1, with a few final thoughts on this project.

Chapter 2

Traveling Salesman Problem

This chapter explains the background context of the famous Traveling Salesman Problem. Some of its existing solutions are also explained in detail with their respective methodology and performance.

2.1 The Problem

The Traveling Salesman Problem (TSP) [1] stands as one of the most known and extensively studied problems in the field of computational mathematics. The problem involves finding the cheapest route between a map of locations, in which each pair of locations is connected by a weighted path. The 'traveling salesman' is required to start at one location, visit every other single location on the map exactly once, and finally returning to the starting location. The 'cheapest' route in this sense is free for interpretation. It can be the cost of the 'traveling salesman', the postal truck in our project's term, the total distance travelled, or the total time travelled. Despite the freedom of interpretation, the goal is to minimize the obtained sum. The final output for an attempt to tackle the problem includes the obtained sum of cost and the order of locations travelled, often referred as a tour, or circuit.

Variations of TSP also exist. General TSP can be classified as a symmetric or asymmetric problem. A symmetric problem refers to the weight of traveling from a location to another is equal to the weight of traveling back from the destined location to the starting location. While an asymmetric problem implies that the travelled path and return path between any pair of locations are different. Asymmetric problem is usually a more realistic interpretation of TSP due to traffic constraints and geometric differences, but it doubles the domain of the problem thus its complexity. TSP can also be classified as static or dynamic. A static problem implies that the weights between each pair of locations are kept constant throughout the tour, while a dynamic problem will have changing weights based on actions such as arriving at a new city after some certain amount of time. In order to keep a comparatively low runtime and memory space required for this project to run on common household machines, this project focus on solving a static symmetric TSP.

Although the first mention of TSP was made by Karl Menger as early as 1930, there is yet an optimised generic solution to tackle this seemingly simple task. TSP therefore became popular among computer scientists to experiment with new technologies and algorithms.

2.2 Existing Solution [2]

While there still does not exist a generic solution for solving large sized TSP (large number of locations to visit) up to this date, there exist a few exact algorithms that are feasible for small problem instances.

2.2.1 Exact Solution Approaches

The simplest exact approach is the brute-force search, which involves trying all possible permutations of paths and concludes on the shortest one. This has the worst time complexity of exponential time $O(n!)$ of all algorithms, making it impractical even for small instances of the problem.

The most known exact algorithm is the Concorde TSP solver, written by Applegate et al. in 2006 [1]. This algorithm is based on branch-and-cut-and-price, meaning that both some constraints and variables are initially relaxed and dynamically generated during the solution process. Recorded by its book, Concorde shows excellent efficiency and consistency for solving TSP with instances less than or equal to 2392. However, there were two exceptions through experimentation with long computational time up to a maximum of 18226404.4s. This shows the worst-case complexity of Concorde to still be impractical, especially for large instances problem. The largest instance solved optimally by Concorde was performed by Applegate et al. in 2009, containing 85900 vertices. The time required equals 136 CPU years.

The Held-Karp lower bound is another exact solution algorithm utilising dynamic programming approach to produce a lower bound to the optimal solution [3]. While finding the optimal solution using Held-Karp is less efficient compared to Concorde, it can approximate a close lower bound much more effectively.

2.2.2 Approximation Approaches

To tackle the high computational time required for exact algorithms, approximating algorithms and heuristic algorithms are introduced to approximate the minimum cost of tour close to the optimal solution, with lower time complexity. The Held-Karp lower bound mentioned in the last section is often used to provide a foundation to justify the performance of heuristic algorithms. Some of the common approximation and heuristic algorithms includes:

Closest Neighbour heuristic

Also known as the Nearest Neighbour algorithm (NN). This is the simplest TSP heuristic. Closest neighbour heuristic begins by choosing a starting location and choosing the next closest unvisited location to travel. Once all locations are visited, the edge between the starting location and ending location is chosen to complete the tour. This approach generally keeps its tour within 25% of the Held-Karp lower bound thus it is not considered to be a highly accurate algorithm. However, this heuristic approach is associated with a polynomial complexity of $O(n^2)$, allowing it to be one of the most computational efficient heuristic algorithms. However, some map structures can be hard for closest neighbour heuristic to perform, such as maps with similar distance between all locations, or centre-populated locations surrounded by a ring of faraway locations.

Christofides-Serdyukov heuristic

Also known as the Christofides heuristic. It combines a minimum spanning tree and a solution to the minimum-weight perfect matching problem to create a Euler cycle from the combined graph, and traverse it taking shortcuts to avoid visited notes. Christofides' algorithm is shown by tests to produce a solution at around 10% above the Held-Karp lower bound. It has been proved to have a polynomial time complexity of $O(n^3)$. The Christofides heuristic still stands as one of the best polynomial time approximation algorithms for TSP. A novel approximation algorithm was introduced by Anna R. Karlin, Nathan Klein and Shayan Oveis Gharan in 2020 to explore Christofides heuristic with a randomly chosen tree instead, which provided slight improvement to the algorithm[4].

Lin-Kernighan

Lin-Kernighan (LK) is a local search algorithm and one of the most effective heuristics to solve the symmetric TSP. It is an optimizing algorithm aims to improve an existing tour (A known Hamiltonian cycle) by exchanging k variable within the tour to locate a shorter cycle. LK can possibly generate a solution within 2% of the Held-Karp lower bound with a time complexity of $O(n^{2.2})$. The algorithm is believed to be fully optimised, and it is hard to achieve further improvement.

k-opt

Similar to Lin-Kernighan, k-opt algorithm studies an existing Hamiltonian cycle and disconnect k edges within the cycle. The once connected nodes are then reconnected using a set of edges to form a new Hamiltonian cycle. The algorithm will repeat the process until the cost of the cycle converges to a minimum. High k value of k-opt algorithms can produce solution with higher accuracy but with a cost of more computational time. 2-opt and 3-opt algorithms are therefore more popular for computer scientists to obtain an approximation compared to k¿4 algorithms, as they can already produce highly accurate solution. 2-opt often results in a tour with a length less than 5% above the Held-Karp bound, while 3-opt can improve the result within a 3% boundary.

Chapter 3

Machine Learning

This chapter explains the background context of Machine Learning. The three categories of machine learning are listed and explained in detail with their speciality, and some of their common algorithms.

3.1 Introduction to Machine Learning

With the rise of the information age and internet being popularize, digital data is being generated and collected from billions of internet users every day. According to a data set published by Statistica, 120 zettabytes of digital data is collected worldwide per day in 2023 [5]. This astronomical number of data records every tiny bit of human activity and by carrying out the correct handling and analysis on these data, it can provide multimillion worth of assets to data scientist. The study of data analysis has since become one of the most popular topics in recent decades, as it carries the solution to many undiscovered human behaviours that cannot be studied by generic computer algorithms. However, the massive amount of newly generated data everyday makes the study of it by human force impossible.

The idea of machine learning was therefore introduced in the 1950s, through the famous Turing Test proposed by mathematician Alan Turing, to determine whether a machine can act like a human, or if humans can't tell the difference between human and machine given answers [6]. The Turing Test introduced the idea of developing machines to think, learn and make decisions like a human would, thus the introduction of artificial intelligence. Between the year 1957 and 1960, the first learning-based machine was introduced by psychologist Frank Rosenblatt from Cornell University, named the 'Perceptron' for recognizing the letters of alphabets. The machine was designed based on the work of the human nervous system and became the first prototype of the modern artificial neural network technology [7].

Modern machine learning, defined as 'the use and development of computer systems that are able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyse and draw inferences from patterns in data.' by Oxford Languages[8], is a field of study in artificial intelligence, which is used by machines to statistically study the unknown trends and correlations of large datasets, to extract the hidden nature behind, and to finally make decisions similar to a human mind level.

Sophisticated machine learning solution provides the advantage of utilising the most of computational resources to understand the relation within large datasets that is unknown to other data analysing strategy. Moreover, once enough data is studied, learnt model can be applied to similar pattern of unseen data to generate sensible response based on experience, without relearning the dataset. This is by far the most significant advantage machine learning provides compared to other algorithms. Ideally, if a perfectly designed and learnt universal model can be found for a high complexity problem, all problems of the same kind can be reduced and solved linearly in $O(n)$ complexity.

3.2 Types of Machine Learning Techniques

The difficulty of data analysis does not only come from the massive number of data, but also the hugely diverse data type it can be presented. The effectiveness of an analysing strategy on numerical data cannot be reproduced if it was carried on language-based data involving strings of words. Different nature of problems and their expected output also suggest that there does not exist a single machine that can learn all types of data and produce suggestive decisions on all types of problem in the world. Different machine learning models are therefore introduced throughout its development to specialise in studying specific problems with suitable datatypes. Models, in the sense of machine learning, refers to a programmable object trained to recognize data patterns. Mathematical algorithms, also known as machine learning algorithms, are then feed into the model, to learn from those data and evaluate them with reasons, such as making a decision or a prediction [9].

These models and algorithms can be classified into three categories, known as:

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

Each of the three categories specialised in recognizing and learning their own range of problems and data patterns, which will be further discussed.

3.2.1 Supervised Learning [10]

The supervised learning technique specialised in generating a model function, meaning that it can be widely used on functional problems that involve taking in inputs X as data parameters and generate an output Y using a dedicated function. Studying the relationship and influences between X and Y, thus generating and adjusting the possible function to map X to Y is the main goal of supervised learning models. Common usage of supervised learning includes:

Classification

A general classification problem requires user to input features of an observation and allow the machine to decide its corresponded category based on the features' constraints. E.g. A machine that determines whether an image of an animal is terrestrial or aquatic (Output Y), by studying the features of the image such as fins and legs (Input X).

Regression problems

A general regression problem involves making prediction based on input parameters. The learning model aims to study the correlation of each parameter and adjust their coefficients to produce an accurate output. E.g. A machine that takes in attributes of a car, such as brand, year, engine and mileage (Input X), to predicts its current value (Output Y).

As the name suggests, supervised learning involves supervision, that is when the model is learning to recognize a given dataset, named as the supervisor, it must contain both input and output data of each instance for the model to justify its correctness. The model training is stopped once enough data is fed into the model and the model function converges. The model will then be fed with an unseen dataset of input parameters to produce their respective outputs. Human supervision is finally required to study whether the generated output is accurate, and further adjust the model if it is proven wrong.

Common supervised learning model includes:

- Logistic Regression
- Naïve Bayes
- Decision Trees
- K Nearest Neighbour (kNN)
- Random Forest

3.2.2 Unsupervised Learning

The unsupervised learning technique specialised in discovering regularities of the input. Unlike supervised learning, a dataset containing only inputs is feed into the model. The model is designed to recognize certain patterns that occurs more frequently than the others within the dataset, to discover possible trends and relationship within the data.

The usage of unsupervised learning is referred as density estimation in statistics, and clustering is a common method to achieve such goal by finding clusters and groupings of similar data pattern. Example of such usage is a search engine, in which the model studies websites traffic to identify user preferences based on common patterns, such as gender and accessed location, to determine the display order of searched results.

Common unsupervised learning model includes:

- K-Means
- Hierarchical Clustering

3.2.3 Reinforcement Learning

The reinforcement learning technique specialises in policy training rather than dataset driven supervised and unsupervised learning models. That is, reinforcement learning models do not necessarily study large dataset but an environment. Within a reinforcement learning model, there exists an agent inside a prebuilt environment and the output of the model will be a set of actions, named policy, for the agent to perform inside the environment. Due to its special design, reinforcement learning specialises in solving optimizing problem as it is able to determine all necessary actions for a strategic problem to obtain a maximum/minimum result or a targeted value.

One common usage of reinforcement learning is game playing, by treating the player as the agent within the game environment, a strategy of actions to obtain the maximum score is learnt and can be reproduced on similar environments. One example is the cart-pole problem, which a cart is introduced in an environment that allows it to move either left or right, and a pole pinned to the centre of the cart is released at a vertical position. The player has to decide whether to move the cart on either direction to balance the pole at its vertical position for as long as possible.

As mentioned above, a reinforcement model must consist of the following attributes:

- **Agent**: The action operator inside the environment
- **Environment**: The rules of the problem
- **States**: The situation the agent experiences between actions. The next action should be decided based on observations of different states.
- **Actions**: Instructions allowed for the agent to perform within the environment
- **Reward**: A scoring system that is used to judge the agent after an action is performed. Maximizing the reward is a common practise of learning.

Reinforcement learning model is self-supervising. Although the learning process is supervised similar to a supervised model, reinforcement learning model is not fed with output data. It studies the environment and generate output solutions by itself, usually with a random number generator, and compare their scoring. The policies generated above a certain score requirement is kept and will be further adjusted to aim for a higher score (evolve), while scraping the ones below, until the achieved score converges.

- Common reinforcement learning model includes:
- Asynchronous Advantage Actor Critic (A2C/A3C)
- Deep Deterministic Policy Gradient
- Q Network / Deep Q Network
- Proximal Policy Optimization
- State–action–reward–state–action (SARSA)

Chapter 4

Methodology

This chapter detailed the methodology used in this projects program in a high level style. The design of the learning model is defined and discussed. This chapter also evaluates on the three existing reinforcement learning algorithms that will be used by this project in training the mentioned model.

4.1 Learning Model Design

This project aims to tackle the static symmetric Traveling Salesman Problem (sTSP) utilising machine learning technique. As suggested in the previous chapter, reinforcement learning specialises in solving optimization problem, hence it will become the focused machine learning technique for this project.

A reinforcement model must specify its five important attributes: agent, environment, state, action and reward (3.2.3). The attributes used by the custom model in this project is detailed as below.

Agent:

The traveling salesman in the map of locations (nodes). It is responsible for performing actions and recording observation after each action is taken. The observation includes the distance (cost/weight) travelled and the order of the nodes travelled, which will also be the path of the found tour.

Environment:

A 2-dimentional square matrix of size n, with n being the number of nodes/instances, containing the cost of traveling from one node to another (referred as weight). As it is an sTSP environment, the square matrix should be symmetrical, representing the cost between two nodes should be equal in both directions. The matrix should follow the following rules:

$$\text{Let } n \text{ be the problem dimension}$$

$$\text{For all } (x, y),\ x \leq n, y \leq n$$

$$weight = Matrix[x][y]$$

$$Matrix[x][x] = 0$$

$$Matrix[x][y] = Matrix[y][x]$$

The problem environment is considered done, if the recorded tour by the agent contains every node within the environment at least once, or it is forced to terminate if the agent has taken $2n$ actions (steps) as it suggests that all nodes can be visited at least twice.

State:

A discrete number of range n, with n being the number of nodes in the problem. It represents the location of the agent, thus the node it is currently visiting.

$$\mathit{State} \in \{0, 1, 2, ..., n-1, n\}$$

A unique observation is corresponded to each state for the agent to decide the next action. The observation space is defined as a box of n continuous value in the range of $[0, +\infty)$. The observation is equivalent to the weight between current state and all other nodes in the environment.

$$\mathit{Observationx} = Matrix[x]$$

Action:

A discrete number of range n, with n being the number of nodes in the problem. It represents the next node the agent is going to visit.

$$\mathit{Action} \in \{0, 1, 2, ..., n-1, n\}$$

Reward:

Rewards can be given to the agent in three different ways. The first reward is given per action taken (step) as the negative of the weight between the current state and the next action, or as a large negative number as penalty, when the agent revisits a visited node or stay stationary (action is equal to state).

The second reward is given as a large positive number when the problem is considered done, to encourage the model to converges on reducing the actions needed to visit all nodes at least once.

The third reward is given as a large positive number when the tour recorded by the agent only contains each unique node once. This encourages the model to converge on finding a Hamiltonian cycle as the optimised tour.

$$\text{Reward} \begin{cases} \text{-weight,} & \text{if next\_action !in tour} \\ \text{-penalty,} & \text{if next\_action in tour} \\ \text{-penalty,} & \text{if next\_action = state} \\ \text{+penalty,} & \text{if done = true} \\ \text{+penalty,} & \text{if tour = Hamiltonian cycle} \end{cases}$$

For a reward maximizing strategy, this encourages the learning model to pick the lowest weight action whenever possible and visit each and all nodes at most once. This reward scheme is in fact a learning model following the Closest Neighbour Heuristic (2.2.2).

4.2 Advantage Actor Critic (A2C) [11]

The designed model will be learnt using three different learning algorithms. The first one being Advantage Actor Critic. Asynchronous Advantage Actor Critic, often referred to as A3C, is an asynchronous variant of deep neural networks designed for reinforcement learning. Although deep neural networks provide rich representations that can enable reinforcement learning algorithms to perform effectively, applying simple learning algorithms with deep neural networks is often unstable. Asynchronous Advantage Actor Critic introduces multiple parallel actor-learners within the same deep learning model which can effectively stabilized and speed up the learning process. Later study, however, showed no evidence that the asynchrony provided by A3C provides any performance benefit. A synchronous deterministic variant of A3C, reduced as A2C is introduced as a more cost-effective model with equal performance. The A2C model involves multiple actors with an updating scheme that operates on fixed-length segments of experience and uses these segments to compute estimators of the returns and advantage function, under a shared layers between the policy and value function Architectures.

4.3 Proximal Policy Optimization (PPO) [12]

Proximal Policy Optimization combines the idea of A2C stabilization (4.2) and Trust Region Policy Optimization (TRPO) [13], which aims to study the next policy taking the biggest possible improvement step, without overshooting to cause performance collapses. While TRPO provides a more accurate estimation of the next improvement step, it is much more computational heavy as it utilises a second-order method. PPO utilises first-order method with high efficiency, but with a slight performance trade off. PPO trains a stochastic policy in an on-policy way. It explores by sampling actions according to the last latest policy, and the randomness on action will gradually decrease and converges by stochastic gradient descent. Although PPO exploration depends on less hyperparameters compared to TRPO, there is no guaranteed that PPO will always avoid overshooting and get trapped in local optimal.

4.4 Deep Q Network (DQN) [14]

Deep Q Network is the neural network version of the standard Q-learning algorithm in reinforcement learning. Q-learning algorithm aims to generate a unique Q-value, which is referred as the reward function, for each action used by the agent in a look up table. Once the look up table is fully learnt, the agent can simply generate the best policy by following actions with the highest Q-value. While its low complexity provides fast computations, the Q-table requires large memory size, especially when the number of state and their respective possible actions get very large. The high space complexity of Q-learning will prove unpractical when a high dimension problem is introduced. Instead of storing the Q-value in a look up table, a neural network is used instead to reduce space complexity. With each iteration, the maximum output is stored inside the network and the next action is determined based on stored experience, to further maximize obtained Q-value and reduce the loss function to stabilize the model.

Chapter 5

Implementation

This chapter detailed the technical implementation of the project program. It listed the setup requirements including the list of system environment the program executed in, the data sources used and their structure, and descriptions of libraries imported. Then the structure of the custom environment class is explained in detail. Finally the ways of evaluation of the model are introduced.

5.1 Implementing Environment

5.1.1 Hardware Environment

Reinforcement learning often requires high computational power and memory space. Computational time and space required for a learning model will therefore vary vastly among different hardware environments. Hence it is important to recognize the hardware environment before carryout computational time and space analysis. The project program implementation is carried using the hardware environment listed below:

- **Processor**: 13th Gen Intel(R) Core(TM) i9-13980HX, 2200 Mhz, 24 Core(s), 32 Logical Processor(s)
- **Physical Memory (RAM)** : 32.0GB 5600Mhz
- **Graphical Processing Unit 0** : Intel(R) UHD Graphics
- **Graphical Processing Unit 1** : NVIDIA GeForce RTX 4080 Laptop GPU

5.1.2 Software Environment

The software environment of this project's program implementation is listed as below:

- Windows 11 Home Version 23H2
- Python 3.12.2
- Jupyter Notebook 7.0.8
- NVIDIA Cuda 12.1

All python code is executed inside an Anaconda environment using Jupyter Notebook. All learning models are processed using GPU acceleration supported by PyTorch and Cuda 12.

5.2 Problem Data

All TSP data used in this project is originated from TSPLIB produced by Gerhard Reinelt [15]. TSPLIB provides problem data for common optimization problem including Symmetric Traveling Salesman problem, Hamiltonian cycle problem, Asymmetric traveling salesman problem, Sequential ordering problem and Capacitated vehicle routing problem. TSP problem files and their attributes are fully documented in its documentation [16]. The optimal solution of each of the provided TSP problem is also listed in its document [17].

The TSP problems studied by this project, and their significant attributes are listed below:

- gr17
  - DIMENSION: 17
  - EDGE_WEIGHT_TYPE: 'EXPLICIT'
  - EDGE_WEIGHT_FORMAT: 'LOWER_DIAG_ROW'
- gr24
  - DIMENSION: 24
  - EDGE_WEIGHT_TYPE: 'EXPLICIT'
  - EDGE_WEIGHT_FORMAT: 'LOWER_DIAG_ROW'
- gr48
  - DIMENSION: 48
  - EDGE_WEIGHT_TYPE: 'EXPLICIT'
  - EDGE_WEIGHT_FORMAT: 'LOWER_DIAG_ROW'
- kroA100
  - DIMENSION: 100
  - EDGE_WEIGHT_TYPE: 'EUC_2D'
- bayg29
  - DIMENSION: 29
  - EDGE_WEIGHT_TYPE: 'EXPLICIT'
  - EDGE_WEIGHT_FORMAT: 'UPPER_ROW'
- bays29
  - DIMENSION: 29
  - EDGE_WEIGHT_TYPE: 'EXPLICIT'
  - EDGE_WEIGHT_FORMAT: 'UPPER_ROW'

The attribute DIMENSION defines the number of nodes to be visited in the problem. The attribute EDGE_WEIGHT_TYPE and EDGE_WEIGHT_FORMAT explains the format of the problem file used to record weights between nodes. LOWER_DIAG_ROW suggested the weight to be given as an array of weights treated as a lower triangular matrix, with each row separated by the value 0 as the diagonal. 'UPPER_ROW' suggested a 2d array of weights formatted as an upper triangular matrix. In the kroA100 problem, no weight but the 2d Euclidean coordinates of each node is given.

The study of problems is divided into three groups with their unique objectives. The study of 'gr' problems aims to analyse the performances of the three different chosen learning model on problems of varying, but small sizes. The study of 'kroA' problem aims to study the performance of learning models on a large size problem. The study of 'bay' problem simulates a pre-trained model being reapplied to unseen environment with similar pattern. 'bayg29' will be used to pre-train the model and 'bays29' will be the verifying (unseen) environment.

5.3 Python Libraries

There exist many python libraries that can support data scientists with pre-built learning models for fast deployment. Pre-built libraries are often more efficient compared to custom instructions to allow for better time complexity analysis. All libraries used in this project and their respective functionality are detailed below:

numpy v1.26.4 [18]

The Numpy library is a commonly used library across all python programs. Numpy supports custom array processing, and it is essential for creating the model environment as 2d matrix used can be defined as a 2d Numpy array. Numpy has an extensive toolkit built for matrix processing.

tsplib95 v0.7.1 [19]

The Tsplib95 library supports file handling for TSBLIB problem files. It helps convert the problem files data into standard problem class for easier access to the data. While it has some built in classes and methods for problem handling, only the load() method is used in our project for customizability.

gymnasium v0.29.1 [20]

The Gymnasium library is a newly maintained fork of the old Gym library developed by OpenAI. It allows the creation and handling of OpenAI Gym environments that is compatible to most reinforcement learning libraries. A snippet of the structure of the environment class is abstracted below:

Class gymnasium.Env

- Methods
  - step()

    Run one timestep of the environment's dynamics using the agent actions.
  - reset()

    Resets the environment to an initial internal state, returning an initial observation and info.
- Attributes
  - Env.action_space

Run one timestep of the environment's dynamics using the agent actions.

  - Env.observation_space

    Resets the environment to an initial internal state, returning an initial observation and info.

The space object contains subclasses of spaces that are used for the environment to define possible action and observation value. The two subclasses used in this project is listed below:

Class gymnasium.spaces.Box

- Parameters
  - low
  - high
  - shape
  - dtype
  - seed

The box object contains n shape real number value of between the range defined by parameters [low, high]. Random sample of the values can be taken with rng. The box can be reduced to Discrete if dtype is set to integer type.

Class gymnasium.spaces.Discrete

- Parameters
  - n
  - seed
  - start

The discrete object contains n integer, with the smallest integer value being start, and following values incrementing by 1 step. Random sample can be taken with rng to return one integer value defined in the space.

stable_baselines3 v2.3.1 [21]

Stable Baselines 3 is a set of reliable implementations of reinforcement learning algorithms defined in PyTorch [22]. It allows PyTorch algorithms to be compatible with Gymnasium environment as well as supporting Tensorboard for model analysis. The three training model classes used by this project are PPO, A2C and DQN, which are subclasses of the base RL class. The defined parameters and used methods in this project are abstracted below:

Class stable_baselines3.common.base_class.BaseAlgorithm

- Parameters
  - policy
  - env

    - verbose
    - tensorboard_log
- Methods
    - learn()
    - load()
    - save()
    - predict()
- When learn() is called, the class applied training algorithm on env using defined policy.
- save() allows saving current model in defined path while load() recalls saved model into BaseAlgorithm class.
- predict() generates the next policy action based on observation and model.
- Defined tensorboard_log path allows Tensorboard to log information during training.
- Policy is defined as MlpPolicy for this project as action is in Discrete space (5.4).

There are also two methods of callbacks and a method of evaluation imported. The abstracts of the two classes are listed below:

Method stable_baselines3.common.callbacks.EvalCallback()

- Evaluate periodically the performance of an agent, using a separate test environment.
- It is used to identify the training instance with the best performance periodically and save the model as best_model to avoid overshooting or diverging.

tensorboard v2.16.2 [23]

Tensorboard is called by Stable Baselines 3 to log the training process. Tensorboard can provide an interactive UI for graphical display of log data, which will be heavily used by this project in the evaluation process.

5.4 Custom Environment Class

The custom environment will be the main class that holds all the required methods and attributes for training. It is a subclass of gymnasium.Env (5.3). The environment class will be passed into Stable Baselines 3's training algorithms as a parameter to achieve training. The attributes, parameters and methods of the custom environment class are detailed below:

Class tspEnv (gymnasium.Env)

- • Parameters
    - problem

      The problem class generated by tsplib95 (5.3)

  - penalty (default value = 10000)

    A large penalty value used as a reward applied by the defined reward scheme explained in section 4.1. The same value is used for completing a tour and producing a Hamiltonian cycle (5.5). A good estimator for this value will be the score achieved by taking random actions to form a Hamiltonian cycle.

- Attributes
  - tspEnv.problem

    A problem class set up by tsplib95. It is passed by the parameter when constructing the object of class tspEnv.

  - tspEnv.action_space

    A discrete space of action with n as the problem dimension, starting from 0. The definition of action is explained in section 4.1.

  - tspEnv.observation_space

    A box space of size n as the problem dimension, with range [0, +infinity]. One observation at state k contains costs of traveling to another node within the environment as defined in section 4.1.

  - tspEnv.tour

    A list recording the nodes travelled by the agent in travelling order.

  - tspEnv.max_length

    An integer value defining the maximum length of the tour before the policy is forced to stop. It is $2n$ for n as the problem dimension, explained in section 4.1.

  - tspEnv.start

    An integer value between $[0, n-1]$ indicating the starting state of the agent (starting position)

  - tspEnv.penalty

    A positive integer value. It is passed by the parameter when constructing the object of class tspEnv. Its use and suitable value is explained above under bullet point 'parameters'.

  - tspEnv.w0_matrix

    A 2d numpy array. This is a 2d square symmetric matrix containing the starting cost of traveling between two nodes in the environment. This attribute should remain constant throughout the training.

  - tspEnv.w_matrix

    A 2d numpy array. This is a 2d square symmetric matrix containing the weight of traveling between two nodes in the environment. This matrix is altered after each step to replace the weight between visited node to tspEnv.penalty to discourage the policy from traveling to already visited node.

- Methods
  - step(action)

    The main method taken by the agent after each action. The functionality of the step method is further explained below.
  - next_rand_action()

    Return a random integer value within the action space that is not yet recorded inside the tour.
  - render()

    Render function of displaying the map of nodes graphically. This method is not used in this project.
  - reset(seed, option)

    Reset the current environment. Emptying the recorded tour, resetting the w_matrix back to w0_matrix, returning starting state to start, and getting the observation of the starting state.
  - _get_state()

    Return the last instance of the tour as the current state
  - _get_reward(state, action)

    Return the negative of the weight between state and next action, recorded in w_matrix.
  - _get_obs(action)

    Return the action indexed row of w_matrix.
  - _get_w_matrix()

    Format the problem data and return a 2d square symmetric matrix as the weight matrix, with diagonal value equivalent to the penalty of the object.
  - _update_matrix(action)

    Replace all weight of the action's column by the penalty value on w_matrix and return it.
  - _is_subset(sub_list, list)

    Return a boolean value based on whether the sub_list is a subset of list.

Method step(action)

The step method represents the agent of the environment (4.1). It is responsible of recording the action taken, the new observation obtained, alternating the environment, and make decision on whether the policy is completed. It is also responsible for applying reward based on the instance. The pseudo-code of the step method is displayed below.

Algorithm 1 Method Step

```
get current state
get observation from next action
get reward according to next action
record the new action into the tour and mark it as 'visited'
if tour contains all nodes at least once then
    done = True
    get reward between next action and starting state        ▷ back to the starting node
    reward is received for completion
    if tour contains all nodes exactly once then
        reward is received for obtaining hamiltonian cycle
    end if
end if
if tour length == 2n then
    force_stop = True
end if
Return observation, reward, done, force_stop
```

5.5 Environment Testing and Model Evaluation

There are five ways of testing and evaluating the environment and the model. To test the environment, the function random_tour is called.

The random_tour function constructs a Hamiltonian cycle by taking random actions returns the average score of all the score obtained from all episodes. This obtained average is used as a penalty when constructing the environment for training to avoid the penalty becoming under or over influential.

After training the model by passing the environment into stable baselines' algorithms and obtaining the best model through callbacks, the performance of the model can be evaluated in the following three ways.

Method stable_baselines3.common.evaluation.evaluated_policy()

- Parameters
  - model
  - env
  - n_eval_episodes

This method applies the input model onto the input environment and run its policy for n_eval_episodes times, finally returning the average score and standard deviation of the obtained scores of all episodes. It is important to note that the obtained score of the policy is offset by the completion reward hence it is not the total cost of the tour.

Function manual_eval()

- Parameters

    - model
    - env
    - episode (default value = 10)

This is a custom function which performs similarly as the evaluated_policy method, but it prints out extra information including the tour obtained, the length of the tour and the total weight of the tour obtained for each episode, calibrated by deducting the offset of completion reward.

Function best_score_eval()

- Parameters
    - model
    - env
    - episode (default value = 10000)

This is a custom function which performs the same instructions as the manual_eval() function. This function is used to run the policy for large amount of episodes and it only prints out the policy's information whenever it obtains a better score (less cost) compared to past episodes.

The final way of evaluation is through the graphical representation of the training process generated using Tensorboard. Graphical logs by Tensorboard includes:

- mean_ep_length using EvalCallBack
- mean_reward using EvalCallBack
- ep_len_mean
- ep_rew_mean
- exploration_rate
- fps
- loss

All plots are plotted against the number of episodes generated. Length represents the length of the obtained tour. Reward represents the total reward obtained including the weight travelled and completion reward. Exploration rate represents the step of the learning algorithm taken when adjusting using gradient descent. Fps represents the number of episodes completed per second. Loss represents the loss function of the policy.

Chapter 6

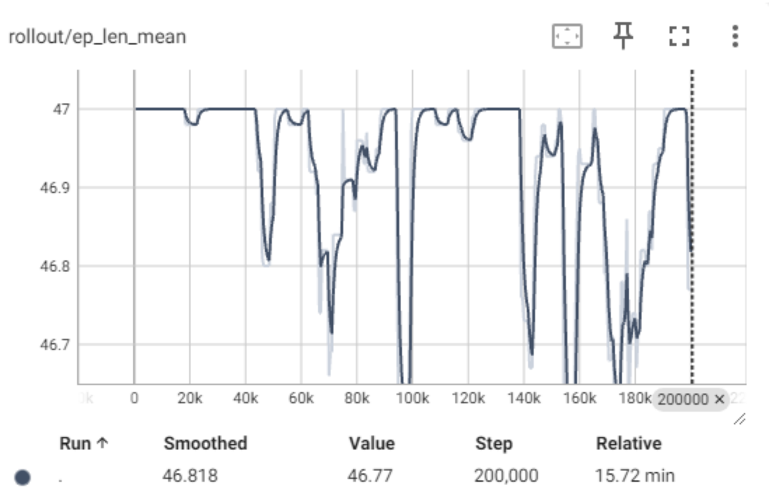
Evaluation

This chapter includes the evaluations of obtained results from the implementation. Evaluations are based on three separate groups of problems, namely the ’gr’, ’kro’ and ’bay’. Each set of problems are evaluated based on different aspect including performance, effectiveness, and reusability.

6.1 A2C and PPO

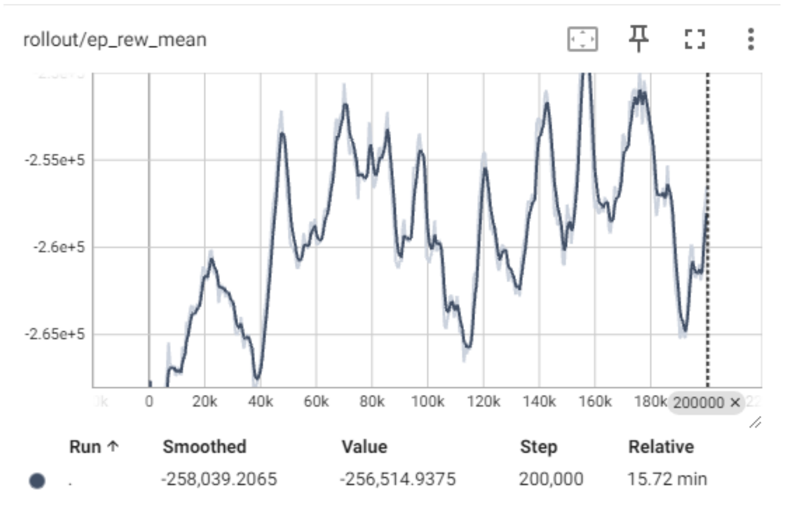
To start off with evaluations, this project first examined the performance of the three different training algorithms, A2C, PPO and DQN. The three algorithms were separately run on the problem gr24 for 200,000 episodes. The log of each run is displayed below.

A2C



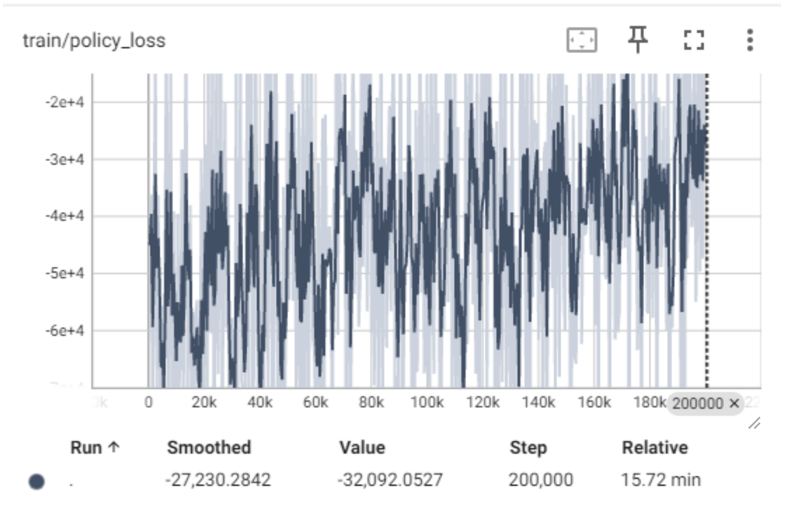
Run ↑	Smoothed	Value	Step	Relative
● .	46.818	46.77	200,000	15.72 min

Figure 1A: A2C tour length mean per episode (gr24)



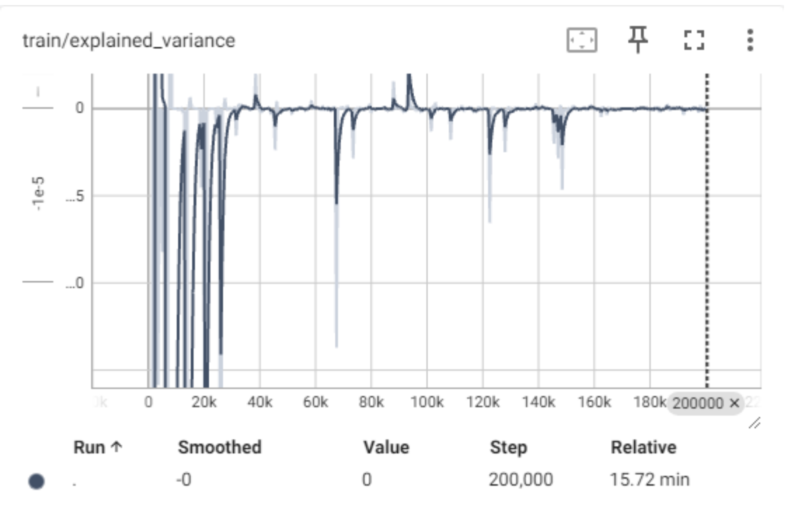
Run ↑	Smoothed	Value	Step	Relative
● .	-258,039.2065	-256,514.9375	200,000	15.72 min

Figure 1B: A2C reward mean per episode (gr24)



Run ↑	Smoothed	Value	Step	Relative
● .	-27,230.2842	-32,092.0527	200,000	15.72 min

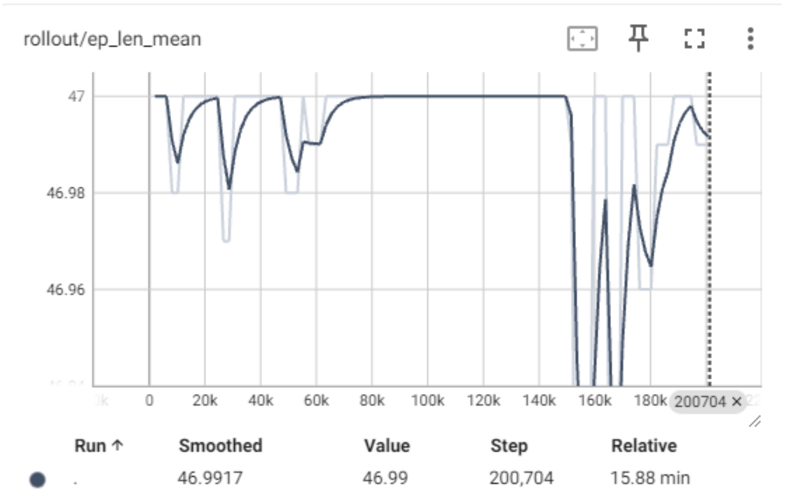
Figure 1C: A2C tour length mean per episode (gr24)



Run ↑	Smoothed	Value	Step	Relative
● .	-0	0	200,000	15.72 min

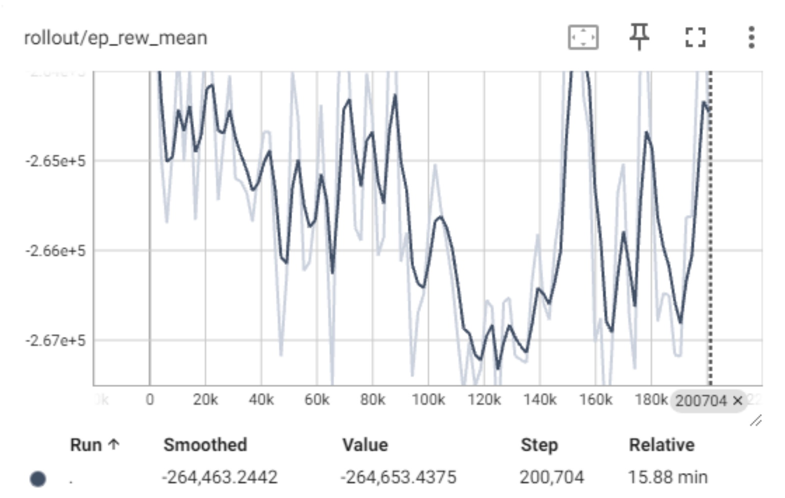
Figure 1D: A2C variance (gr24)

PPO



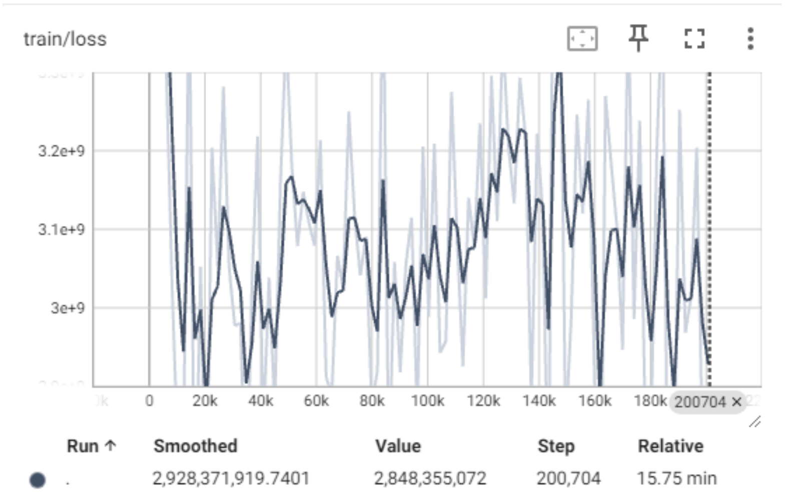
Run ↑	Smoothed	Value	Step	Relative
● .	46.9917	46.99	200,704	15.88 min

Figure 2A: PPO tour length mean per episode (gr24)



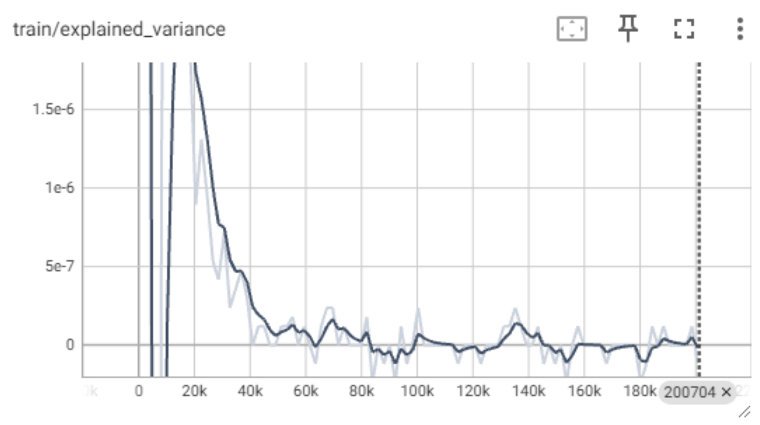
Run ↑	Smoothed	Value	Step	Relative
● .	-264,463.2442	-264,653.4375	200,704	15.88 min

Figure 2B: PPO reward mean per episode (gr24)



Run ↑	Smoothed	Value	Step	Relative
● .	2,928,371,919.7401	2,848,355,072	200,704	15.75 min

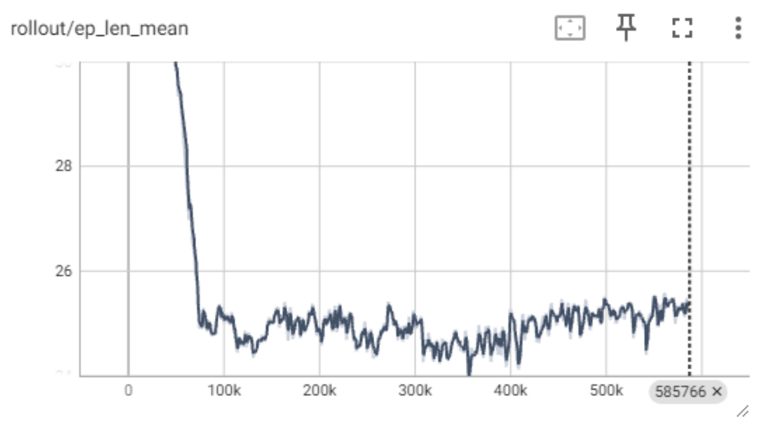
Figure 2C: PPO loss function (gr24)



Run ↑	Smoothed	Value	Step	Relative
● .	-0	-0	200,704	15.75 min

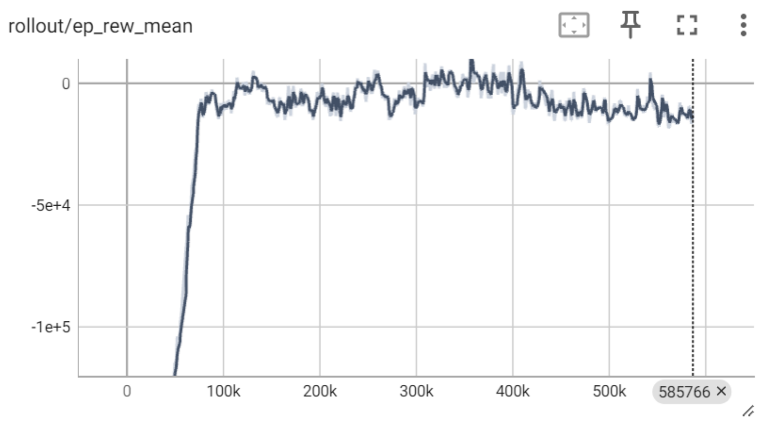
Figure 2D: PPO variance (gr24)

DQN



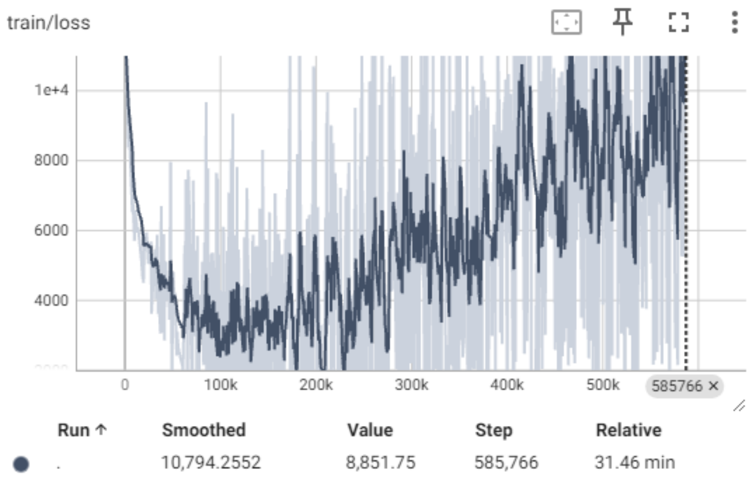
Run ↑	Smoothed	Value	Step	Relative
● .	25.3252	25.27	585,766	31.46 min

Figure 3A: DQN tour length mean per episode (gr24)



Run ↑	Smoothed	Value	Step	Relative
● .	-13,629.063	-12,272.8701	585,766	31.46 min

Figure 3B: DQN reward mean per episode (gr24)



```
Run ↑   Smoothed       Value      Step      Relative
●  .    10,794.2552    8,851.75   585,766   31.46 min
```

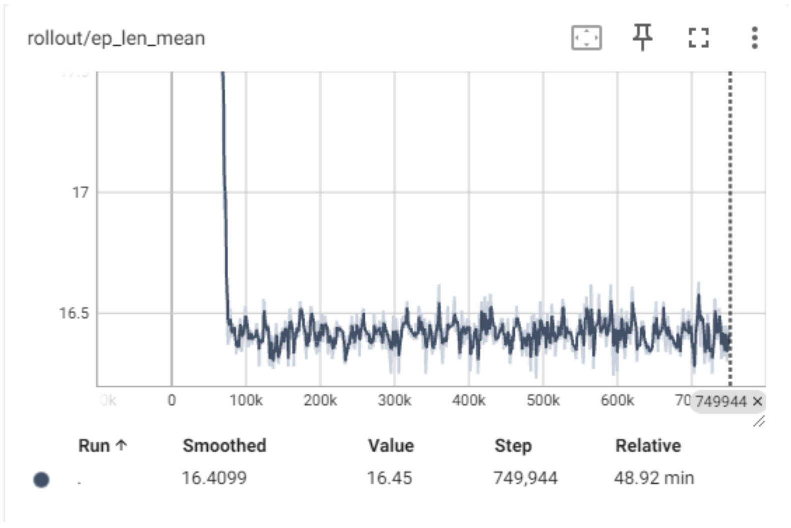
Figure 3C: DQN loss function (gr24)

According to the graph of loss function, both A2C and PPO show no convergence with large amplitude of noise, while the variances of training are approaching zero, whereas the loss function of DQN shows a global minimum at around steps 170,000. At around steps 75,000 DQN is successful in reducing the average length of the tour close to 24, the dimension of the problem and the score achieved is approaching to the positive region, stating that the policy is able to obtain the completion reward and the Hamiltonian cycle reward. On the other hand, both A2C and PPO shows no significant result in both reducing the length of the found tour and the obtained score. The convergence to 0 of the variances of both A2C and PPO suggested that they are both trapped within local optimal which are destructive for the learning model. Both A2C and PPO are therefore treated as unsuitable for this project's environment and are no longer used for evaluating other problems.

6.2 Overall Performance

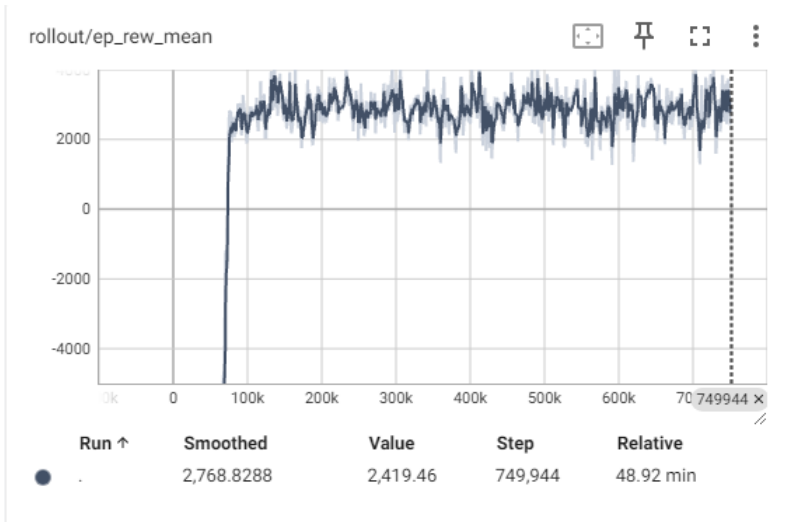
As suggested in the previous section, the performance analysis only accounts for DQN algorithm. Once proofing DQN to be the most effective on solving this project's environment, each problem, except problem bays29, is trained using 750,000 steps, with callback evaluation every 10,000 steps to obtain the best model during training. The log of problem gr17, bayg29 and gr48 are displayed below.

gr17



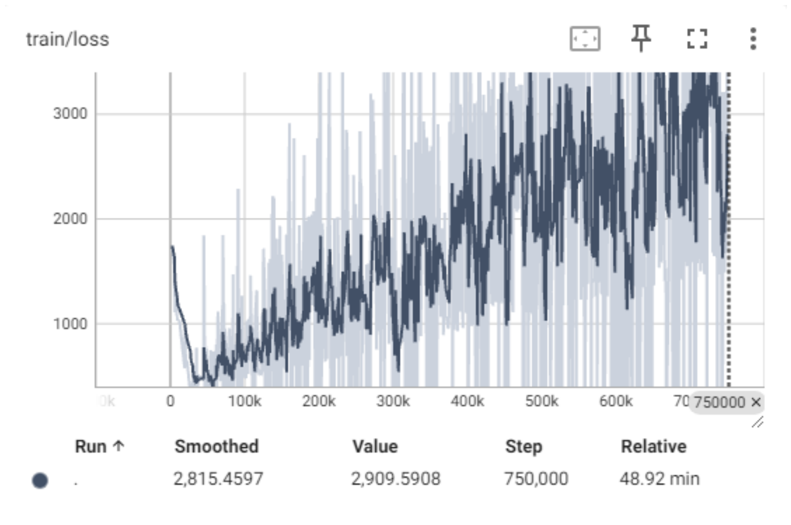
Run ↑	Smoothed	Value	Step	Relative
● .	16.4099	16.45	749,944	48.92 min

Figure 4A: gr17 tour length mean per episode



Run ↑	Smoothed	Value	Step	Relative
● .	2,768.8288	2,419.46	749,944	48.92 min

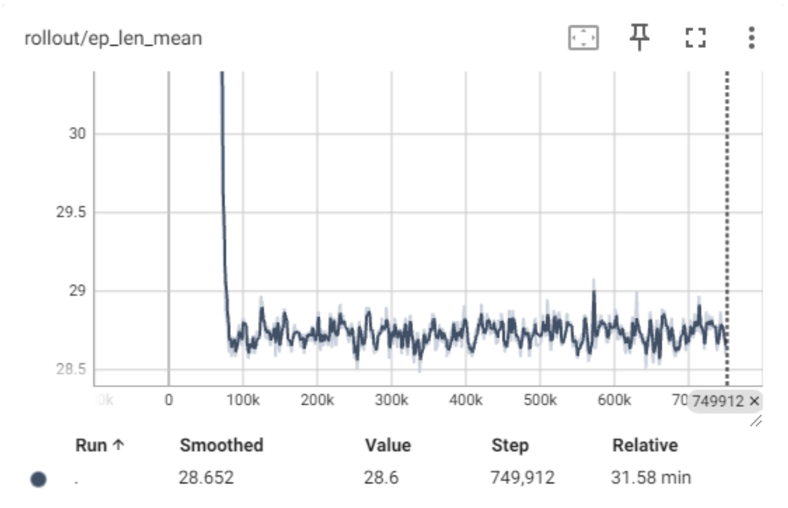
Figure 4B: gr17 reward mean per episode



Run ↑	Smoothed	Value	Step	Relative
● .	2,815.4597	2,909.5908	750,000	48.92 min

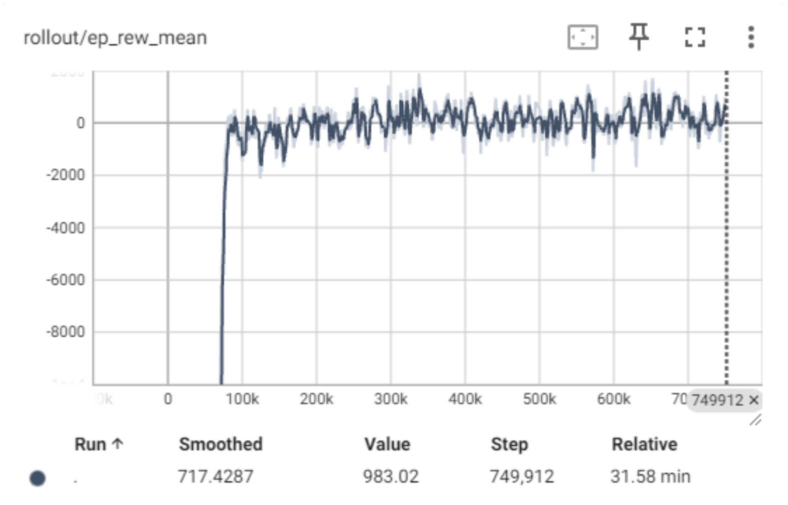
Figure 4C: gr17 loss function

bayg29



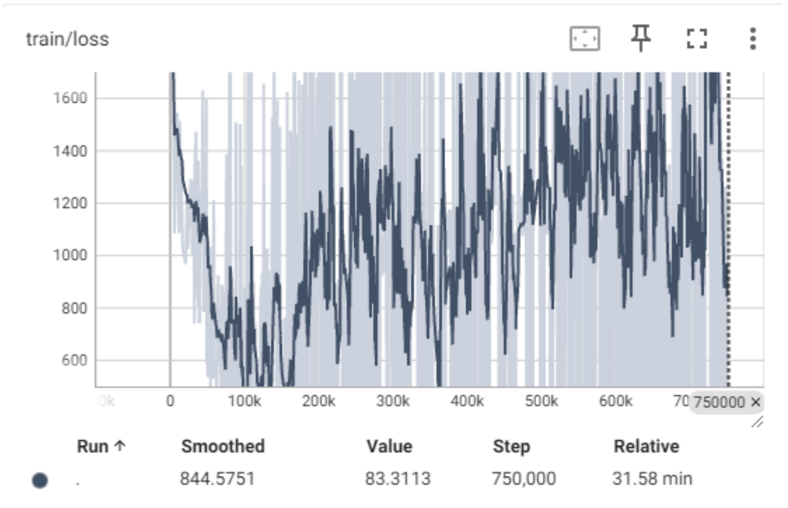
Run ↑	Smoothed	Value	Step	Relative
● .	28.652	28.6	749,912	31.58 min

Figure 5A: bayg29 tour length mean per episode



Run ↑	Smoothed	Value	Step	Relative
● .	717.4287	983.02	749,912	31.58 min

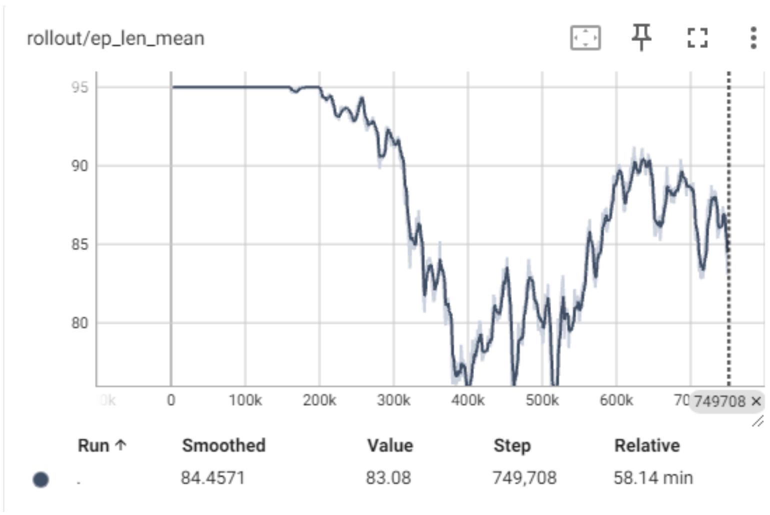
Figure 5B: bayg29 reward mean per episode



Run ↑	Smoothed	Value	Step	Relative
● .	844.5751	83.3113	750,000	31.58 min

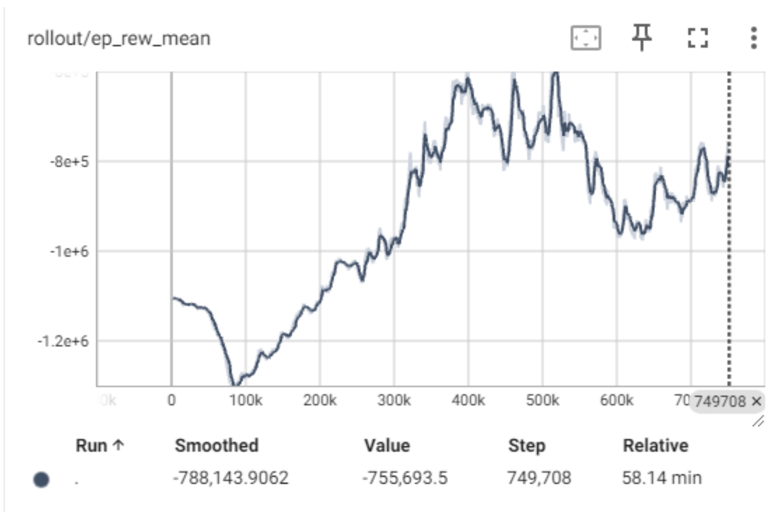
Figure 5C: bayg29 loss function

gr48



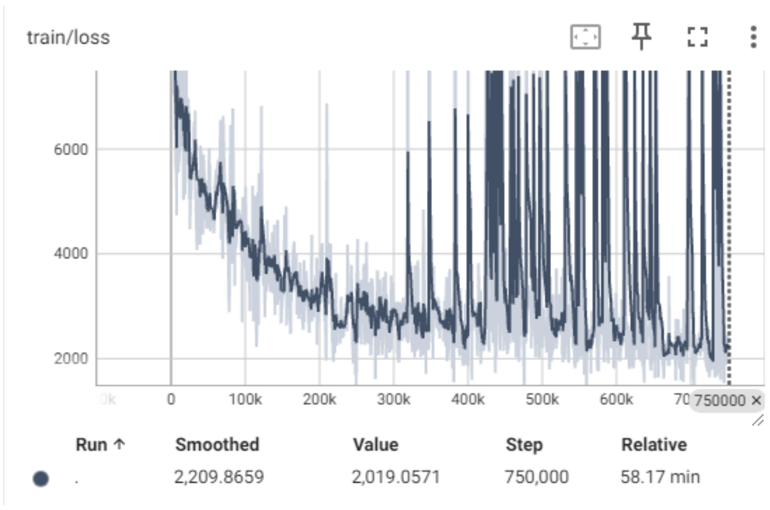
Run	Smoothed	Value	Step	Relative
● .	84.4571	83.08	749,708	58.14 min

Figure 6A: gr48 tour length mean per episode



Run	Smoothed	Value	Step	Relative
● .	-788,143.9062	-755,693.5	749,708	58.14 min

Figure 6B: gr48 reward mean per episode



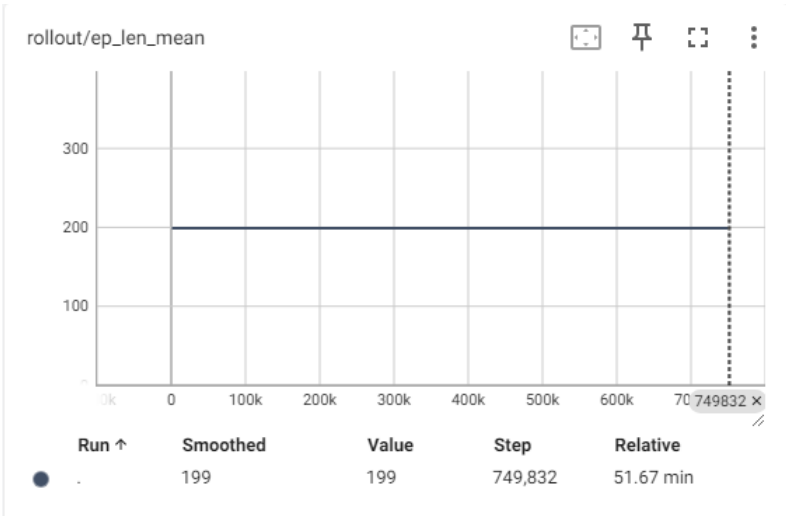
Run	Smoothed	Value	Step	Relative
● .	2,209.8659	2,019.0571	750,000	58.17 min

Figure 6C: gr48 loss function

Similar to the gr24 problem, all graphs of gr17 and bayg29 show similar performance. It is able to quickly identify the Hamiltonian cycle and reduce the length of tour to the problem's dimension. However, when the dimension of the problem doubled in gr48, the algorithm showed signs of divergence of a V-shaped graph instead of a converging shape. After reaching its optimal point, huge spikes were observed in the loss function and both the mean length and reward obtained diverged away from the optimal solution. It should be noted that both gr17 and gr48 loss function showed divergence as the steps increased, but this was expected as the model approaches optimal, gradient descent would mostly overshoot.

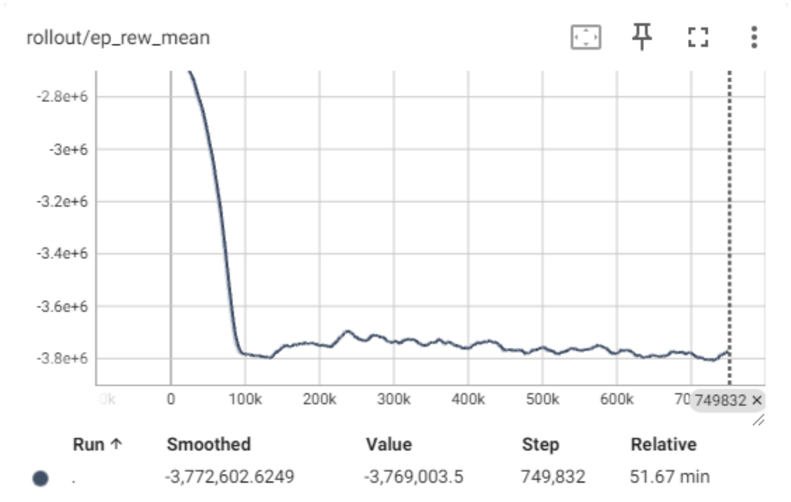
The divergence problem observed by solving larger instances problem is further justified when studying the kroA100 problem.

kroA100



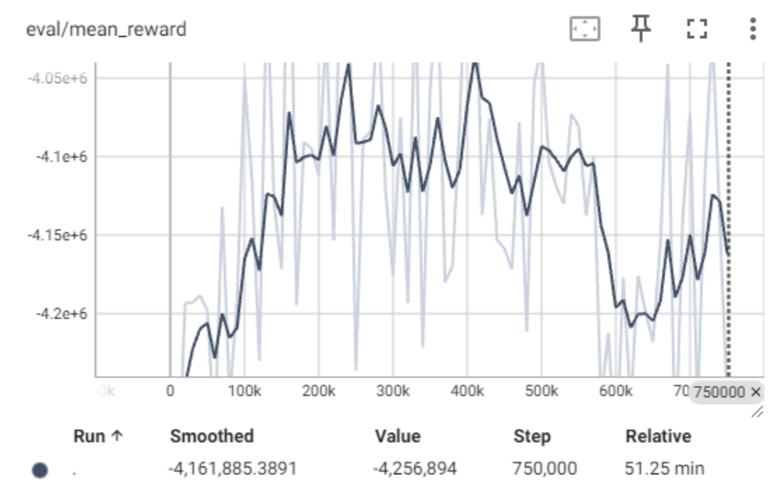
Run ↑	Smoothed	Value	Step	Relative
● .	199	199	749,832	51.67 min

Figure 7A: kroA100 tour length mean per episode



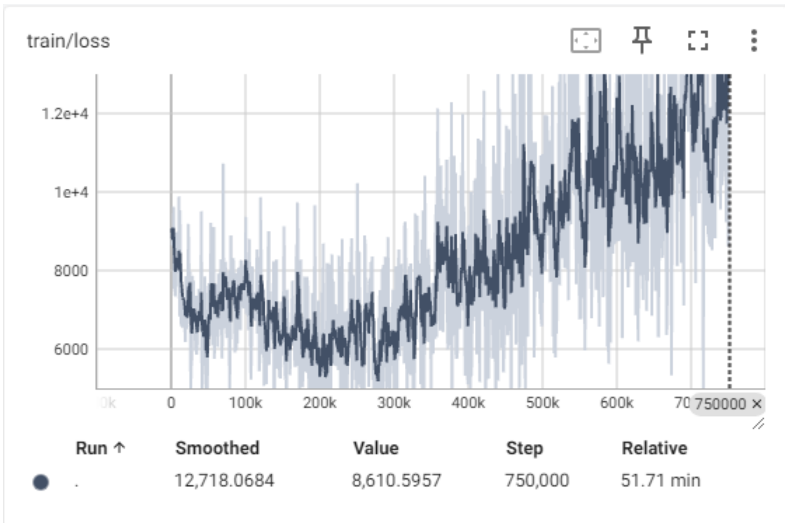
Run ↑	Smoothed	Value	Step	Relative
● .	-3,772,602.6249	-3,769,003.5	749,832	51.67 min

Figure 7B: kroA100 reward mean per episode



Run	Smoothed	Value	Step	Relative
● .	-4,161,885.3891	-4,256,894	750,000	51.25 min

Figure 7C: kroA100 reward mean per callback



Run	Smoothed	Value	Step	Relative
● .	12,718.0684	8,610.5957	750,000	51.71 min

Figure 7D: kroA100 loss function

When the problem dimension was increased to 100, the model was proved to fail. Although the similar shaped loss function showed a local optimal was reached, the mean length of the tour was stuck at the maximum limit throughout the entire 750,000 steps. The mean reward, unlike the other three problems, decreased as more steps were taken, opposing the reward maximizing objective of the model. The mean reward of callbacks was included in the graphs above which showed a better picture. The same V-shaped trend was observed, suggesting that the model diverged after reaching the local optimal. The model is proven to perform well when solving this project's environment, when the problem dimension is small and tends to have an increased chance of divergence as the size of the problem increases. The effectiveness of the obtained policy is then studied. As suggested in section 4.1, the reward policy of this project's environment is similar to a Closest Neighbour heuristic. The obtained tour cost is expected to lie within 25% of the Held-Karp lower bound to be considered effective. The optimal tour cost of each problem, given by TSPLIB(5.2) is used as the Held-Karp lower bound for this evaluation. All problems except kroA100 and bays29 have their best model obtained through callbacks and they were executed on their respective environment for 10,000 episodes to obtain the best results. The results and their respective problems' optimal solution is displayed below.

Problem	Cost(Rand)	Cost(Model)	Cost(Opt)	Optimal+25%	%Error($\frac{Model-Opt}{Opt}$)	Length
gr17	4708	2183	2085	2606	4.7%	17
gr24	3540	2486	1272	1590	95.4%	24
gr48	20870	18043	5046	6307	257.6%	48
bayg29	4546	2373	1610	2012	47.7%	29

Table 6.1: Held-Karp lower bound test with 25% Boundary (all)

According to the result obtained above, the model performed extremely well on problem gr17, while other results all exceeded the 25% threshold. The percentage error increases as the problem dimension, especially shown in the problem gr48. However, it could be argued that the best model obtained when learning gr48 was not suitable for this analysis as it was highly unstable. Nonetheless, a Hamiltonian cycle for each problem can be found by the model and improvement were made compared to a random tour.

6.3 Efficiency (Problem gr)

This section explores the efficiency of the learning algorithm on this project's custom environment, with respect to n, the problem's dimension. Notice this section of evaluation is based on a dynamic environment, where the exploration rate and episode per second is dynamically changed by stable baselines predefined algorithm. This section only evaluates on the simple picture of obtained result against time but not a fair and accurate comparison of the algorithms' actual complexity. The time and steps taken for DQN to learn, and allow its loss function to reach local minimum for each problem through 750,000 steps are listed below.

Problem	g17	gr24	gr48	kroA100	bayg29
Time(mins)	3.068	9.113	29.07	14.27	4.329
Steps	56000	161000	381000	208000	136000

Table 6.2: Time-Steps Table

The recorded result shows no significant correlation between the problem dimension and time required to compute the local minimum, or the number of steps required. It is a possibility that the computational time and steps required to find an optimal solution depends on the design pattern of the map, that is how the nodes are positioned in the environment. Certain patterns of how the nodes are located will greatly increase the computational time required to find an optimal solution.

However, if only consider the 'gr' problems, a possible polynomial relation between the problem size and the computational time, and between the problem size and number of steps required, can be observed, since all 'gr' problems can be considered to have similar data pattern. Although there is no way to justify this relation since the amount of data points is too low to consider a correlation.

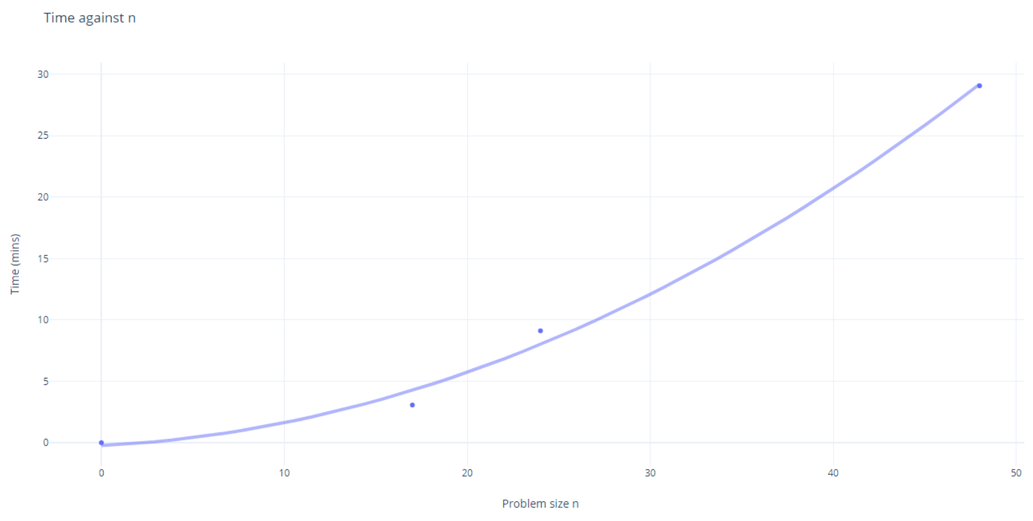


Figure 8A: Time(mins) against n

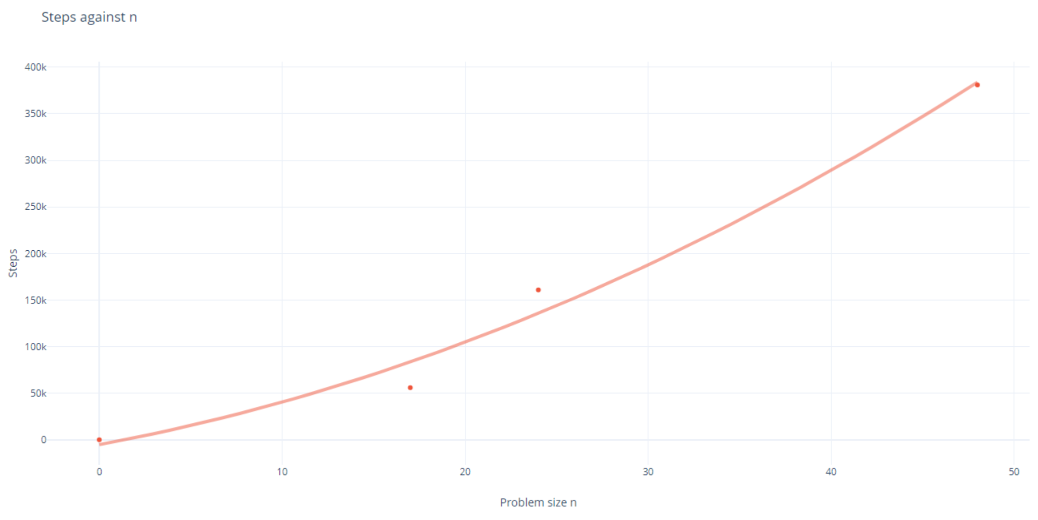


Figure 8B: Steps against n

6.4 Reapply model on unseen data (Problem bayg/bays)

The last section studies the possibility of reapplying pre-trained model of similar pattern to unseen data. This is considered the most important objective for machine learning solutions to justify for its computational heavy nature from heuristic algorithms. The problems used for testing are the bayg29 and bays29 problem, in which both consists of 29 nodes and a similar data pattern. The DQN model is first trained on the bayg29 environment and reapplied to bays29 environment. The results obtained are listed below.

Problem	Cost(Rand)	Cost(Model)	Cost(Opt)	Optimal+25%	%Error($\frac{Model-Opt}{Opt}$)	Length
bayg29	4546	2373	1610	2012	47.4%	29
bays29	4558	2351	2020	2525	16.4%	29

Table 6.3: Held-Karp lower bound test with 25% Boundary (bay)

The trained model using bayg29 performed even better on bays29, successfully obtained a Hamiltonian cycle and a result within the 25% threshold. This justifies the performance of transferring models between environment in solving TSP. Problems of large size can highly benefit from this property as solving multiple different computational heavy large sized problems, instead of deploying high time complexity algorithms multiple times, learning one model can solve all of them, providing they have similar data pattern and structure.

6.5 Compare with Existing Solutions

Machine learning solutions on solving TSP provides a unique alternative to data scientist from existing heuristic algorithms. The custom environment used in this project is only an example of simplicity. Solutions with professionally calibrated neural networks pairing with specialised environment design is where reinforcement learning will be able to produce highly accurate results. Strategies like designing sophisticated reward functions and hyper-parameters tuning may allow reinforcement learning solutions to become competitive with existing heuristic algorithms in terms of performance, in addition with the ability to apply onto other similar problems with close to none extra computational resources. However, similar to other algorithms, the model introduced in this project suffers the most from model unfriendly data patterns that will lengthen the computational time, or even failing the learning process. The other disadvantage of machine learning solutions compared to other algorithms is the high demand of memory space. This project has restricted the learning step of each model to a maximum of 750,000 step due to memory limitations, but an industrial levelled learning model usually requires millions or even billions of steps to exhaust all possibilities and locate the optimal policy.

Chapter 7

Conclusion

This chapter concludes the finding of this project and evaluations on project objectives. Possible future work is also discussed to improve this project's findings.

7.1 Project Summary

In this project a custom environment based on the static symmetric Traveling Salesman Problem is built successfully. Reinforcement learning algorithms are able to learn and produce high performance, competitive solutions to solve some TSP problems introduced in this project with the designed environment. This project also studied and evaluated on both the advantages and disadvantages of reinforcement learning on TSP, supported by experimental evidence. The most important achievement of this project is the prove of the ability of learnt model to be transferred to unseen environment and produce highly accurate solutions on TSP. Theoretical wise, this project explores the fundamentals of the famous TSP and its many existing solutions. Each solution is evaluated with their respective advantages, disadvantages and limitations. The recent development of machine learning technology and its points of interest are also studied.

7.2 Future Work

Tracing back to the original problem of this project. The impreciseness of postage delivery time is still unlikely to benefit from this project's solution. That is because the main obstacle of preventing postal service companies to produce a more precise estimation is the potential incident happening in real time traffic. A sudden accident in a high traffic area, or road maintenance carrying on the usual traveling path alternates the weight of the travel, of forces the postal van to visit a completely different location first. The sudden change in weights while the traveling salesman has already begun his journey is the dynamic Traveling Salesman Problem. Although it can be solved by reapplying a model which previously studied another problem of similar pattern to the new altered map, it is unrealistic to prepare all models that have already learned the close to infinity different kinds of data pattern. A new dynamic learning environment should be constructed instead. However, this further complicates the study and it is unable to be studied by this project due to time limitation.

Besides, further improvements can be applied to the static environment introduced in this project as many solutions to the problems studied were still proved ineffective. Stochastic reward functions can be designed instead of taking the direct weight as the reward can further improve the accuracy of the model. Hyperparameter tuning is also an advanced technique to improve learning and model convergence, which may be able to solve larger sized problems suggested in this project, without being stuck in local minimums. Moreover, hyperparameter tuning can be applied to A2C and PPO to better fit the TSP environment, which may provide unexpected performance on patterns of problems that DQN struggles to learn.

7.3 Final Thoughts

With the positive evaluations and results obtained, this project's can be concluded as success and its objectives are achieved. Before this project I was a mere beginner in the field of machine learning and I certainly learned a lot from my supervisor throughout this research. The technologies of machine learning are improving day by day and the scope of its applications are continuously expanding. I cherish the skills I learned throughout this project and I am satisfied with my performance.

Bibliography

[1] Applegate, David L., Robert E. Bixby, Vašek Chvatál, and William J. Cook. “The Problem.” In The Traveling Salesman Problem: A Computational Study, 1–58. Princeton University Press, 2006. http://www.jstor.org/stable/j.ctt7s8xg.4.

[2] Rajesh Matai, Surya Prakash Singh and Murari Lal Mittal. ”Traveling Salesman Problem: An Overview of Applications, Formulations, and Solution Approaches” In *Traveling Salesman Problem, Theory and Applications.* Edited by Davendra, Donald, 2010. https://directory.doabooks.org/handle/20.500.12854/64896

[3] D. S. Johnson, D. S. Johnson, E. E. Rothberg. ”Asymptotic Experimental Analysis for the Held-Karp Traveling Salesman Bound” 7th Annual ACM-SIAM Symposium on Discrete Algorithms, 341-350, 1996. http://dimacs.rutgers.edu/archive/Challenges/TSP/papers/HKsoda.pdf.

[4] Anna R. Karlin, Nathan Klein, Shayan Oveis Gharan. ”A (Slightly) Improved Approximation Algorithm for Metric TSP” arXiv:2007.01409, Oct 26, 2023. https://doi.org/10.48550/arXiv.2007.01409

[5] Petroc Taylor. “Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025” Published Nov 16, 2023. https://www.statista.com/statistics/871513/worldwide-data-created/.

[6] Turing, A.M. ”Computing machinery and intelligence”. Mind, 59, 433-460. 1950. https://www.cs.ox.ac.uk/activities/ieg/e-library/sources/t_article.pdf

[7] Alexander L. Fradkov, ”Early History of Machine Learning”, IFAC-PapersOnLine, 53-2, ISSN 2405-8963, 1385-1390, 2020, https://doi.org/10.1016/j.ifacol.2020.12.1888.

[8] Oxford Languages. Accessed May 10, 2024. https://languages.oup.com/google-dictionary-en/

[9] Windows, ”What is a machine learning model?”. In *Windows Machine Learning*, Apr 17, 2024. https://learn.microsoft.com/en-us/windows/ai/windows-ml/what-is-a-machine-learning-model

[10] Alpaydin, E. ”Introduction to machine learning” Fourth edition. The MIT Press. 2020. https://search-ebscohost-com.libproxy.ucl.ac.uk/login.aspx?direct=true&AuthType=ip,shib&db=nlebk&AN=2957329&site=ehost-live&scope=site

[11] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu. ”Asynchronous Methods for Deep Reinforcement Learning” arXiv:1602.01783, Jun 16, 2016. https://doi.org/10.48550/arXiv.1602.01783

[12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov. ”Proximal Policy Optimization Algorithms” arXiv:1707.06347. Aug 27, 2017. https://doi.org/10.48550/arXiv.1707.06347

[13] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, Pieter Abbeel. ”Trust Region Policy Optimization” arXiv:1502.05477. Apr 20, 2017. https://doi.org/10.48550/arXiv.1502.05477

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller. ”Playing Atari with Deep Reinforcement Learning” arXiv:1312.5602, Dec 19, 2013. https://doi.org/10.48550/arXiv.1312.5602

[15] Gerhard Reinelt. TSPLIB. Universität Heidelberg. Accessed May 10, 2024. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/index.html

[16] Gerhard Reinelt, ”TSBLIB 95”. Universität Heidelberg. Accessed May 10, 2024. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp95.pdf

[17] Gerhard Reinelt, ”Optimal solutions for symmetric TSPs”. Universität Heidelberg. Accessed May 10, 2024. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/STSP.html

[18] NumPy Developers. ”NumPy documentation” Accessed May 10, 2024. https://numpy.org/doc/stable/

[19] Robert Grant. ”Welcome to TSPLIB 95’s documentation!” Accessed May 10, 2024. https://tsplib95.readthedocs.io/en/stable/

[20] Farama Foundation. Gymnasium. Accessed May 10, 2024. https://gymnasium.farama.org/

[21] Antonin Raffin, Ashley Hill, Adam Gleave3, Anssi Kanervisto, Maximilian Ernestus, Noah Dormann. ”Stable-Baselines3 Docs - Reliable Reinforcement Learning Implementations”, Nov, 2021. https://stable-baselines3.readthedocs.io/en/master/

[22] The Linux Foundation. ”PyTorch documentation” Accessed May 10, 2024. https://pytorch.org/docs/stable/index.html

[23] TensorFlow. ”Get started with TensorBoard” Accessed May 10, 2024 https://www.tensorflow.org/tensorboard/get_started

Appendix A

Source Code and Related Document

All written source code can be found in the Github Repository using the link:

https://github.com/KinC16/TSP-with-Reinforcement-Learning.git

This repository is not anonymous.

The snippet of the class TspEnv is included below:

```
import math
import numpy as np
from gymnasium import Env
from gymnasium.spaces import Discrete, Box

class tspEnv(Env):
    def __init__(self, problem, penalty = 10**4):
        self.problem = problem
        self.action_space = Discrete(self.problem.dimension)
        self.observation_space = Box(0,np.inf,
                                     (self.problem.dimension,))
        self.tour = []
        self.max_length = 2*self.problem.dimension
        self.start = 0
        self.penalty = penalty
        self.w0_matrix = self._get_w_matrix()
        self.w_matrix = self.w0_matrix

    def step(self, action):
        # Get current state
        state = self._get_state()
        new_obs = self._get_obs(action)

        # Get reward for such a move
        reward = self._get_reward(state, action)

        # Append reached node to tour
        self.tour.append(int(action))
        self._update_matrix(action)

        done = False
        force_stop = False
        if self._is_subset(list(range(self.problem.dimension))
                            , self.tour):
            done = True
            reward += self.penalty
            reward -= np.array(self.w0_matrix[action])[self.start]
            if len(self.tour) == self.problem.dimension:
                reward += self.penalty

        if len(self.tour) == self.max_length:
            force_stop = True

        info = {"tour": self.tour}

        return new_obs, reward, done, force_stop, info

    def next_rand_action(self):
        while True:
            a = self.action_space.sample()
            if (((a not in self.tour) and (a != self.start))
            and (a != self._get_state())):
                break
        return a

    def render(self):
        pass

    def reset(self, seed = None, option = None):
        super().reset(seed=seed)
        self.tour = []
        self.tour.append(self.start)
        self.w_matrix = self.w0_matrix
        self._update_matrix(self.start)
        info = {}
        return self._get_obs(self.start), info

    def _get_state(self):
        return self.tour[-1]

    def _get_reward(self, state, action):
        return -self.w_matrix[state][action]

    def _get_obs(self, action):
        return np.array(self.w_matrix[action])

    def _get_w_matrix(self):

        # convert lower triangle matrix to square matrix
        if self.problem.edge_weight_type == "EXPLICIT":
            if self.problem.edge_weight_format == "LOWER_DIAG_ROW":
                data = []
                weight = []
                for i in self.problem.edge_weights:
                    for j in i:
                        data.append(j)
                for x in range(self.problem.dimension):
                # format lower triangle matrix
                    node = []
                    w = data.pop(0)
                    while w != 0:
                        node.append(w)
                        w = data.pop(0)
                    while len(node) != self.problem.dimension:
                        node.append(0)
                    weight.append(node)
                matrix = np.triu(np.array(weight).T,1) + weight
                #convert to square matrix
            # convert upp triangle matrix to square matrix
            elif self.problem.edge_weight_format == "UPPER_ROW":
                matrix = np.zeros((self.problem.dimension,
                                   self.problem.dimension))
                weight = self.problem.edge_weights
                for x in weight:
                    while len(x) != self.problem.dimension:
                        x.append(0)
                weight = np.vstack((weight,
                         np.zeros(self.problem.dimension)))
                matrix = matrix + weight
                matrix = np.flipud(matrix)
                matrix = np.triu(matrix.T,1) + matrix

        # extract weight between nodes
        # in euclidean 2d coordinate system
        elif self.problem.edge_weight_type == "EUC_2D":
            matrix = np.zeros((self.problem.dimension,
                               self.problem.dimension))
            for i in range(self.problem.dimension):
                for j in range(self.problem.dimension):
                    x1, y1 = self.problem.node_coords[i+1]
                    x2, y2 = self.problem.node_coords[j+1]
                    w = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
                    matrix[i,j] = round(w)

        else: raise Exception("Invalid edge_weight_type")

        matrix[matrix == 0] = self.penalty
        return matrix

    def _update_matrix(self, action):
        self.w_matrix[:, action] = self.penalty

    def _is_subset(self, sub_list, list):
        if set(sub_list).intersection(set(list)) == set(sub_list):
            return True
        else:
```

```
        return False
```

Appendix B

Project Plan

Name: Chun Kin Cheung

Project Title: Estimation of Postal Service Delivery Time via Machine Learning

Supervisor's Name: Carlo Ciliberto

B.1 Aims

To improve the estimation of postal service arriving time with higher accuracy by modelling and implementing the service as Machine Learning solutions, starting with route optimization.

B.2 Objectives

- Collect large traffic data set, including road congestion rate at different hours, traffic light locations and destinations for the learning model.
- Review algorithms and techniques to route optimization problem. Analyse each methods pros and cons and determine the most suitable method for the scenario.
- Develop the learning model and software tools to process collected data.
- Calculate errors and graphical data to justify the effectiveness of the model.
- Expand the model with more parameters outside of traffic concerns to improve its suitability for generating a more realistic solution.
- Evaluate the effectiveness of the solution compared to current postal service by leading companies in the sector.

B.3 Deliverables

- Design specification for the learning model
- Documentation for the learning model with functional algorithms and error analysis.
- Data set used for the model to learn and test.
- Review and justification of the result obtained

B.4 Work Plan

25/09/2023 – 12/11/2023 – Literature Review. Research the current Postal Service market and identify its challenges.

06/09/2023 – 03/12/2023 – Data Collection. Obtain required data suitable for constructing a learning model by going through open-source databases and websites.

04/12/2023 – 31/12/2023 – Analysis and Modelling. Analyse collected data and design the learning model.

01/01/2024 – 11/02/2024 – System Design and Implementation. Coding the learning model and carry out data testing.

12/02/2024 – 03/03/2024 – Error Analysis and System Refinement. Carry out error analysis and adjust the model accordingly to enhance accuracy.

04/03/2024 – 14/04/2024 – Result Evaluation / Final Report. Construct the Final Report based on the result generated by the learning model.

14/04/2024 – 26/04/2024 – Project Review. Review and finalize the project with administrations. Submit project before 26/04/2024.