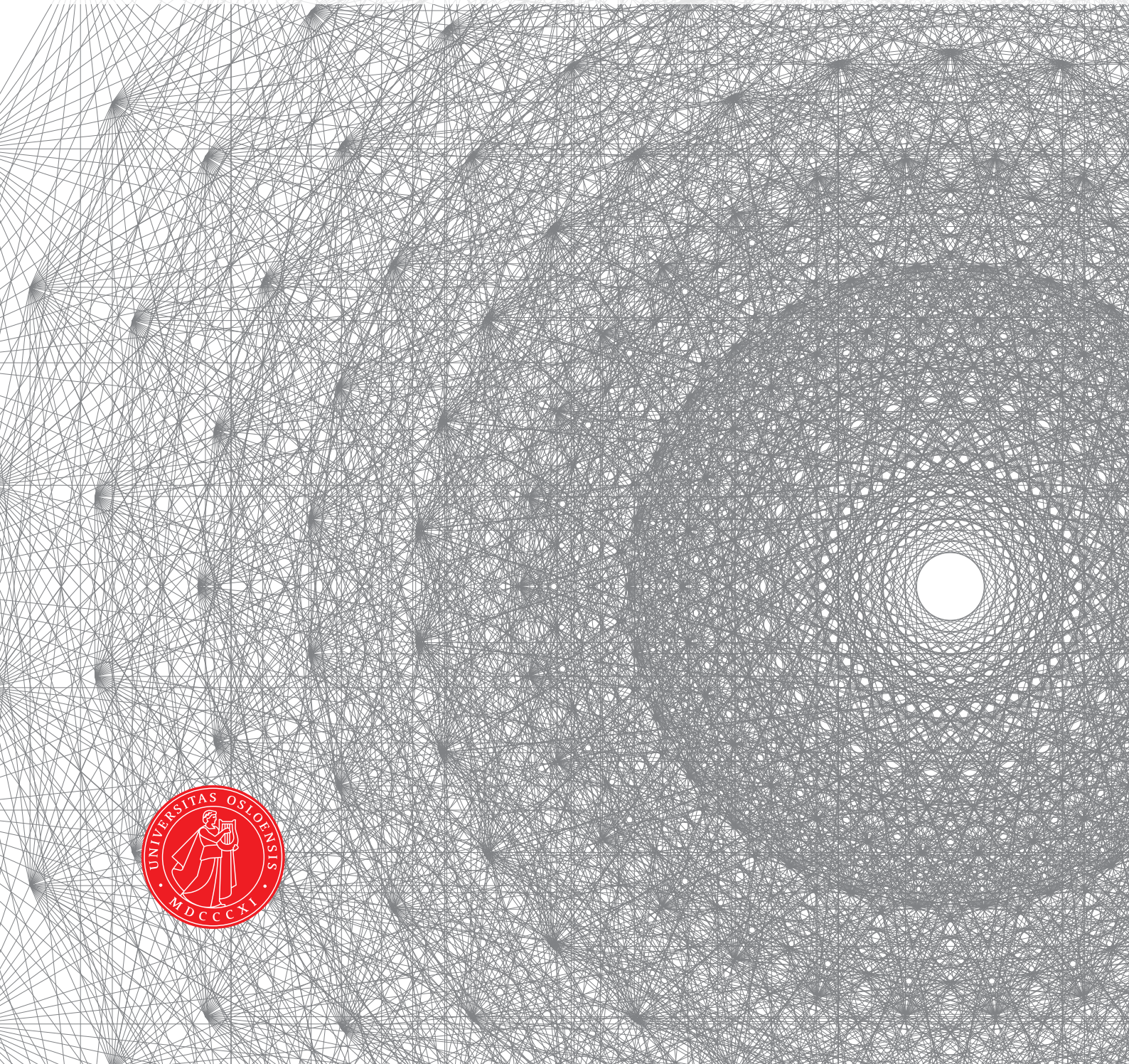
UiO : Department of Mathematics
University of Oslo

Optimizing Coordination on Road Construction Sites with a Reinforcement Learning Framework

Øystein Høistad Bruce

Master's Thesis, Spring 2023



This master's thesis is submitted under the master's programme *Computational Science*, with programme option *Applied Mathematics and Risk Analysis*, at the Department of Mathematics, University of Oslo. The scope of the thesis is 60 credits.

The front page depicts a section of the root system of the exceptional Lie group E_8, projected into the plane. Lie groups were invented by the Norwegian mathematician Sophus Lie (1842–1899) to express symmetries in differential equations and today they play a central role in various parts of mathematics.

Abstract

Road construction sites are often inefficient, with construction machines frequently idling for extended periods, wasting fuel and time. One way to increase efficiency is to optimize the scheduling of the dumpers transporting materials across the site. This thesis proposes and investigates a multi-agent reinforcement learning framework designed to coordinate dumpers and excavators on construction sites. The framework generates time schedules for all vehicles, considering multiple criteria such as fuel consumption, completion time, and cost. Users can choose a plan that best aligns with their preferences, ensuring maximum efficiency. The framework has a negligible training time and generally outperforms a baseline constructed from human behavior. In addition, we developed a predictive fuel consumption model for a dumper using high-resolution data logged over 12 working days. By associating each dumper with such a model, we can more accurately predict their fuel consumption while driving, further improving the planning.

Figure 1: Data driven road construction sites [SIN20]

Acknowledgements

First, I want to express my sincere gratitude to my supervisor at SINTEF, Signe Riemer-Sørensen, for allowing me to work on such an interesting task. Her guidance and expertise throughout this thesis have been invaluable, and I feel fortunate to have had the opportunity to learn from her. In addition, thanks to SINTEF for providing me with resources and support during my thesis.

I would also like to thank my supervisors at UiO, Øyvind Ryan and Vegard Antun, for their guidance and support in helping me to develop the skills and knowledge necessary to write a good master's thesis.

Second, I would like to express my appreciation to *CLOPEN*[1] [UiO], my study group throughout my bachelor's and master's degrees. Working with my fellow *CLOPEN* team members, Sigurd Holmsen and David Händler Andersen, has been an incredible experience. Our collective efforts have resulted in better results and increased knowledge, and I am grateful for the sense of community and collaboration we have fostered.

Finally, I would like to thank all my friends, family, and girlfriend who have supported me throughout my academic journey. Your encouragement, love, and support have been a constant source of motivation, and I am grateful for everything you have done for me.

Øystein Høistad Bruce,

Oslo 2023

[1] https://www.mn.uio.no/math/studier/aktuelt/aktuelle-saker/2020/maec-studenter-vant-nb-casenm.html

Contents

Acknowledgements ii

Contents iii

1 Introduction 1
1.1 Outline . . . 3

I Reinforcement Learning Framework 4

2 Theory 5
2.1 Reinforcement Learning . . . 5
2.2 Neural Networks . . . 10
2.3 Graphs . . . 13
2.4 Bruce's algorithm . . . 15

3 Method 17
3.1 Route Optimization on Construction Site . . . 17
3.2 An Overview of the Problem's Environment . . . 18
3.3 The Roles of the Agents: An In-Depth View . . . 19
A Closer Look at the Planning Agents . . . 20
The Coffee Break Agent: A Solution for Temporary Breaks . . . 24
Node Agents: Guiding the Way through Graphs . . . 25
3.4 A Detailed Breakdown of an Episode . . . 28
3.5 Exploration vs. Exploitation . . . 30
3.6 A Guide to the Framework's Configuration Options . . . 31

4 Results 33
4.1 Evaluate Performance: Two Baselines . . . 33
4.2 Testing the Framework's Capability on Basic Scenarios . . . 35
4.3 Framework Evaluation: Analysis on Multiple Maps . . . 40

5 Discussion 45

II Fuel Model **48**

6 Towards more advanced route planning **49**

7 Theory **50**
7.1 A Brief Overview of How to Evaluate Machine Learning Models 50
7.2 Understanding Model Interpretability: Exploring SHAP and SAGE Values 50

8 Data **53**
8.1 Exploring the GPS and Fuel Data 53
8.2 Data Selection and Cleaning 54
8.3 Merging Data 56
8.4 Statistics of a Route 56
8.5 The Final Data Set 58
8.6 Investigation of New Features 58

9 Method **59**
9.1 Predictive Model for Fuel Consumption on a Route 59
9.2 Fine-tuning Models: Hyperparameter Tuning 59
9.3 Feature Investigation for Improved Model Performance 59

10 Results and Discussion **65**
10.1 Measuring Model Performance 65
10.2 Analyzing Model Predictions 66
10.3 Final Considerations: Key Takeaways and Insights 67

III Summary and Conclusions **68**

IV Appendices **72**

A Appendix for Part I **73**
A.1 Universal Approximation Theorem 73
A.2 The Default Configuration of the Framework 76
A.3 Determining the Starting Order of Dumpers 78
A.4 Map 1: Demonstrating Baseline 1 79
A.5 Map 1: Dumper and Loader Scheduling and Performance . . . 80

B Appendix for Part II **84**
B.1 A Brief Introduction to Decision Trees 84
B.2 Tables of Selected Data Points 85
B.3 Data Summary: A Table of Statistics 88

Bibliography **92**

CHAPTER 1

Introduction

Road construction is essential for the development of infrastructure and transportation systems. However, it also significantly impacts the environment, as it is a major source of greenhouse gas emissions [HN15]. Moreover, road construction sites are often inefficient, as construction machines frequently idle for extended periods, wasting fuel and time. Therefore, there is a need to optimize the planning and execution of road projects to reduce emissions, costs, and duration.

Skanska, in collaboration with SINTEF, Volvo, and Ditio [Lei+23] [SIN23] [Ska19], conducted a research project from 2020 to 2022 to improve the coordination of machinery on road construction sites using data and artificial intelligence techniques.

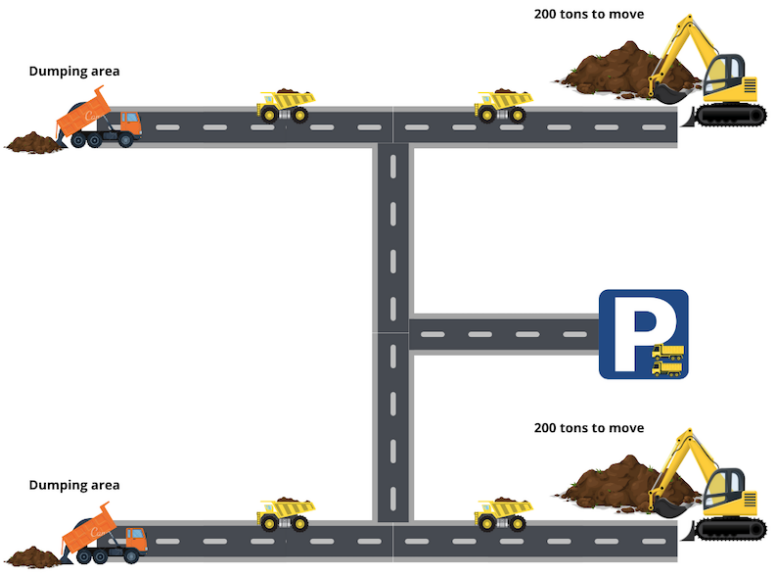


Figure 1.1: Representation of the problem on a road construction site.

Figure 1.1 depicts a road construction site comprising loading and dumping locations, roads, intersections, and a parking area. The objective is to find the most efficient plan to transport all the required mass while minimizing fuel consumption, time, and cost with the available dumpers.

This problem belongs to the class of capacitated vehicle routing problems, where vehicles have a limited carrying capacity and must move materials between different locations.

To address this problem, SINTEF used a stochastic time-front construction heuristic to generate an initial plan for moving materials. This plan was further optimized using a local search algorithm.

This thesis explores an alternative approach and investigates the following research questions:

- Can reinforcement learning be effectively applied to the scheduling problem on construction sites?
- How does such a framework handle dynamic changes in the construction site environment?
- Can we use the continuously logged GPS data to predict the fuel consumption of dumpers and further optimize the scheduling?

This thesis proposes a framework to address the scheduling problem using multi-agent reinforcement learning. The framework enables agents to be responsible for route planning, navigation, and decision-making concerning directing dumpers to a parking spot. In addition, an agent is responsible for scheduling temporary breaks for dumpers in cases where it can improve overall efficiency. These agents learn through trial and error, using a reward signal based on the coordination's efficiency, including distances and waiting times. Our framework aims to improve productivity and efficiency at road construction sites through intelligent dumper scheduling.

The framework was developed entirely by the author of this thesis and required significant coding efforts. The GitHub repository[1] [Bru23] contains the full code for this thesis.

In addition to the reinforcement learning framework, this thesis also develops a fuel consumption model that can be integrated with the framework to improve the accuracy of fuel consumption estimates during route planning. The fuel consumption model predicts the amount of fuel consumed by the machinery when traversing the roads on the construction site by considering factors such as altitude, acceleration time, and the amount of mass loaded. This integration is necessary for more advanced evaluations, as the current method of converting distance directly to fuel consumption based on averages can lead to inaccurate estimates. Due to time constraints, we did not implement this integration. However, future work could incorporate fuel consumption as the cost of traversing a road rather than physical distance.

By reducing waiting times and fuel consumption, our framework has the potential to enhance both the sustainability and cost-effectiveness of road construction sites.

[1] https://github.com/oystehbr/MasterThesis_Bruce

1.1 Outline

Part I Reinforcement Learning Framework

Chapter 2 presents an overview of the fundamental concepts in reinforcement learning. It includes a brief introduction to neural networks and an explanation of the concepts in graph theory relevant to our application. Finally, the author presents his algorithm to solve the all-pairs shortest path problem in weighted graphs.

Chapter 3 details the development and setup of the proposed reinforcement learning framework, explaining its capabilities and limitations.

Chapter 4 presents the performance of the proposed framework by evaluating it in several scenarios. The framework is first tested on basic scenarios to demonstrate its capabilities. Then, we conduct a larger analysis with more complicated maps.

Chapter 5 discusses the reinforcement learning framework with suggestions for potential improvements. In addition, we explore areas that we did not test but could be interesting to investigate in further research.

Part II Fuel Model

Chapter 6 outlines the necessity of the fuel model and its potential to enhance the evaluation process of the reinforcement learning framework.

Chapter 7 provides the theoretical background needed to understand the fuel model part of this thesis.

Chapter 8 provides a detailed overview of the data used in the fuel model analysis. It explains the data preprocessing and guides the structuring of the final data set before training the model.

Chapter 9 presents how we select the optimal model based on our data, including feature selection techniques to identify the most relevant features.

Chapter 10 interprets and discusses the final fuel model's results.

Part III Summary and Conclusions

Part IV Appendices

Appendix A covers the appendix for Part I.

Appendix B covers the appendix for Part II.

PART I

Reinforcement Learning Framework

CHAPTER 2

Theory

2.1 Reinforcement Learning

Learning by interacting with the environment around us is the most basic and standard way of acquiring knowledge for humans. *Reinforcement learning* is a machine learning technique that mimics this learning process by trial and error. The goal is for an agent to learn to make more informed responses to different situations.

Researchers have successfully applied reinforcement learning to a wide range of problems in domains such as robotics [Lev+16], game intelligence [Sil+18], and language models [Ouy+22]. Several educational resources are available in the field; for instance, *Spinning Up* [AA], produced by OpenAI, provides a comprehensive introduction to reinforcement learning and practical advice on how to implement and apply it. Additionally, the book [SB18] offers a more detailed and technical treatment of the subject.

There are many different models in reinforcement learning, but they all follow the same basic concept of an agent performing actions in an environment and being rewarded based on the desirability of the outcome.

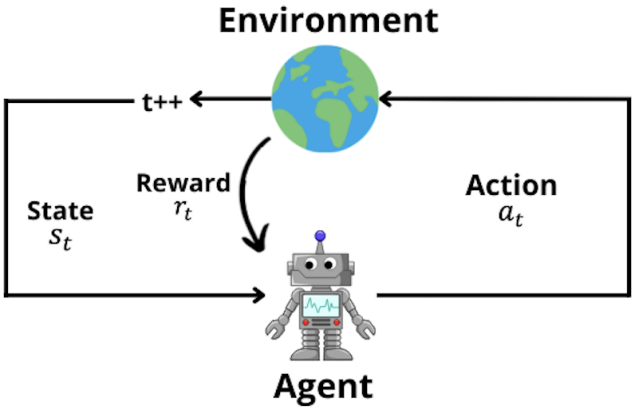


Figure 2.1: The Reinforcement Learning Cycle: The agent observes the environment's current state, s_t. It then performs an action a_t, which leads to a new state s_{t+1} and a reward r_t.

Agent, environment, state, action space

An *agent* can be considered a player that learns by interacting with its environment. The *environment* is the world in which the agent lives and operates. An *action* is a move that an agent makes, which results in changes in the environment.

To select an action, the agent must have some knowledge of the current state of its environment. A *state*, s, is a complete description of the environment.

The set of all valid actions an agent can perform is called the *action space*. The environment and the particular task the agent aims to learn influences the action space. Depending on the nature of the environment and the task, the action space can be either discrete or continuous.

The agent, environment, state, and action relationship are visualized in Figure 2.1.

Policy

A *policy* refers to the decision-making process of an agent or its brain. It determines the agent's actions based on information about the environment. A policy can be deterministic (as shown in Equation 2.1) or stochastic (as shown in Equation 2.2).

$$a_t = \mu_\theta(s_t) \tag{2.1}$$

$$a_t \sim \pi_\theta(\cdot|s_t) \tag{2.2}$$

where μ, π refers to a function and probability distribution (respectively), and s_t is the observed state of the environment at time t. The θ represents that the policy function could depend on a set of adjustable parameters resulting in different policies. For example, in the case of a neural network-based policy, the parameters could include weights, biases, and activation functions.

The policy could be as simple as a look-up table, which could be applied where the action space is discrete and relatively small (e.g., a game of Tic Tac Toe). A more advanced policy may be necessary for more complex scenarios where the action space is larger and continuous, such as using a neural network.

Trajectory, episode

By repeating the cycle of reinforcement learning (as shown in Figure 2.1) until we reach either a failure or a predefined stopping criterion, we obtain a *trajectory*, also called an *episode*.

A trajectory, τ, is a sequence of states and actions in the environment:

$$\tau = (s_0, a_0, s_1, a_1, ...) \tag{2.3}$$

where the initial state s_0 can be fixed or randomly sampled from the *starting-state distribution*.

The laws of the environment determine the state transitions, which rely on the most current state and action. State transitions may be deterministic (as shown in Equation 2.4) or stochastic (as shown in Equation 2.5).

$$s_{t+1} = f(s_t, a_t) \tag{2.4}$$
$$s_{t+1} = P(\cdot|s_t, a_t) \tag{2.5}$$

where the actions $\{a_t\}_{t>0}$ are chosen according to the agent's policy.

Reward and Return

The *reward function*, R, measures if the action taken by the agent is good or bad, and it is critical to be able to train the policy. The reward function often depends on some or all of the following: current state (s_t), current action (a_t), and next state (s_{t+1}). The reward, r_t, at time t in an episode is of the form:

$$r_t = R(s_t, a_t, s_{t+1}) \tag{2.6}$$

The agent aims to maximize the cumulative reward, called *return* $R(\tau)$, over an episode. The return is often a linear combination of all rewards obtained in the given episode. However, this linear combination could be *discounted* and controls the importance of future rewards.

Goal

The main objective of reinforcement learning is determining the policy that generates the maximum *expected return*, denoted by $J(\pi)$. The *optimal policy*, denoted by π^*, is expressed mathematically as:

$$\pi^* = \arg\max_{\pi} J(\pi) \tag{2.7}$$

where

$$J(\pi) = E_{\tau\sim\pi}[R(\tau)] = \int_{\tau} P(\tau|\pi)R(\tau) \tag{2.8}$$
$$P(\tau|\pi) = P(s_0)\prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \tag{2.9}$$

Equation 2.8 and 2.9 define the following terms: τ represents an episode, $P(\tau|\pi)$ denotes the probability of reaching the episode τ given the policy π, and $R(\tau)$ expresses the return of the episode τ.

Exploration vs. Exploitation

While training an agent, a key consideration is a trade-off between *exploration* and *exploitation*. Exploration refers to the agent trying out new actions in unfamiliar states to gain new knowledge about the environment. By doing so, the agent can discover potentially better policies and avoid getting stuck in suboptimal ones. Exploitation involves selecting reasonable actions based on the agent's experience, usually by following the trained policy without knowing if any variation would lead to a greater return.

Vanilla - and Deep Q-Learning

Q-learning [WD92] is a reinforcement learning technique that enables an agent to learn how to act optimally in a given environment by estimating the maximum expected return achievable for each action. The main objective is to train the agent's policy, $Q_\theta(s, a)$, such that it approximates the optimal action-value function, $Q^*(s, a)$, which we define as:

$$Q^*(s, a) = \max_\pi E_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a], \tag{2.10}$$

where $R(\tau)$ is the return of the trajectory τ, (s, a) is a state-action pair, and π is the policy. $Q^*(s, a)$ represents the maximum expected return achievable by acting according to the *optimal* policy, given an initial state s and arbitrary action a.

The agent's policy $Q_\theta(s, a)$ is a function that maps each state-action pair to a *Q-value*, which estimates the maximum expected return of taking an action in a given state. The agent tries to maximize the expected return for any state. Hence, it always selects the action with the highest Q-value, as shown in Equation 2.11.

$$a(s) = \arg\max_a Q_\theta(s, a), \tag{2.11}$$

where $a(s)$ is the action yielding the highest Q-value in a given state s

To improve the approximator $Q_\theta(s, a)$, Q-learning uses a form of the Bellman equation (presented in Equation 2.12), which was first introduced in [Bel52]. This equation allows the agent to update its estimates of the Q-values based on the rewards it receives while taking actions in the environment.

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_a Q(s_{t+1}, a)\right], \tag{2.12}$$

where α is the learning rate, γ is the discount factor, and $Q(s_t, a_t)$ and $Q(s_{t+1}, a)$ are the Q-values for the state-action pairs (s_t, a_t), (s_{t+1}, a), respectively. We denote the reward for taking action a_t at time t as r_t.

Equation 2.12 updates the Q-value of a state-action pair (s_t, a_t) by combining the current Q-value and the *newly proposed Q-value*. The newly proposed Q-value, $[r_t + \gamma \max_a Q(s_{t+1}, a)]$, estimates a new maximum expected return based on the current reward, r_t, and the discounted maximum of the Q-value of the next state s_{t+1}. The discount factor $\gamma \in [0, 1]$ ensures that rewards received soon are given more weight than those received in the distant future. The learning rate $\alpha \in [0, 1]$ determines the trade-off between the old and the newly proposed Q-value. A higher α means more weight is given to the newer estimate. The Bellman equation is recursive because it uses the same formula to calculate the maximum Q-value of the next state, which depends on another state-action pair (s_{t+1}, a_{t+1}).

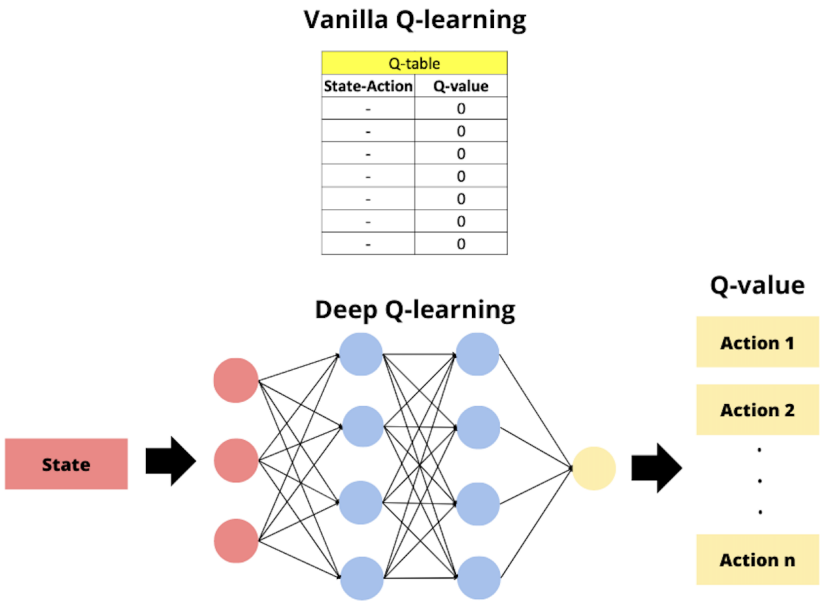


Figure 2.2: The approximator $Q_\theta(s, a)$ in Vanilla - and Deep Q-learning. While Vanilla Q-learning utilizes a look-up table as its approximator, Deep Q-learning replaces it with a neural network.

Vanilla Q-learning uses a table-based approach for the policy, known as the Q-table (Figure 2.2). The Q-table contains a Q-value for every state-action pair, which it updates using Equation 2.12. This method is limited to environments with a low and finite number of discrete states and actions. As a result, it may struggle to generalize to unseen states, particularly in environments with continuous state spaces.

Deep Q-learning overcomes this limitation by using a neural network as the function approximator $Q_\theta(s, a)$. We feed the state into the neural network and obtain the Q-values for all possible actions. Using a neural network allows the policy to handle large and continuous state spaces, as the network can generalize to similar states even if they are not identical.

In Deep Q-learning, we train our policy by first inputting the state s_t into the neural network to obtain the Q-values. We then update the Q-value for the action, a_t, using Equation 2.12, while keeping the other Q-values unchanged. Finally, we use the updated Q-values and the corresponding state s_t as a training sample for the neural network.

We will use the Deep Q-learning approach in our reinforcement learning framework.

2.2 Neural Networks

Our agents in the reinforcement learning framework use neural networks as their policy. Neural networks are models composed of interconnected layers of nodes that learn to recognize patterns in data and make predictions based on those patterns. Two books that provide a comprehensive introduction to neural networks are *Deep Learning* [GBC16] and *Neural Networks and Deep Learning* [Nie15].

The fundamental goal of a neural network is to approximate a mathematical function that maps a set of input variables x to a set of output variables y. In other words, the goal is to train a neural network, denoted as f, to approximate the underlying function such that $f(x_i) \approx y_i$ for all observations i.

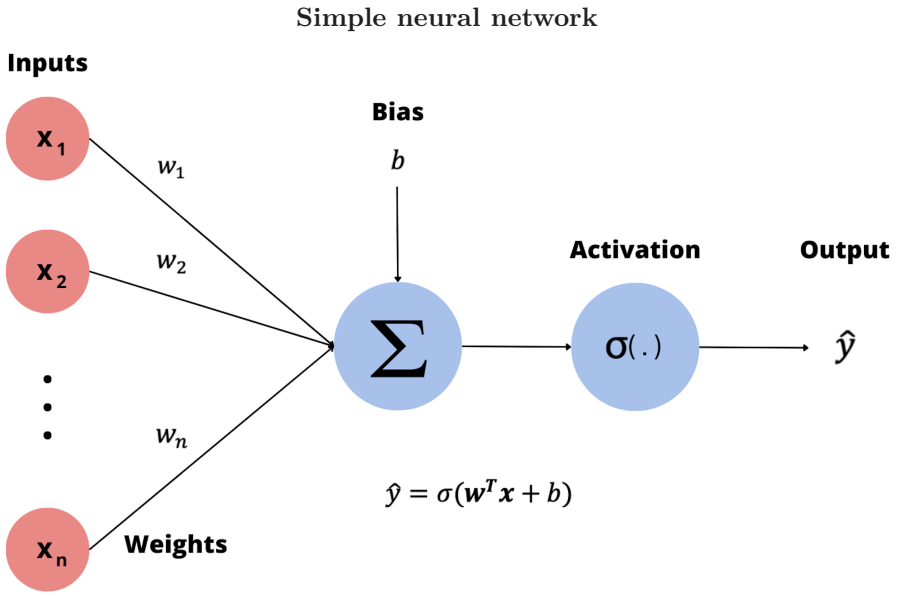


Figure 2.3: Illustrates two types of neural network computations: **(a)** at node-level **(b)** a simple neural network

We apply the following terminology:

(1) Neuron, layer: these are the building blocks of a neural network. The neuron (node) consists of weights and a bias term. It performs a dot product between the weights and the output of the previous nodes, then adds a bias term, and finally, applies an activation function to get the neuron's output. A layer is a stack of nodes.

(2) Input -, hidden- and output layer: are terms used to talk about the structure of a neural network. The input layer is the first layer and represents a sample of input data. The hidden layer(s) is the layer(s) between the input layer and the output layer and decides the depth of the network. The output layer is the last and outputs the predicted value(s).

(3) Weights and bias: are used in calculating the output of a node. The weights indicate the importance of the connection between any two nodes. The bias term is a constant value used to speed up or delay the activation of a given node.

(4) Activation function: is a function used in a node that can introduce non-linearity into the neural network. Often used functions are, e.g., tanh, (Leaky) ReLU, and sigmoid, in addition to linear mappings for simple information passing.

Neural networks are scalable; Figure 2.3 can illustrate **(a)** a single neuron or **(b)** a simple neural network.

(a) Refers to a node that takes the previous layer as its input. The input features $(x_1, x_2, ..., x_n)$ are multiplied with their corresponding weights $(w_1, w_2, ..., w_n)$ and added the bias term. The resulting value is then passed through an activation function, σ, to get the node's output. If the network contains additional layers, the output of the current node is the input for the next layer. If the current node is the output node, its value represents the neural network's prediction.

(b) Refers to a neural network with an input size of n, no hidden layers, and a single output prediction $\hat{y}$.

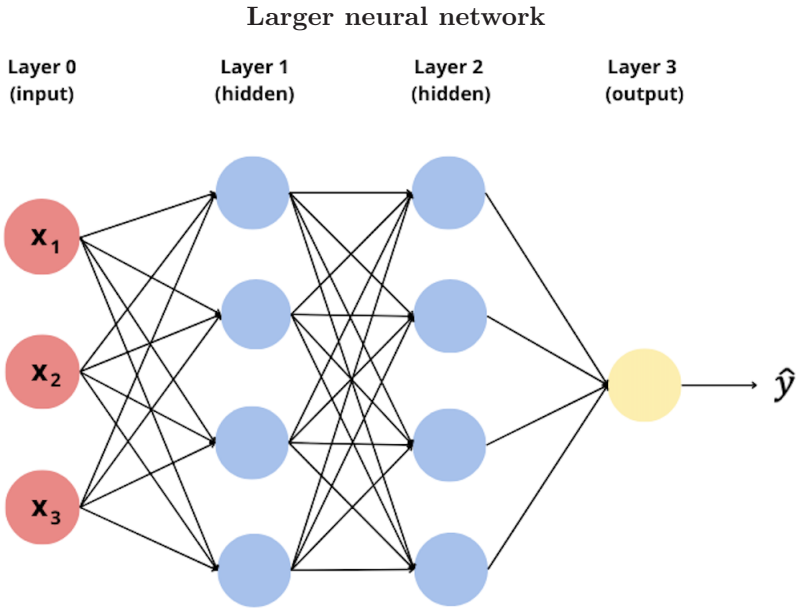


Figure 2.4: A neural network structure. Three input features, two hidden layers with four nodes each, and a single output node.

Figure 2.4 depicts a more extensive neural network, where all nodes in the hidden and output layers are constructed similarly to Figure 2.3 (with the interpretation of **(a)**), but with different weights and biases. This structure is commonly known as a *fully-connected feed-forward neural network*, where each node in a layer connects to every node in the previous layer.

Evaluation and training

A neural network starts with random values for its weights and biases, often centered around 0. The neural network learns by iteratively tuning these parameters through a process of feed-forwarding data, making predictions, evaluating the results using a *cost function*, and then updating the parameters with *back-propagation* [RHW86]. This common type of machine learning is called *supervised learning*, where a model trains on labeled data and aims to make accurate predictions for new, unseen data.

The cost function evaluates the predictions $\{\hat{y_i}\}_{i=1}^{m}$ against the known targets $\{y_i\}_{i=1}^{m}$, which we aim to minimize to achieve better predictions. A cost function could, for instance, look like this:

$$E(\theta) = \frac{1}{m}\sum_{i=1}^{m} L(y_i, \hat{y_i}) \tag{2.13}$$

where y_i and $\hat{y_i}$ are the target label and prediction (respectively), θ is a collective variable that indicates that $E(\theta)$ depends on multiple parameters, and m refers to the number of samples, The term $\frac{1}{m}$ acts as a weight, indicating that the cost considers the loss of each observation as equally important in this example. The function L evaluates the loss/error for each target-prediction pair. The cost function and the loss have the same dimension as the output layer.

The back-propagation algorithm is a technique used for tuning the parameters of neural networks by iteratively updating the weights and biases based on the error calculated at the output layer. This process involves using an optimization algorithm, such as the *gradient descent*, which aims to find the optimal weights and biases that minimize the cost function. The computation of the update for each weight w and bias b in gradient descent goes as follows:

$$w_{t+1} = w_t - \epsilon\frac{dE(\theta)}{dw} \tag{2.14}$$

$$b_{t+1} = b_t - \epsilon\frac{dE(\theta)}{db} \tag{2.15}$$

where ϵ is the *learning rate*, a hyper-parameter controlling the step size taken.

The Adam optimizer [KB14], a modified version of the gradient descent algorithm, was utilized for training the neural networks in this thesis.

Universal Approximation Theorem

The Universal Approximation Theorem A.1.5, first introduced in [Cyb89], is an essential theorem in the field of neural networks as it highlights the strengths of these models. It states that a feed-forward neural network with a single hidden layer can approximate any continuous function to an arbitrary accuracy greater than 0. However, the theorem does not offer a method for finding the optimal network; it simply states that such construction is possible. For a detailed proof of this theorem, please refer to Appendix A.1.

2.3 Graphs

In this thesis, we use graph theory to model and solve problems related to road construction sites. Graph theory is a branch of mathematics that studies the structure and properties of graphs, which are abstract models of relationships between objects. We refer to the book [BM08] for the core concepts of graph theory.

A *graph* G, consisting of a set of *vertices* $V(G)$, also called *nodes*, and a set of *edges* $E(G)$. A vertex represents an object, such as a location, and an edge represents a relationship between two vertices, such as a connection or a road.

In the context of our problem, we represent the road network as a graph. The edges of this graph have associated costs, known as *edge weights*, resulting in a *weighted graph*. Depending on the weighting system used, the weighted graph can be either directed or undirected.

A *directed graph* (example in Figure 2.5) is a graph where edges have a designated direction, with each edge only traversable one way. Conversely, *undirected graph* (example in Figure 2.6) is a graph in which edges are *bidirectional*, allowing for traversal in either direction. If one uses a distance-based weighting system, the graph will be undirected because the weight of each edge remains constant regardless of the direction of traversal. However, if the edge weights depend on factors such as fuel consumption, the graph will likely be directed since the weight may differ in each direction.

Two vertices in a graph are *adjacent* if an edge is connected. In Figure 2.6, it can be observed that vertex $N5$ is adjacent to the vertices $N1, N4, N6$, and $N9$.

In graph theory, a *path* refers to a sequence of vertices connected by edges. A *cycle* is a path that begins and ends at the same vertex.

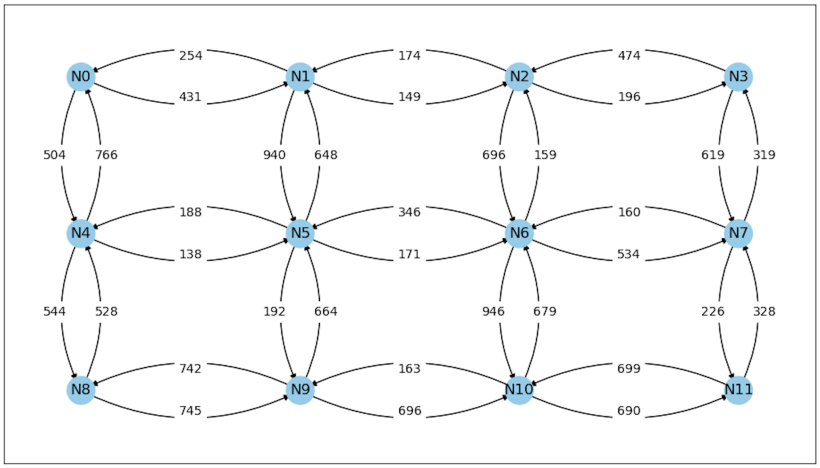


Figure 2.5: Example of a directed graph, with arrows indicating the direction of the edges.

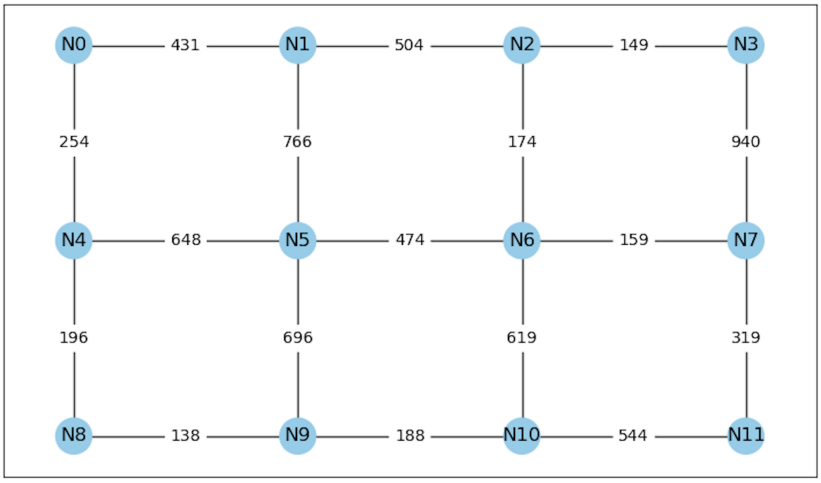


Figure 2.6: Example of an undirected graph, where edges are bidirectional.

All-pairs shortest path problem

In graph theory, the *all-pairs shortest path problem* involves finding the shortest path between every pair of nodes in a weighted graph. The problem arises in various applications, such as routing and navigation systems, where finding the most efficient way to travel between multiple nodes is crucial. In this problem, the *shortest path* is the set of nodes one must traverse to achieve the shortest distance between two nodes in the graph. The *shortest distance* between two nodes is the minimum sum of all edge weights along a path connecting them.

Floyd–Warshall algorithm

The Floyd–Warshall algorithm, presented in Algorithm 1, solves the *all-pairs shortest path problem*, by finding the shortest distance between all pairs of nodes rather than the complete information of the paths. This information is enough to find the shortest path in simple operations (one way: Algorithm 3).

Algorithm 1 Floyd–Warshall [Hou10]

Input: A digraph G with $V(G) = \{1, 2, ..., n\}$ and weights $c : E(G) \to \mathbb{R}$

Output: An $n \times n$ matrix M such that $M[i, j]$ contains the length of the shortest path from vertex i to vertex j.

```
1:  M[i,j] := ∞   ∀ i ≠ j
2:  M[i,i] := 0   ∀ i
3:  M[i,j] := c((i,j))   ∀ (i,j) ∈ E(G)
4:  for i := 1 to n do
5:      for j := 1 to n do
6:          for k := 1 to n do
7:              if M[j,k] > M[j,i] + M[i,k] then
8:                  M[j,k] = M[j,i] + M[i,k]
9:  for i := 1 to n do
10:     if M[i,i] < 0 then return ('graph contains a negative cycle')
```

2.4 Bruce's algorithm

The author of this thesis has developed an algorithm to solve the all-pairs shortest path problem (as explained in Section 2.3) in weighted graphs. The algorithm searches for the shortest distance between every pair of nodes in the graph, similar to the well-known Floyd–Warshall algorithm (presented in Algorithm 1). Our algorithm works for both undirected and directed graphs but assumes the absence of negative cycles in the graph. This requirement is not limited in its intended application, as the graph represents a road network, and the edges represent the traversal cost. We assume positive cost as it should represent distance or fuel consumption.

Bruce's algorithm, presented in Algorithm 2, utilizes a process of *adjacent inquiry* to determine the shortest distances between all nodes in the graph. Adjacent inquiry means that each node should ask its adjacent nodes about their knowledge of the shortest distance. Initially, the algorithm sets the distance between all pairs of nodes to the maximum possible value, which is either the sum of the absolute values of all edge weights or a large number representing infinity. If two nodes are adjacent, the algorithm stores the corresponding edge weight.

The inquiry begins with a node pair: a start node and an end node. The start node asks all its adjacent nodes for their estimated shortest distance to the end node. The same process applies to every node pair. The algorithm repeats this process until the estimated shortest distances do not change, confirming that they are indeed the shortest.

Algorithm 2 Bruce's algorithm (Find Shortest Distances)

Input: A directed - or undirected weighted graph G with $V(G) = \{1, 2, ..., n\}$ and weights $w : E(G) \to \mathbb{R}$

Output: An $n \times n$ matrix S such that $S[i, j]$ contains the length of the shortest path from vertex i to vertex j.

```
1:  S[i, j] := ∞   ∀ i ≠ j
2:  S[i, i] := 0   ∀ i
3:  S[i, j] := w((i, j))   ∀ (i, j) ∈ E(G)
4:  updates = 1
5:  while updates > 0 do
6:      updates = 1
7:      for i := 1 to n do
8:          for j := 1 to n and j ≠ i do
9:              for (k, w) in (i.edges.other_node, i.edges.weight) do
10:                 if w + S[k, j] < S[i, j] then
11:                     S[i, j] = w + S[k, j]
12:                     S[j, i] = w + S[k, j]          ▷ if the graph is undirected
13:                     updates = updates + 1
```

Algorithm 3 shows how to get the shortest path between any two nodes based on the information of the shortest distances. We obtain the path through adjacent inquiry by following the edge that satisfies this condition: the shortest distance from the current node equals the sum of the shortest distance from the adjacent

node and the weight of the edge connecting them. We repeat this process until we reach the end node.

Algorithm 3 Bruce's algorithm (Find Shortest Path)

Input: `start_node, end_node`
Output: the shortest path: `[start_node, ..., end_node]`

```
path = [start_node]
next_node = start_node
while next_node != end_node do
   shortest_distance = next_node.shortest_distance_to(end_node)

   for edge in next_node.edges() do
      tmp_node = edge.to_node
      tmp_shortest = tmp_node.shortest_distance_to(end_node)
      shortest_using_edge = tmp_shortest + edge.weight

      if shortest_using_edge == shortest_distance then
         next_node = tmp_node
         path = path + [next_node]
```

Return: `path`

Table 2.1 shows the computation time of Bruce's algorithm. As n increases, the computation time grows significantly. However, the algorithm is still suitable for our problem because the road construction site expects to have fewer than 1000 nodes. These nodes represent intersections or decision points within the construction site.

If this algorithm is a bottleneck for the problem, one can improve performance by parallelizing the algorithm and running it on a computer with higher processing power.

Table 2.1: Speed test of Bruce's algorithm 2, serialized version

n	Nodes	Edges	Time (s)
10	100	340	0.1
20	400	760	1.2
30	900	1740	8.6
40	1600	3120	33
50	2500	4900	103
60	3600	7080	270
70	4900	9660	604
80	6400	12640	1419
90	8100	16020	2512
100	10000	19800	5188

The graphs are generated as a grid of $n \times n$ nodes. The edges connect the nodes as in Figure 2.6, resulting in $2n(n-1)$ edges. The test was performed with *MacBook Pro (2020) Apple 8-core M1 CPU GHz, 8GB RAM, 512GB SSD* [APP20].

CHAPTER 3

Method

The author of this thesis developed the framework presented in this chapter from scratch, incorporating some ideas from [Qin+21].

Section 3.6 provides a guide to the framework's configuration options. This section, in addition to the default values of the framework presented in Appendix A.2, contains the necessary information for using the framework effectively.

However, before exploring the configuration options in detail, it is essential first to thoroughly understand the problem at hand and the framework itself.

3.1 Route Optimization on Construction Site

This thesis aims to find the most efficient assignment of dumpers to jobs on a construction site. To accomplish efficient plans, we need to consider various factors, including the map of the construction site, the locations of the dumpers and loaders, and the requirements of each job, such as the volume of mass transported. The *dumpers* are the vehicles transporting the materials to the designated dumping areas, while the *loaders* are the stationary units that load the materials onto the dumpers. The loaders will always fill the dumpers to their maximum capacity if there is sufficient remaining mass. Typically, a loader on a construction site is an excavator, although it may also take the form of a conveyor belt or a person using a shovel. The type of loader used impacts the loading process and fuel consumption rates.

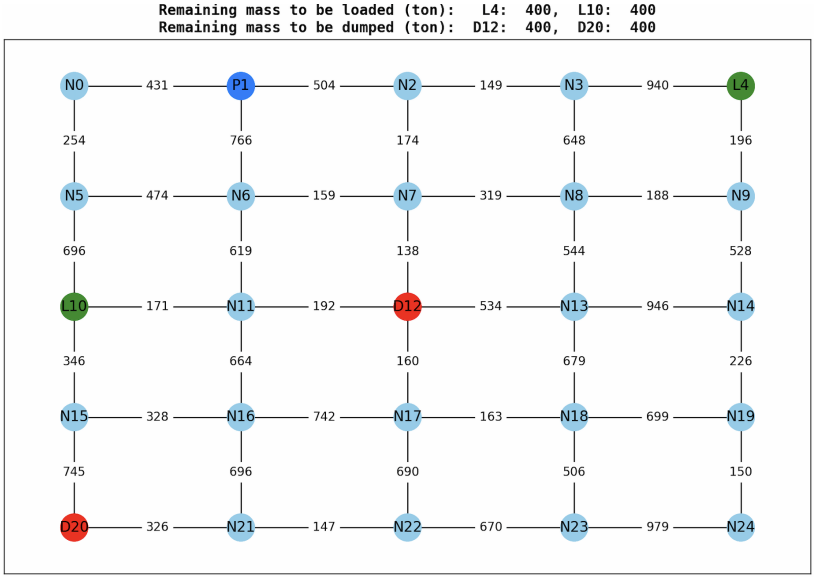
There are several ways to optimize this system, such as by minimizing fuel consumption, time, or project cost. If the algorithm can learn to optimize for one of these objectives, it can be adapted to optimize for the others, as the difference primarily affects the reward function. The main challenge is incorporating the project leader's chosen optimization objective into the reward functions.

The framework consists of several agents with distinct responsibilities and reward functions. During training, the framework generates a new plan (dumper schedule) for each episode, which is evaluated based on a range of metrics, including driving distance, waiting times for dumpers and loaders, and overall execution time. To simplify evaluation, a single metric, such as fuel consumption, could be used by accurately weighting the other measures.

3.2 An Overview of the Problem's Environment

In this context, the *environment* is the map of the road construction site, which includes the locations of the jobs and the dumpers and loaders (an example in Figure 3.1). We represent the map as a weighted graph, where the *nodes* represent the locations of the intersections/decision points on the construction site, and the *edges* represent the connections between them (roads). By using the information provided by the nodes and edges, such as the distances or the predicted fuel consumption, we can optimize the assignment of the dumpers to the jobs. In practice, we could generate the graph from the same GPS data used for the fuel model (Part II). However, for the sake of this master's thesis, let us assume that we have the graph already.

Figure 3.1: Representation of a road construction site as a weighted graph



Node colors: loading (green), dumping (red), parking (dark blue), and intersection (light blue)
Task: Transport 400 tons from each loading node to each dumping node.
This visualization of the graph was generated automatically by the framework.

Each job on the construction site specifies the amount and type of mass to transport from a designated *loading node* to a *dumping node*. The mass is loaded at the loading node, transported to the dumping node, and then dumped at its destination. When multiple jobs contain the same type of mass, it is possible to transport the materials between jobs rather than directly from the specified loading node to the dumping node.

A node cannot serve as both a loading and a dumping node. An additional node should be added to the map adjacent to the desired node to accommodate multiple jobs at one location.

Furthermore, the environment includes a parking node for the dumpers, which can function as both the starting and ending points for the dumpers at the beginning and end of their day. Moreover, the framework can utilize this node to let dumpers park whenever deemed more efficient.

3.3 The Roles of the Agents: An In-Depth View

For our problem, we found it relevant to explore the application of *multi-agent reinforcement learning* (inspired by [Qin+21]). Multi-agent reinforcement learning involves multiple agents coexisting and interacting within a shared environment, each with its own goals and rewards. In our case, the agents should cooperate to achieve a common goal: optimize the coordination on a road construction site.

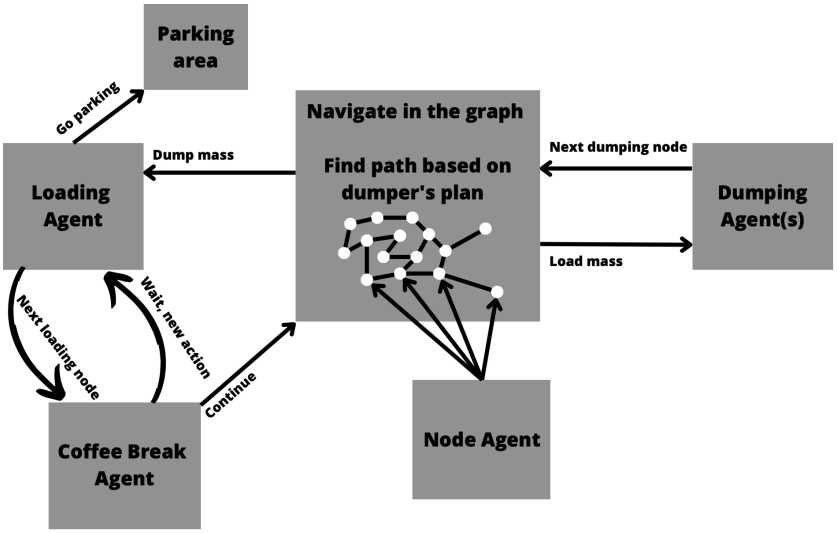
The problem involves a set of dumpers that should behave similarly, assuming all other factors, such as speed, mass capacity, and dumping ability, are equal. Each dumper should alternate between loading nodes and dumping nodes while seeking to optimize the path to its final destination. Specifically, when situated at a loading node, the dumper should identify the most *suitable dumping node* to go to next, considering its job status and ability to accept the type of mass loaded. On the other hand, when stationed at a dumping node, the dumper should identify the most *suitable loading node* to travel to, one that has not yet fulfilled its job and has short queues. Determining the optimal loading and dumping nodes involves a weighted decision among both the distance and idle time required.

Using a single agent attached to each dumper is not an efficient approach since it does not allow for learning from the experiences of other dumpers. Alternatively, having a single agent that steers all the dumpers may seem reasonable, but it can quickly become computationally expensive and challenging to scale. However, the structure naturally lends itself to be divided into more straightforward tasks that are easier to learn, leading to faster and more accurate learning. Our work has resulted in the implementation of four distinct types of agents:

1. Loading Agent: determining the next loading plan (optionally: send it to a parking node).
2. Dumping Agent(s): determining the next dumping plan, one agent responsible for one type of mass.
3. Coffee Break Agent: evaluating the Loading Agent's plan, and it can allow dumpers to take temporary breaks.
4. Node Agent(s): determining the next edge to follow, given a plan.

Figure 3.2 illustrates how the agents work together within the framework.

Figure 3.2: Cooperation among the agents and the cycle of the dumpers



Policy

In our framework, we utilize the Deep Q-learning method described in Section 2.1, where we equip each agent with a neural network. The neural network takes the state information as input and outputs a vector containing a Q-value for each possible action. The agent selects its action by choosing the index of the maximum Q-value, where each index corresponds to a specific action.

Training individual policies in our multi-agent problem is relatively straightforward since there is no clear *continuation* of states for an agent. As a result, we found it efficient to use only the current state to train the model and not consider the next or previous states. This approach necessitates setting $\gamma = 0$ in the Bellman Equation 2.12, which updates the Q-values.

A Closer Look at the Planning Agents

State - Loading Agent

Equation 3.1 represents the state information for the Loading Agent when the environment consists of N loading nodes.

$$S^L = (D_{\text{parking}}, I_1, I_2, ..., I_N, D_1, D_2, ..., D_N) \tag{3.1}$$

In Equation 3.1, I_i represents the predicted waiting (idling) time in minutes at loading node i, and D_i denotes the shortest distance to it from our current location. The distance measure can be physical distance, fuel consumption, or time (which is the default). There is correspondence between $\{I_i\}_{i=1}^N$ and $\{D_i\}_{i=1}^N$ for all N possible loading nodes. Additionally, D_{parking} represents the distance to the nearest parking node.

At first, we set the shortest distance between every pair of nodes on the map to equal the sum of all edge weights (in absolute value). However, as dumpers

find more efficient routes than the current best, we update the shortest distance accordingly.

The predicted waiting time is calculated based on the presence of dumpers at the destination node and the estimated arrival time of other dumpers. Additionally, the calculation considers the shortest distance obtained so far, the speed of the dumpers, and the estimated duration of loading/dumping at the destination node. The waiting time is allowed to be negative, with the interpretation of *How long has the loader been available*. This possibility is an essential factor to consider in the decision-making process since the loaders consume fuel and should aim to avoid idling during their work, thus minimizing fuel waste.

State - Dumping Agent

Equation 3.2 shows the state information for a Dumping Agent(s) responsible for M dumping nodes.

$$S^D = (\Delta m_1, \Delta m_2, ..., \Delta m_M, D_1, D_2, ..., D_M) \tag{3.2}$$

In Equation 3.2, we calculate the values $\{\Delta m_i\}_{i=1}^{M}$ using Listing 3.1. Furthermore, D_i denotes the shortest distance from our current location to dumping node i and exhibits identical behavior to that described in the state of the Loading Agent (Section 3.3). There is a relationship between $\{\Delta m_i\}_{i=1}^{M}$ and $\{D_i\}_{i=1}^{M}$ for all M possible dumping nodes.

The value of Δm_i represents how many dumps by which dumping node i is ahead of the dumping node with the lowest completion rate, based on their respective completion rates. The completion rate refers to the progress made by each dumping node regarding the number of tons of mass they have received out of the total amount required. The Dumping Agent can learn to distribute dumpers more evenly among all dumping nodes by considering the imbalance in dump completion rates. This consideration ensures that all dumping nodes remain accessible longer during the episode and prevents some nodes from finishing early. Furthermore, it may reduce waiting times and prevent suboptimal planning toward the end of the episode.

Listing 3.1: Pseudocode: Calculation of $\{\Delta m_i\}_{i=1}^{M}$, `delta_mass`

```
# Calculate the completion rate for each dumping node
received_mass = [received_mass_1, received_mass_2, ..., received_mass_M]
total_mass = [total_mass_1, total_mass_2, ..., total_mass_M]
completion_rate = mass_dumped / total_mass

# Find the progression relative to the lowest completion rate
base_index = completion_rate.index(min(completion_rate))
progress_ahead = completion_rate - min(completion_rate)

# Calculate the delta_mass included in the state information
delta_mass = progress_ahead * (total_mass[base_index] / dumper_capacity)
```

Action - Loading Agent

The *Loading Agent* selects the next loading node to serve and is called upon whenever an empty dumper requires a new loading destination. The policy outputs align with the loading nodes; however, if the agent's preferred option becomes unavailable due to job completion, the agent automatically chooses the next best option based on the policy. We can enable an additional action to the Loading Agent's action space, allowing it to park dumpers instead of assigning them a loading plan if it determines this to be more beneficial. Once a dumper is parked, it cannot be assigned any new plan in the current episode.

Action - Dumping Agent

The *Dumping Agent(s)* operates similarly to the Loading Agent but selects the next dumping node instead of a loading node. Each mass type has its own Dumping Agent, which imposes restrictions on where specific types of mass can dump. Consequently, the dumper must consult the Dumping Agent responsible for the loaded mass type. It is important to note that a Dumping Agent cannot direct dumpers to a parking node; only empty dumpers can park.

Rewards - Loading Agent

Let us consider a Loading Agent that issues a plan. Upon completing the loading plan, defined as the dumper reaching its designated final destination and successfully executing the task at the location, the dumper will provide Information 3.1 to the Loading Agent.

Information 3.1: Post-information of a loading plan

1. T_D (driving time in seconds): time to arrive at the destination node
2. W_D (waiting time dumper in seconds): queue time at the destination node
3. W_L (waiting time loader in seconds): time since loader was last used

Equation 3.3 shows the setup of the reward function and is dependent on Information 3.1.

$$R(W_L, W_D, T_D) = \min\left[\frac{K}{W_D + w_1 \cdot T_D - w_2 \cdot W_L}, 80\right] \tag{3.3}$$

In Equation 3.3, K is a constant used to determine the magnitude of the rewards. The weights (w_1, w_2) corresponds to T_D and W_L, respectively. The weights indicate that T_D is w_1 times more costly than W_D, and W_L is w_2 times more expensive than W_D. The negative sign to w_2 indicates that the Loading Agent should prioritize a loader expected to idle for a long time to prevent longer idling times.

Setting the weights $(w_1, w_2) = (8, 3)$ indicates that an idling loader consumes three times more fuel than an idling dumper. Moreover, the fuel consumption of a moving dumper is, on average, eight times greater than when it is idling. The values align with those expected from a heavy-duty excavator and a 40-ton dumper.

We want to set K such that the rewards fall in the domain of [0, 80]. To achieve this, we first set the shortest time between any pair of loading and

dumping nodes (also possible parking nodes) to T_{SD}. In Equation 3.3, setting $K = 50 \cdot (w_1 - w_2) \cdot T_{\text{SD}}$ results in a reward of 50 for the agent if the dumper follows the shortest possible route, do not have to wait in a queue, and the loader is ready for the entire duration of the trip. This is achieved when $T_D = T_{\text{SD}}, W_D = 0, W_L = T_D$.

Determining an appropriate reward for a dumper directed to a parking node is challenging as it requires information about the remaining part of the episode. The reward system for parking suggests that if there are no idle loaders during an episode, decreasing the number of dumpers in the planning may be beneficial. Conversely, increasing the number of dumpers may be advantageous if idle loaders exist. To overcome this issue, we devised a solution that addresses all such cases at the end of each episode when the necessary information is available. We set the parameter T_D equal to the driving time to the parking node and impose a penalty for parking through W_D, computed using the formula in Equation 3.4.

$$W_D(t_0) = w_4 \cdot \frac{1}{N} \sum_{i=1}^{N} W^{\text{total}}_{\text{loader}_i}(t_0) \tag{3.4}$$

In Equation 3.4, we define $W_D(t_0)$ as a measure of the cost for parking at time t_0, where w_4 is a hyperparameter that determines the degree of this cost. To calculate $W_D(t_0)$, we first define $W^{\text{total}}_{\text{loader}_i}(t)$ as a function that computes the total idle time of loader i after time t in the current episode. We do not add any idling time if a dumper arrives loader i at the earliest possible time in an episode. Moreover, N is the number of loaders in the environment.

By setting $w_4 = 1$, the Loading Agent is penalized for parking a dumper at time t_0 as if the dumper was idling for the average idling time experienced by all loaders after time t_0.

Rewards - Dumping Agent

To calculate the reward for the Dumping Agent(s), we follow a procedure similar to the Loading Agent (Section 3.3). The reward function takes the same form as Equation 3.3 but with a different interpretation of W_L. As explained in Section 3.3, the state information provided to the Dumping Agent contains details of the completion rate imbalance between the dumping nodes. To quantify this imbalance, we need to determine how many dumps ahead the dumping node selected by our plan is compared to the dumping node with the lowest completion rate. We assign this value to W_L, and the corresponding weight w_2 in Equation 3.3 is negative to penalize imbalance in the completion rates. For example, setting $w_2 = -60 \cdot w_1$ indicates a willingness to travel an extra minute per unit of imbalance to compensate for the imbalance.

Since the dumping time is shorter than the loading time, dumping sites are less likely to have queues. Thus, we set $W_D = 0$ and do not prioritize it.

The Coffee Break Agent: A Solution for Temporary Breaks

The *Coffee Break Agent* was introduced late in the research as we found putting dumpers on a temporary break helpful, allowing other dumpers to select their next loading plan first. Without this feature, the Loading Agent would assign plans to dumpers as they become available without considering that the next available dumper might be much closer to the desired loading node.

As Illustrated in Figure 3.2, the Loading Agent and Coffee Break Agent work in close collaboration. Whenever the Loading Agent has decided on a plan, the Coffee Break Agent evaluates it. The Coffee Break Agent decides whether the dumper should continue with it or wait for the next dumper to select its plan before making a decision. The process continues until the Coffee Break Agent approves the plan.

If the temporary break lasts until the end of an episode, we consider the dumper going directly to the parking area at the start of the break. This situation highlights the similarity between the Loading Agent's parking option and the Coffee Break Agent. However, a key difference is that when a dumper is parked, it remains so for the entire episode, whereas the Coffee Break Agent allows for temporary pauses. Disabling the parking option and utilizing all available dumpers in the environment can be advantageous in reducing completion time in some cases. In such scenarios, the Coffee Break Agent becomes even more critical, especially in environments with many dumpers.

In some scenarios, the first available dumper is far from the relevant loading node, while another dumper much closer becomes available shortly after. In such cases, it is clear that the distant dumper should not be assigned the plan and instead wait for a more suitable one, as illustrated in Section 4.2. This extreme scenario serves to highlight the necessity of the Coffee Break Agent in ensuring optimal plan assignments.

It is important to note that the Coffee Break Agent is not involved with the Dumping Agent.

State

The state information for the Coffee Break Agent, as shown in Equation 3.5, is based on the plan issued by the Loading Agent and the potential *competitors* for the same loading node.

$$S^C = \left(\frac{T_D}{T_{\text{MD}}}, \frac{T_{\text{SD}}}{T_{\text{MD}}}, \frac{T_{\text{SL}}}{T_{\text{MD}}}, Q \right) \tag{3.5}$$

Equation 3.5 specifies four variables that make up the state information: the current dumper's arrival time (T_D), the shortest possible arrival time for dumpers on their way to or at a dumping node (T_{SD}), and the shortest possible arrival time for dumpers on their way to or at a loading node that requires a trip via a dumping node (T_{SL}). Scaling all arrival times by 1/10 of the maximum time between any pair of loading and dumping nodes (or the parking node) (T_{MD}) allows applying the approach in various environments. The Loading Agent's policy generates the final information, the Q-value for the intended loading plan (Q).

Incorporating the arrival times of other dumpers in the state can enable the Coffee Break Agent to make more informed decisions about when to take breaks.

Action

The Coffee Break Agent's actions are straightforward: it must either approve or reject the Loading Agent's plan, instructing the dumper to proceed with its current plan or advising it to wait for a future opportunity.

Rewards

The Coffee Break Agent's reward function is similar to that of the Loading Agent (as shown in Equation 3.3). If the dumper follows the recommended plan, the Coffee Break Agent receives the same reward as the Loading Agent. However, if the dumper takes a temporary break, the duration of the break is added to the waiting time for the dumper, with a weight w_5. The weight w_5 must be lower than w_2 because the agent should not learn that it is advantageous to let loaders idle while a dumper is on a break, as the dumper may wait for a long time before deciding to resume work.

If a dumper remains on a break until the end of an episode, we treat it as if it went to the parking node at the start of the break. Assigning a meaningful reward for this situation is difficult since it is hard to determine if remaining on a break was a smart move and, at the same time, align it with other rewards. As a result, the Coffee Break Agent does not receive any reward for those dumpers, which means there are no samples from which to learn.

Node Agents: Guiding the Way through Graphs

The final type of agent is the Node Agent. Instead of allowing the dumpers to find their route, we make the nodes into agents (inspired by [Qin+21]).

When a dumper arrives at a new node, the corresponding Node Agent selects the next edge to follow based on the available information from the dumper. The state information for the Node Agent is a one-hot vector, where the activated entry represents the dumper's next destination node. Each index in the state corresponds to a unique node in the map.

The Node Agent's action specifies the next edge the dumper should follow. The policy of the Node Agent can either be fixed or trainable. When operating under a fixed policy, the Node Agent consistently selects the same action for a given state. In contrast, a trainable policy will adapt and change based on the experience it gains over time.

Rewards

Whenever a dumper is driving, it must have a plan provided by a planning agent. The dumpers should share their plan with all Node Agents on the path to the destination node. Rewards are calculated based on the path after a dumper completes its plan. Initially, the Node Agents make random choices, resulting in many repetitive nodes along the path. Given a path, we deploy the following algorithm to provide rewards to the Node Agents:

Algorithm 4 Pseudocode: Give reward to Node Agents based on a path

Input: *path.* the nodes a dumper visited while finding its way to the plan's destination node.

Note: - Whenever the reward R occurs, the corresponding Node Agent will receive this information and learn.
- ** ... ** means it should be done.

```
path_indices = [0, 1, ..., len(path) - 1]
end_node = path[-1]
final_path = []
i = 0

# Find final path
while i < (len(path) - 1) do
    j = ** index of last occurrence of path[i] in path **
    final_path.add(path[j])
    i = j + 1
final_path.add(end_node)

# Calculate reward, R, for nodes in final path
for idx in range(final_path - 1) do
    ### Shortest observed distance is base_dist ###
    shortest_observed_dist = ** final_path[idx] to end_node **
    curr_dist = ** distance by traversing final_path[idx:]  **

    R = 10 * shortest_observed_dist / curr_dist          ▷ Green

# Rewarding remaining nodes in the path
for idx in path_indices do
    if path[idx] in final_path then
        if ** same state-action pair (Green case) ** then
            Continue                                     ▷ Gray
    R = 0                                                ▷ Red
```

green: will receive a positive reward based on path optimality
gray: will be skipped in training (duplicates of green)
red: will receive 0 as reward

Algorithm 4 categorizes the nodes in the path into three different cases, each with a different reward system. Green cases include nodes forming the final path, defined as the shortest distance from the starting node to the destination node based on the dumper's path. We implemented the algorithm such that the final path is composed of the last occurrence of each node. This implementation does not make any difference, as it is more of a coding matter. The duplicates of the green cases, called gray cases, are skipped. The remaining nodes, the red cases, are rewarded with 0, which sets a lower and upper bound for the reward at 0 and 10, respectively.

However, this approach may lose some information the agents hold, as it

excludes information about detours. Table 3.1 provides an example of such a case.

Table 3.1: Example path, and the splitting happening in Algorithm 4

path	[L3, N4, N5, N4, L3, N2, L3, N4, N5, N4, N5, D6]
(path_colored)	[L3, N4, N5, N4, L3, N2, L3, N4, N5, N4, N5, D6]
path_indices	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
final_path	[L3, N4, N5, D6]

Comments: L, D, and N are the Loading node, Dumping node, and (intersection) node (respectively).

The example in Table 3.1 shows that we could give the Node Agent placed at node $N2$ a more suitable reward by adding it first in the final_path. Missing this information is not a problem because the discovery of the transition from $N2 \rightarrow L3$ will most likely occur in another path.

Enhancing Node Agents with Bruce's Algorithm

We use Bruce's algorithm (see Section 2.4) to determine the shortest path between nodes. This algorithm always finds the optimal solution and is faster than having the Node Agents train from scratch.

If our sole objective is to follow the shortest path, we can freeze the Node Agents and instruct them only to follow the path discovered by Bruce's algorithm. However, if our goal is to expand the knowledge of our Node Agents beyond just the shortest path, providing the shortest distance to the environment can make their training more effective. This increase in effectiveness occurs because the Node Agents use the shortest observed distance to calculate their rewards. If this distance remains constant throughout training, it improves the accuracy of the rewards and leads to faster learning.

In situations where edges may become more costly due to traffic and road closures having additional knowledge beyond the shortest path can be beneficial. By training our Node Agents to consider those factors, we can improve their ability to navigate around obstacles and find alternative paths.

3.4 A Detailed Breakdown of an Episode

We define an *episode* of the framework as completing all jobs provided or expiring a pre-defined timer within the algorithm.

Initialization

(1) Jobs and timer
The user must specify the map of the road construction site, the capabilities of the loaders, and the jobs to complete. The environment regulates the amount of mass that is moved and tracks the progress of the jobs. We control the episode's duration by setting a pre-defined timer. The episode terminates if the dumpers do not complete the tasks within the designated time frame.

(2) Dumpers starting node
We must specify the maximum number of dumpers on the road construction site and determine how to map the dumpers to starting node(s). In a real-case scenario, we can use GPS coordinates to map each dumper to the nearest node. During testing, we mapped all dumpers to the parking node. Alternatively, we can distribute them evenly among all loading nodes or distribute them randomly among all nodes.

(3) Starting order of dumpers
Establishing the starting order is crucial for optimizing the efficiency of the initial stage of an episode. The state information may need to be clarified at the beginning of an episode since the framework initializes the dumpers sequentially. An algorithm that prioritizes closeness and availability ensures that dumpers receive their first loading plan in an organized order. This organized order allows the agents to access more accurate information when deciding which plan to follow. See Appendix A.3 for details on determining the starting order.

Actions and events within an episode

Let $T = \{t_1, t_2, ..., t_d\}$ represent the set of timers for the d available dumpers in the environment. Timer t_i corresponds to dumper i and indicates the time remaining until dumper i must act.

(4) Beginning of an episode
At the beginning of an episode, when $T = \{0, 0, ..., 0\}$, all dumpers are required to act. If dumper i starts at its plan's destination node, it reserves the corresponding loader/dumping area or is placed in a queue if multiple dumpers have the same plan. Conversely, if dumper i is away from its intended destination node, it provides its plan to the Node Agent at the current node. The Node Agent will then direct dumper i to traverse an edge based on its plan. Regardless of its action, add the duration until dumper i must perform another action to t_i (measured in seconds). If the destination node is the parking node, then the corresponding timer t_i is set to the maximum time, preventing dumper i from taking further actions.

(5) Repeat until the dumpers complete all jobs or the pre-defined timer expires
After initializing the dumpers in the environment, they make decisions dynamically. Whenever $t_i = 0$ for some $i = 0, 1, ..., d$, the dumper(s) i must act. For instance, after completing a loading or dumping task, it calls the Dumping or Loading Agent (respectively) for a new plan. If it has reached its destination node, it may initiate or join a queue for loading or dumping. Alternatively, if it is en route to its final destination, it calls a Node Agent to determine which edge to traverse next. All tasks consume time, and we add their duration to the corresponding dumper's timer. However, if the plan's destination is the parking node, dumper i remains parked throughout the episode. Figure 3.2 illustrates how the cycle of all dumpers works in the framework.

After the dumpers complete all necessary actions or events in a time step, we update the time system T by subtracting the minimum value of T from all its elements $T := T - min(T)$. This update means that T takes uneven increments based on the shortest time until the next dumper(s) must act. As a result, the environment's time also advances unevenly.

Post-episode

(6) Store information, train agents
After each episode, the framework saves a significant amount of information, including the plans executed by the dumpers, the schedule of the loaders, and various performance metrics. These metrics include completion time, job completion rate, total distance driven by the dumpers, and idle time for dumpers and loaders.

To make it easier to compare plans, we assign a fuel consumption and cost value to each plan. We calculate these values based on the previously mentioned metrics and pre-defined constants such as fuel price, fuel consumption rates, and salaries.

The agents train both during and after each episode. During an episode, the agents store the state-action pairs they have executed and the corresponding rewards in their memory. These samples remain in the agents' memory for several episodes. In the post-episode stage, the agents use a random batch of their stored samples for training.

3.5 Exploration vs. Exploitation

The framework allows for two different approaches to agent exploration. The first approach involves setting a fixed *random choice probability* for the agents. The second approach involves setting an *exploration parameter* that allows the agents to act gradually according to their trained policy. It is important to note that the agents can learn simultaneously or sequentially.

When setting the exploration parameter to n, the agent selects its action as follows:

$$a(s) = \begin{cases} \arg\max_a Q_\theta(s,a) & \text{with probability} \quad \min(\frac{c}{n}, 1) \\ \text{random } a & \text{with probability} \quad \max(1 - \frac{c}{n}, 0) \end{cases} \tag{3.6}$$

where $c = 0, 1, ..., n, ...$ is the current episode number and n is the exploration parameter.

The planning agents will disregard actions that cannot be executed due to the job status at the loading/dumping nodes (as explained in Section 3.3).

In the simplest form, the Node Agents are pre-computed and not trained at all (see Section 3.3). However, if we train them from scratch, the author has achieved favorable results by splitting the exploration of the agents. This splitting includes letting the Node Agents explore while the planning agents make random actions, and the Coffee Break Agent is deactivated. This strategy is motivated by the fact that the planning agents' reward function relies on the route taken by the dumper and that the Coffee Break Agent depends heavily on the Loading Agent. Consequently, if the Node Agents make unreasonable choices, such as taking a detour, the planning agent should avoid learning from such situations.

Repeated exploration

Using multiple exploration numbers instead of relying on a single high one can help broaden the search and avoid getting stuck in local minima. This approach allows the agents to explore a more diverse set of states and increases the chances of finding the global optimum. Specifically, we can set the *exploration vector* to (n, k, m), where n is the exploration number used in Equation 3.6, k represents the number of episodes in which the agent's actions are solely determined by the policy (without exploration), and m is the number of times we repeat this process. This exploration vector will make our framework follow the repeated exploration setup for $m(n + k)$ episodes.

3.6 A Guide to the Framework's Configuration Options

The framework offers several configuration options for the environment, jobs, dumpers, loaders, and agents. In this section, we will present and discuss these different options. The framework's default values are in Appendix A.2.

Environment, Map

The framework represents the road network as a weighted graph and categorizes the nodes in the graph as one of the following types: loading node, dumping node, parking node, or (pure) intersection node. Loading, dumping, and parking nodes are intersection nodes with an additional specialty. It is important to note that the framework does not support nodes with multiple specialties. If a node needs more than one specialty, one should place a new node next to the desired node. The framework can only handle one parking node.

It is possible to manipulate the agents into believing that the loading nodes are waiting at the beginning of an episode, which may result in earlier utilization of them. Conversely, the loading nodes may remain inactive until their initial use, potentially resulting in each loading node completing its tasks before moving on to the next.

All edges in the graph should be assigned a distance and, optionally, a fuel consumption value. We can use GPS data to calculate these distances in a real-case scenario. At the same time, each edge can obtain a fuel consumption value by integrating the fuel model (Part II) into the framework.

We set a maximum time limit (pre-defined timer) to end an episode if the dumpers do not complete all jobs within the allotted time.

Jobs

A job description within the framework should include the following:

- A loading node
- A dumping node
- The loader positioned at the loading node
- An identifier for the type of material to transport
- The quantity of material to transport

It is important to note that the framework does not support assigning multiple mass types to a single loading/dumping node. However, adding a new node to the map can circumvent this restriction.

Dumpers

The framework necessitates the specification of the maximum number of dumpers available. We must assign each dumper an average speed and maximum carry capacity for use during graph traversal. The user must also define the dumping rate in seconds per ton and set an independent dumping time as a fixed duration that applies regardless of the quantity of mass dumped. Dumpers will always be loaded to max capacity if permitted by job requirements.

At the start of each episode, the framework offers several options for setting the initial positions of the dumpers. The framework can either set all dumper's starting positions at the parking node, distribute them evenly across all loading nodes, or map them randomly.

During an episode, the dumper operator will only turn off its dumper if parked. Otherwise, they will either be driving or idling.

Loaders

Each loader should have a loading rate representing the duration required to load one ton of mass. In addition, it is necessary to specify an independent loading time, which is a constant duration that applies regardless of the quantity of mass loaded.

The loaders are activated upon the arrival of the first dumper and remain operational until job completion or until the end of the episode.

Node Agents

It is possible to choose between using the Node Agents' policy or only following edges that lead to the shortest distance using Bruce's algorithm (see Section 2.4). If we train the Node Agents, we could set them to trainable or finished training. The Node Agents can explore, as described in Section 3.5.

Planning agents

The planning agents can explore, as described in Section 3.5. It is necessary to set the weights in the reward function for the Loading and Dumping Agents, as shown in Equation 3.3. The Loading Agent can enable the option to park dumpers and must set the weighting w_4 in the penalty term in Equation 3.4.

Coffee Break Agent

We can let the Coffee Break Agent be active or inactive. The penalty weight w_5 must be specified if the agent is active, as detailed in Section 3.3. The Coffee Break Agent can explore, as described in Section 3.5.

Additional parameters

To effectively compare final plans within the proposed framework, we must define fuel consumption rates for idling and driving vehicles, salaries for vehicle operators, and fuel prices. Combining metrics from a plan and these parameters allows us to assign each plan a comparable measure, such as cost or fuel consumption. These metrics from each plan include completion time, job completion rate, total distance driven by the dumpers, and idle time for dumpers and loaders.

CHAPTER 4

Results

In this chapter, we present the results of the framework described in Chapter 3. Although the settings have not been validated against empirical data, the framework is robust enough to handle all possible settings.

Unless we specify different settings for a particular analysis, we presume that we use the default settings in Appendix A.2.

4.1 Evaluate Performance: Two Baselines

We will evaluate each map based on two key factors: fuel consumption and the time required to complete all jobs. To assess performance, we will establish two baselines. Baseline 1 is based on a logical and practical approach, while Baseline 2 represents the lower bound for each map, which may not be realistically achievable.

Baseline 1: Using Practical Approach

Baseline 1 is a practical human perspective on a potential solution and serves as a reference point for evaluating the performance of our framework. The strategy consists of two steps: pairing each loading node with a dumping node and assigning dumpers to each pair.

In the first step, we pair each loading node with a dumping node. We prioritize the loading node with the maximum distance to any dumping node and let it select the nearest available dumping node. This way, we avoid having dumpers travel the longest distances on a map repeatedly. We repeat this process until all loading nodes have a partner. Appendix A.4 illustrates the pairing process used for this baseline.

In the second step, we assign a set of dumpers to each pair and let them alternate between the loading and dumping nodes until they finish the jobs. We limit the number of dumpers assigned to each pair to ensure no dumper is idle during an episode. This limit depends on the time it takes for a dumper to travel to the dumping node, dump the mass, and return to the loading node. Once we determine the maximum number of dumpers for each pair, we distribute all available dumpers evenly without exceeding the maximum number. If there are any leftover dumpers, we assign them to the pairs that need the most dumpers.

By executing the strategy of Baseline 1, we can measure its performance in terms of total fuel consumption and completion time.

Baseline 2: The Lower Bound

Please note that achieving the lower bound for completion time and fuel consumption is possible only in specific scenarios. While attaining the lower bound for time may be feasible in some cases, achieving the lower bound for fuel consumption is rare due to the challenge of simultaneously satisfying all of its assumptions.

To determine the lower bound for the time required to complete all jobs within a given map, we first calculate the travel time for a dumper to reach each loading node from the parking spot. Moreover, we assume that each loading node remains occupied until the completion of its job. Next, we add the travel time for the dumper to return to the parking spot via the optimal dumping node, minimizing travel time. Finally, we select the maximum time among all loading nodes as the optimal time to complete the map.

To find the minimal fuel consumption, we make several assumptions. First, we assume that vehicles never idle. Second, we include only one distance from the parking spot to each loading node, equivalent to sending one dumper to each loading node. Third, we assume that mass can be delivered to the nearest dumping node, even if it has reached the maximum capacity. The single dumper on each loading node will alternate between the nearest dumping node and the loading node. On the last trip from each loading node, we assume that the dumper returns to the parking spot via the optimal dumping node, minimizing distance. Finally, we sum up the total distance each loading node required and convert them into fuel consumption using the settings described in Appendix A.2.

Comparing Plans: An Overview of our Scoring System

The proposed scoring system calculates a score for a given plan based on its fuel consumption and completion time relative to Baseline 2. Baseline 2 represents the lower bound for fuel consumption and completion time (see Section 4.1).

Equation 4.1 calculates the score by assigning an 80% weight to the fuel consumption ratio and a 20% weight to the completion time ratio. This weighting prioritizes fuel efficiency while still considering completion time. However, this scoring system reflects only some considerations. If priorities for the various metrics attached to each plan differ, adjusting the weighting and possibly adding a new metric into the scoring becomes necessary.

$$\text{Score} := \left(\frac{4}{5} \cdot \frac{\text{F}_{\text{B2}}}{\text{F}_{\text{M}}} + \frac{1}{5} \cdot \frac{\text{T}_{\text{B2}}}{\text{T}_{\text{M}}} \right) \cdot 100 \tag{4.1}$$

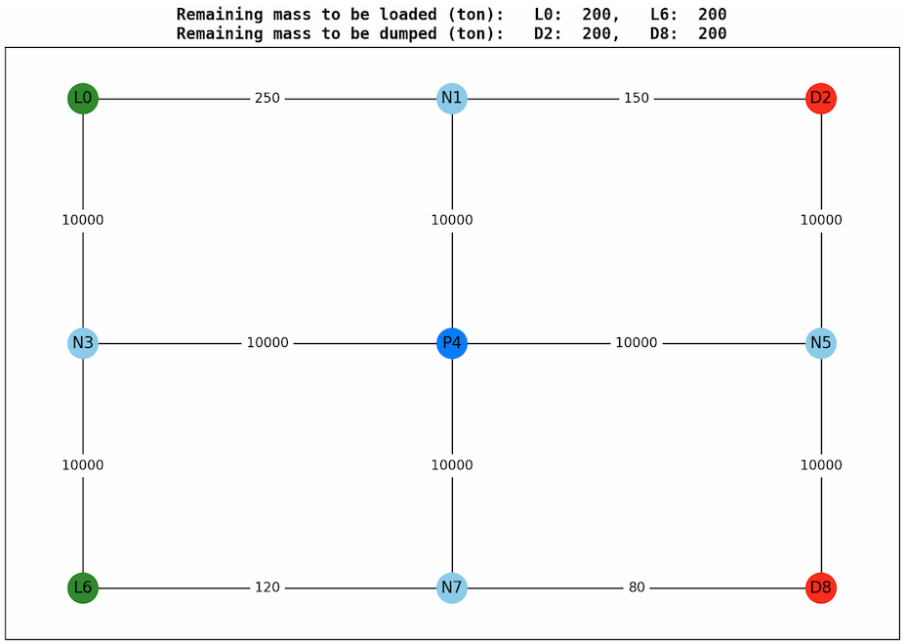
In Equation 4.1, F_{B2} and T_{B2} represent the fuel consumption and completion time of Baseline 2 (respectively). The variables F_{M} and T_{M} refer to the corresponding metrics of another method's plan.

4.2 Testing the Framework's Capability on Basic Scenarios

To verify the framework's ability to identify optimal solutions, we should first test it on simple scenarios with well-known optimal outcomes.

We used the map presented in Figure 4.1 to demonstrate such scenarios.

Figure 4.1: Simple Map



Scenario 1: The Benefits of the Coffee Break Agent

We are constructing a scenario using the map depicted in Figure 4.1, where the optimal solution is straightforward. First, we assign two dumpers to each loading node. Specifically, we let the loading time at node $L0$ equal the travel time from $L0$ to $D2$, plus the time to dump the mass at $D2$ and return to $L0$. Similarly, we let the loading time at node $L6$ equal the travel time from $L6$ to $D8$, plus the time to dump the mass at $D8$ and return to $L6$. As a result, after the initial loading at each node, dumpers traveling *horizontally* across the map will encounter no waiting time. Furthermore, to test the capabilities of the Coffee Break Agent, we disabled the parking option for the Loading Agent.

To achieve the desired outcome, we applied a loading rate of 4 s/ton at $L0$ and 2 s/ton at $L6$. The task involves transporting a total of 200 tons of mass from both loading nodes and evenly distributing it among the two dumping nodes. Consequently, dumpers on the $L6 - D8$ route will finish earlier than those on the $L0 - D2$ route. Due to the significant distance between these routes, other alternatives are not beneficial. A proficient Coffee Break Agent should recognize that dumpers on the $L6 - D8$ route should take a temporary break until the episode ends.

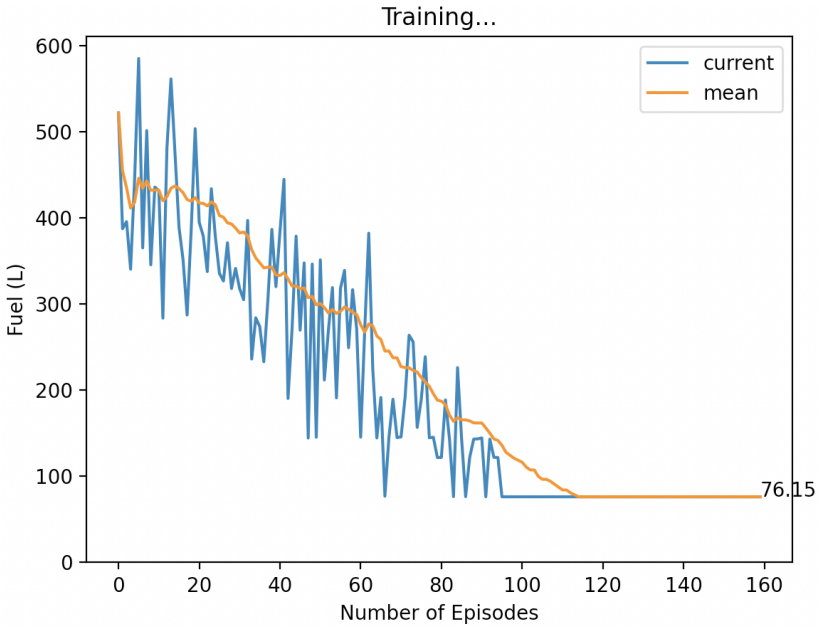


Figure 4.2: Fuel consumption while training the agents on Scenario 1.

In Figure 4.2, the agents' performance is assessed based on fuel consumption per episode. The framework has successfully converged to a solution to the problem. The training process involved setting the exploration number, as described in Section 3.5, to 100 for both the planning agents and the Coffee Break Agent.

Listing 4.1 provides the schedule for each dumper for the converged solution.

Listing 4.1: Time schedule of the converged solution obtained in Figure 4.2

```
          Dumper num: 0              |            Dumper num: 1
      ---------------------          |        ---------------------
Time:                    Plan:       |   Time:                    Plan:
[00:00:00, 00:03:10]     L0 => L0    |   [00:00:00, 00:01:51]     L6 => L6
[00:03:10, 00:05:00]     L0 => D2    |   [00:01:51, 00:03:01]     L6 => D8
[00:05:00, 00:09:32]     D2 => L0    |   [00:03:01, 00:05:33]     D8 => L6
[00:09:32, 00:11:22]     L0 => D2    |   [00:05:33, 00:06:43]     L6 => D8
[00:11:22, 00:15:54]     D2 => L0    |   [00:06:43, 00:09:15]     D8 => L6
[00:15:54, 00:17:44]     L0 => D2    |   [00:09:15, 00:10:25]     L6 => D8
-------------------------------------------------------------------------

          Dumper num: 2              |            Dumper num: 3
      ---------------------          |        ---------------------
Time:                    Plan:       |   Time:                    Plan:
[00:00:00, 00:06:21]     L0 => L0    |   [00:00:00, 00:03:42]     L6 => L6
[00:06:21, 00:08:11]     L0 => D2    |   [00:03:42, 00:04:52]     L6 => D8
[00:08:11, 00:12:43]     D2 => L0    |   [00:04:52, 00:07:24]     D8 => L6
[00:12:43, 00:14:33]     L0 => D2    |   [00:07:24, 00:08:34]     L6 => D8
```

Listing 4.2 highlights the Coffee Break Agent's critical role in optimizing coordination. Near the end of the episode, the Loading Agent must choose loading node $L0$ due to job completion of $L6$. Dumper 3 would receive this plan as it is first in line for the final loading plan. However, the Coffee Break Agent intervenes and identifies that a dumper positioned on the $L0 - D2$ route is better for this task. As a result, the scheduling is improved by letting Dumper 1 and 3 take temporary breaks instead of executing the plan.

Listing 4.2: Coffee Break Agent's actions in Listing 4.1

```
          Dumper num: 0          |          Dumper num: 1
 ---------------------           |  ---------------------
00:05:00 - FOLLOW PLAN           |  00:00:00 - FOLLOW PLAN
00:11:22 - FOLLOW PLAN           |  00:03:01 - FOLLOW PLAN
                                 |  00:06:43 - FOLLOW PLAN
                                 |  00:10:25 - COFFEE BREAK
-------------------------------------------------------------------------

          Dumper num: 2          |          Dumper num: 3
 ---------------------           |  ---------------------
00:00:00 - FOLLOW PLAN           |  00:00:00 - FOLLOW PLAN
00:08:11 - FOLLOW PLAN           |  00:04:52 - FOLLOW PLAN
                                 |  00:08:34 - COFFEE BREAK
                                 |  00:09:16 - COFFEE BREAK
                                 |  00:09:33 - COFFEE BREAK
                                 |  00:10:26 - COFFEE BREAK
```

Scenario 2: The Benefits of the Parking Option

In this scenario, we want to test a situation where parking is more advantageous for dumpers than being a part of the environment. This advantage might occur if the queues are too long or the parking node is nearby. To test this scenario, we will use the map shown in Figure 4.1 and the same configurations as in Scenario 1. However, we will change the initial mapping of the dumpers to the parking node instead of the loading nodes and enable the Loading Agent to direct dumpers to the parking spot. Furthermore, we will deactivate the Coffee Break Agent, as it performs a similar function to the parking action.

The optimal schedule for this setup involves having two dumpers, each driving in opposite directions. Specifically, one dumper should follow the $L0 - D2$ route while the other takes the $L6 - D8$ route. This arrangement is optimal because the two routes are significantly distant from the parking node, and the number of jobs on each route is relatively low.

As in Scenario 1, the loader at $L6$ will complete its job before the loader at $L0$. After the dumper on the $L6 - D8$ route completes the jobs assigned to those nodes, it would be optimal for this dumper to park.

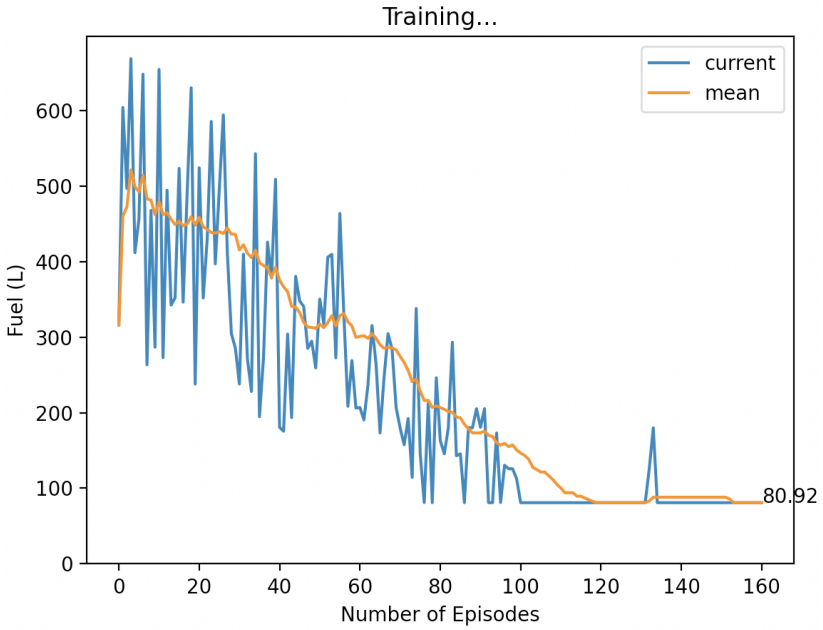


Figure 4.3: Fuel consumption while training the agents on Scenario 2.

The agents' performance was evaluated based on fuel consumption per episode, as shown in Figure 4.3. The exploration number for the planning agents is 100.

The framework converged to a solution but not to the optimal one. The top plans achieved while training the agents on Scenario 2 are shown in Table 4.1.

Table 4.1: Performance of top plans for Scenario 2

Score	Rank	Plan	Fuel	Cost	Time	Distance	L idle	D idle	D
93.8	332	96	81.1	2482	5900	46.00	1192	0	2
85.9	110	79	80.9	2476	10974	45.87	1192	0	1
85.8	220	77	81.0	2479	10986	45.93	1192	0	1
75.1	441	100	112.4	3347	5145	66.00	436	6	3
61.0	550	74	114.4	3438	14960	65.80	1192	0	1
60.1	667	99	125.8	3916	9454	65.87	4745	0	2
60.1	778	98	125.8	3916	9455	65.87	4747	0	2
57.6	880	97	130.5	4112	10106	65.93	6197	0	2
50.8	003	72	157.5	4800	8954	85.87	4135	0	3
50.2	990	85	143.0	4648	14960	65.80	10252	0	1
43.9	004	62	190.4	5741	9019	105.87	3875	115	4
34.9	005	50	259.3	7739	9293	145.73	4492	2	4
33.8	006	33	271.2	8248	9379	145.73	8185	193	4
31.8	009	49	294.6	8777	9539	165.66	5063	2	4

Score (see Section 4.1), **D**: dumpers (required amount).
Rank XYZ: Plan ranking based on Fuel (Xth), Cost (Yth), Time (Zth). A '0' indicates that the plan did not rank in the top 9 for any categories.
Plan (Game number), **Fuel** (liter), **Cost** (NOK), **Time** (seconds), **Distance** (kilometer).
L idle: Loaders idling (seconds), **D idle**: Dumpers idling (seconds).

Table 4.1 shows that the framework converged to a plan with second place in scores, despite ranking first in fuel and cost. This converged solution is plan number 79 in the table. According to the scoring system (see Section 4.1), which tries to find the most suitable plan balancing several factors, the best plan is the one we expected to be optimal. Our framework successfully found this solution as plan number 96.

The converged plan uses a single dumper to complete all jobs within the environment, starting with the jobs at the route $L6 - D8$. After completing all jobs at $L6$, the dumper follows the path $[D8, N7, L6, N3, L0]$. This approach results in the dumper traversing the edge connecting $(L0, N1)$ one less time and the edge connecting $(N7, L6)$ one additional time compared to the optimal solution using two dumpers. As a result, the total distance driven is slightly shorter than for the optimal plan. Consequently, this plan has slightly lower fuel consumption and cost than the optimal solution but takes about twice as much time.

4.3 Framework Evaluation: Analysis on Multiple Maps

This section presents a comparative analysis of our framework's performance against Baseline 1. For each method, we identify the *best plan* for a given scenario. We define the best plan as having the highest score according to the scoring system outlined in Section 4.1. First, we provide an overview of the maps used in our analysis. Finally, we examine the metrics obtained from applying the methods to multiple maps.

The Maps Structures: Similarities and Differences

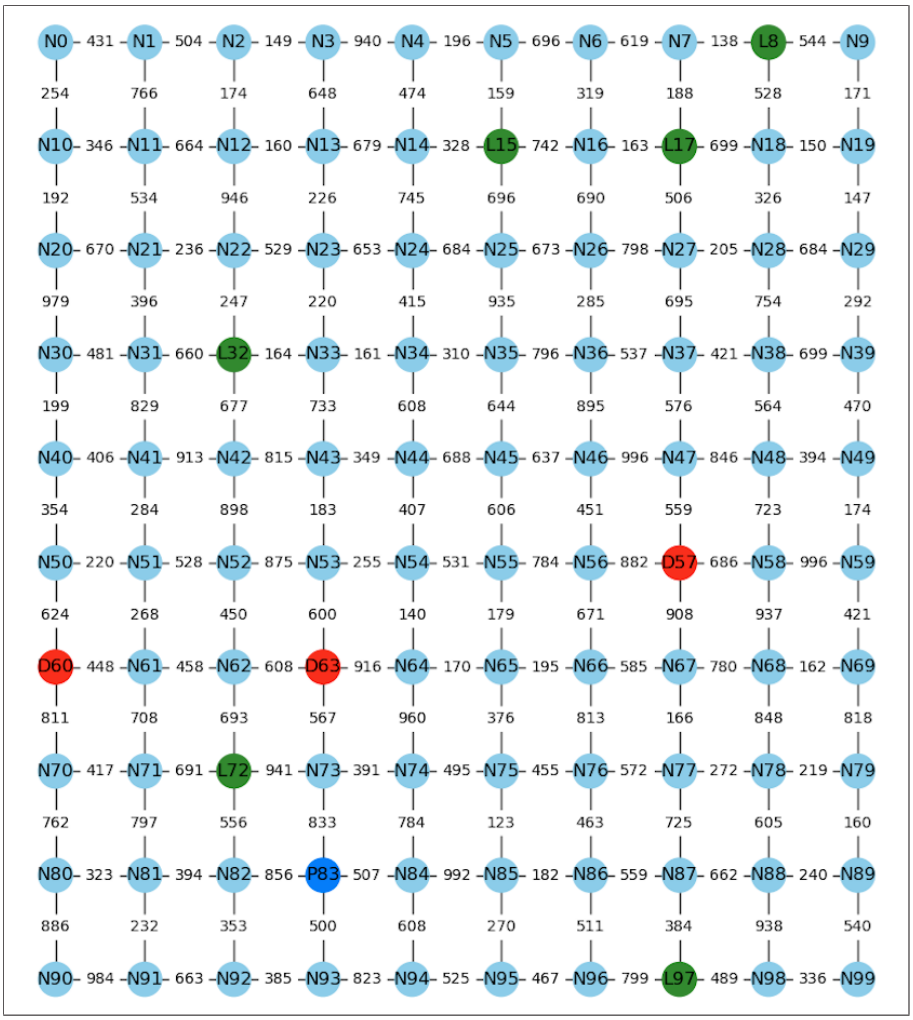
In our study, we represent road construction sites as undirected weighted graphs on a 10×10 grid. Each scenario in the study includes one parking node, six loading nodes, and three dumping nodes. The job involves transporting 400 tons of mass from each loading node and distributing it evenly among the three dumping nodes, resulting in 800 tons of mass across all dumping nodes. We assume the materials to be homogeneous, meaning dumpers can transport all mass to any dumping node. This assumption of homogeneous materials is actually more challenging for our framework since the number of possible solutions increases.

We use a random and independent uniform distribution over the interval $[100, 1000]$ to assign edge weights, representing the distance in meters between nodes.

Our analysis considers scenarios where we can deploy up to 25 dumpers onto the road construction site. However, our framework may find that using only some of the available dumpers is more efficient in finding the optimal plan.

Figure 4.4 (Map 1 in the study) illustrates the structure of the maps in our research. In our analysis, we maintain the general structure of each map while varying the edge weights and the placement of the special nodes (parking, loading, and dumping) across different maps. In other words, the edge weights and special node placements remain constant for each map but differ between maps.

Figure 4.4: Map 1



Jobs: loading nodes: 400 ton, dumping nodes: 800 ton
Loading nodes: L8, L15, L17, L32, L72, L97
Dumping nodes: D57, D60, D63
Parking node: P83

Performance Analysis of Map 1: A Detailed Presentation

Our initial map, Map 1, as illustrated in Figure 4.4, serves as the primary training ground for our agents. Therefore, we will thoroughly explain the procedure to achieve the best plans and contrast our findings with those of the two baselines, as discussed in Section 4.1.

In this scenario, Baseline 2, also known as *The Lower Bound*, provides a fuel consumption of 483.0 (liter) and a completion time of 4140 seconds.

To establish the performance of Baseline 1, we will evaluate its performance with different numbers of dumpers as deploying fewer dumpers could yield better plans.

Table 4.2 presents the results of Baseline 1 on Map 1, indicating that employing too few dumpers will result in significant idle time for the loaders. Conversely, deploying too many dumpers will increase the overall distance traveled. Therefore, to find the plan with the lowest fuel consumption for Baseline 1, it is necessary to strike a balance between the fuel consumption from the distance traveled with the loaders' idling time. Specifically, 14 dumpers would offer the plan with minimum fuel consumption for Baseline 1, as seen in Table 4.2.

Table 4.2: Baseline 1: Map 1

D	Score	Fuel (L)	Time (s)	Distance (km)	Idling Loader (s)
10	66.1	651.9	12207	333	29150
11	69.4	640.6	9141	337	23604
12	71.6	632.2	7939	339	19496
13	71.7	631.9	7939	342	17898
14	71.9	629.4	7939	343	16408
15	72.2	635.9	7287	350	14933
16	72.6	642.8	6661	357	13589
17	72.8	644.7	6464	360	12252
18	72.7	646.0	6464	363	11202
19	72.6	646.9	6464	366	9984
20	72.7	645.6	6464	367	8874
21	73.2	653.3	5942	374	7779
22	74.0	661.4	5369	380	6815
23	73.7	664.6	5369	384	5858
24	73.5	667.0	5369	387	5188
25	73.3	669.1	5369	390	4350

Measuring the performance done by Baseline 1 on Map 1, given the number of dumpers (**D**) available. We calculate **Score** as described in Section 4.1.

Considering fuel consumption and completion time is essential to ensure an efficient and sustainable plan. Therefore, we use the scoring system presented in Section 4.1. Based on the results presented in Table 4.2, the plan with 22 dumpers scores the highest.

Our framework requires training the agents to generate effective plans.

In Map 1, we train the agents from scratch, while pre-trained agents are used in the remaining maps to achieve better performance in earlier episodes. To train our agents, we set the exploration vector to $(20, 10, 10)$ (described in Section 3.5), indicating that we explore more than we rely solely on our policy. We trained the agents by running 360 episodes, which took slightly longer than 2 minutes to complete on the laptop [APP20]. After training, we use the exploration vector $(5, 20, 20)$ and run 510 episodes because the agents can start discovering optimal plans immediately. This exploration vector indicates that we rely more on our policy while exploring less frequently. This process took slightly longer than 3 minutes to complete on the laptop [APP20].

In Map 1, the best plan generated by our framework achieved a score of 75.3, with a fuel consumption of 617 liters and a completion time of 6575 seconds. Further details about the results for Map 1 are available in Appendix A.5. Table A.2 presents metrics for the highest-scoring plans. Furthermore, the schedules

for both loaders and dumpers in the best plan for Map 1 are provided in Listing A.1 and A.2, respectively.

On Map 1, our framework found a schedule that consumes 44 liters less than Baseline 1. This decrease in fuel consumption comes at the cost of an additional 20 minutes of time. However, our scoring system favors this trade-off.

Performance Analysis of All Maps: An Overview and Comparison

We will perform the same analysis on 20 maps with similar structures as we did with Map 1. All maps will use the pre-trained agents obtained from Map 1 and an exploration vector of $(5, 20, 20)$ to run 510 episodes. We will record metrics of the best plan and the plan with the minimum fuel for our framework and the two baselines.

Table 4.3 presents the performance of our framework on 20 maps and compares it to the two baselines. We focus on the results indicated by b, representing the best plan on the map.

Our framework achieved an average score of 72.9, compared to Baseline 1's 69.8, outperforming the baseline in 75% of cases. While the score itself is challenging to interpret, it indicates that our framework outperforms Baseline 1 on average.

The framework consistently provided plans that consumed less fuel across all maps analyzed, resulting in an average reduction in fuel consumption of 10% compared to Baseline 1. However, this reduction came at the cost of a 23% increase in time.

Two factors contribute to this increase. First, Baseline 1 initially sends dumpers to all loading nodes, while our framework may not use all loading nodes after a while in an episode. Second, our framework found deploying an average of 20.6 dumpers effective, while Baseline 1 deployed on average 2.5 more dumpers.

Thus, this time increase results in lower fuel consumption and fewer dumpers to complete the jobs. Our scoring system favors this trade-off between fuel consumption and time. By adjusting the weighting in the scoring system, we may achieve a smaller increase in time at the cost of reduced fuel savings.

Table 4.3: Performance on several maps

Map	**Framework**				**Baseline 1**				**Baseline 2**	
	D	Fuel	Time	Score	D	Fuel	Time	Score	Fuel	Time
1-a	22	617	6575	75.3	22	661	5369	74.0	483	4170
1-b	22	613	7072	74.8	14	629	7939	71.9	483	4170
2-a	25	425	4230	66.4	21	438	4068	65.6	265	3504
2-b	16	400	6349	64.1	14	423	5403	63.1	265	3504
3-a	17	744	7476	63.2	25	925	6104	55.3	488	3996
3-b	14	719	9130	63.1	14	884	9364	52.7	488	3996
4-a	25	673	6913	61.5	25	721	5770	60.3	423	3873
4-b	25	669	7221	61.3	25	721	5770	60.3	423	3873
5-a	21	474	6117	68.1	25	520	4076	68.7	337	3447
5-b	18	471	6644	67.6	14	494	6603	65.0	337	3447
6-a	21	454	5431	75.6	25	543	4553	68.0	351	3710
6-b	15	450	6629	73.5	20	539	5540	65.5	351	3710
7-a	18	543	5947	84.9	25	610	4392	81.3	492	3681
7-b	14	538	6962	83.7	13	582	6803	78.4	492	3681
8-a	25	538	6090	74.3	23	584	5155	71.8	412	3962
8-b	25	538	6090	74.3	15	571	8680	66.9	412	3962
9-a	25	490	5156	77.4	25	502	4047	79.5	395	3344
9-b	25	490	5156	77.4	21	500	4736	77.3	395	3344
10-a	11	437	7374	66.9	23	534	4374	64.0	308	3899
10-b	11	436	7727	66.6	11	473	7287	62.8	308	3899
11-a	14	650	8699	80.4	21	754	6517	74.0	573	4317
11-b	12	641	10883	79.5	12	719	10028	72.4	573	4317
12-a	25	659	6380	80.7	25	702	5585	78.5	559	4123
12-b	25	659	6380	80.7	20	700	6828	76.0	559	4123
13-a	10	413	8055	71.8	9	447	8342	66.9	316	4284
13-b	10	413	8055	71.8	9	447	8342	66.9	316	4284
14-a	25	450	4927	75.7	25	506	4524	70.0	352	3228
14-b	25	448	5478	74.6	25	506	4524	70.0	352	3228
15-a	14	424	6157	83.3	23	576	4340	69.5	377	3716
15-b	14	424	6157	83.3	10	525	8696	65.9	377	3716
16-a	13	525	8012	60.7	24	593	4477	62.8	333	3989
16-b	13	523	8810	60.0	11	531	8336	59.7	333	3989
17-a	25	719	5881	68.9	22	779	6124	64.1	499	3929
17-b	19	703	6626	68.7	16	777	7892	61.3	499	3929
18-a	25	475	5088	71.7	24	484	3888	74.6	349	3283
18-b	25	470	5739	70.9	20	481	4655	72.2	349	3283
19-a	25	499	4713	65.3	22	505	4320	65.9	320	3284
19-b	15	472	6542	64.3	13	492	6511	62.1	320	3284
20-a	25	584	5437	85.1	25	620	5206	81.7	515	3950
20-b	25	584	5437	85.1	25	620	5206	81.7	515	3950

D represents the number of dumpers, and Equation 4.1 calculates the **Score**. We measure **Fuel** and **Time** in liters and seconds (respectively). Indicator Map-a provides the plan with the highest score, while Map-b represents the plan with minimal fuel.

CHAPTER 5

Discussion

We have presented a multi-agent reinforcement learning framework for planning the coordination of dumpers on a road construction site. Our framework includes agents responsible for route planning, navigation, and decision-making concerning directing dumpers to a parking spot. We used Q-learning to train our agents to learn optimal policies. We evaluated our approach through experiments on various problem instances and compared its performance with two baselines. Our results demonstrate that our approach produces, on average, more efficient plans than a rule-based method derived from information on current construction site planning practices. The information originates from interviews with foremen, site managers, and the planning department in the project *Data driven road construction sites* [SIN20].

In this chapter, we discuss potential shortcomings of the developed framework and explore ways to overcome them.

Exploring Alternative Agent Architectures and Parameters

We used a zero gamma value for Q-learning in our approach, meaning our agents only considered the immediate reward in each state rather than the long-term return. This approach may result in suboptimal plans that are locally greedy at the expense of global efficiency. To improve this, we could use a positive gamma and ensure a more concise state transition function for each agent. Using a positive gamma would allow our agents to learn to balance the trade-off between local and global rewards by considering the future consequences of their actions. However, using a positive gamma may also introduce some challenges, such as dealing with delayed rewards and tuning the gamma value for different scenarios.

This thesis excludes experiments with different gamma values for two reasons: inconsistent state transitions and time constraints. The setup with multiple agents controlling all dumpers causes inconsistency in state transitions. When an agent performs the same action in a given state, the resulting state transition can vary significantly. The next state depends not only on the previous state and the action but also on the actions taken by other agents and the locations of the dumpers in the environment. This inconsistency violates the Markov chain property [SB18], which assumes that the next state depends solely on the current state and action. Violating the Markov chain property can be problematic because it may lead to suboptimal decision-making, as the agent's

understanding of the environment may be incorrect. Consequently, a positive gamma may not be applicable in our scenario.

Another improvement would be integrating the Coffee Break Agent into the Loading Agent since it only evaluates its plan. We did not prioritize this due to time constraints. Integration may improve computational time because we could rely on one neural network instead of running two networks whenever the Loading Agent issues a plan. The challenge would be to add more information to the state of the Loading Agent, but this could have both positive and negative effects. This information could make the state transition more consistent but may also add irrelevant details to the loading plans.

In the first draft of our framework, we trained the Node Agents. However, these agents did not always find the optimal route for a dumper's next destination. To address this issue, we utilized a method for solving the all-pairs shortest path problem to provide the Node Agents with information about the edge leading to the shortest path. We did not test the ability of Node Agents to train in situations where the environment changes continuously during an episode, such as when roads open and close. Implementing this functionality would likely have required significant additional coding time, so we decided to defer it for the sake of this thesis. Additionally, since an all-pairs shortest path algorithm can quickly update changes in the environment for the sizes of graphs we consider, it was not necessary to implement this functionality into the environment.

Although hyperparameter optimization is a common task for neural networks, we found it unnecessary for this thesis. We performed some manual hyperparameter optimization, but the neural network did not limit the framework's performance, so a full systematic optimization was unnecessary.

A Smarter Approach for Plan Comparison

In this thesis, the framework underwent evaluation using Baseline 1 (see Section 4.1), which generated plans based on specified logic. Visual inspection revealed that the plans generated by Baseline 1 appeared reasonable.

As described in Chapter 1, similar functionality to the presented framework was developed in the project [SIN20]. However, the frameworks were not directly comparable due to practical reasons. The software used in SINTEF's project relies on proprietary code, which prevented us from running it locally and feeding in comparable graphs during the project time.

In addition, comparing the framework's plans to actual road construction site operations could provide valuable insight into its effectiveness. Moreover, verifying the framework's performance on real maps could help us understand if it works better on such maps than the maps used in our analysis. It would also be beneficial to test the framework on various map structures to determine whether it prefers one over another.

If additional time were available, we would have sought an alternative method for comparison, such as finding a similar framework or conducting a more comprehensive assessment of its effectiveness against real-world road construction site operations. This would have allowed us to better understand the strengths and limitations of our framework in relation to other approaches.

The framework generates plans accompanied by metrics that measure their success. We base our plan selection on the scoring system discussed in Section

4.1. Selecting the optimal plan for a construction site could benefit from a thorough understanding of the priorities and goals of the construction site management team. By taking these factors into account, the plan selection process could be more effective in meeting the needs of the project.

Applying Our Framework to Real-World Scenarios

The proposed framework has certain restrictions that may limit its applicability in real-world scenarios. One such restriction is the availability of only one parking node. Moreover, for optimal use of the framework, all dumpers must start on this parking node at the beginning of the planning phase. However, this restriction may not always be feasible in real-world scenarios where dumpers may be distributed across multiple locations.

To address this issue, one potential solution would be to map dumpers to the nearest node on the map during the starting phase. Although additional coding can quickly address this issue, we did not include it in this thesis.

Additionally, we could have implemented the possibility for dumpers to start with materials on their vehicle. However, we deemed this feature unnecessary for the scope of this thesis.

In a real-world scenario, decision-makers responsible for the coordination of dumpers can use the proposed framework to improve planning. They can compare the plan provided by the framework with their original plan using metrics. If the metrics indicate that the framework's plan is superior, decision-makers can choose to use it. If not, they can retain their original plan.

One significant advantage of our framework is that it has a negligible training time and can generate informative plans in just a few minutes. This short training time means that decision-makers can quickly generate new plans based on environmental changes without incurring a high computational cost.

If computational power permits, decision-makers should initialize the framework several times with random initialization of the weights and biases of the neural networks. This approach introduces randomness and can result in the framework generating different best plans. Additionally, experimenting with other hyperparameters, such as parameters in the agents' reward function, Q-learning parameters, and neural network architecture, can be beneficial.

Running multiple instances of the framework with different hyperparameters and introducing randomness is similar to starting with a new solution in a combinatorial optimization algorithm. For instance, in greedy search, when it has exhausted all possible improvements, it generates a new random solution and starts over. Ultimately, the algorithm selects the best solution from all its visited solutions. Similarly, by running multiple instances of the framework with different hyperparameters and introducing randomness, decision-makers can generate a range of plans and select the best one based on their evaluation criteria.

PART II

Fuel Model

CHAPTER 6

Towards more advanced route planning

Developing a fuel model is crucial to improve the accuracy of evaluating plans generated by the reinforcement learning framework. Currently, the framework estimates fuel consumption based on pre-defined fuel consumption averages. This approach does not account for variations in fuel consumption due to factors such as vehicle load, terrain, and driving behavior, leading to inaccurate estimates and suboptimal planning. Instead, a more precise approach would be to create individualized fuel models for each dumper that can be updated in real-time as the vehicle collects more data while driving. These fuel models could provide a more accurate estimate of fuel consumption based on the specific characteristics of each dumper and its operating conditions.

Most currently used models are based on outdated and long-term aggregated data [Str01]. To circumvent this, we use a small sample of high-resolution fuel data combined with GPS data to create a data-driven model with the XGBoost framework [CG16].

XGBoost is a powerful machine learning algorithm that combines multiple decision trees [Qui86] to improve predictive accuracy. By leveraging the strengths of decision trees, XGBoost can achieve high performance on a wide range of tasks. One advantage of XGBoost over neural networks is its faster training time, making it a more efficient choice for certain applications. As decision trees serve as the building blocks of the XGBoost algorithm, we include a brief overview of their functionality in Appendix B.1.

We selected the XGBoost framework for its excellent performance on tabular data. [SA21] systematically compares various models' performance and computational efficiency when applied to tabular data. They found that XGBoost generally outperforms deep learning models in terms of performance and requires less tuning, which is practical for real-world applications. In terms of overall performance, the best results were achieved using an ensemble of deep-learning models and XGBoost. However, due to the limited data sample size, we decided to apply XGBoost alone.

CHAPTER 7

Theory

7.1 A Brief Overview of How to Evaluate Machine Learning Models

Train, test, validation

When training and evaluating a machine learning model, splitting the data into three sets: training, validation, and test [XG18] is common practice. The training set trains the model, while the validation set tunes its hyperparameters and selects its structure. Model selection involves finding the model that gives the lowest validation error. After identifying the best model, we apply it to the unseen test set to estimate its predictive performance. It is important to note that the test set estimates the model's performance but does not guarantee its performance on real-world data, which may come from a different distribution.

Cross-validation

K-fold cross-validation is a popular method for evaluating the performance of a machine learning model [Koh95]. The basic idea is to divide the data into K mutually exclusive subsets of approximately equal size. The model is then trained and evaluated K times, with each fold serving as the test set once and the remaining K-1 folds as the training set. This method evaluates the model across multiple data sets and helps determine how well the input data describes the target variable. Additionally, k-fold cross-validation can provide valuable insights into the data by revealing consistent trends and relationships across multiple data sets. Moreover, it can help identify potential issues or biases affecting the model's accuracy.

7.2 Understanding Model Interpretability: Exploring SHAP and SAGE Values

SHAP [LL17] and SAGE [CLL20] are two methods to interpret machine learning models and are based on the Shapley values from game theory [Sha53]. Ian Covert, the author of the papers on SHAP and SAGE, overviews the concepts here [Cov20]. This thesis will calculate these values using the PyPI packages shap [Lun] and sage-importance [Cov].

SHAP is a framework that explains how each feature contributes to a model's prediction for a single sample, allowing for *local* interpretation. SHAP values can be used for this purpose regardless of how well the model describes the data. SAGE values represent how much a model depends on each feature over the whole data set, allowing for a *global* interpretation.

Shapley values

Shapley values provide a fair method for distributing rewards among players in a cooperative game. Let us say we have a set of players $C = \{1, 2, ..., c\}$ and a profitability function:

$$\pi : \mathcal{P}(C) \to \mathbb{R} \tag{7.1}$$

that calculates the total reward using only a subset of players $S \subseteq C$. Here $\mathcal{P}(C)$ represents the power set of C, containing all 2^c possible subsets of C.

In order to distribute rewards fairly among the players, we must establish certain properties. Let us denote the reward for each player $1, 2, ..., c$ as $\phi_1(\pi), \phi_2(\pi), ..., \phi_c(\pi)$, then the properties are as follows:

1. (efficiency) $\sum_{i=1}^{c} \phi_i(\pi) = \pi(C) - \pi(\emptyset)$. We should distribute all rewards generated by the players.
2. (symmetry) For $i, j \in C$, if $\pi(S \cup \{i\}) = \pi(S \cup \{j\}) \quad \forall\ S \subseteq C$, then $\phi_i(\pi) = \phi_j(\pi)$. If two employees perform identically, they should receive the same reward.
3. (dummy) For $i \in C$, if $\pi(S \cup \{i\}) = \pi(S) \quad \forall\ S \subseteq C$, then $\phi_i(\pi) = 0$. A player should receive a reward of 0 if it does not contribute to any profitability increase.
4. (linearity) If $\pi = a_1\pi_1 + a_2\pi_2 + ... + a_k\pi_k$, then $\phi_i(\pi) = a_1\phi_i(\pi_1) + a_2\phi_i(\pi_2) + ... + a_k\phi_i(\pi_k)$. If the profitability function is a linear combination of sub-functions, the player's reward is also a combination of rewards from each sub-function.

The Shapley values are the unique way to satisfy all these properties. The method for distributing rewards is given by:

$$\phi_i(\pi) = \frac{1}{c} \sum_{S \subseteq C \setminus \{i\}} \binom{c-1}{|S|}^{-1} [\pi(S \cup \{i\}) - \pi(S)] \quad \text{for } i = 1, 2, ..., c \tag{7.2}$$

where $|S|$ is the number of players in the subset S.

SHAP (SHapley Additive exPlanations)

We use the SHAP method to explain the predictions made by a model f for a given set of input features $x = (x_1, x_2, ..., x_n)$. This method assigns a value ϕ_i to each feature x_i, where $i = 1, 2, ..., n$. The value ϕ_i represents the contribution of feature i in deviating the prediction from the average prediction $E_X[f(X)]$.

We introduce the cooperative game for making a prediction, where the input features x act as the players. We define the profitability function for a subset of features S as:

$$\pi_{f,x}^{\text{SHAP}}(S) = E_{X_{\bar{S}}}\left[f(X) \mid X_S = x_S\right] \tag{7.3}$$

where x_S is the known features and the remaining subset of features, denoted by $\bar{S}$, will be stochastic variables.

We can now apply the profitability function, Equation 7.3, to Equation 7.2 to determine the importance of each feature in making the prediction.

SAGE (Shapley Additive Global importancE)

To explain how much a model f relies on each feature over the entire data set, we use the SAGE values.

We introduce a cooperative game for making accurate predictions, where the input features x act as players. We define the profitability function for a subset of features S as:

$$\pi_{f}^{\text{SAGE}}(S) = -E_{XY}[L(E_{X_{\bar{S}}}[f(X) \mid X_S], Y)] \tag{7.4}$$

where X represents the entire input data set with the corresponding target labels Y. X_S is the known features in the inner expectation, while the remaining subset of features, denoted by $\bar{S}$, will be stochastic variables. L is a loss function describing the closeness of the model to the data, and the negative sign indicates that decreasing the loss increases profitability.

We can now apply the profitability function, Equation 7.4, to Equation 7.2 to determine the global importance of each feature in making predictions for the model.

CHAPTER 8

Data

The analyzed data comprises 12 working days of a single dumper [Vol]. Ditio collects the GPS data. At the same time, Volvo extracts fuel data directly from the CAN BUS on the dumper control system using an Aplicom unit.

Before using the data, we must preprocess the data (Section 8.2), including identifying and removing erroneous observations due to issues with the data logging unit. Additionally, resampling the data points is necessary to efficiently merge the two data sources (Section 8.3).

Finally, we partition the data and create summary statistics (Section 8.4) to train and evaluate the fuel prediction model.

8.1 Exploring the GPS and Fuel Data

GPS Data

We can utilize GPS data as a source of information to predict fuel consumption for a given route. The logging frequency of GPS data depends on the movement and activity of the vehicle. Typically, the logging frequency while driving is around two seconds. However, when the dumper is stationary, there will be longer gaps in the data logging. We consider the following features of the GPS data:

1. **Altitude [meters]**: height above sea level
2. **Timestamp [date and time]**: the time of the recorded data point.
3. **x, y [meters]**: coordinates, according to a given centering on the construction site, representing the dumper's location (placement of center is irrelevant). We use them to calculate distances between time points.
4. **Type [number]** : 0: Waiting to obtain mass, 1: Driving with mass, 2: Driving without mass
5. **Quantity [ton]**: the amount of mass on the vehicle. The quantity is 0 when the *Type* is not 1.
6. **Course [degrees]**: the angle at which the dumper is facing.

7. **HorizontalAccuracy, VerticalAccuracy** [**meters**]: a measure of the uncertainty of the measurement in the horizontal and vertical directions (respectively).

8. **load_x, load_y, dump_x, dump_y** [**meters**]: coordinates, according to a given centering on the site, indicating where the dumper loads and dumps the mass (placement of center needs to be the same as for the features **x** and **y**).

Fuel Data

The fuel data represent the fuel consumption rate at a specific time point. Typically, the logging frequency for the fuel data is around 5 seconds, although there may be some deviations. The fuel data contains the following noteworthy features:

1. **longitude, latitude** [**degrees**] two out of three values of the geographic coordinate system. We use these coordinates to determine the position of the dumpers, and they need to correspond to the GPS data to be trustable.

2. **unitId** [**number**]: the fuel tracker ID connected to the vehicle.

3. **data_Time** [**date and time**]: the time of the recorded data point.

4. **data_Fuel** [**Liter/ hour**]: the fuel consumption rate, measured in liters per hour.

8.2 Data Selection and Cleaning

The data set presents several complications, including time gaps, inaccurate fuel consumption measurements, and significant variations in altitude. During the filtering process, any data points that exhibit these issues in either the GPS or fuel data are marked as untrustworthy based on the analysis described below. As described in Section 8.4, we will partition the data into 30-second intervals to create summary statistics. For this reason, having frequent consecutive observations is important due to the short time range.

Time gaps

Time gaps occur in both data sets, and including them in the final merged data set would result in poor interpolation points. As described in Section 8.3, we will interpolate data points to achieve a second-by-second frequency of our data.

Time gaps are particularly problematic for the fuel data since, with the 30-second route statistics, we only use about six logged fuel values and interpolate the rest. Fewer values will lead to high uncertainty in the values. For the GPS data, a longer time gap would have some impact, but the effect of each missing point is limited since we measure continuous quantities without abrupt changes.

We should deem data with significant time gaps untrustworthy and retain data with time gaps below a certain threshold as trustworthy. An optimal

threshold for time gaps is five seconds based on the typical intervals observed. However, we implemented a limit of 10 seconds to preserve data as much as possible.

Poor measurements of fuel data

Poor measurements in the fuel data can manifest signatures in several ways. One is when the same fuel rate repeats several times. It is unrealistic for the fuel rate to remain constant over several consecutive time points, as various factors, such as the driver, terrain, or sensor noise, can affect the rate. When filtering the data, we will exclude data points with three or more similar consecutive fuel rate values and split the routes accordingly. Table B.1 in Appendix B.2 shows an example of such data points, where we marked the fuel data as failed.

The other problem is that the fuel rate can be 0 between non-zero (and possibly high) fuel rates, which is illogical. A fuel rate of 0 should refer to a turned-off dumper, so we cannot trust such values. Table B.2 in Appendix B.2 shows the problem. Even values lower than 2 L/h are not reliable when the dumper is driving since the fuel rate is around 2.3 - 3 when the dumper is standing still (shown in Table B.3, Appendix B.2).

One solution is to have a linear interpolation between the observation before and after a fuel rate observation below the threshold of 2, but this will again reduce the actual fuel rate usage in the statistics of a route. We will allow such linear interpolation if the interval is within our time gap threshold of 10 seconds. An alternative approach would be to consistently partition the data and eliminate any instances where the fuel rate falls below a threshold of 2. However, this method may result in a higher loss of data.

A final issue to consider is the occurrence of two consecutive fuel rate values below a threshold of 2. In such instances, we find it appropriate to exclude those data points and partition the data.

Large variation in altitude

The GPS data includes a feature that indicates the accuracy of the measurements in both the vertical and horizontal directions. Since we base our model on a fixed time interval, altitude is one of the most essential features. Moreover, significant oscillations in altitude would drastically affect the prediction. So, whenever we get an *VerticalAccuracy* above the threshold of 12 (in the data set, higher *VerticalAccuracy* results in noisy altitude), then we will mark those data points as untrustable. Table B.4 in Appendix B.2 illustrates such an example.

Figure 8.1 illustrates the fluctuation of the altitude over distance based on Table B.4. The figure also suggests how the altitude may appear with more accurate data collection. We considered smoothing the altitude but ultimately decided to exclude these instances.

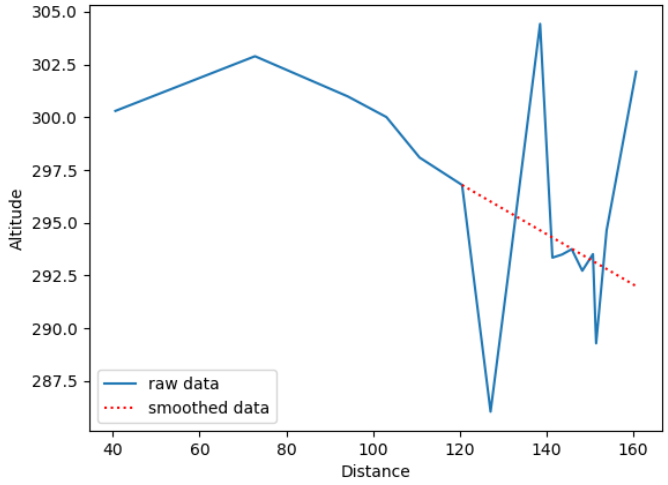


Figure 8.1: Altitude fluctuation over distance using data from Table B.4.

8.3 Merging Data

After filtering the data, we must merge the GPS and fuel data sets. To accomplish this, we convert both data sets into second-by-second frequency. This conversion results in a consistent time for all observations and makes the data easier to use. For categorical features, we will use the last known value and assign it to the subsequent time points until the next observed value occurs. We use linear interpolation to estimate missing values between existing observations for features varying over time.

8.4 Statistics of a Route

We will use the merged data set to create a predictive model. However, analyzing individual data points alone provides no insight. To extract more meaningful information, we will conduct statistics to summarize routes by calculating metrics such as distance and altitude changes. This approach will enable us to identify crucial patterns and trends for accurately predicting fuel values.

Our first challenge is determining the time interval size for performing the statistics. It could be dynamic, where the time intervals vary, or static, with a constant time interval for every route. Using dynamic time intervals would require more data, as the statistics could become too unique and difficult to generalize. On the other hand, using static time intervals avoids these problems but requires adapting the model's usage to work with varying time intervals.

For a static time approach, choosing the appropriate time interval size requires considering two main effects. Firstly, if the intervals are too small, we may need more logged fuel data points to base our statistics. Moreover, delays or noise in individual time points may result in less reliable statistics. Conversely, if the intervals are too large, detailed information in the data could be lost, as

some statistics calculate average values. Furthermore, larger intervals require more data points, which may reduce the total amount of reliable statistics.

After careful consideration, we determined that 30-second routes provide the minimum duration necessary to generate meaningful statistics. The routes will then contain six logged fuel rate values when the logging unit is operating normally.

We considered sampling the statistics from overlapping tracks but ultimately opted against it due to the risk of producing inaccurate, low-error estimates. Additionally, using multiple samples that contain a significant amount of noise could compromise our model's performance. Hence, we chose to exclusively use uniquely determined statistics to ensure the most reliable results.

Statistics

In the below statistics, we calculated *changes* as the increase/decrease between two consecutive data points. We summarized each route with the following statistics:

1. **LengthDistance** [**meters** (m)]: distance of the route.
2. **AltitudeGain** [m]: sum of all positive changes in the altitude .
3. **AltitudeLoss** [m]: sum of all the negative changes in the altitude.
4. **AltitudeDeltaEndStart** [m]: the difference in altitude from the start to the end of the route.
5. **AltitudeDeltaMaxMin** [m]: the difference between the max and min altitude recorded.
6. **AltitudeChange** [m]: **AltitudeGain** + | **AltitudeLoss** |
7. **SpeedMean** [m/s]: the average speed through the route. We calculated the mean speed as **LengthDistance**/30.
8. **Quantity** [**ton**]: the amount of mass on the vehicle for transportation.
9. **UpInclination** [**num**]: sum of all positive incline changes. We calculated the incline as ΔAltitude/ΔDistance between all consecutive data points.
10. **DownInclination** [**num**]: sum of all negative incline changes. We calculated the incline as ΔAltitude/ΔDistance between all consecutive data points.
11. **AccTime** [**s**]: the amount of seconds used for positive acceleration. Every second, where the increase is more than 1m/s.
12. **Fuel** [**L**]: the fuel consumption of the route in liter.

8.5 The Final Data Set

After completing our filtering and merging process, we generated a set of 30-second statistics (as shown in Section 8.4) for each day by randomly sampling non-overlapping routes. Table 8.1 shows the number of statistics achieved through sampling. We will use these statistics to train our fuel model.

Date	number of statistics
2021-09-28	53
2021-09-29	18
2021-10-13	33
2021-10-14	37
2021-10-15	18
2021-10-18	11
2021-10-19	56
2021-10-20	25
2021-10-21	23
2021-11-03	1
2021-11-04	0
2021-11-15	2

Table 8.1: Daily statistics overview, more details in Appendix B.3.

As shown in Table 8.1, there is limited data passing our filtering criteria for days after and including *2021-11-03*. Therefore, we will exclude those days from our analysis, as they may contain more noise than informative samples.

We will exclude the data from *2021-10-21* from our feature selection and use it to evaluate the final model. We will use cross-validation to evaluate the model using different features, as it is appropriate for our small data set. We split the data into folds for cross-validation, so each day is a fold. This approach will allow us to detect potentially unusual or atypical samples.

Using folds of different sizes for cross-validation when partitioning the data by date may introduce potential issues. A primary concern is the possibility of bias, as the model's performance can vary significantly on specific subsets of the data due to an over or under-representation of statistics in the training or test sets. However, despite this potential issue, we will proceed with this approach.

8.6 Investigation of New Features

Integrating additional features such as weather and road vibration into the data set and model could provide valuable insights, particularly regarding the impact of different road friction scenarios.

To ensure the model performs well under varying weather conditions, gathering data from several days with diverse weather patterns would be necessary. However, if we lack sufficient data on different weather situations, including weather as a feature may result in a less effective model.

Moreover, road vibration could serve as a significant feature of the model, as it differentiates between driving on a paved versus a dirt road.

CHAPTER 9

Method

9.1 Predictive Model for Fuel Consumption on a Route

The proposed approach for predicting fuel consumption for dumpers involves partitioning the route into 30-second intervals, as explained in Section 8.4. The statistical properties of each partition are calculated and used as inputs to a trained predictive model, which generates a fuel consumption prediction for each partition. The final estimated fuel consumption for the complete route is obtained by summing up all the predictions.

If the route length is not evenly divisible by 30 seconds, we handle the remaining portion like a regular 30-second partition. We then linearly scale the resulting prediction according to the length of the remainder.

This approach assumes that the statistical properties of the partition can sufficiently explain fuel consumption patterns within each 30-second partition. We must validate this assumption through further research to improve the predictions' accuracy.

As described in Section 8.5, we use the data from *2021-10-21* as the final test set. The final test set is first introduced in Table 9.6 to evaluate the last models.

9.2 Fine-tuning Models: Hyperparameter Tuning

Given a set of features, we divide the samples into training, testing, and validation sets (see Section 7.1). First, we exclude a test set containing data from a specific date, as explained in Section 8.5. Then, we split the remaining samples into a training and validation set using an 80-20 split.

We use the *Hyperopt* [BYC13] Python library to set the hyperparameters. Our goal is to select hyperparameters that generalize well to unseen data and achieve good performance on the test data by minimizing the validation error.

9.3 Feature Investigation for Improved Model Performance

To determine the most and least important features for predicting fuel consumption, we use SAGE values (see Section 7.2). Specifically, we will use the weighted SAGE values, where the weights are determined by the size of the data set being evaluated, over all the models developed by the cross-validation (Section 7.1).

By iteratively excluding the worst-performing feature, we can identify the best features and stop once the prediction error on the validation and test set significantly decreases. This process, known as feature selection, is commonly used in machine learning to improve model performance and simplify the model through fewer features.

Correlation matrix

First, we look at the correlation matrix of the data included in the data set.

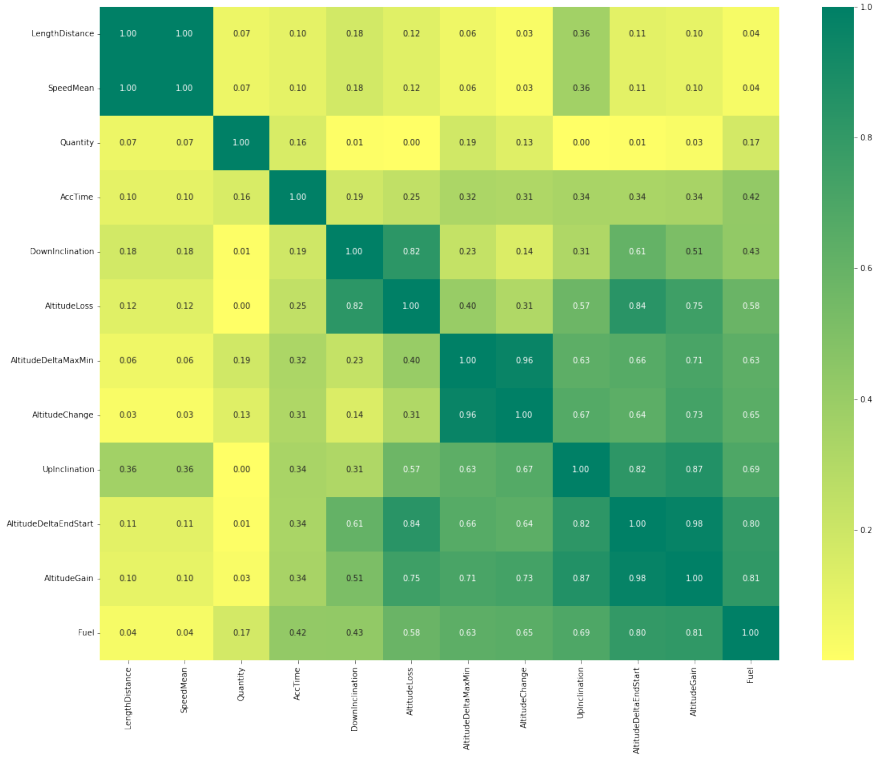


Figure 9.1: Correlation matrix (spearman) for features and the target in the data set.

The correlation matrix shown in Figure 9.1 provides insights into the direct relationship between the studied features. It is important to note that all samples/routes have the same duration, which implies that distance and mean speed are 100% correlated. Interestingly, distance does not correlate with fuel consumption in any significant way. This observation suggests that using distance as the only feature in the model would not be optimal, but combining it with other features may be more insightful.

The most correlated features with fuel are those related to altitude changes, particularly altitude gain, which makes sense.

Although many features are similar, some are inherently correlated due to their construction. Still, including them all in the feature selection process is necessary because the model may prefer one feature to another. However, we

will exclude distance from the process due to its 100% correlation with the mean speed of the route.

Feature selection

After conducting cross-validation and evaluating the importance of each feature to the model based on SAGE values (and also the in-built feature importance evaluation within XGBoost), we obtained the ranking of the different features shown in Table 9.1. We base the ranking on the weighted average of the SAGE values and feature importance from the cross-validation.

VAL	FEAT IMPORTANCE	(FEATURE)
0	0	AltitudeDeltaEndStart
1	1	AltitudeGain
2	2	AccTime
3	6	UpInclination
4	7	AltitudeChange
5	3	AltitudeDeltaMaxMin
6	5	Quantity
7	8	AltitudeLoss
8	4	SpeedMean
9	9	DownInclination

Table 9.1: Ranking of features in the first iteration of selection.

Table 9.2 shows the mean squared error for the train, test, and validation sets.

Date	Test	Val	Train
2021-09-28	0.00661	0.00531	0.00003
2021-09-29	0.00886	0.00566	0.00081
2021-10-13	0.00538	0.00800	0.00092
2021-10-14	0.00397	0.00481	0.00140
2021-10-15	0.00720	0.00465	0.00159
2021-10-18	0.00807	0.00648	0.00122
2021-10-19	0.01020	0.00481	0.00051
2021-10-20	0.00449	0.00305	0.00150

Table 9.2: Mean squared error for training, testing, and validation sets. The test data is from a date and refers to the cross-validation folds in this context.

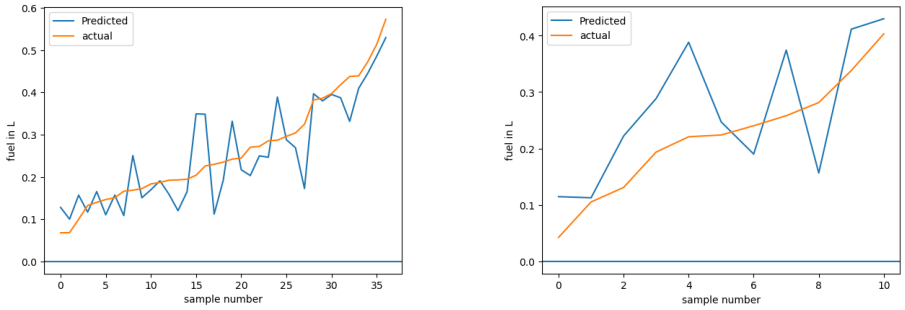
By analyzing the errors for all folds in the cross-validation, as shown in Table 9.2, it reveals that two test folds (dates *2021-10-13* and *2021-10-14*) exhibits a lower error than the validation data used for hyperparameter optimization. This lower error could potentially happen because the fold had well-recorded samples. These samples may better conform to the underlying distribution of the data, making them easier to predict.

We used an early stopping criterion. However, it is important to note that the model appears to overfit for the dates *2021-09-28* and *2021-10-19*. This is evident from the low training error and high testing error.

A detailed examination of individual predictions, presented in Table 9.3, reveals that some predictions appear reasonable based on their statistics but deviate significantly from the corresponding logged actual values. This

observation suggests that inaccurate measurements may contribute to a significant portion of the error variation. In addition, the number of samples included in each fold may also affect errors.

As seen in Figure 9.2, a visual representation of the predictions versus actual values highlights the model's performance on the best and worst days.



(a) test set: 2021-10-14 (b) test set: 2021-10-18

Figure 9.2: Prediction vs actual value, sorted by the *actual* target value

The results of our prediction model on the data from *2021-10-14* show a high degree of fit, as demonstrated in Figure 9.2a. While there are minor deviations from the target values, overall, the model accurately captures the data's dynamics. However, when we analyze the samples from *2021-10-18*, depicted in Figure 9.2b, we can see that they exhibit a trend, but many of the samples are poorly predicted. This observation suggests there may be errors or inaccuracies in the data, or the model may not be sufficiently robust to predict the samples accurately. Upon further examination of the complete data set, we can see that many of the samples that were predicted poorly actually make logical sense in the context of other samples. Let us investigate one of the worst predictions within the data of *2021-10-18*.

No	Date	Fuel	Distance	AltitudeDeltaEndStart	Prediction
Bad prediction					
113	2021-10-18 07:17:21	0.22	178	6.0	0.42
Similar to target fuel					
109	2021-10-13 17:00:20	0.21	198	-3.4	-
110	2021-10-19 17:21:55	0.21	130	-0.5	-
111	2021-10-19 18:02:20	0.22	110	2.5	-
112	2021-09-28 11:49:31	0.22	60	4.3	-
114	2021-09-29 06:20:40	0.22	139	-0.7	-
115	2021-10-18 12:32:53	0.22	59	2.0	-
116	2021-10-19 18:44:41	0.23	126	-3.8	-
117	2021-10-13 09:52:12	0.23	111	2.4	-
Similar to predicted fuel					
220	2021-10-19 14:55:12	0.42	80	5.4	-
221	2021-09-28 14:31:24	0.42	107	3.7	-
222	2021-10-14 14:57:26	0.42	115	4.6	-
223	2021-10-20 13:33:07	0.42	111	11.0	-

Table 9.3: Shows our prediction vs. similar samples. A part from Table B.5 in Appendix B.3

The data in Table 9.3 suggests that the target fuel value may be influenced by measurement errors or noise. This is evidenced by the poorly predicted sample being more similar to samples with fuel consumption of 0.42 rather than 0.22.

This observation suggests there may be errors in the logging unit on the dumper or elsewhere in the process. Further analysis and investigation are needed to ensure accurate measurement of fuel consumption.

Repeated feature selection

We can perform iterative feature selection to identify and select the most important features in our data set using the SAGE values for the validation data. In the first iteration, the least important feature overall was the *DownInclination* feature (as seen in Table 9.1). After repeating this process, we achieve Table 9.4. This table presents the weighted mean squared error of the validation data over all days given a set of features. The weighted mean squared error gives equal importance to all samples, regardless of the amount of data in each evaluation. We determine the weighting by comparing the number of samples for a specific day to the total number of samples across all days in the cross-validation.

Model	MSE	excluded feature (including those above)
model0	0.00528	-
model1	0.00522	DownInclination
model2	0.00517	SpeedMean
model3	0.00544	AltitudeLoss
model4	0.00559	AltitudeDeltaMaxMin
model5	0.00620	Quantity

Table 9.4: The models weighted error on validation data over all days.

Table 9.4 shows that the model error decreases in the beginning when we exclude features, reaching its minimum value at *model2*. However, the *SpeedMean* had high importance in the in-built feature importance in XGBoost, as shown in Table 9.1. To further check for improvement, we started with the features in *model1* and excluded the next worst-performing feature instead of *SpeedMean*. The resulting models' performances are listed in Table 9.5.

Model	MSE	excluded feature (including those above)
model0	0.00528	-
model1	0.00522	DownInclination
model6	0.00527	AltitudeLoss
model7	0.00539	AltitudeDeltaMaxMin
model8	0.00523	AltitudeChange

Table 9.5: The models weighted error on validation data over all days.

We can see that there is no better MSE in our second attempt to find a better combination of features, and the final model will then be based on *model2* with the following features: *AltitudeDeltaEndStart, AltitudeGain, AccTime, UpInclination, AltitudeChange, AltitudeDeltaMaxMin, Quantity, AltitudeLoss*.

Since each model in this table corresponds to a group of models, we will determine the overall best model by applying all the models to a completely

unseen final test data set from *2021-10-21* and selecting the model that performs best on that data (Table 9.6).

Although model averaging, which combines predictions from multiple models, is generally considered more accurate and robust, we will not use it in this case.

Test set	MSE of data 2021-10-21
2021-09-28	0.00513
2021-09-29	0.00396
2021-10-13	0.00623
2021-10-14	0.00515
2021-10-15	0.00375
2021-10-18	0.00519
2021-10-19	0.00502
2021-10-20	0.00527

Table 9.6: Results of the final test data set on the models developed with the final features

CHAPTER 10

Results and Discussion

Based on the analysis in Chapter 9, we developed the final model with the following features: *AltitudeDeltaEndStart, AltitudeGain, AccTime, UpInclination, AltitudeChange, AltitudeDeltaMaxMin, Quantity, AltitudeLoss*.

10.1 Measuring Model Performance

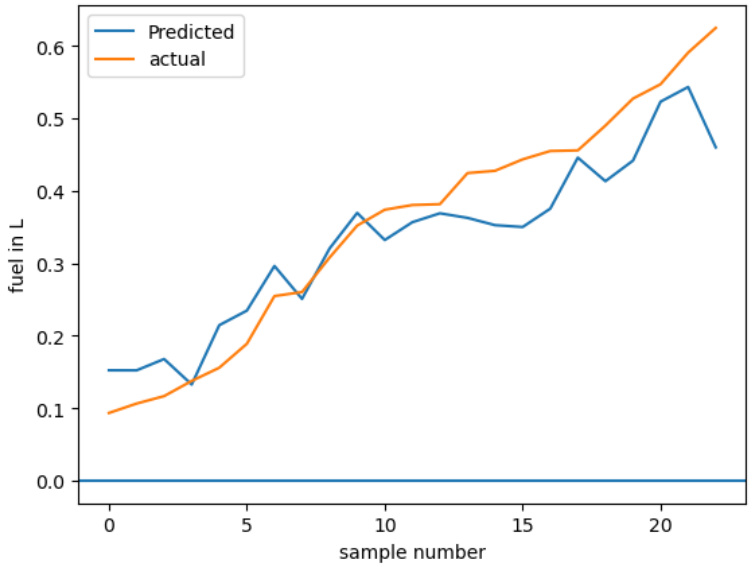
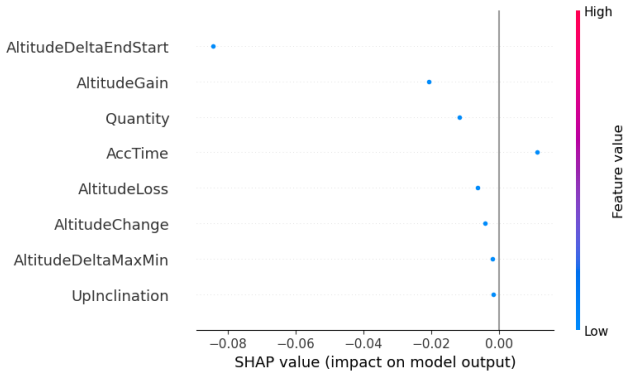


Figure 10.1: Prediction for unseen data from *2021-10-21*, sorted by actual target value.

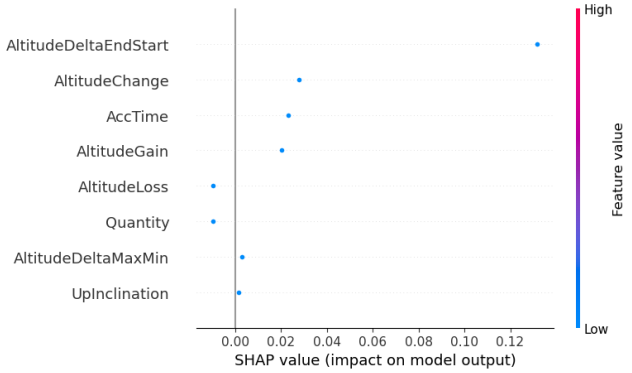
Figure 10.1 shows the final model's prediction on the unseen data. The predictions align well with the target values in this final unseen test data, indicating that the model is a good predictor of fuel consumption for routes lasting 30 seconds.

10.2 Analyzing Model Predictions

We can examine the SHAP values (see Section 7.2) for specific samples to understand how the model arrives at its predictions. Analyzing the SHAP values will give us insight into the specific features and how they contribute to the model's predictions. Figure 10.2 displays the SHAP values for two samples taken on *2021-10-21*. To provide additional context, the feature values for these samples are presented in Table 10.1.



(a) lowest fuel target (target: 0.093, prediction: 0.152)



(b) highest fuel target (target: 0.625, prediction: 0.460)

Figure 10.2: SHAP values for two samples from *2021-10-21* with final model

feature	lowest prediction sample	highest prediction sample
AltitudeDeltaEndStart	-5.3	10.5
AltitudeGain	0.0	10.5
Quantity	0	0
AccTime	18	14
AltitudeLoss	-5.3	0.0
AltitudeChange	5.3	10.5
AltitudeDeltaMaxMin	5.3	10.5
UpInclination	0.0	0.174

Table 10.1: The statistics of the samples in Figure 10.2

According to the results presented in Figure 10.2, the feature *AltitudeDeltaEnd-Start* has the greatest impact on the prediction's deviation from the average predicted value. This finding is consistent with our prior expectations, as variations in altitude are likely to affect fuel consumption significantly.

On the other hand, both *AltitudeDeltaMaxMin* and *UpInclination* have relatively small effects on the predictions.

It is worth noting that, in both examples in Table 10.1, the *Quantity* is 0, indicating no mass on the dumper. This indicates no mass on the dumper, contributing to a lower prediction than the average prediction.

10.3 Final Considerations: Key Takeaways and Insights

Figure 10.1 shows that the highest target value from *2021-10-21* was 0.625, while the prediction was 0.460. This deviation may be due to the limited data available at the extremes of the target distribution. As shown in Figure 10.3, the distribution of fuel consumption and the most important feature over the entire data set may not capture the full range of possible values. Further analysis and potentially increased data coverage in these areas may be necessary to improve the model's ability to accurately predict values at the extremes.

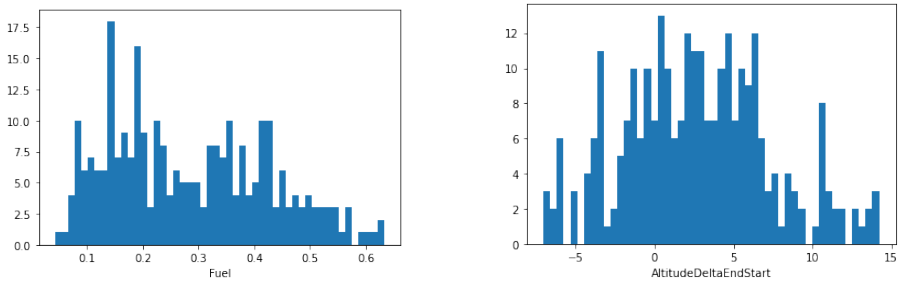


Figure 10.3: feature distribution in all the data

Increasing the amount of data available for analysis is necessary to improve the model further. In our current study, we used 274 statistics collected over 9 days (Table 8.1). This sums up to 137 minutes, or just over 2 hours, of data. This indicates that we only had an average of 15 minutes of usable data per day.

Upon examining the raw data, as described in Section 8.2, we identified several instances of noisy or poorly reported values from the logging unit on the dumper. Investigating this issue is important to increase the amount of usable data for each day. In addition, we can apply more stringent criteria to the data filtering process, ensuring we only train the model with high-quality data. Currently, we applied relatively lenient criteria in some cases.

PART III

Summary and Conclusions

This thesis explores ways to increase the efficiency of road construction sites. One of the key research questions we investigated was whether reinforcement learning can be effectively applied to the scheduling problem on construction sites. To address this question, we developed a reinforcement learning framework to optimize the coordination between dumpers and loaders, minimizing their time idling, driven distance, and overall time spent on tasks. Our results demonstrate that reinforcement learning can indeed provide a powerful tool for improving efficiency on road construction sites. In addition, we created a fuel model to predict the fuel consumption by dumpers based on their routes within the construction site. In summary, our work presents an approach to reducing fuel consumption and increasing efficiency using reinforcement learning and a fuel predictive model.

Reinforcement Learning Framework

Our reinforcement learning framework enables intelligent scheduling of dumpers to improve coordination and efficiency on road construction sites. The framework comprises several agents, each with its responsibilities. When a dumper needs to make a new move, such as driving to a new loading or dumping location or navigating to the next road within the construction site, an agent is called upon to assist.

The Loading Agent provides the dumper with a loading plan, directing it to drive to a specific loading location and load mass with the help of a loader. Before executing the loading plan, the Coffee Break Agent evaluates its reasonability. If approved, the dumper navigates to the loading location. If rejected, the dumper must wait for another dumper in the system to receive its plan before trying again.

The Dumping Agent(s) operates similarly to the Loading Agent but directs the dumper to a dumping location where it can unload its mass. There are multiple Dumping Agents, each responsible for a specific type of mass.

Regardless of the plan, the dumpers deliver it to the Node Agents along the route to the destination. These agents are located on all nodes within the graph and will guide the dumper along the shortest path to its destination by directing it to the next road.

In reinforcement learning, agents learn through a reward signal that guides their decision-making. As agents operate within their environment, they receive rewards for making good decisions and smaller rewards for less optimal choices. In our case, we may only know the efficiency of an agent's action after other agents complete their actions, which delays the rewards. The rewards calculation varies between our agents, but all share the same goal: to reduce the distance driven by dumpers, minimize idling, and increase overall efficiency. Over time, as our agents receive more rewards and learn from their experiences, they become better at making smart decisions.

Our framework will rarely converge to a single solution, which allows for the generation of multiple plans for the user to choose from. Each plan provides metrics that describe its overall efficiency, including the total completion time, distance driven by dumpers, fuel consumption, idling time for loaders and dumpers, and the cost of executing the plan. By providing multiple plans with

different trade-offs between these metrics, our framework allows users to select the plan that best meets their needs and preferences.

It is important to note that these metrics only consider variable costs, which optimization can reduce. Fixed costs, such as the fuel consumption of an operating loader or the salary of a loader operator when loading, are not included in the calculations as they must always be covered regardless of the plan chosen.

We first evaluated our framework in two scenarios with easily interpretable solutions to verify its abilities. Our framework found the best solution in both cases, demonstrating that our agents' responsibilities worked as expected.

We then analyzed 20 more advanced maps and compared our framework to two baselines: Baseline 1, the logical approach, and Baseline 2, the absolute minimum. Our framework outperformed Baseline 1 in 75% of cases, consistently providing plans with lower fuel consumption and an average reduction of 10% compared to Baseline 1. However, this came at the cost of a 23% increase in time due to our framework's deployment of fewer dumpers, among other factors. Our scoring system, used for selecting the best plan, favors this trade-off between fuel consumption and time.

One research question we investigated was how a reinforcement learning framework handles dynamic changes in the construction site environment. Our analysis suggests that our framework is well-equipped to adapt to such changes. It took only a few minutes to simulate 510 episodes and find effective plans for each of the 20 maps analyzed. These maps had similar structures and could represent dynamic changes in the environment. Furthermore, when the maps have the same number of loading and dumping nodes as those used in training, we can employ trained agents to search for good plans from the first simulation. Given its ability to quickly adapt to new conditions, our framework shows promise for handling dynamic changes in real-world construction site environments.

In real-world applications, decision-makers coordinating dumpers can use our framework to improve planning. They can compare the framework's plan with their original plan using metrics and choose the superior one. If computation power permits, users should run multiple instances of the framework with randomized agent initialization and varied hyperparameters to generate a range of plans. Ultimately, users can select the best plan based on their evaluation criteria.

Fuel Model

This thesis introduces a fuel consumption model that aims to enhance the accuracy of fuel consumption estimates during route planning. Ideally, each dumper should have a unique fuel model based on its specific data. As the dumper operates and collects more data, the model can be retrained to provide even more precise estimates. This fuel model can help further optimize the reinforcement learning framework by accurately predicting the fuel consumption of its plans.

The data sets used in this study comprised fuel and GPS data collected from a single dumper over several days. The data points were recorded at varying step lengths, introducing several issues when merging the two data sets.

Individual data points alone are insufficient to gain meaningful insights from the data. Thus, we decided to apply statistics to summarize routes by calculating metrics describing the route. Each route includes several data points, allowing us to calculate metrics such as distance and altitude changes. This approach enabled us to identify critical patterns and trends for accurately predicting fuel consumption.

To generate reliable statistical samples for summarizing routes using data points, we needed a sufficient number of consecutive and accurately recorded data points in our data set. For instance, relying solely on a route's first and last measurements would not provide a sufficient sample. As a result, data cleaning was necessary to ensure the accuracy and reliability of our samples.

During our data cleaning process, we excluded poorly measured fuel rates to ensure the accuracy of the fuel consumption of our samples. We also excluded consecutive data points with unrealistically large variations in altitude. When selecting the final routes, we ensured they contained minimal time gaps, resulting in more data points to rely on for our summary of the route.

Ultimately, we chose to summarize the routes in 30-second time intervals and make our model predict fuel consumption for routes of that duration. The model partitions the data into 30-second summaries for longer routes and predicts a fuel value for each partition. We then sum the predicted fuel values for each partition to make the final prediction for the entire route.

For our fuel consumption model, we found that the most important features were altitude-related, given our fixed time length of 30 seconds. Interestingly, the distance driven was not included as one of the final features, likely because other features already capture this information due to the fixed time length. In addition, we included the amount of time spent accelerating and the quantity of mass loaded in the final model.

Although time constraints prevented us from integrating the fuel model into the reinforcement learning framework, this future work could enhance the accuracy of the framework's plan evaluations. Our fuel model demonstrated promising results in predicting dumper fuel consumption. This addresses our research question of whether continuously logged GPS data can predict the fuel consumption of dumpers and further optimize the scheduling. Integrating the fuel model could further improve the optimization process through more precise evaluations.

PART IV

Appendices

APPENDIX A

Appendix for Part I

A.1 Universal Approximation Theorem

The Universal Approximation Theorem A.1.5 states that a feed-forward neural network with a single hidden layer can approximate any continuous function to an arbitrary accuracy greater than 0.

In the proof of this theorem, we will use a variation of the Hahn-Banach Theorem A.1.1 and the Riesz Representation Theorem A.1.2.

Theorem A.1.1 (**Hahn-Banach** [Rud87, p. 107])**.** *Let S be a linear subspace of a normed linear space X, and let $f_0 \in X$. Then f_0 is in the closure $\bar{S}$ of S if and only if there is no bounded linear functional L on X such that $L(f) = 0 \quad \forall\, f \in S$ but $L(f_0) \neq 0$*

Let $C(\Omega)^*$ be the set of linear functionals on $C(\Omega)$ and let $C(\Omega)$ be the set of all continuous functions from Ω to $\mathbb{R}$. Let $M(\Omega)$ denote the space of finite, signed regular Borel measures on Ω.

Theorem A.1.2 (**Riesz Representation** [MW13, p. 492])**.** *Let Ω be a compact Hausdorff space. Then $L \in C(\Omega)^*$ if and only if there exists a $\mu \in M(\Omega)$ such that*

$$L(f) = \int_{\Omega} f(x)\, d\mu(x) \quad \forall\, f \in C(\Omega) \tag{A.1}$$

Before examining the Universal Approximation Theorem A.1.5, it is important to introduce several definitions. Let I_n denote the n-dimensional unit cube $[0,1]^n$.

Definition A.1.3 ([Cyb89, p. 306])**.** We say that σ is *discriminatory* if for a measure $\mu \in M(I_n)$,

$$\int_{I_n} \sigma(w^T x + b)\, d\mu(x) = 0$$

for all $w \in \mathbb{R}^n$ and $b \in \mathbb{R}$ implies that $\mu = 0$.

Definition A.1.4 ([Cyb89, p. 306])**.** We say that σ is *sigmoidal* if

$$\sigma(t) \to \begin{cases} 1 & as \quad t \to +\infty \\ 0 & as \quad t \to -\infty \end{cases}$$

Theorem A.1.5 (**Universal Approximation** [Cyb89, p. 306])**.** *Let σ be any continuous discriminatory function. Then finite sums of the form*

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(w_j^T x + b_j) \tag{A.2}$$

are dense in $C(I_n)$. In other words, given any $f \in C(I_n)$ and $\epsilon > 0$, there is a sum, $G(x)$, of the above form, for which

$$|G(x) - f(x)| < \epsilon \quad \forall \ x \in I_n$$

Proof. Let $S \subset C(I_n)$ be the set of all functions of the form given in Equation A.2. We want to prove that the subspace S is dense in $C(I_n)$; this is equivalent to the closure of S being equal to $C(I_n)$, i.e., $\bar{S} = C(I_n)$.

Assume that $\bar{S} \neq C(I_n)$:

Applying the Hahn-Banach Theorem A.1.1, stating that there exists a bounded linear functional, L, on the space $C(I_n)$ such that $L(\bar{S}) = L(S) = 0$ and $L \neq 0$. Further, by the Riesz Representation Theorem A.1.2 there exists some $\mu \in M(I_n)$ such that L takes this form:

$$L(h) = \int_{I_n} h(x)\, d\mu(x) \quad \forall\, h \in C(I_n)$$

Since $\sigma(w^T x + b) \in S \subset C(I_n)$,

$$\int_{I_n} \sigma(w^T x + b)\, d\mu(x) = 0$$

We reached a contradiction since σ is assumed to be discriminatory, implying that $\mu = 0$, so the linear functional must be $L = 0$. Hence, the assumption is incorrect, and we have proved by contradiction that $\bar{S} = C(I_n)$, equivalently that S is dense in $C(I_n)$. ■

Remark A.1.6. It is proven [Cyb89] that all continuous sigmoidal functions, as defined in Definition A.1.4, are discriminatory. This result is easier to understand when considering the context of a neural network, as sigmoidal functions are common activation functions.

Remark A.1.7. Equation A.2 in Theorem A.1.5 can be interpreted as representing a single-hidden-layer neural network with N nodes. The network takes x as input, has weights w_j for each node j in the hidden layer, a bias term b, and an activation function σ. We represent the weights for the output node as $\alpha = (\alpha_1, \alpha_2, ..., \alpha_N)$.

A.2 The Default Configuration of the Framework

It is important to note that we still need to validate the default settings against empirical data, so their logic may be limited.

Agent Policy Configurations

- Neural Network Architecture:
  - Fully connected with three hidden layers.
  - Hidden layer node count: 100, 50, and 100 (respectively).
  - ReLU activation function applied to each hidden layer.
- Optimizer:
  - Criterion: MSELoss [PyT]
  - Optimizer: Adam
  - Learning rate: 0.001
- Q-learning Parameters:
  - $\gamma = 0$ (framework restriction, can only be 0)
  - $\alpha = 0.5$
- Memory:
  - Size: 5000 samples.
  - FIFO (First in, First Out) memory management.
  - Up to 500 samples of memory are trained between episodes.

Environment, Map

- Edge weights represent the distance between nodes in meters.
- Loading nodes seem active from the start of an episode.
- The pre-defined timer: 1000000000 seconds (ensuring job completion).

Jobs

- Each job has a mass quantity of 400 tons.
- The mass type is identical.

Dumpers

- Initial position: the parking node
- Average speed: 5 m/s
- Maximum mass capacity: 40 tons
- Independent dumping time: 30 seconds
- Dumping rate: 0 seconds/ton

Loaders

- Independent loading time: 30 seconds
- Loading rate: 4 seconds/ton

Node Agents

- Use only pre-obtained shortest path (see Section 3.3)

Planning agents

- Weights in the reward function (see Equation 3.3):
  - Loading Agent: $(w_1, w_2) = (8, 3)$
  - Dumping Agent(s): $(w_1, w_2) = (8, -60 \cdot 8)$.
    - By setting $w_2 = -60 \cdot 8$, the cost of one unit of imbalance, based on the completion rates, is equated to the cost of driving an additional 60 seconds.
- The parking action is enabled with weighting $w_4 = 1$ in Equation 3.4.

Coffee Break Agent

- The agent is active.
- The weight of taking a coffee break is $w_5 = 1$ (see Equation 3.3).

Additional parameters

- Fuel Consumption:
  - While idling for a dumper: $3.785 \cdot 1$ L/h
  - While idling for a loader: $3.785 \cdot 3$ L/h
  - While driving for a dumper: $3.785 \cdot 8$ L/h
- Fuel Price: 21 NOK/L

- Salaries:
  - Loader operator: 242.15 NOK/h
  - Dumper operator: 233.33 NOK/h

A.3 Determining the Starting Order of Dumpers

This procedure is done to determine the starting order of the dumpers.

1. Set all loading nodes to be unallocated.
2. For each unallocated loading node, identify the nearest unallocated dumper and add it to a list.
3. For each dumper in the list from Step 2, select its nearest loading node from among the loading nodes that have chosen that dumper and allocate that loading node.
4. For each dumper in the list from Step 2, request a plan from the Loading Agent in the order of the shortest distance to any loading node. Then, allocate the dumpers.
5. If not all dumpers are allocated:
   a) If all loading nodes are allocated, return to Step 1.
   b) If not all loading nodes are allocated, return to Step 2.
6. All dumpers have received their initial plan, and the episode can begin.

A.4 Map 1: Demonstrating Baseline 1

Table A.1 presents the shortest distances between all loading and dumping nodes for Map 1, as depicted in Figure 4.4. In Map 1, each loading node is responsible for transporting 400 tons, while each dumping node should receive 800 tons. Consequently, the pairing process will select each dumping node twice. The data lead to the following pair-up of nodes in Baseline 1:

- First, $L8$ is the most distant from any dumping node (specifically, node $D60$) and pairs up with $D57$, the closest dumping node among the three.
- Second, $L17$ is the next most distant from any dumping node (specifically, node $D60$) and pairs up with $D57$. Since $L8$ and $L17$ have paired up with $D57$, it is unavailable for additional pairing.
- Next, $L97$ chooses $D63$. Although $L97$ is closer to $D57$, it is unavailable due to two loading nodes already paired with it. Therefore, $L97$ pairs up with its second-best option.
- $L15$ selects $D63$.
- Finally, $L72$ and $L32$ get $D60$, completing the pairing process.

Thus, the resulting pairs are:

$(L8, D57)$, $(L17, D57)$, $(L97, D63)$, $(L15, D63)$, $(L72, D60)$ and $(L32, D60)$.

Table A.1: Distances between loading and dumping nodes in Map 1

Loading node	**Dumping node**	**Distance (m)**
L8	D57	2662
	D60	5166
	D63	4200
L15	D57	3241
	D60	3935
	D63	2969
L17	D57	2336
	D60	4840
	D63	3874
L32	D57	3103
	D60	2318
	D63	1680
L72	D57	3891
	D60	1599
	D63	1301
L97	D57	2183
	D60	4215
	D63	2701

A.5 Map 1: Dumper and Loader Scheduling and Performance

Table A.2 shows metrics of the plans with the best performance, where the best plan is plan number 459. Listing A.1 provides the loaders' schedule for the best plan. Listing A.2 provides the dumpers' schedule for the best plan.

Table A.2: Performance of top plans for Map 1

Score	Rank	Plan	Fuel	Cost	Time	Distance	L idle	D idle	D
75.3	590	459	617.3	19019	6575	356.0	3691	6517	22
75.2	050	487	619.6	18933	6481	359.8	3171	4224	22
74.8	110	485	613.4	18733	7072	355.3	3904	3268	22
74.6	970	165	619.2	18951	6834	356.7	4909	3444	21
74.3	060	346	622.1	18941	6835	361.2	3679	2733	22
74.2	680	233	618.5	18976	7096	354.5	5885	3469	19
74.0	340	449	614.8	18933	7477	347.0	9161	2126	15
74.0	003	224	638.4	19627	6192	370.1	2548	7438	25

Score (see Section 4.1), **D**: dumpers (required amount).
Rank XYZ: Plan ranking based on Fuel (Xth), Cost (Yth), Time (Zth). A '0' indicates that the plan did not rank in the top 9 for any of the categories.
Plan (Game number), **Fuel** (liter), **Cost** (NOK), **Time** (seconds), **Distance** (kilometer).
L idle: Loaders idling (seconds), **D idle**: Dumpers idling (seconds).

Listing A.1: Best plan (Map 1): Schedule for the loaders

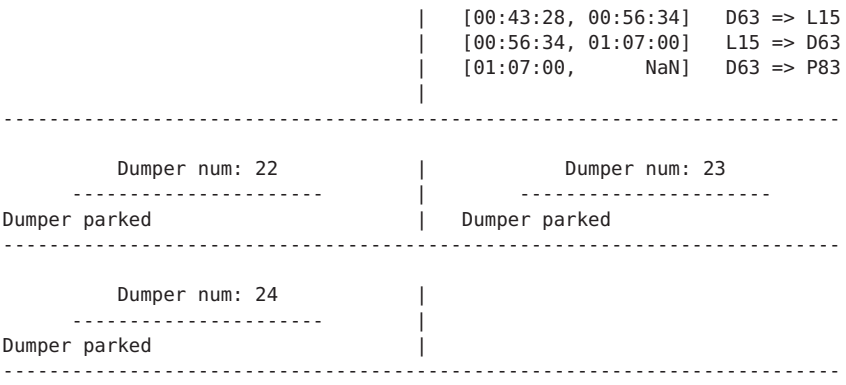
```
         Loader (node:  8)          |        Loader (node: 15)
       ----------------------       |     ----------------------
Start first job: 00:33:46           |   Start first job: 00:14:58
Finish last job: 01:22:04           |   Finish last job: 01:15:40
                                    |
Breaks (FROM - TO):                 |   Breaks (FROM - TO):
     00:18:08 - 00:32:14            |        00:41:37 - 00:43:34
     00:35:24 - 00:47:05            |
     00:50:15 - 00:53:24            |
-------------------------------------------------------------------------

         Loader (node: 17)          |        Loader (node: 32)
       ----------------------       |     ----------------------
Start first job: 00:25:43           |   Start first job: 00:10:25
Finish last job: 00:59:28           |   Finish last job: 00:56:27
                                    |
Breaks (FROM - TO):                 |   Breaks (FROM - TO):
     00:41:37 - 00:43:34            |        00:13:35 - 00:24:41
                                    |        00:31:02 - 00:34:11
-------------------------------------------------------------------------

         Loader (node: 72)          |        Loader (node: 97)
       ----------------------       |     ----------------------
Start first job: 00:04:44           |   Start first job: 00:08:52
Finish last job: 00:36:33           |   Finish last job: 00:40:41
                                    |
Breaks (FROM - TO):                 |   Breaks (FROM - TO):
-------------------------------------------------------------------------
```

Note: The loaders at nodes 72 and 97 do not have any breaks.

Listing A.2: Best plan (Map 1): Time schedule for dumpers

```
          Dumper num:  0            |            Dumper num:  1
      ----------------------        |        ----------------------
Time:                    Plan:      |  Time:                    Plan:
[00:00:00, 00:07:54]   P83 => L72   |  [00:00:00, 00:11:05]   P83 => L72
[00:07:54, 00:12:45]   L72 => D63   |  [00:11:05, 00:16:56]   L72 => D60
[00:12:45, 00:28:53]   D63 => L17   |  [00:16:56, 00:27:51]   D60 => L32
[00:28:53, 00:37:12]   L17 => D57   |  [00:27:51, 00:33:58]   L32 => D63
[00:37:12, 00:49:55]   D57 => L17   |  [00:33:58, 00:50:05]   D63 => L32
[00:49:55, 00:58:14]   L17 => D57   |  [00:50:05, 00:58:20]   L32 => D60
[00:58:14, 01:12:31]   D57 =>  L8   |  [00:58:20, 01:08:05]   D60 => P83
[01:12:31, 01:30:19]    L8 => D60   |  Dumper parked
-------------------------------------------------------------------------

          Dumper num:  2            |            Dumper num:  3
      ----------------------        |        ----------------------
Time:                    Plan:      |  Time:                    Plan:
[00:00:00, 00:14:16]   P83 => L72   |  [00:00:00, 00:17:27]   P83 => L72
[00:14:16, 00:20:07]   L72 => D60   |  [00:17:27, 00:22:18]   L72 => D63
[00:20:07, 00:31:02]   D60 => L32   |  [00:22:18, 00:35:24]   D63 => L15
[00:31:02, 00:37:09]   L32 => D63   |  [00:35:24, 00:45:50]   L15 => D63
[00:37:09, 00:50:15]   D63 => L15   |  [00:45:50, 01:06:07]   D63 => L15
[00:50:15, 01:00:41]   L15 => D63   |  [01:06:07, 01:16:33]   L15 => D63
[01:00:41, 01:05:22]   D63 => P83   |
Dumper parked                       |
-------------------------------------------------------------------------

          Dumper num:  4            |            Dumper num:  5
      ----------------------        |        ----------------------
Time:                    Plan:      |  Time:                    Plan:
[00:00:00, 00:20:38]   P83 => L72   |  [00:00:00, 00:12:02]   P83 => L97
[00:20:38, 00:26:29]   L72 => D60   |  [00:12:02, 00:19:50]   L97 => D57
[00:26:29, 00:40:32]   D60 => L32   |  [00:19:50, 00:32:04]   D57 => L17
[00:40:32, 00:48:47]   L32 => D60   |  [00:32:04, 00:40:23]   L17 => D57
[00:48:47, 01:15:40]   D60 => L15   |  [00:40:23, 00:56:17]   D57 => L17
[01:15:40, 01:26:06]   L15 => D63   |  [00:56:17, 01:04:36]   L17 => D57
                                    |  [01:04:36, 01:18:53]   D57 =>  L8
                                    |  [01:18:53, 01:36:41]    L8 => D60
-------------------------------------------------------------------------

          Dumper num:  6            |            Dumper num:  7
      ----------------------        |        ----------------------
Time:                    Plan:      |  Time:                    Plan:
[00:00:00, 00:23:49]   P83 => L72   |  [00:00:00, 00:13:35]   P83 => L32
[00:23:49, 00:29:40]   L72 => D60   |  [00:13:35, 00:19:42]   L32 => D63
[00:29:40, 00:43:43]   D60 => L32   |  [00:19:42, 00:36:56]   D63 =>  L8
[00:43:43, 00:51:58]   L32 => D60   |  [00:36:56, 00:46:21]    L8 => D57
[00:54:05, 01:03:50]   D60 => P83   |  [00:46:21, 01:09:18]   D57 => L15
Dumper parked                       |  [01:09:18, 01:19:44]   L15 => D63
-------------------------------------------------------------------------

          Dumper num:  8            |            Dumper num:  9
      ----------------------        |        ----------------------
Time:                    Plan:      |  Time:                    Plan:
[00:00:00, 00:27:00]   P83 => L72   |  [00:00:00, 00:30:11]   P83 => L72
[00:27:00, 00:31:51]   L72 => D63   |  [00:30:11, 00:36:02]   L72 => D60
[00:31:51, 00:46:54]   D63 => L32   |  [00:36:02, 00:53:16]   D60 => L32
[00:46:54, 00:53:01]   L32 => D63   |  [00:53:16, 01:01:31]   L32 => D60
[00:53:01, 00:57:42]   D63 => P83   |
Dumper parked                       |
-------------------------------------------------------------------------
```

```
          Dumper num: 10            |           Dumper num: 11
      ----------------------        |       ----------------------
Time:                    Plan:      |  Time:                    Plan:
[00:00:00, 00:33:22]   P83 => L72   |  [00:00:00, 00:36:33]   P83 => L72
[00:33:22, 00:39:13]   L72 => D60   |  [00:36:33, 00:42:24]   L72 => D60
[00:39:13, 00:56:27]   D60 => L32   |  [00:42:24, 01:02:56]   D60 => L15
[00:56:27, 01:02:34]   L32 => D63   |  [01:02:56, 01:16:36]   L15 => D60
[01:02:34, 01:07:15]   D63 => P83   |
Dumper parked                       |
-------------------------------------------------------------------------

          Dumper num: 12            |           Dumper num: 13
      ----------------------        |       ----------------------
Time:                    Plan:      |  Time:                    Plan:
[00:00:00, 00:15:13]   P83 => L97   |  [00:00:00, 00:18:24]   P83 => L97
[00:15:13, 00:23:01]   L97 => D57   |  [00:18:24, 00:26:12]   L97 => D57
[00:23:01, 00:35:15]   D57 => L17   |  [00:26:12, 00:38:26]   D57 => L17
[00:35:15, 00:43:34]   L17 => D57   |  [00:38:26, 00:46:52]   L17 => D57
[00:43:34, 00:59:28]   D57 => L17   |  [00:46:52, 01:12:29]   D57 => L15
[00:59:28, 01:07:47]   L17 => D57   |  [01:12:29, 01:22:55]   L15 => D63
[01:07:47, 01:22:04]   D57 =>  L8   |
[01:22:04, 01:39:52]    L8 => D60   |
-------------------------------------------------------------------------

          Dumper num: 14            |           Dumper num: 15
      ----------------------        |       ----------------------
Time:                    Plan:      |  Time:                    Plan:
[00:00:00, 00:21:35]   P83 => L97   |  [00:00:00, 00:24:46]   P83 => L97
[00:21:35, 00:29:23]   L97 => D57   |  [00:24:46, 00:32:34]   L97 => D57
[00:29:23, 00:41:37]   D57 => L17   |  [00:32:34, 00:44:39]   D57 =>  L8
[00:41:37, 00:49:56]   L17 => D57   |  [00:44:39, 00:54:04]    L8 => D57
[00:49:56, 01:02:01]   D57 =>  L8   |  [00:54:04, 01:06:09]   D57 =>  L8
[01:02:01, 01:19:49]    L8 => D60   |  [01:06:09, 01:23:57]    L8 => D60
-------------------------------------------------------------------------

          Dumper num: 16            |           Dumper num: 17
      ----------------------        |       ----------------------
Time:                    Plan:      |  Time:                    Plan:
[00:00:00, 00:27:57]   P83 => L97   |  [00:00:00, 00:31:08]   P83 => L97
[00:27:57, 00:35:45]   L97 => D57   |  [00:31:08, 00:38:56]   L97 => D57
[00:35:45, 00:46:44]   D57 => L17   |  [00:38:56, 00:53:06]   D57 => L17
[00:46:44, 00:55:03]   L17 => D57   |  [00:53:06, 01:01:25]   L17 => D57
[00:55:03, 01:09:20]   D57 =>  L8   |  [01:01:25, 01:15:42]   D57 =>  L8
[01:09:20, 01:27:08]    L8 => D60   |  [01:15:42, 01:33:30]    L8 => D60
-------------------------------------------------------------------------

          Dumper num: 18            |           Dumper num: 19
      ----------------------        |       ----------------------
Time:                    Plan:      |  Time:                    Plan:
[00:00:00, 00:34:19]   P83 => L97   |  [00:00:00, 00:37:30]   P83 => L97
[00:34:19, 00:43:59]   L97 => D63   |  [00:37:30, 00:45:18]   L97 => D57
[00:43:59, 00:59:45]   D63 => L15   |  [00:45:18, 00:57:23]   D57 =>  L8
[00:59:45, 01:10:11]   L15 => D63   |  [00:57:23, 01:15:11]    L8 => D60
-------------------------------------------------------------------------

          Dumper num: 20            |           Dumper num: 21
      ----------------------        |       ----------------------
Time:                    Plan:      |  Time:                    Plan:
[00:00:00, 00:40:41]   P83 => L97   |  [00:00:00, 00:18:08]   P83 => L15
[00:40:41, 00:50:14]   L97 => D63   |  [00:18:08, 00:28:34]   L15 => D63
[00:50:14, 00:54:55]   D63 => P83   |  [00:28:34, 00:37:21]   D63 => L32
Dumper parked                       |  [00:37:21, 00:43:28]   L32 => D63
```



Note: The dumpers 22, 23, and 24 are parked at the start and remain unused.

APPENDIX B

Appendix for Part II

B.1 A Brief Introduction to Decision Trees

A decision tree [Qui86] is a powerful tool for making informed decisions. It visually displays the different options and their potential outcomes, allowing for a thorough evaluation of each choice. The final value of each branch presents the predicted outcome.

Figure B.1 shows an example of a decision tree predicting the fuel consumption of a dumper on a specific route. The decision tree indicates that routes longer than 100 meters tend to have higher fuel consumption compared to shorter routes.

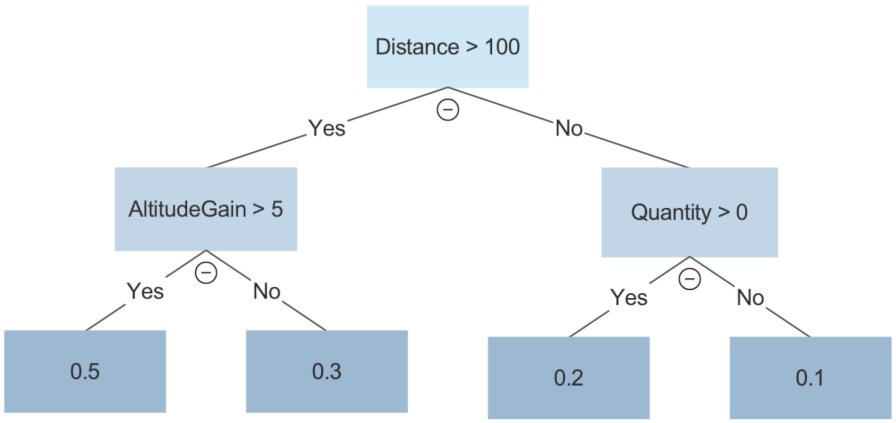


Figure B.1: An example of a decision tree, predicting fuel consumption of a route.

B.2 Tables of Selected Data Points

Table B.1: Constant fuel rate, raw data

data_Time	data_Fuel	diff_xy	fuel_data_fails	track_data_fails
2021-10-15 05:18:46	26.95	4.0	False	False
2021-10-15 05:18:51	74.50	19.0	True	False
2021-10-15 05:18:56	74.50	14.0	True	False
2021-10-15 05:19:01	74.50	8.0	True	False
2021-10-15 05:19:06	74.50	29.0	True	False
2021-10-15 05:19:11	74.50	28.0	True	False
2021-10-15 05:19:16	74.50	11.0	True	False
2021-10-15 05:19:21	74.50	39.0	True	False
2021-10-15 05:19:26	74.50	32.0	True	False
2021-10-15 05:19:31	74.50	11.0	True	False
2021-10-15 05:19:36	74.50	15.0	True	False
2021-10-15 05:19:41	74.50	17.0	True	False
2021-10-15 05:19:46	74.50	15.0	True	False
2021-10-15 05:19:51	74.50	15.0	True	False
2021-10-15 05:19:56	74.50	7.0	True	False
2021-10-15 05:20:01	74.50	5.0	True	False
2021-10-15 05:20:06	74.50	10.0	True	True
2021-10-15 05:20:11	74.50	1.0	True	True
2021-10-15 05:20:16	74.50	0.0	True	True
2021-10-15 05:20:21	74.50	0.0	True	True
2021-10-15 05:20:26	74.50	2.0	True	True
2021-10-15 05:20:31	74.50	1.0	True	False
2021-10-15 05:20:36	74.50	11.0	True	False
2021-10-15 05:20:41	74.50	14.0	True	False
2021-10-15 05:20:46	74.50	11.0	True	False
2021-10-15 05:20:51	74.50	30.0	True	False
2021-10-15 05:20:56	74.50	39.0	True	False
2021-10-15 05:21:01	32.10	13.0	False	False

The table shows constant fuel rate over several time points. Such data points will be marked as failed.

Table B.2: Zero in between high fuel rate

data_Time	data_Fuel	diff_xy
2021-10-19 12:12:58	81.15	31.0
2021-10-19 12:13:03	61.05	31.0
2021-10-19 12:13:08	0.00	28.0
2021-10-19 12:13:13	35.10	13.0
2021-10-19 12:13:18	6.70	21.0
2021-10-19 12:13:23	14.10	14.0
2021-10-19 12:13:28	38.85	10.0
2021-10-19 12:13:33	32.15	3.0
2021-10-19 12:13:38	67.25	40.0
2021-10-19 12:13:43	34.45	16.0

The table illustrates an unstable fuel rate, characterized by a sudden spike to 0 when the dumper is still moving (as indicated by a positive diff_xy value).

Table B.3: Low fuel rate

data_Time	data_Fuel	diff_xy	fuel_data_fails	track_data_fails
2021-10-15 05:23:02	2.25	1.0	False	True
2021-10-15 05:23:07	2.45	0.0	False	True
2021-10-15 05:23:12	2.40	0.0	False	True
2021-10-15 05:23:17	2.35	0.0	False	True
2021-10-15 05:23:22	2.25	0.0	False	True
2021-10-15 05:23:27	2.45	0.0	False	True
2021-10-15 05:23:32	2.55	0.0	False	True
2021-10-15 05:23:37	2.65	0.0	False	True
2021-10-15 05:23:42	2.65	0.0	False	True
2021-10-15 05:23:47	2.15	0.0	False	True
2021-10-15 05:23:52	3.10	0.0	False	True
2021-10-15 05:23:57	2.15	0.0	False	True
2021-10-15 05:24:02	2.30	0.0	False	True
2021-10-15 05:24:07	2.30	0.0	False	True
2021-10-15 05:24:12	2.70	0.0	False	True
2021-10-15 05:24:17	2.70	0.0	False	True
2021-10-15 05:24:22	3.60	0.0	False	True
2021-10-15 05:24:27	2.45	0.0	False	True
2021-10-15 05:24:32	2.40	0.0	False	True
2021-10-15 05:24:37	2.00	0.0	False	True
2021-10-15 05:24:42	2.50	0.0	False	True
2021-10-15 05:24:47	2.25	0.0	False	True

The table shows that the fuel rate varies from 2 to 3, whenever the dumper is standing still (**diff_xy** ≈ 0).

Table B.4: Altitude oscillating

Timestamp	Distance	Altitude	VerticalAccuracy	Over threshold
2021-10-19 17:41:07	36.47	301.48	7.16	False
2021-10-19 17:41:13	40.59	300.30	10.02	False
2021-10-19 17:41:19	32.19	302.89	7.46	False
2021-10-19 17:41:24	21.38	300.99	10.41	False
2021-10-19 17:41:31	8.93	300.01	8.20	False
2021-10-19 17:41:37	7.65	298.09	8.29	False
2021-10-19 17:41:45	9.80	296.79	12.37	True
2021-10-19 17:41:56	6.57	286.05	16.24	True
2021-10-19 17:42:07	4.09	292.57	18.02	True
2021-10-19 17:42:13	7.33	304.42	11.53	True
2021-10-19 17:42:20	2.84	293.35	17.15	True
2021-10-19 17:42:27	2.22	293.50	18.78	True
2021-10-19 17:42:33	2.24	293.79	11.55	True
2021-10-19 17:42:41	2.43	292.73	13.87	True
2021-10-19 17:42:49	2.46	293.52	14.68	True
2021-10-19 17:42:55	0.73	289.29	9.48	True
2021-10-19 17:43:07	2.41	294.64	14.21	True
2021-10-19 17:43:50	6.79	302.16	11.60	True

The raw data indicates that the **Altitude** tends to oscilliate when the **VerticalAccuracy** exceeds the threshold of 12. It appears to be illogical, as the observed changes in **Altitude** are too large for the relatively low time and **Distance**.

B.3 Data Summary: A Table of Statistics

Table B.5: Some statistics of each route, from data set Section 8.5.
LD: LengthDistance, **ADES**: AltitudeDeltaEndStart,
AT: AccTime, **Q**: Quantity

No	Date	Fuel (L)	LD (m)	ADES (m)	AT (s)	Q (ton)
1	2021-10-18 10:48:01	0.04	60	-1.2	20.0	0.0
2	2021-09-29 09:51:25	0.06	194	-1.5	12.0	0.0
3	2021-10-14 17:45:17	0.07	64	-0.5	9.0	0.0
4	2021-09-29 07:18:18	0.07	118	-3.3	3.0	0.0
5	2021-10-14 18:01:51	0.07	53	-0.4	3.0	0.0
6	2021-09-28 14:33:25	0.08	76	-1.8	9.0	0.0
7	2021-10-19 15:16:57	0.08	112	-1.1	12.0	40.0
8	2021-10-20 05:26:53	0.08	35	1.5	7.0	0.0
9	2021-10-13 08:47:06	0.08	110	-5.8	17.0	0.0
10	2021-09-28 10:20:44	0.08	101	-3.6	11.0	0.0
11	2021-10-13 06:04:17	0.08	103	-4.3	11.0	0.0
12	2021-10-19 16:26:29	0.08	54	2.2	6.0	0.0
13	2021-09-28 11:02:44	0.08	104	-3.3	17.0	0.0
14	2021-09-28 08:16:57	0.08	101	-3.7	11.0	0.0
15	2021-10-19 16:03:29	0.09	103	-1.9	14.0	0.0
16	2021-10-19 16:08:13	0.09	19	0.2	5.0	0.0
17	2021-09-28 16:27:10	0.09	123	-2.1	12.0	0.0
18	2021-09-28 11:41:08	0.09	114	0.0	6.0	0.0
19	2021-10-21 10:31:26	0.09	69	-5.3	18.0	0.0
20	2021-09-28 07:54:45	0.10	108	-1.2	17.0	0.0
21	2021-09-28 16:17:53	0.10	107	-2.0	10.0	0.0
22	2021-10-14 10:37:47	0.10	79	0.1	0.0	40.0
23	2021-09-29 14:03:25	0.10	68	-0.3	10.0	0.0
24	2021-11-15 18:59:28	0.10	90	-0.7	9.0	0.0
25	2021-10-15 05:05:37	0.11	87	-3.2	12.0	0.0
26	2021-10-18 07:18:01	0.11	102	-0.1	6.0	0.0
27	2021-10-21 10:54:02	0.11	55	-1.3	24.0	0.0
28	2021-10-20 15:23:47	0.11	105	1.0	10.0	0.0
29	2021-10-20 15:06:45	0.11	139	-3.4	18.0	0.0
30	2021-10-19 11:31:44	0.11	108	-1.0	11.0	0.0
31	2021-10-15 12:34:31	0.11	86	-6.8	11.0	40.0
32	2021-10-20 10:30:24	0.11	35	-4.1	13.0	40.0
33	2021-09-28 15:19:30	0.11	99	0.2	24.0	0.0
34	2021-09-28 16:46:34	0.12	104	-3.2	10.0	0.0
35	2021-10-21 11:00:53	0.12	46	-1.5	16.0	0.0
36	2021-10-13 14:48:05	0.12	61	1.4	10.0	0.0
37	2021-10-19 15:45:46	0.13	126	2.8	13.0	0.0
38	2021-10-18 15:10:53	0.13	105	2.9	0.0	0.0
39	2021-10-15 05:44:05	0.13	96	-7.0	7.0	0.0
40	2021-10-14 14:41:35	0.13	82	-5.9	9.0	0.0
41	2021-09-28 09:43:36	0.13	67	-1.4	22.0	0.0
42	2021-09-28 07:05:08	0.13	115	-5.0	25.0	0.0
43	2021-10-21 06:35:57	0.14	100	-2.0	4.0	0.0
44	2021-10-13 10:57:38	0.14	111	-0.3	4.0	40.0
45	2021-09-28 07:17:33	0.14	101	-2.7	12.0	0.0
46	2021-09-29 10:44:58	0.14	137	-3.5	10.0	0.0
47	2021-10-13 10:35:22	0.14	121	-6.3	9.0	0.0
48	2021-10-19 19:30:45	0.14	110	0.7	9.0	0.0
49	2021-10-13 14:47:07	0.14	90	0.5	17.0	0.0
50	2021-10-14 10:01:59	0.14	124	-1.1	22.0	0.0
51	2021-10-15 12:08:32	0.14	94	-6.1	6.0	0.0
52	2021-10-19 17:52:27	0.14	91	2.6	2.0	0.0
53	2021-10-19 07:03:04	0.14	118	1.5	6.0	0.0
54	2021-09-28 07:33:10	0.14	128	-4.1	13.0	0.0
55	2021-10-19 15:57:47	0.14	108	2.6	2.0	0.0
56	2021-09-28 15:48:43	0.15	122	-3.3	6.0	0.0
57	2021-10-13 10:15:16	0.15	107	0.4	18.0	0.0
58	2021-10-14 16:30:01	0.15	111	-3.9	13.0	0.0
59	2021-10-19 16:20:52	0.15	94	-2.0	13.0	40.0
60	2021-09-28 15:48:11	0.15	144	-0.6	12.0	0.0
61	2021-10-14 16:38:24	0.15	117	-6.3	7.0	0.0
62	2021-10-13 12:50:39	0.15	54	0.8	12.0	40.0
63	2021-09-28 10:32:20	0.16	123	-4.2	14.0	0.0
64	2021-10-21 06:25:13	0.16	55	2.2	21.0	0.0
65	2021-09-28 11:30:22	0.16	134	0.3	23.0	0.0
66	2021-10-19 16:10:39	0.16	102	3.0	6.0	0.0
67	2021-10-13 07:11:47	0.16	51	-0.6	15.0	40.0
68	2021-09-28 15:16:57	0.16	54	2.0	12.0	0.0
69	2021-09-28 14:32:54	0.16	137	-1.4	24.0	0.0
70	2021-10-19 15:22:18	0.16	119	2.6	15.0	0.0
71	2021-11-03 09:32:44	0.17	69	0.0	11.0	0.0
72	2021-10-14 10:59:29	0.17	103	-5.3	7.0	0.0
73	2021-09-29 08:32:14	0.17	146	-2.0	17.0	0.0

Continued on next page

Table B.5 – continued from previous page

No	Date	Fuel (L)	LD (m)	ADES (m)	AT (s)	Q (ton)
74	2021-09-28 17:32:10	0.17	164	-1.8	21.0	0.0
75	2021-10-14 11:44:03	0.17	91	-3.8	23.0	0.0
76	2021-09-28 15:20:10	0.17	136	-4.0	17.0	0.0
77	2021-10-14 11:17:39	0.17	110	-0.5	12.0	40.0
78	2021-10-13 10:02:08	0.18	110	1.9	5.0	40.0
79	2021-09-29 06:10:53	0.18	150	-3.3	11.0	0.0
80	2021-10-20 15:23:09	0.18	67	2.8	21.0	0.0
81	2021-10-15 12:07:11	0.18	111	1.0	8.0	40.0
82	2021-09-29 05:58:49	0.18	108	-1.6	8.0	30.0
83	2021-10-14 10:10:47	0.18	120	0.3	10.0	40.0
84	2021-09-28 14:10:46	0.18	120	-3.2	11.0	0.0
85	2021-10-20 07:44:47	0.19	122	0.9	15.0	0.0
86	2021-10-14 14:09:30	0.19	119	1.5	6.0	40.0
87	2021-10-20 13:56:35	0.19	100	2.4	17.0	0.0
88	2021-10-21 06:23:11	0.19	105	-0.6	11.0	28.0
89	2021-10-20 08:10:23	0.19	127	1.0	12.0	0.0
90	2021-09-28 17:40:47	0.19	136	-6.1	17.0	0.0
91	2021-10-19 17:15:49	0.19	104	1.2	17.0	0.0
92	2021-10-14 14:39:17	0.19	108	0.4	4.0	40.0
93	2021-09-28 11:51:21	0.19	90	0.4	29.0	0.0
94	2021-10-14 14:10:58	0.19	107	-4.0	18.0	0.0
95	2021-09-28 17:30:14	0.19	137	0.1	8.0	42.0
96	2021-09-28 11:19:58	0.19	123	1.0	8.0	42.0
97	2021-10-18 14:03:03	0.19	64	3.1	12.0	40.0
98	2021-10-13 08:18:06	0.19	106	-6.7	7.0	0.0
99	2021-10-14 10:44:55	0.19	112	0.1	6.0	40.0
100	2021-09-28 17:39:00	0.20	151	0.2	10.0	42.0
101	2021-10-19 16:48:34	0.20	117	0.4	19.0	0.0
102	2021-10-20 05:26:18	0.20	81	1.0	23.0	0.0
103	2021-09-29 08:58:20	0.20	129	-1.0	11.0	0.0
104	2021-09-28 12:00:55	0.20	122	-1.1	6.0	0.0
105	2021-09-28 17:40:14	0.20	133	-0.7	16.0	0.0
106	2021-10-14 17:52:00	0.20	173	-1.0	15.0	40.0
107	2021-10-19 17:04:00	0.20	124	1.4	10.0	0.0
108	2021-10-15 06:04:48	0.21	125	1.2	9.0	40.0
109	2021-10-13 17:00:20	0.21	198	-3.4	18.0	40.0
110	2021-10-19 17:21:55	0.21	130	-0.5	14.0	0.0
111	2021-10-19 18:02:20	0.22	110	2.5	18.0	0.0
112	2021-09-28 11:49:31	0.22	60	4.3	7.0	42.0
113	2021-10-18 07:17:21	0.22	180	6.2	11.0	0.0
114	2021-09-29 06:20:40	0.22	139	-0.7	20.0	0.0
115	2021-10-18 12:32:53	0.22	59	2.0	12.0	40.0
116	2021-10-19 18:44:41	0.23	126	-3.8	17.0	0.0
117	2021-10-13 09:52:12	0.23	111	2.4	13.0	40.0
118	2021-10-14 17:49:19	0.23	133	4.0	12.0	40.0
119	2021-10-20 07:13:34	0.23	121	4.2	7.0	40.0
120	2021-10-14 14:18:54	0.23	110	-6.1	10.0	0.0
121	2021-09-28 05:23:13	0.23	80	2.9	14.0	0.0
122	2021-10-20 07:23:19	0.23	103	1.6	13.0	0.0
123	2021-10-19 15:33:01	0.23	130	1.7	9.0	0.0
124	2021-10-14 16:28:38	0.24	124	0.8	10.0	40.0
125	2021-10-20 08:16:53	0.24	108	-0.5	15.0	40.0
126	2021-10-18 14:59:56	0.24	56	2.9	17.0	0.0
127	2021-09-28 17:19:09	0.24	129	1.2	13.0	42.0
128	2021-10-19 16:09:25	0.24	78	5.6	17.0	0.0
129	2021-10-14 17:54:57	0.24	137	4.7	11.0	0.0
130	2021-10-19 12:13:31	0.24	141	2.1	13.0	0.0
131	2021-10-14 17:56:23	0.24	152	-2.5	25.0	40.0
132	2021-10-19 16:08:44	0.25	36	4.2	9.0	0.0
133	2021-10-20 13:55:26	0.25	86	4.3	29.0	0.0
134	2021-10-21 09:36:13	0.25	117	2.5	5.0	31.0
135	2021-09-28 17:57:32	0.26	158	3.5	11.0	0.0
136	2021-10-18 07:10:35	0.26	209	4.8	11.0	0.0
137	2021-10-21 06:44:01	0.26	131	0.4	7.0	24.0
138	2021-10-19 17:00:14	0.26	58	5.5	22.0	0.0
139	2021-09-28 09:41:53	0.26	93	3.3	6.0	42.0
140	2021-10-19 16:47:24	0.27	89	5.8	29.0	0.0
141	2021-10-14 10:00:25	0.27	114	1.6	14.0	40.0
142	2021-10-20 13:33:53	0.27	108	2.7	26.0	0.0
143	2021-10-20 13:32:30	0.27	59	4.8	23.0	0.0
144	2021-10-14 12:50:13	0.27	107	2.1	10.0	40.0
145	2021-10-19 12:25:28	0.28	94	6.6	20.0	0.0
146	2021-10-13 17:07:30	0.28	160	-1.3	8.0	40.0
147	2021-10-18 16:00:13	0.28	66	0.3	17.0	40.0
148	2021-10-14 18:00:33	0.29	120	3.4	7.0	40.0
149	2021-10-19 12:12:25	0.29	103	5.4	24.0	0.0
150	2021-10-14 11:42:27	0.29	95	3.5	16.0	40.0
151	2021-09-28 17:20:08	0.29	135	-1.9	17.0	0.0
152	2021-09-28 15:46:28	0.29	138	-0.1	14.0	42.0
153	2021-10-14 09:59:29	0.30	40	3.2	13.0	0.0
154	2021-09-28 10:28:12	0.30	88	3.1	18.0	0.0
155	2021-10-19 12:01:33	0.30	140	2.4	11.0	0.0

Continued on next page

Table B.5 – continued from previous page

No	Date	Fuel (L)	LD (m)	ADES (m)	AT (s)	Q (ton)
156	2021-10-14 17:56:56	0.30	164	2.3	10.0	0.0
157	2021-10-13 08:16:32	0.31	116	4.2	12.0	0.0
158	2021-10-21 06:10:21	0.31	124	3.1	9.0	36.0
159	2021-09-28 15:45:41	0.32	103	3.2	13.0	0.0
160	2021-10-13 14:43:35	0.32	112	5.0	18.0	40.0
161	2021-09-28 08:03:59	0.32	92	2.2	12.0	0.0
162	2021-09-28 08:15:20	0.32	102	4.0	11.0	0.0
163	2021-09-28 05:47:41	0.32	74	1.9	23.0	0.0
164	2021-09-28 07:12:08	0.32	61	3.7	8.0	42.0
165	2021-10-13 14:21:21	0.32	45	3.3	15.0	0.0
166	2021-10-14 18:02:21	0.32	125	-1.1	19.0	0.0
167	2021-10-13 14:10:19	0.33	114	4.4	18.0	40.0
168	2021-10-19 16:47:58	0.33	101	10.7	9.0	0.0
169	2021-09-29 06:09:25	0.33	146	1.9	12.0	36.0
170	2021-10-19 17:14:17	0.33	59	6.2	29.0	0.0
171	2021-10-19 17:51:00	0.33	60	5.5	27.0	0.0
172	2021-09-29 06:18:36	0.34	123	6.3	17.0	0.0
173	2021-09-28 11:37:43	0.34	107	5.0	13.0	0.0
174	2021-09-29 08:56:54	0.34	140	1.5	12.0	39.0
175	2021-10-18 07:16:51	0.34	126	6.9	20.0	0.0
176	2021-10-19 12:00:57	0.34	92	10.5	6.0	0.0
177	2021-10-13 11:49:34	0.35	118	-6.1	6.0	0.0
178	2021-10-15 06:30:34	0.35	116	3.6	13.0	40.0
179	2021-09-28 11:10:38	0.35	102	4.7	13.0	0.0
180	2021-10-13 07:02:07	0.35	92	4.6	15.0	40.0
181	2021-10-19 15:45:04	0.35	117	10.5	9.0	0.0
182	2021-10-19 12:00:26	0.35	109	7.6	23.0	0.0
183	2021-10-19 18:01:44	0.35	118	13.4	11.0	0.0
184	2021-10-21 10:28:51	0.35	19	3.8	22.0	36.0
185	2021-09-29 09:49:30	0.35	130	2.7	15.0	39.0
186	2021-10-19 16:36:36	0.35	118	6.4	23.0	0.0
187	2021-09-28 16:24:29	0.36	95	3.8	13.0	0.0
188	2021-10-20 05:25:20	0.36	100	8.5	15.0	0.0
189	2021-09-29 06:08:49	0.36	121	4.5	13.0	36.0
190	2021-10-19 15:09:58	0.36	124	13.6	8.0	0.0
191	2021-10-19 17:51:36	0.36	121	11.8	14.0	0.0
192	2021-11-15 16:54:49	0.36	73	3.4	11.0	40.0
193	2021-10-19 11:49:05	0.37	134	13.8	7.0	0.0
194	2021-09-29 08:56:22	0.37	131	2.4	19.0	39.0
195	2021-10-20 17:22:06	0.37	207	4.5	18.0	0.0
196	2021-10-13 11:38:15	0.37	124	4.7	19.0	40.0
197	2021-10-21 08:00:23	0.37	150	0.3	8.0	36.0
198	2021-09-28 07:03:44	0.38	76	2.5	8.0	42.0
199	2021-10-19 15:21:30	0.38	128	12.9	10.0	0.0
200	2021-10-20 13:56:00	0.38	96	10.6	8.0	0.0
201	2021-10-21 06:34:39	0.38	125	0.2	10.0	24.0
202	2021-10-21 10:29:28	0.38	75	6.1	13.0	36.0
203	2021-10-14 10:18:07	0.38	112	5.3	20.0	40.0
204	2021-09-29 07:16:02	0.38	166	4.1	15.0	34.0
205	2021-10-19 16:25:18	0.39	115	11.0	22.0	0.0
206	2021-09-28 08:50:47	0.39	111	4.2	11.0	0.0
207	2021-10-14 18:05:58	0.39	136	8.2	8.0	40.0
208	2021-10-19 17:02:59	0.40	110	10.8	13.0	0.0
209	2021-10-19 20:09:47	0.40	160	0.9	8.0	40.0
210	2021-10-14 10:57:43	0.40	113	4.9	20.0	40.0
211	2021-09-28 11:19:07	0.40	117	4.7	16.0	0.0
212	2021-10-18 07:10:05	0.40	182	6.9	29.0	0.0
213	2021-10-19 19:44:16	0.41	194	-3.4	19.0	0.0
214	2021-10-13 12:48:09	0.41	93	5.9	16.0	0.0
215	2021-10-19 15:21:00	0.41	94	6.5	29.0	0.0
216	2021-10-20 13:16:17	0.41	102	6.0	29.0	0.0
217	2021-09-29 05:58:07	0.41	129	1.3	13.0	30.0
218	2021-10-20 12:56:16	0.41	125	9.1	24.0	0.0
219	2021-09-28 17:38:25	0.42	133	4.1	17.0	0.0
220	2021-10-19 14:55:12	0.42	80	5.4	19.0	0.0
221	2021-09-28 14:31:24	0.42	107	3.7	14.0	42.0
222	2021-10-14 14:57:26	0.42	115	4.6	19.0	40.0
223	2021-10-20 13:33:07	0.42	111	11.0	16.0	0.0
224	2021-09-29 08:30:12	0.42	142	4.6	18.0	39.0
225	2021-09-28 17:55:19	0.42	89	4.6	10.0	42.0
226	2021-09-28 17:29:31	0.42	122	4.4	17.0	42.0
227	2021-10-21 10:38:32	0.42	34	5.8	23.0	0.0
228	2021-10-13 17:57:42	0.43	117	14.2	16.0	0.0
229	2021-10-21 10:52:17	0.43	33	5.6	22.0	0.0
230	2021-10-13 08:35:09	0.43	64	6.6	19.0	0.0
231	2021-10-19 15:44:34	0.43	102	7.1	29.0	0.0
232	2021-10-19 19:42:40	0.43	158	-0.9	13.0	40.0
233	2021-10-20 07:12:38	0.43	132	10.7	19.0	40.0
234	2021-10-14 16:46:00	0.44	96	3.2	16.0	40.0
235	2021-10-14 11:27:57	0.44	87	4.9	20.0	40.0
236	2021-10-21 17:33:44	0.44	79	5.0	13.0	30.0
237	2021-10-19 11:36:07	0.45	125	9.0	29.0	0.0

Continued on next page

Table B.5 – continued from previous page

No	Date	Fuel (L)	LD (m)	ADES (m)	AT (s)	Q (ton)
238	2021-10-19 08:47:17	0.45	114	10.1	6.0	0.0
239	2021-10-15 06:04:15	0.45	63	5.9	23.0	0.0
240	2021-10-21 08:12:32	0.45	143	2.1	9.0	36.0
241	2021-10-21 10:52:50	0.46	97	6.4	14.0	32.0
242	2021-10-20 14:58:35	0.46	123	11.4	25.0	0.0
243	2021-10-15 05:56:42	0.46	90	5.2	22.0	0.0
244	2021-10-20 08:09:27	0.46	123	12.6	12.0	0.0
245	2021-10-20 08:21:43	0.47	137	14.2	16.0	0.0
246	2021-10-13 05:25:55	0.47	87	7.7	17.0	0.0
247	2021-10-14 11:07:02	0.47	97	6.5	19.0	40.0
248	2021-10-13 10:23:56	0.48	102	6.1	23.0	40.0
249	2021-10-15 05:49:25	0.48	86	6.0	22.0	0.0
250	2021-10-15 06:13:13	0.48	96	6.4	17.0	40.0
251	2021-10-19 17:14:48	0.49	132	11.6	23.0	0.0
252	2021-10-21 16:54:26	0.49	91	5.8	16.0	32.0
253	2021-10-13 17:13:52	0.49	110	7.8	15.0	40.0
254	2021-10-15 05:34:28	0.49	88	6.4	18.0	0.0
255	2021-10-13 14:32:37	0.49	97	5.6	20.0	40.0
256	2021-10-15 05:42:00	0.50	66	5.3	20.0	0.0
257	2021-10-15 08:19:04	0.51	106	5.2	17.0	0.0
258	2021-10-19 14:43:26	0.51	132	8.6	29.0	0.0
259	2021-10-14 16:36:40	0.51	86	5.3	22.0	40.0
260	2021-10-13 11:47:41	0.52	82	6.3	20.0	0.0
261	2021-10-13 14:21:59	0.52	87	6.2	21.0	40.0
262	2021-10-19 14:33:14	0.52	156	13.2	23.0	0.0
263	2021-10-21 10:45:41	0.53	55	6.9	19.0	0.0
264	2021-10-13 11:26:37	0.53	73	7.2	17.0	0.0
265	2021-10-13 17:14:29	0.53	165	12.1	24.0	40.0
266	2021-10-19 15:56:38	0.54	122	10.6	29.0	0.0
267	2021-10-21 17:11:17	0.55	64	9.3	19.0	0.0
268	2021-10-15 14:55:09	0.55	70	6.6	25.0	40.0
269	2021-10-15 06:39:18	0.56	84	6.8	20.0	0.0
270	2021-10-19 14:10:21	0.57	121	8.7	29.0	0.0
271	2021-10-19 14:20:32	0.57	143	10.8	28.0	0.0
272	2021-10-14 14:38:47	0.57	82	9.2	20.0	0.0
273	2021-10-21 10:59:04	0.59	59	8.5	19.0	0.0
274	2021-10-15 14:47:07	0.60	85	7.7	20.0	40.0
275	2021-10-13 05:16:36	0.62	86	8.4	15.0	0.0
276	2021-10-21 11:25:05	0.62	60	10.5	14.0	0.0
277	2021-10-15 14:32:18	0.63	58	7.0	23.0	40.0

Bibliography

[APP20] APPLE. *MacBook Pro (13-inch, M1, 2020) - Technical Specifications*. https://support.apple.com/kb/SP824?viewlocale=en_US&locale=no_NO. Accessed on May 13, 2023. Apple, 2020.

[Bel52] Bellman, R. 'On the Theory of Dynamic Programming'. In: *Proc Natl Acad Sci U S A* vol. 38, no. 8 (1952), pp. 716–719.

[BM08] Bondy, A. and Murty, U. *Graph Theory (graduate texts in mathematics 244)*. Springer, 2008.

[Bru23] Bruce, Ø. *MasterThesis_Bruce*. https://github.com/oystehbr/MasterThesis_Bruce. 2023.

[BYC13] Bergstra, J., Yamins, D. and Cox, D. 'Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures'. In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Dasgupta, S. and McAllester, D. Vol. 28. Proceedings of Machine Learning Research 1. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 115–123.

[CG16] Chen, T. and Guestrin, C. 'XGBoost: A Scalable Tree Boosting System'. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, 2016, pp. 785–794.

[CLL20] Covert, I., Lundberg, S. M. and Lee, S. 'Understanding Global Feature Contributions Through Additive Importance Measures'. In: *CoRR* vol. abs/2004.00668 (2020). arXiv: 2004.00668.

[Cov] Covert, I. *sage-importance - PyPI*. https://pypi.org/project/sage-importance/. Version: 0.0.4.

[Cov20] Covert, I. *Explaining machine learning models with SHAP and SAGE*. https://iancovert.com/blog/understanding-shap-sage/. Accessed: December 15, 2022. 2020.

[Cyb89] Cybenko, G. 'Approximation by superpositions of a sigmoidal function'. In: *Mathematics of Control, Signals and Systems* vol. 2 (1989), pp. 303–314.

[GBC16] Goodfellow, I., Bengio, Y. and Courville, A. *Deep learning*. MIT press, 2016.

[HN15] Hanson, C. S. and Noland, R. B. 'Greenhouse gas emissions from road construction: An assessment of alternative staging approaches'. In: *Transportation Research Part D: Transport and Environment* vol. 40 (2015), pp. 97–103.

[Hou10] Hougardy, S. 'The Floyd–Warshall algorithm on graphs with negative cycles'. In: *Information Processing Letters* vol. 110, no. 8-9 (2010), pp. 279–281.

[KB14] Kingma, D. P. and Ba, J. 'Adam: A Method For Stochastic Optimization'. In: *arXiv preprint arXiv:1412.6980* (2014).

[Koh95] Kohavi, R. 'A study of cross-validation and Bootstrap for accuracy estimation and model selection'. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. Morgan Kaufmann, 1995, pp. 1137–1143.

[Lei+23] Leivestad Hall, E. et al. *Vil kutte utslipp med kunstig intelligens*. https://www.nrk.no/osloogviken/anleggsmaskiner-pa-tomgang-er-klimaversting-_-kunstig-intelligens-kan-vaere-losningen-1.16338194. Accessed: April 14, 2023. 2023.

[Lev+16] Levine, S. et al. 'Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection'. In: *CoRR* vol. abs/1603.02199 (2016). arXiv: 1603.02199.

[LL17] Lundberg, S. M. and Lee, S. 'A unified approach to interpreting model predictions'. In: *CoRR* vol. abs/1705.07874 (2017). arXiv: 1705.07874.

[Lun] Lundberg, S. *shap - PyPI*. https://pypi.org/project/shap/. Version: 0.41.0.

[MW13] McDonald, J. N. and Weiss, N. A. *A Course in Real Analysis*. 2nd ed. Academic Press, 2013.

[Nie15] Nielsen, M. A. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, USA, 2015.

[Ouy+22] Ouyang, L. et al. 'Training language models to follow instructions with human feedback'. In: *Advances in Neural Information Processing Systems* vol. 35 (2022), pp. 27730–27744.

[PyT] PyTorch. *torch.nn.MSELoss*. https://pytorch.org/docs/stable/generated/torch.nn.MSELoss.html. Accessed: April 17, 2023.

[Qin+21] Qin, W. et al. 'Multi-agent reinforcement learning-based dynamic task assignment for vehicles in urban transportation system'. In: *International Journal of Production Economics* vol. 240 (2021), p. 108251.

[Qui86] Quinlan, J. R. 'Induction of decision trees'. In: *Machine Learning* (1986), pp. 81–106.

[RHW86] Rumelhart, D. E., Hinton, G. E. and Williams, R. J. 'Learning representations by back-propagating errors'. In: *nature* vol. 323, no. 6088 (1986), pp. 533–536.

[Rud87] Rudin, W. *Real and Complex Analysis*. 3rd ed. McGraw-Hill Boston, 1987.

[SA21] Shwartz-Ziv, R. and Armon, A. 'Tabular Data: Deep Learning is Not All You Need'. In: *CoRR* vol. abs/2106.03253 (2021). arXiv: 2106.03253.

[SB18] Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. The MIT Press, 2018.

[Sha53] Shapley, L. S. '17. A Value for n-Person Games'. In: *Contributions to the Theory of Games (AM-28), Volume II*. Ed. by Kuhn, H. W. and Tucker, A. W. Princeton: Princeton University Press, 1953, pp. 307–318.

[Sil+18] Silver, D. et al. 'A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play'. In: *Science* vol. 362, no. 6419 (2018), pp. 1140–1144.

[SIN20] SINTEF. *Data-driven road construction sites*. https://www.sintef.no/en/projects/2020/data-driven-road-construction-sites/. Accessed: March 15, 2023. 2020.

[SIN23] SINTEF. *Artificial intelligence cuts emissions and costs of road projects*. https://www.sintef.no/en/latest-news/2023/artificial-intelligence-aids-with-cuts-in-emissions-and-costs-of-road-projects/. Accessed: April 14, 2023. 2023.

[Ska19] Skanska. *Skanska vil kutte utslipp med kunstig intelligens*. https://www.skanska.no/hvem-vi-er/media/aktuelt/skanska-vil-kutte-utslipp-med-kunstig-intelligense/. Accessed: April 14, 2023. 2019.

[Str01] Stripple, H. *Life cycle assessment of road. A pilot study for inventory analysis*. 2001.

[UiO] UiO, M. i. *MAEC-studenter ble vinnerne av Norges Bank Case-NM 2020!* https://www.mn.uio.no/math/studier/aktuelt/aktuelle-saker/2020/maec-studenter-vant-nb-casenm.html. Accessed: April 21, 2023.

[Vol] Volvo. *A45G FS Articulated Haulers*. https://www.volvoce.com/norge/nb-no/volvo-maskin-as/products/articulated-haulers/a45gfs/. Accessed: February 20, 2023.

[WD92] Watkins, C. J. and Dayan, P. 'Q-learning'. In: *Machine learning* vol. 8 (1992), pp. 279–292.

[XG18] Xu, Y. and Goodacre, R. 'On Splitting Training and Validation Set: A Comparative Study of Cross-Validation, Bootstrap and Systematic Sampling for Estimating the Generalization Performance of Supervised Learning'. In: *Journal of Analysis and Testing* vol. 2 (Oct. 2018).

[AA] Achiam, J. and Abbeel, P. *OpenAI: Spinning Up*. https://spinningup.openai.com/en/latest/index.html#. Accessed: December 7, 2022.