Machine Learning and Optimization Techniques Applied to Haulage Trucks Fueling in Surface Mining Operations

by
Luis Fernando Larota Machacca

A thesis submitted to the Graduate Division
in partial fulfillment of the requirements for the degree of

Master of Science in Mining Engineering

South Dakota School of Mines and Technology
Rapid City, South Dakota

Date Defended: September 06, 2023

Approved by:

Major Professor — Robert Hall, Ph.D., Department of Mining Engineering and Management	Date
Graduate Division Representative — Saurav Kumar Dubey, Ph.D., Department of Industrial Engineering	Date
Committee Member — Andrea Brickey, Ph.D., Department of Mining Engineering and Management	Date
Head of the Mining Engineering and Management Department — Robert Hall, Ph.D.	Date
Dean of Graduate Education — Maribeth Price	Date

Abstract

Fuel dispatching for haulage trucks in open pit mines is a crucial task impacting mining operations' efficiency. Long wait times for fueling can reduce equipment productivity, leading to increased costs and reduced profits. Mine dispatchers use their intuition or respond to radio requests from operators to assign fuel stops, which may not be optimal and may not consider future implications. This research proposes an automated fueling assignment system for haulage trucks that uses machine learning and optimization techniques to optimize fueling assignments.

The proposed optimization model maximizes the match factor of truck-shovel allocations while dispatching trucks to fuel, considering road networks, haul cycles, fueling stops, fuel consumption, and cycle time information to provide optimal solutions for mine dispatchers. The results show that the match factor is ideal for determining how trucks are scheduled to fuel. The results of the optimization model act as a recommendation tool for mine dispatchers, providing them with a solution that improves decision-making. The proposed model contributes to optimizing and automating fuel dispatching in open pit mines, improving efficiency and reducing costs.

Furthermore, this thesis evaluates six machine learning algorithms' performance in predicting fuel consumption based on effective flat haul distances and tonnage hauled. The Artificial Neural Network achieved the highest coefficient of determination, R^2 value, with a relatively high tuning time. In comparison, Multi-Linear Regression performed well in speed and obtained a respectable R^2 value, making it a good option for simple models. Even though the artificial neural network model reached a higher tuning time to find the best hyperparameters, it can be deployed fast and as a pre-trained model for further use by other researchers.

The proposed solution provides a baseline for researchers to build more complex models to improve fuel dispatching, decrease operational delays, and assist the decision-making process of mine dispatchers. In addition, a website application is created and deployed to show the selected machine learning model and the optimization model running online to replicate how the dispatchers would use this tool.

Acknowledgments

I want to express my deepest gratitude to all those who have made this research project possible. In particular, I am incredibly grateful to my major professor, Dr. Robert A. Hall, whose tremendous sacrifice and effort were instrumental in completing this work. In addition, his expertise in mining and fleet management systems has been invaluable in enhancing my understanding of the subject area.

I also sincerely thank Dr. Andrea Brickey and Dr. Saurav Dubey for their insightful contributions, discussions, and feedback that helped me write up this project successfully. Additionally, I am grateful for the input and support of my research colleagues at the Caterpillar Lab and the encouragement and support of my fellow graduate students and friends, including Edward Anokye, Michael Tetteh, Dr. Judith Buaba, Dr. Amy McBrayer, Kristin Guerin, Sharon Arrieta, Michael Michael, Darren Osei, and Clint Kling.

My dear family, Julia, and Walter, have always provided me with their unwavering support, and I am indebted to them for everything they have done to help me succeed in my career. I also look up to my sister, Janet, and appreciate her support and encouragement.

Lastly, I want to thank the entire Mine Engineering and Management department at the South Dakota School of Mines Technology for providing me with the opportunity and support to undertake this research project.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Mining is a global industry producing raw materials (Reichl and Schatz, 2022). Mining aims to extract, process, and provide minerals and metals to support the future need of the world, making it a critical industry now and in the future (Chropra and Swart, 2022).

Mining is a complex process that involves five distinct stages. The first stage is prospecting, which involves measuring the earth's properties to identify potential mineral deposits. Geologists then estimate the mineral concentration and variability of the deposits through exploratory drilling. The second stage is exploration, during which the mine site is thoroughly studied to determine the feasibility of mining. Finally, this stage involves collecting and analyzing data to assess the size and quality of the mineral deposit (Whittle, 2019).

The third stage is development, which includes obtaining permits, removing overlying waste, and conducting mine planning studies. During this stage, the most appropriate mining method is determined, capital investments are estimated, and production capacities are planned. The fourth stage is production, where ore is extracted from the ground and transported to stockpiles and processing plants. Waste material is transported to waste dumps and tailings. Finally, in the reclamation stage, the area where mining occurred is restored to its natural state through activities such as soil stabilization, re-vegetation, and water management Whittle (2019); Newman et al. (2010).

To extract an ore body, mining operations are split into business units (e.g., drilling, blasting, loading, and hauling processes (Hustrulid et al., 2013)). Approximately 50 percent of operating costs are allocated to haulage and materials handling; to reduce these costs and improve the efficiency of loading and hauling, surface mining operations apply fleet management systems (FMS) from multiple providers, e.g., Caterpillar, Modular Mining Systems, and Hexagon (Moradi and Askari-Nasab, 2019).

1.1 Fleet Management Systems

Fleet management systems (FMS) are computer systems used in surface mining operations to manage and optimize the movement of haulage and loading equipment, including auxiliary equipment. These systems provide real-time information about the fleets' location, status, and utilization, enabling better decision-making and improved operational efficiency. In addition, implementing FMS can reduce fuel consumption and improve safety, leading to lower costs and increased sustainability (Aguirre-Jofré et al., 2021).

The use of FMS in mining operations is becoming increasingly critical due to the improved accessibility to GPS, 4G, and wireless radio network technology, as outlined in Table 1.1 (Carter, 2012). The primary goal of FMS is to improve mine production by maximizing material movement and efficiency with real-time data from the equipment. The primary users of FMS are dispatchers, who make decisions on truck and shovel assignments based on FMS recommendations, such as assigning a truck to a loader based on proximity (Moradi and Askari-Nasab, 2019).

Table 1.1: Elements of a Fleet Management System in Surface Mining Operations
Source: Moradi and Askari-Nasab (2019)

Component	Usage
1. Global Positioning System (GPS)	- Bring situational awareness of loading and haulage equipment - Measure traveling time and distance - Build Road Network
2. Wireless Radio Network	- Communicate and store data in servers
3. Optimization Solver	- Run allocation algorithms for loading and hauling equipment
4. Data Servers	- Store data for further reporting

Data collected from Fleet Management Systems (FMS) helps surface mining operations build a time usage model, a graphical representation of productive and non-productive events. Time usage models are first divided into categories, such as calendar time, representing the total available time for equipment to work. Next, unscheduled time, such as planned shutdowns, is subtracted from the calendar time to determine the scheduled time (refer to Figure 1.1). Mining operations can then use these categories to calculate their productivity indicators, including availability and production effectiveness (Lukacs, 2020).

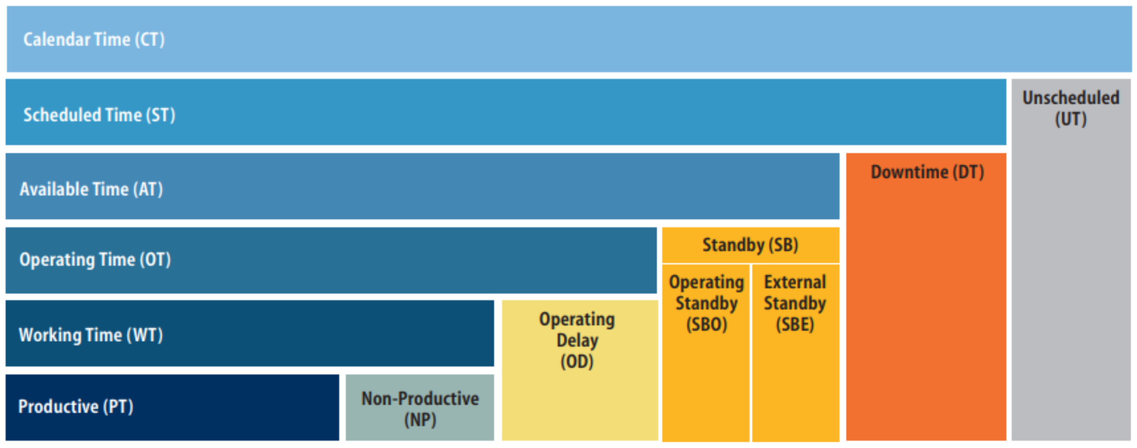


Figure 1.1: Generic Time Usage Model
Source: Lukacs (2020)

Operational delays, such as shift changes, haul road maintenance, blasting, and haulage truck fueling, negatively impact productivity in the shovel-truck system of surface mining operations. Therefore, minimizing these delays is crucial for increasing efficiency and productivity in the loading and hauling process. For example, reducing shift changes and equipment inspection times by 30 and 15 minutes can prevent downstream processes from being interrupted (Schmidt, 2015).

1.2 Haulage Trucks Fueling

The fueling process of haulage trucks in surface mining operations has four stages: travel to the fueling station, waiting at the station, fueling, and return travel to the loading location. Mine dispatchers are responsible for manually assigning trucks to either stationary or mobile fuel stations. The dispatchers make decisions based on truck operators communicating when their fuel level drops below 40% (Schmidt, 2015). However, this manual process can lead to concerns due to the need for more analysis of the potential queues it may cause at fueling stations and the impact on the number of trucks a shovel needs to load (match factor).

Burt and Caccetta (2007) and Modular Mining (2019) state that the match factor measures the effectiveness of trucks and shovel allocations. The computation of the match factor can be run online with data recorded from FMS. The match factor matches the cycle times of trucks assigned to loading equipment with its loading time. If the match factor is zero, the shovel needs more trucks to load (low utilization of shovel); on the other hand, if one, the shovel has enough trucks to load (Burt and Caccetta, 2007).

Studies have shown that improving dispatch processes and implementing fuel consumption monitoring systems can minimize operational delays and increase efficiency in the shovel-truck system (Adams and Bansah, 2016). Additionally, Khorasgani et al. (2020) explored the application of deep reinforcement learning techniques in dynamically dispatching trucks to shovels and found that the current manual practices for fueling assignments need improvement through the use of efficient dispatching algorithms. This would streamline the dispatcher's work and improve downstream activities, such as reducing delays in loading equipment.

1.3 Optimization Applied to Loading and Hauling Dispatching

Dispatching algorithms to allocate hauling to loading equipment has been widely studied and improved academically (Moradi and Askari-Nasab, 2019). Different techniques such as queuing theory, discrete event simulation, and linear and dynamic programs have improved surface mine operations' loading and hauling process (Newman et al., 2010).

Linear programs (LP) are widely used with fleet management systems in the market. For instance, according to Moradi and Askari-Nasab (2019), DISPATCH®, an FMS from Modular Mining Systems, uses a linear program to allocate trucks to loaders and shovels maximizing the fleet production considering operational and production constraints. In addition, Wenco International Mining Systems use a linear program to find the optimal hauling-loading configuration through multiple iterations. This enables them to dynamically dispatch trucks according to a mining plan and automatically account for any operational disruptions, such as equipment failures, weather changes, or delays (Wenco Mining Systems, 2013).

Therefore, this proposal takes a proactive approach to develop an optimization model to automatically assign trucks to the fuel station, ensuring the match factor of the truck and its assigned shovel is impacted by the delays to the smallest possible extent. To create the optimization model, data on fuel consumption through a road network has been computed, to which machine learning algorithms were applied. The objectives that will drive this research are as follows.

- Primary objectives
  - To demonstrate that an optimization model can improve fuel dispatching of haulage trucks in surface mining operations by maximizing their match factor.
  - To develop a fuel consumption model for a single fleet of haul trucks using six machine learning models for linear regression.
- Secondary objectives
  - To develop an online application that replicates the interaction between a dispatcher and the developed machine learning and optimization tools.
  - To show that the running time of the optimization model to provide an optimal solution is accurate and fast for dispatchers' decision-making.

This thesis is structured as follows: Chapter 2, ”Previous Work,” provides an overview of machine learning for regression problems and how it has been applied to predict fuel consumption. This includes the algorithms, features, labels, and performance used. The chapter then briefly overviews fuel dispatching solutions from academic and industry sources. Chapter 3, ”Methods,” covers the materials used, including data collection and preparation, as well as the assumptions and limitations of the data. It also explains the techniques used to tune and select the machine learning algorithms for predicting fuel consumption and the programming language and frameworks used. This chapter also presents the optimization model that solves the fuel dispatching problem, including the programming language used to model it. Chapter

4, ”Results,” presents the outcomes of the trained machine learning algorithms, including the learned parameters and chosen hyper-parameters. Finally, Chapters 5 and 6, ”Conclusions,” and ”Future Work,” summarize the answers to the research objectives stated in Chapter 1 and outline future work that is aimed at improving the results of this research.

Chapter 2

Previous Work

2.1 Machine Learning in the Mining Industry

2.1.1 Machine Learning Overview

Machine learning (ML) has gained significant popularity in recent years. For instance, Chollet (2021) mentioned that chatbots, autonomous vehicles, and virtual assistants have been promised by machine learning and its power, often portrayed in a bleak or idealistic manner. Machine learning grew from computer science, and it was first introduced as ”machine learning” by Arthur Samuel in 1959 (Wuest et al., 2016). Bishop (2006) describes machine learning as a field of study that enables computers to learn without explicit programming - the core concept of machine learning is that computer programs can be trained to enhance their performance in a specific task through the experiences gained from the data they process. Machine learning works as follows: a model is trained with a dataset and uses the patterns it learns from the data to make predictions about new data that it has never encountered before (Wuest et al., 2016). As the model is exposed to more (unseen) data, its performance improves, leading to a deeper understanding of the relationships and patterns (Bishop, 2006).

The classification of machine learning algorithms depends on how the data is labeled. There are four main types of machine learning algorithms: supervised, unsupervised, semi-supervised, and reinforcement learning (Hastie et al., 2009). For instance:

1. Supervised learning: the most commonly used, where the data is labeled, and the algorithm learns to predict the correct output for a given input. This type of learning is used in many applications, including image recognition, speech recognition, natural language processing, and many others. In addition, supervised learning algorithms include linear regression, decision trees, and neural networks (Hastie et al., 2009).

2. Unsupervised learning: employed when the data is unlabeled, and the algorithm has to identify patterns and structure without prior knowledge of what to look for. This type of learning is utilized for clustering, dimensionality reduction, and anomaly detection. Some examples of unsupervised learning algorithms include k-means clustering, principal component analysis (PCA), and generative adversarial networks (GANs) (Hastie et al., 2009).

Chollet (2021) and Bishop (2006) researched extensively on the remaining two types of machine learning algorithms. A brief explanation of them is as follows:

1. Semi-supervised learning: employed when the data is partly labeled. The algorithm acquires the ability to make predictions by relying on labeled and unlabeled data. This type of learning is beneficial when labeling data is either costly or time-consuming (e.g., somebody is labeling them manually before the learning process). Semi-supervised learning algorithms encompass self-training, co-training, and multi-view learning (Chollet, 2021).

2. Reinforcement learning: category of machine learning, which makes the algorithm learn how to make decisions based on the feedback received in the form of rewards or penalties while interacting with the environment. This type of learning is applied in different fields like robotics, game playing, and autonomous driving. For example, Q-learning, SARSA, and deep reinforcement learning are some reinforcement learning algorithms (Sutton and Barto, 2018)

Machine learning is used to solve a wide variety of problems. For instance, Ray (2019) classifies machine learning algorithms based on the type of problem meant to be solved:

1. Classification: to classify data into different classes or categories. For example, recognize whether haulage trucks are in an image (Ali and Frimpong, 2021).

2. Regression: to predict a continuous value. For example, estimate diesel particulate matter in underground mines (Buaba and Brickey, 2021).

3. Clustering: to group data based on similarities or patterns. For example, different group images of minerals to discover similar patterns (Liu et al., 2019).

Machine learning has become an essential tool in addressing many real-world problems. It has diverse applications in finance, healthcare, and marketing (see details in Table 2.1below).

Table 2.1: Use Cases of Machine Learning in Different Industries

Industry	Machine learning - Use case	Resource
Mining	• Predict the quality of ores • Improve fatigue control • Rock fragmentation prediction • Predict fuel consumption of equipment • Improve the efficiency of mineral processing	• Hyder et al. (2019); • Amoako et al. (2022)
Retail	• Analyze customer behavior • Recommendation systems for customers • Predict customer demands • Improve supply chain management	Mathur and Mathur (2019)
Marketing	• Customer segmentations • Predict the behavior of buyers • Personalize advertisement	Ngai and Wu (2022)
Manufacture	• Predict equipment failures • Improve quality control • Automate production processes	Wuest et al. (2016)
Healthcare	• Diagnose diseases • Predict patient outcomes • Develop specialized treatment plans	Shailaja et al. (2018)

The mining industry generates massive amounts of data, which can be analyzed with the help of machine learning algorithms. By processing large volumes of data, machine learning can help mining companies predict future mineral deposit trends and improve business units within mining operations. In addition, using traditional methods, machine learning can be used to analyze geological data to identify potential mineral deposits that may have gone undetected (Hyder et al., 2019).

The application of machine learning in safety is also crucial in the mining industry as it can help identify patterns in data that could lead to accidents or injuries. For instance, machine learning algorithms can analyze mining equipment sensor data to detect patterns indicating workers are in danger. Moreover, it can monitor worker behavior to identify potential safety hazards and suggest ways to mitigate them, ultimately improving worker safety. In summary, machine learning has the potential to significantly enhance productivity, profitability, and worker safety in the mining industry (Hyder et al., 2019; Ouanan and Abdelwahed, 2019).

2.1.2 Machine Learning Algorithms for Regression Problems

2.1.2.1 Linear Regression

Linear regression is a statistical method used to find the best-fitting straight line that models the relationship between one or more input variables and an output variable. The method aims to predict future values of the dependent variable using the importance of the independent variables (Hastie et al., 2009).

While univariate linear regression is a simple form of linear regression that predicts one output variable from a single input variable using Equation 2.1, multivariate linear regression predicts an output variable from multiple input variables as shown in Equation 2.2 (Hastie et al., 2009):

$$\hat{y} = h_\theta(x) = \theta_0 + \theta x \tag{2.1}$$

$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \tag{2.2}$$

Where $\hat{y}$ is the output variable, x is the input variable, θ_0 is the intercept, and θ is the slope of the line. The method aims to minimize the sum of the squared differences between the observed values y and predicted values $\hat{y}$ (Hastie et al., 2009). It can be illustrated as follows:

The gradient descent algorithm helps multivariate linear regression find the best $\theta_0, \theta_1, ..., \theta_n$ through the least squares method (James et al., 2013):

$$LeastSquares = J(\theta) = \frac{1}{n} \sum_{j=1}^{n} (h_\theta(x_j) - \hat{y}_j)^2 \tag{2.3}$$

$$\theta = \min \sum_j \text{Least Squares}(\hat{y}_j, h_\theta) \tag{2.4}$$

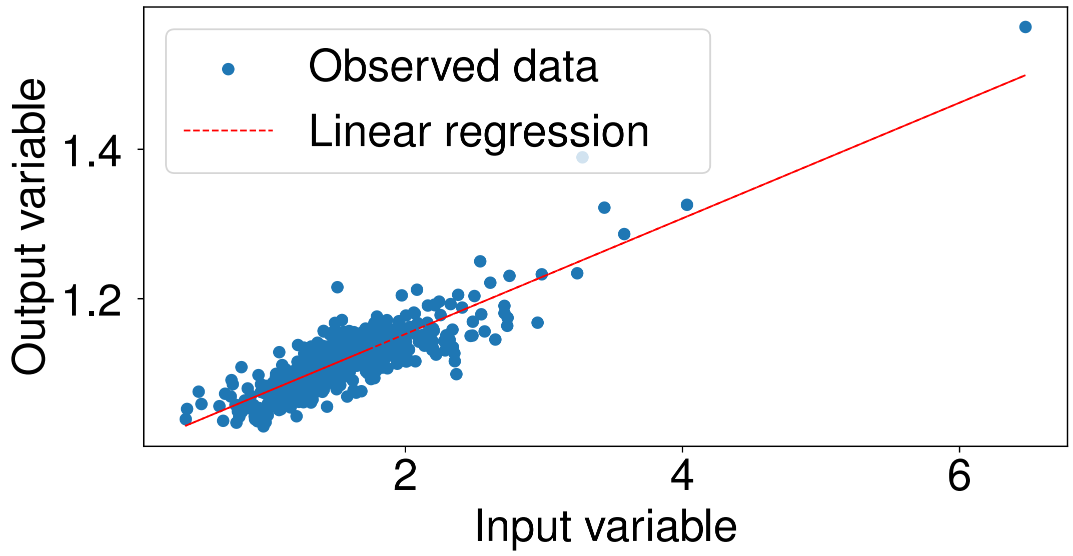


Figure 2.1: Univariate Linear Regression

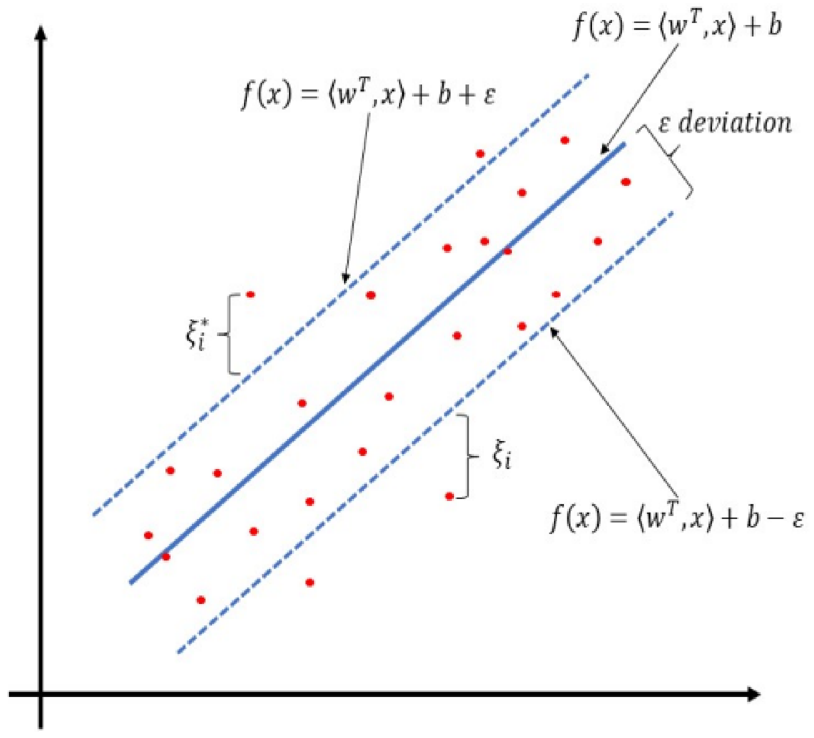
2.1.2.2 Support Vector Regression (SVR)

According to James et al. (2013), support vector regression (SVR) is a statistical method used to predict a continuous variable based on input variables. One notable feature of SVR is its tolerance for outliers within a pre-defined limit, represented as ε. This means that the algorithm allows data points to deviate from the expected pattern within a certain margin of error, as specified by epsilon. In essence, SVR is a robust regression technique that can handle noise and outliers in the input data (Hastie et al., 2009). The SVR's estimate function is similar to linear regression called the kernel function. However, SVR tries to transform the data into two dimensions with a decision, upper and lower boundary, as shown in Figure 2.2.

The following function is used by SVR (Bhoraskar, 2019):

$$\hat{y} = f(x) = \langle w^T, x \rangle + b \tag{2.5}$$

Where $w \in R^N$ and $n \in R$. ensures that data points within a radius of ε are not penalized, meaning $\hat{y} - y \leq \varepsilon$



$f(x) = \langle w^T, x \rangle + b$
$f(x) = \langle w^T, x \rangle + b + \varepsilon$
ε deviation
ξ_i^*
ξ_i
$f(x) = \langle w^T, x \rangle + b - \varepsilon$

Figure 2.2: Illustration of SVR Algorithm - Classification
Source: Bhoraskar (2019)

There are different choices for kernels, functions that transform data into a high dimensional space allowing non-linear relationships (Vovk, 2013), in SVR. These are (Bhoraskar, 2019):

$$Linear\ Kernel : k(x, x_i) = x^T x_i \tag{2.6}$$

$$Polynomial\ Kernel : k(x, x_i) = (1 + x^T x_i)^d \tag{2.7}$$

$$Gaussian\ Radial\ Bias\ Kernel : k(x, x_i) = \exp\left(-\frac{|x - x_i|^2}{2\sigma^2}\right) \tag{2.8}$$

Where d is the degree of the polynomial in the polynomial kernel, and σ is a parameter defining the width of the kernel in the RBF kernel. The main disadvantage of using SVR is tuning multiple variables to find appropriate values for all free variables in the equations. The value of ε affects the model's accuracy, and a low value may lead to overfitting, while a high value may lead to underfitting (Bhoraskar, 2019).

2.1.2.3 Decision Tree Regression

According to Loh (2011) and Breiman (2017), decision tree regression is a supervised learning algorithm that predicts the target variable based on input features. It builds a tree-like model, where internal nodes represent decision rules and leaf nodes represent predictions. Decision tree regression recursively partitions the input feature space into subsets based on the input feature value. The predicted output for the input feature vector x is represented as:

$$y(x) = \sum_{m=1}^{M} c_m I(x \in R_m) \tag{2.9}$$

Where c_m is the constant value associated with the leaf node m, I is the indicator function that returns one if x belongs to the region R_m and 0 otherwise, and M is the total number of leaf nodes in the decision tree (Breiman, 2017).

The algorithm selects the best feature to split the data based on the reduction in the variance of the target variable. The variance reduction is measured using the mean squared error (MSE), given by (Breiman, 2017; Loh, 2011):

$$\Delta = \frac{1}{N}\left[\sum_{x_i \in R_1} (y_i - \bar{y}1)^2 + \sum x_i \in R_2 (y_i - \bar{y}_2)^2\right] \tag{2.10}$$

where Δ is the reduction in variance, N is the total number of samples, y_i is the target variable for the i^{th} sample, $\bar{y}_1$ and $\bar{y}_2$ are the mean target variables for the two subsets R_1 and R_2, respectively (Breiman, 2017; Loh, 2011).

Once the best feature and threshold value are selected, the algorithm splits the data into two subsets. Then, it recursively applies the process to each subset until a stopping criterion is met. The stopping criterion can be based on the maximum depth of the tree, the minimum number of samples required to split a node, or the minimum variance reduction needed to perform a split (Breiman, 2017; Loh, 2011).

2.1.2.4 Gradient Boosting Regression

Gradient boosting regression is a popular machine learning algorithm that produces a prediction model as an ensemble of weak prediction models, typically decision trees. The basic idea is to add new decision trees to the model iteratively, each one correcting the errors of the previous ones until the overall error is minimized (Natekin and Knoll, 2013).

At each iteration, the algorithm fits a decision tree to the negative gradient of the loss function concerning the current model's prediction. This means that the new tree focuses on the examples that the current model misclassified or predicted with high error. The prediction of the new tree is then added to the current model, with a weight determined by a learning rate hyperparameter. This process is repeated for a fixed number of iterations or until the error reaches a threshold (Mayr et al., 2014; Natekin and Knoll, 2013).

The prediction of the final model is given by the sum of the predictions of all the individual trees, weighted by their learning rate. Formally, let $\hat{y}_i$ be the predicted output of the final model for input x_i, M be the total number of trees, $f_m(x)$ be the predicted output of tree m for input x, and α be the learning rate (Natekin and Knoll, 2013). Then:

$$\hat{y}_i = \sum m = 1^M \alpha f_m(x_i) \tag{2.11}$$

The choice of loss function depends on the problem and can be customized. A common choice for regression problems is the mean squared error (Natekin and Knoll, 2013):

$$L(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.12}$$

where y_i is the true output for input x_i and $\hat{y}_i$ is the predicted output of the final model.

Figure 2.3 provides an example of how gradient boosting regression works with decision trees. Initially, the model predicts a flat line that does not fit the data well. Next, the negative gradient of the loss function concerning this prediction is computed and used to train a new decision tree that captures the trend in the data. The new decision tree is then added to the model, with its weight determined by the learning rate, resulting in a better fit. This process is repeated for several iterations until the error is minimized (Mayr et al., 2014).

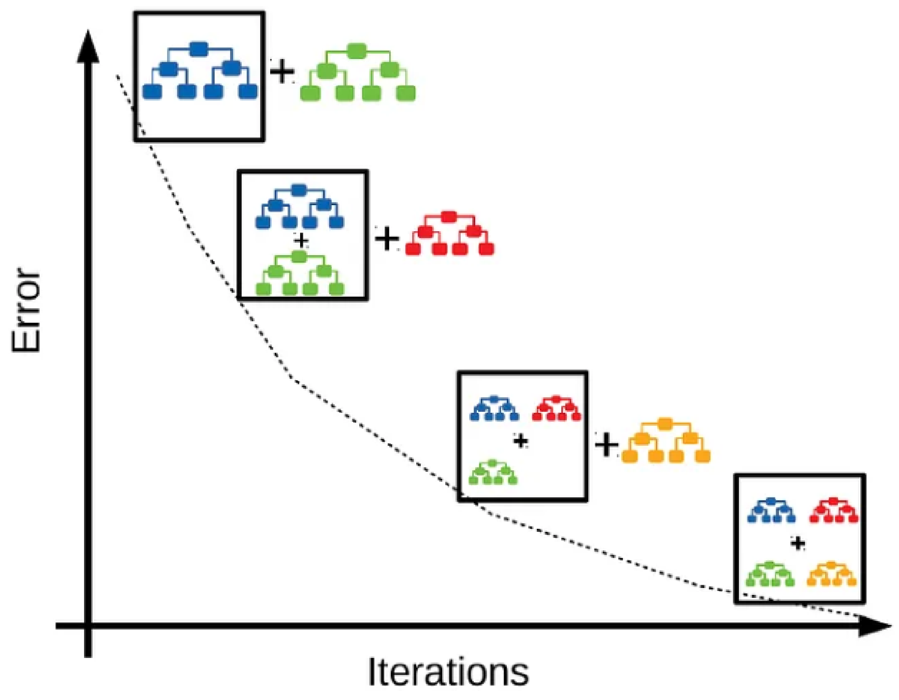


Figure 2.3: Illustration of Gradient Boosting Regression
Source: Pal (2020)

In practice, gradient boosting regression can be used for various regression problems, such as predicting housing or stock prices. It is a robust algorithm that often outperforms other regression algorithms, but it can be sensitive to overfitting and requires careful hyperparameter tuning to achieve optimal performance (Bentéjac et al., 2021).

2.1.2.5 Random Forest Regression

Random Forest Regression is a versatile supervised learning algorithm that uses an ensemble of decision trees to predict continuous numerical data (Segal, 2004). It can handle missing values, nonlinear relationships, and high-dimensional datasets (Li et al., 2013). According to Bhoraskar (2019), to construct multiple decision trees, the algorithm randomly selects subsets of training data and features, which helps to prevent overfitting and improve accuracy. Each decision tree is grown using a random set of features, and nodes are split based on the best feature and value. The final prediction is made by averaging the predictions from all the trees.

Decision trees narrow the range of possible values through a series of questions to arrive at a prediction. Random forests combine at least two decision trees (Li et al., 2013). Figure 2.4 briefly explains how decision trees work.

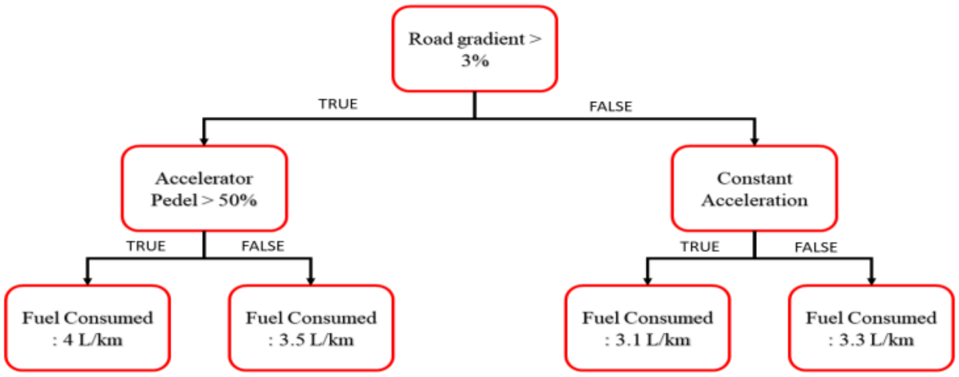


Figure 2.4: Illustration of Decision Trees
Source: Bhoraskar (2019)

2.1.2.6 Artificial Neural Networks

According to Chollet (2021), Artificial Neural Networks (ANNs) are a subset of machine learning algorithms inspired by biological neuron models. They can be used for classification and regression tasks, but in this study, they are used for regression since the output is a continuous variable. ANNs consist of interconnected nodes, with each node representing a neuron. A neuron in ANNs is activated when a particular threshold is exceeded, and the weighted sum of inputs determines its activation. Each input is associated with a weight, and the summation of inputs (multiplied by their weights) determines whether the threshold is exceeded.

ANNs can have multiple layers of interconnected nodes, allowing for more complex computations. Figure 2.5 shows the arrangement of an artificial neuron or a node in ANNs. ANNs are trained by adjusting the weights on the connections between nodes to minimize prediction error. They can be used to model complex nonlinear relationships between input and output variables (Bishop, 2006).

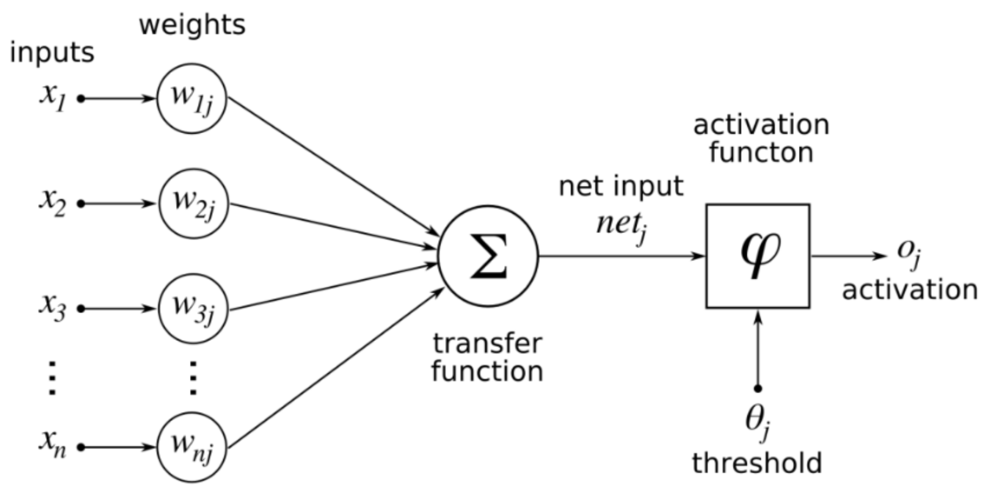


Figure 2.5: Illustration of ANNs
Source: Krogh (2008)

Where $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$; $\mathbf{w} = [w_{1,1}, w_{1,2}, \ldots, w_{1,n}]^T$ represents the input data and weights connecting the input layer to the first hidden layer in an artificial neural network (ANN), respectively (Krogh, 2008), the activation of the hidden layer neurons is determined by applying a transfer function f, such that $z_j = f(\sum_{i=1}^{n} w_{i,j} x_i)$, where z_j is the activation of the j^{th} neuron in the hidden layer. The threshold, denoted as θ, determines whether the neuron is activated, and the activation function, represented as g, is used to determine the neuron's output based on its activation. The output of the j^{th} neuron in the hidden layer is denoted as $o_j = g(z_j - \theta_j)$. The process of forward propagation in an ANN can be summarized by the equation (Chollet, 2021; Krogh, 2008):

$$\mathbf{o} = \mathbf{g}(\mathbf{W}^T \mathbf{x} - \theta) \tag{2.13}$$

Where $\mathbf{W}$ is the weight matrix connecting the input layer to the hidden layer, $\mathbf{g}$ is the activation function applied element-wise to the vector $\mathbf{W}^T \mathbf{x} - \theta$, and θ is the vector of thresholds (Chollet, 2021; Krogh, 2008).

Finally, Table 2.2 shows the six regression algorithms and their corresponding parameters that can be tuned.

Table 2.2: Regression Algorithms and Corresponding Hyper-parameters. Source: Pedregosa et al. (2011)

Algorithm	**Parameters**
Multilinear Regression	None
Support Vector Regression	• Kernel • C • gamma
Decision Tree Regression	• Max depth • Min samples split
Gradient Boosting Tree Regression	• Learning rate • Max depth
Random Forest Regression	• Number of trees • Max depth • Min samples split
Artificial Neural Networks	• Number of hidden layers • Number of neurons • Learning rate

2.1.3 Machine Learning Applied to Fuel Consumption of Haulage Trucks

Machine learning has been widely used in the mining industry to accurately predict fuel consumption, leading to improvements in haul road designs, mine plans, and decisions from dispatchers in fleet management systems (Alamdari et al., 2022). In recent years, several studies have been conducted to identify the critical determinants of fuel consumption in mining trucks and develop models to predict fuel consumption accurately.

One study by Siami-Irdemoosa and Dindarloo (2015) presented an artificial neural network (ANN) model that used different haulage truck features, such as payload and cycle time, to predict fuel consumption per cycle. After training and testing, the authors concluded that their chosen ANN provided an accurate prediction of fuel consumption with a mean absolute error

(MAE) of 10%. Furthermore, they found that queuing was a needless energy consumption, which can be minimized to improve fuel efficiency.

Similarly, Dindarloo and Siami-Irdemoosa (2016) conducted a study on a surface mine of haulage trucks involving different models and load capacities. The authors used regression analysis to identify features significantly impacting fuel consumption per cycle. They found that queuing times of shovels have the highest impact on fuel consumption when analyzed by different regression techniques, such as partial-least square regression. The authors suggested that different dispatching strategies could help minimize queuing of truck-shovel allocations, resulting in better fuel usage and cost-effectiveness for the haulage business unit. In addition, they identified engine power, distance traveled, and haul road grade as the most significant determinants of fuel consumption in mining trucks.

In addition, in 2016, Soofastaei et al. (2016) studied how payload variance on haulage trucks influenced fuel consumption and gas emission in a surface mine operation and how the relationship between them was by using different linear and non-linear correlation metrics; the authors concluded that payloads loaded on haulage trucks in surface mining operations have a significant impact on their fuel consumption and gas emissions. The weight of the payload carried by trucks is a primary factor affecting fuel consumption. A heavier payload increases the power required to accelerate, climb, and maintain the truck's speed, which results in higher fuel consumption. In addition, the engine has to work harder to move the heavier load, which reduces its efficiency, leading to further fuel consumption.

Moreover, a heavier payload increases the truck's gas emissions, mainly greenhouse gases such as carbon dioxide (CO2). The engine's efficiency decreases when the truck carries a heavier payload, which results in higher gas emissions. The increase in gas emissions the truck produces can have significant environmental impacts, including air pollution and climate change. Even though Soofastaei et al. (2016) does not apply any machine learning algorithms but studies the relationship between payload and fuel consumption, it gives this research a hint on which features to use to predict fuel consumption.

Furthermore, Wang et al. (2021) focused on developing a model to predict fuel consumption patterns of haulage trucks in open-pit mines. They proposed a model based on multi-dimensional features and XGBoost, a gradient-boosting algorithm. The study showed that the model accurately predicted fuel consumption per cycle with a correlation coefficient of 0.93 and a mean absolute percentage error of 8.78%. The authors also evaluated the weight of all features used to train the XGBoost algorithm.

Recently, Alamdari et al. (2022) proposed different machine learning techniques to predict the fuel consumption of haul trucks in open-pit mines. Their models, support vector regression, random forest, and artificial neural networks were based on four input parameters, including the weight of the loaded truck, the distance traveled, the slope of the road, and the altitude of the mine to predict fuel consumption rate in liters per hour. The authors collected data from a large open-pit mine in Iran and trained and tested the different machine learning algorithms. The results showed that the artificial neural network model was more accurate in predicting fuel consumption, with a mean absolute error of 13.44 liters per hour and an accuracy of 90%. The authors suggested the model can be used in other mining operations with similar characteristics. Future research can investigate using additional input parameters, perform feature selection algorithms, and other machine learning techniques to enhance the model's performance.

In summary, these studies demonstrate the importance of using machine learning techniques to analyze the determinants of fuel consumption in mining trucks. Accurately predicting fuel consumption can help mining companies optimize their operations, reduce costs, and minimize their environmental impact. Table 2.3 provides an overview of previous authors' different machine learning algorithms to predict fuel consumption based on specific features. Future research can further explore more advanced machine learning techniques to improve fuel consumption predictions.

Table 2.3: Performance Comparison of Different Algorithms for Fuel Consumption Prediction.

Author	Algorithm	Features	Label	Metric	Result
Siami-Irdemoosa and Dindarloo (2015)	Artificial neural networks	1. Payload 2. Haul cycle status	Fuel consumption per cycle	Mean absolute percentage error (MAPE)	10%
Dindarloo and Siami-Irdemoosa (2016)	Partial-least squares regression	1. Haul cycle status 2. Payload	Fuel consumption per cycle	MAPE	6.01%
Wang et al. (2021)	XGBoost	1. Haulage distance 2. Cycle time 3. Uphill distance	Fuel consumption per cycle	MAPE	8.78%
Alamdari et al. (2022)	Several regression algorithms (ANN winner)	1. Payload 2. Total resistance 3. Speed	Fuel consumption per hour	R squared	90%

2.2 Fuel Dispatching

Fleet Management Systems (FMSs) are widely used in surface mining operations because they automate data collection, production reporting, truck optimization algorithms, and real-time monitoring of equipment conditions. FMS providers, such as Caterpillar's Minestar and Modular Mining Systems' DISPATCH, are integrating supporting modules into their systems to make them more appealing to mining companies. One such module is the fueling module (Carter, 2012).

A case study conducted in an Australian coal mine operation found that 95% of fueling assignments to haulage trucks were scheduled by dispatchers when fuel levels dropped below 40%. At the same time, the remaining 5% occurred with fueling levels between 30 and 27%, resulting in unproductive trips to and from the fuel bay (Modular Mining, 2019). To address this issue, Modular Mining gradually lowered the fueling ratios until they worked with ratios between 24% and 20% to assign trucks to fuel. This significantly reduced non-productive travel time across the fleet from 80 hours to 42 hours. However, this technique must be tailored to each mine site's unique characteristics, such as road networks and locations for loading, dumping, and fueling areas.

To enhance fuel dispatching, Caceres and Wells (2017) presented a case study in a large surface copper mine to increase filling volumes during fueling, reduce wait time at fuel locations, and minimize person-hours required to schedule trucks to fuel manually. The improvements achieved are shown in Table 2.4. The authors developed an algorithm that estimated the remaining fuel hours of trucks based on the truck model, status, and travel terrain (uphill or downhill). They suggested two types of thresholds: one below 40% of fuel level, which would include a truck when running the assignment algorithm, and critical with 10% of fuel level, which would force the assignment of a truck. The algorithm had fixed and mobile fuel stations and constrained trucks to fuel when exiting a dumping location while traveling empty. This fuel dispatching algorithm estimated the projected travel time of a truck (average of recorded data from FMS) and the projected queuing and fueling time and assigned the fuel location with the lowest sum of traveling, queuing, and fueling time. Although the authors explained the functionality of their algorithm, they did not provide the mathematical formulation of the optimization model, which requires further investigation.

Table 2.4: Improvements of Fueling Dispatching
Source: Caceres and Wells (2017)

Indicator	Fueling Assignment	
	Before	After
Fill volume per filling	3,400 liters	3,900 liters
Wait time at fuel location		Decreased 10%
Person-hours		Decreased 2,000 hours each year

In addition, Leonida (2022) mentioned that Modular Mining Systems' FMS, DISPATCH, offers a fueling module within its product, which sends haulage equipment to fuel when needed, maximizing fuel utilization and minimizing unnecessary travel to fueling stations. This fueling module is assumed to run an optimization model with a multi-objective function. However, more information is needed on how the fueling module works, and its optimization model may need to be simplified - the objective functions can have conflicts when seeking the optimal solution(s). Rardin (2016) suggested that single-objective models are preferable over multi-objective optimization models since they are more tractable.

Although optimization techniques have been presented by Caceres and Wells (2017) and Leonida (2022) as solutions for the fuel dispatching problem, Khorasgani et al. (2020) stated that fuel dispatching is typically arranged when trucks are low on fuel and then assigned to the nearest fuel station. Assigning trucks to specific tasks is related to the assignment problem, where the assignee (such as machines, people, etc.) is assigned particular tasks. Assignment problems are one of the primary types of integer programming problems that seek the best assignments to optimize an objective function (Hillier and Lieberman, 2010).

Integer programming (IP) programs are optimization models that use linear constraints and objective functions, considering discrete values for the decision variables (Rardin, 2016). IP models are widely used in the mining industry, such as in long-term mine planning to solve the ultimate-pit limit problem and in short-term mine planning to allocate shovels and loaders with haulage trucks (Newman et al., 2010; Moradi and Askari-Nasab, 2019). Optimization solvers, such as Gurobi, Solver in Excel, and CPLEX, are used to run these programs, which offer student licenses and differ in time complexity for different generic optimization problems (Anand et al., 2017).

In conclusion, FMSs are used in mining operations due to their various automated features and real-time monitoring capabilities, with fuel dispatching and monitoring among them. However, some providers, such as Minestar from Caterpillar and Modular Mining Systems, do not disclose information about their dispatching algorithms because of proprietary information. This thesis assumes that the optimization models of fueling modules from FMS providers are kept hidden due to competitors and the competitive advantages of each in the market for FMS in the mining industry. Furthermore, there have been case studies where some features of fueling modules in FMS have been shown, yet full disclosure of the data or the logic on how they work has yet to be provided.

Moreover, machine learning has been ”mining” the data recorded in FMS thanks to the increased hardware and improved algorithms (Khorasgani et al., 2020). Its ability to predict data is appealing for usage in this research to find relationships between the behavior of haul trucks and fuel consumed.

Finally, the literature review shows that even though fuel dispatching of haul trucks, included in FMS, can improve the productivity of the loading and hauling process, there needs to be full disclosure on how they work or would work. Therefore, this research presents a technique using machine learning and optimization models to improve fueling operations in surface mines, machine learning to predict fuel consumption of haul trucks, and optimization models to maximize the match factor of truck-shovel allocations when fueling is needed.

2.2.1 Linear Programming

Linear programming is a widely used mathematical optimization technique to solve problems with a linear objective function subject to linear constraints that are either maximized or minimized. LP problems have various applications in engineering, economics, and business (Hillier and Lieberman, 2010). The general form of LP problems can be represented as follows (Sen, 2009):

$$\text{Maximize or Minimize } Z = c_1x_1 + c_2x_2 + \cdots + c_nx_n \tag{2.14}$$

subject to:

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq b_1 \tag{2.15}$$

$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \leq b_2 \tag{2.16}$$

$$\ldots \tag{2.17}$$

$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \leq b_m \tag{2.18}$$

and

$$x_i \geq 0, \forall i = 1, 2, ..., n \tag{2.19}$$

Where x_1, x_2, ..., x_n are decision variables, c_1, c_2, ..., c_n are coefficients of the objective function, a_{ij} and b_i are coefficients of the constraints. The objective function is either maximized or minimized, and the constraints are linear inequalities.

Integer programming (IP) is a linear problem (LP) problem where the decision variables must take integer values. The general form of an IP problem can be represented as follows (Sen, 2009):

$$\text{Maximize or Minimize } Z = c_1x_1 + c_2x_2 + \cdots + c_nx_n \tag{2.20}$$

subject to:

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq b_1 \tag{2.21}$$

$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \leq b_2 \tag{2.22}$$

$$\cdots \tag{2.23}$$

$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \leq b_m \tag{2.24}$$

and

$$x_i \geq 0, \forall i = 1, 2, \ldots, n \tag{2.25}$$

$$x_i \in Z, \forall i = 1, 2, \ldots, n \tag{2.26}$$

Where $x_i \in Z$ represents that the decision variables must take integer values. The feasible region of IP problems is restricted to discrete values, which requires more computational effort than LP problems (Rardin, 2016).

While linear programs have linear objective functions and constraints, non-linear programming (NLP) is an optimization problem where the objective function or constraints are non-linear. These types of problems are generally more challenging to solve than LP problems because the solutions cannot be found by simple mathematical operations (Hillier and Lieberman, 2010).

Non-linear programs require specialized optimization techniques such as heuristic search methods or gradient-based optimization techniques. Heuristic methods use rules of thumb, experience, or trial-and-error to generate reasonable solutions. On the other hand, gradient-based optimization techniques rely on using calculus to find the optimal solution (Sen, 2009).

Several methods for solving LP problems are available, including the simplex, interior-point, and branch-and-bound methods for integer programming (IP) problems. The simplex method is the most widely used method for solving LP problems, and it works by iteratively improving the feasible solution until the optimal solution is reached. Interior-point methods are a more recent development that solves the dual of the LP problem by minimizing a barrier function (Rardin, 2016).

Overall, the choice of optimization method will depend on the specific problem requirements and constraints. Therefore, different methods may have different strengths and weaknesses, and the best approach will depend on factors such as problem size, complexity, and the available computational resource (Rardin, 2016; Hillier and Lieberman, 2010).

Chapter 3

Methodology

3.1 Machine Learning Algorithm Selection

Various studies (Alloghani et al., 2020; Maulud and Abdulazeez, 2020; Lee and Shin, 2020) have investigated supervised and unsupervised machine learning techniques for data science, including linear and multi-linear regression algorithms. These studies have discussed various applications of machine learning in enterprises and provided a list of regression algorithms that can be used to solve a regression problem. The list includes linear and multi-linear regression, support vector regression, decision tree regression, gradient boosting regression, random forest regression, and artificial neural network algorithms.

To select the best algorithm for a regression problem, Doan and Kalita (2015); Ray (2019) suggest balancing accuracy and simplicity. Accuracy measures how well the algorithm predicts, while simplicity refers to how easy it is to interpret the algorithm. For example, the following steps can be used to select the best algorithm based on accuracy (Hastie et al., 2009; James et al., 2013):

1. **Data Splitting:** The first step is to split the data into training and testing sets. The training set will be used to train the algorithms, while the testing set will be used to evaluate their performance. This research will split the dataset into 80% for training and 20% for testing when training the multi-linear regression algorithm with its closed-form solution (Joseph and Vakayil, 2022). However, cross-validation will be used for the additional algorithms to find the best hyperparameters and overcome the disadvantage of not having a large dataset for training and testing.

   - **Cross-validation:** Involves splitting the data into several subsets and using each subset as a testing set while training on the remaining subsets. K-fold cross-validation is commonly used, where the data is split into k subsets, and each subset is used as a testing set once (Anguita et al., 2012). The value of k will be determined through *For Loops* up to 10 to avoid computation limitations Anguita et al. (2012), and the value of k that achieves the highest accuracy will be chosen.

- **Feature Selection and Engineering:** Feature selection and engineering will be performed based on the collected data, features matched with the literature, and insights from the multilinear regression model. Linear regression can provide information on which features to select, as suggested by Guyon and Elisseeff (2003). The goal is to choose the most relevant features and create new ones (e.g., combining two of them) that can improve the accuracy of the predictive model.
- **Parameter Tuning:** The machine learning algorithms used in this problem are parametric, and some of their parameters are hyperparameters that need to be input by the developer/user. Several techniques can be used to tune these hyperparameters, such as grid search or random search, to search through a range of hyperparameter values and select the optimal combination of hyperparameters that produce the best results. Grid search is proven to find the best hyperparameters, but it is computationally expensive (Shahhosseini et al., 2022). Therefore, this research will use grid search to find the hyperparameters for all models except for multi-linear regression and artificial neural networks, given the available resources mentioned in Table 3.2.

2. **Final Comparison:** After evaluating the algorithms using cross-validation and tuning their parameters, the best-performing algorithm will be selected based on its accuracy; coefficient of determination R^2 for regression problems as advised by Chollet (2021). However, as reported by Ray (2019), simplicity should also be considered - referring to the number of hyper-parameters to be tuned and the time taken. Therefore, a qualitative interpretation needs to be made.

Machine learning involves splitting data into training and testing sets, using cross-validation to find the best hyperparameters for some algorithms, and tuning hyperparameters using a grid or random search. The best-performing algorithm is selected based on its accuracy (R^2) and simplicity (run-time), which are evaluated by the number of parameters that need to be tuned and the computational resources required.

3.2 Binary Integer Models

The optimization model looks for improvements during the fueling process of trucks in surface mining operations by maximizing the match factor, which measures the continuity of truck-shovel allocations. A higher match factor indicates enough trucks to load (Burt and Caccetta, 2007). The match factor is calculated using Equation 3.1, where MF is the match factor, $\sum_{i=1}^{P}(trucks_i \times truck\ loading\ time_i)$ is the sum of the product of the number of trucks and their loading times for all P loading cycles, $(number\ of\ loaders)$ is the total number of loaders, and $(truck\ cycle\ time)$ is the average time taken by a truck to complete an entire cycle (loading and dumping).

$$MF = \frac{\sum_{i=1}^{P}(trucks_i \times truck\ loading\ time_i)}{(number\ of\ loaders) \times (truck\ cycle\ time)} \tag{3.1}$$

- Trucks cannot be fueled simultaneously.
- Each truck can only be fueled once.
- All trucks with a fuel ratio below 20% must be fueled.
- Each truck requires a 15-minute time window for fueling - it is assumed that all trucks necessitate a 15-minute window. However, this assumption may lead to infeasible solutions (This issue will be addressed in subsequent sections).
- The minimum fuel level ratio allowed at the fuel station should be above 0% to ensure that trucks do not run out of fuel. In reality, a 0% fuel level is not an ideal scenario, as

it would cause delays if trucks cannot start. However, for the purpose of this thesis, the chosen value is 0% to demonstrate the functionality of the optimization model.

Sending trucks to refuel creates delays for the haulage fleet. Furthermore, it reduces the number of available trucks for assigned loaders or shovels, leading to hang time and decreased productivity of the loading and hauling unit (Schmidt, 2015), which is why this research considers the match factor of truck-shovel allocations as the objective function for the optimization model.

3.2.1 Mathematical Formulations

3.2.1.1 First Formulation - Fuel Time Window

A binary integer programming model will be employed to solve the initial problem described in Section 3.2. The optimization model is defined as follows:

Sets and Indices:[1]

- $i \in I$: It represents the set of all trucks with a fuel level ratio lower than 20
- $t \in T$: It denotes the set of expected arrival times at the fuel station.
- $k \in K$: It signifies the set of all shovels loading trucks with a fuel level ratio lower than 20

- $m_{k,i,t}$: Match factor for shovel k, truck i, and time period t (unitless, ranging from 0 to 1).
- $f_{i,t}$: Fuel ratio for truck i (and any other trucks in I) during time period t (unitless, ranging from 0 to 1).
- N: Number of trucks that need to be fueled.

Decision Variables:

[1] The lower and upper bounds of these sets and indices will depend on when the optimization model is run.

- $y_{i,t}$: $\begin{cases} 1, & \text{if truck } i \text{ goes to fuel at time period } t \\ 0, & \text{otherwise} \end{cases}$

Optimization model:

- Objective Function:
  - **Match factor:** Maximizes the match factor of truck and shovel allocations (unitless, from 0 to 1) to satisfy the fueling of all trucks.

$$\text{Maximize} \quad \sum_{i \in I} \sum_{t \in T} \sum_{k \in K} m_{k,i,t} \cdot y_{i,t} \tag{3.2}$$

- Constraints
  - **One truck at a time:** Trucks should not fuel simultaneously, at most once each period.

$$\sum_{i \in I} y_{i,t} \leq 1 \qquad \forall t \in T \tag{3.3}$$

  - **Trucks fueling once:** Trucks can not fuel more than once each period.

$$\sum_{t \in T} y_{i,t} \leq 1 \qquad \forall i \in I \tag{3.4}$$

  - **All trucks should fuel:** All trucks selected with a fuel ratio below 20% should fuel

$$\sum_{i \in I} \sum_{t \in T} y_{i,t} \geq N \qquad \forall i \in I \tag{3.5}$$

  - **15-minute time window:** All trucks fueling should have a 15-minute time window to fuel; otherwise, they will be queuing for fuel.

$$\sum_{i \in I} y_{i,t} + \sum_{i \in I} \sum_{j=t+1}^{t+14} y_{i,j} \leq 1 \qquad \forall t \in T \tag{3.6}$$

  - **Fuel ratio allowed:** No trucks should run out of fuel when arriving at the fuel station.

$$\sum_{t \in T} fuelratio_{i,t} \cdot y_{i,t} \geq 0\% \qquad \forall i \in I \tag{3.7}$$

– **Binary constraint**

$$y_{i,t} \in \{0,1\} \qquad \forall (i,t) \in (I \times T) \tag{3.8}$$

The summarized optimization model is as follows (find Python code in Appendix 14):

$$\begin{aligned}
\text{Maximize} \quad & \sum_{i\in I}\sum_{t\in T}\sum_{k\in K} m_{k,i,t} \cdot y_{i,t} && && (3.9)\\
\text{subject to} \quad & \sum_{i\in I} y_{i,t} \leq 1 && \forall t \in T \quad \text{(Not at the same time)} && (3.10)\\
& \sum_{t\in T} y_{i,t} \leq 1 && \forall i \in I \quad \text{(Trucks fueled once)} && (3.11)\\
& \sum_{i\in I}\sum_{t\in T} y_{i,t} \geq N && && (3.12)\\
& \sum_{i\in I} y_{i,t} + \sum_{i\in I}\sum_{j=t+1}^{t+14} y_{i,j} \leq 1 && \forall t \in T \quad \text{(15-minute window)} && (3.13)\\
& \sum_{t\in T} fuelratio_{i,t} \cdot y_{i,t} \geq 0\% && \forall i \in I \quad \text{(No running out of fuel)} && (3.14)
\end{aligned}$$

and

$$y_{i,t} \in \{0,1\} \qquad \forall (i,t) \in (I \times T) \tag{3.15}$$

Fueling is an intrinsic process in a mining operation involving diesel trucks. Ideally, the fuel dispatching should be aligned with the allocation of trucks and shovels. However, the presence of 15-minute time windows may introduce infeasibility, particularly when two trucks arrive at the fuel station with a gap of only 3 minutes. Consequently, the researchers considered the possibility of removing Constraint 3.6, modifying it, and incorporating it into the objective function. The goal of this adjustment is to minimize the overlapping times of trucks at the fuel station.

The following optimization model presents a modification to the initial model shown in Section 3.2.1.1. This modification includes a penalty in the objective function and removes the time-window constraint.

To introduce the penalty, a new parameter $p_{i,t}$ was incorporated, representing the arrival time of truck i at the fuel station. The objective function aims to maximize the match factor $mfactor_{k,i,t}$ while simultaneously minimizing the absolute difference between the arrival times of different trucks i and h at time t. This objective is mathematically represented by the expression: $\sum_{i\in I}\sum_{h\in I,h\neq i}\sum_{t\in T}|p_{i,t}-p_{h,t}|\cdot y_{i,t}$.

Parameters:

- $m_{k,i,t}$: Match factor for shovel k, truck i, and time period t (unitless, ranging from 0 to 1).
- $f_{i,t}$: Fuel ratio for truck i (and any other relevant trucks) during time period t (unitless, ranging from 0 to 1).
- N: Number of trucks that need to be fueled.
- $p_{i,t}$: Expected arrival time of truck i at time period t (epoch timestamp).

New Objective Function:

- **Match factor and overlapping times:** Maximizes the match factor of truck and shovel allocations (unitless, from 0 to 1) to satisfy the fueling of all trucks while minimizing the overlapping times at the fuel station.

$$\text{Maximize} \quad \sum_{i\in I}\sum_{t\in T}\sum_{k\in K} mfactor_{k,i,t}\cdot y_{i,t} - \sum_{i\in I}\sum_{h\in I,h\neq i}\sum_{t\in T}|p_{i,t}-p_{h,t}|\cdot y_{i,t} \tag{3.16}$$

However, the introduction of the penalty results in non-linearity within the objective function. To address this and convert it into a linear expression, the linearization technique for the absolute value function (Hillier and Lieberman, 2010) can be employed. This involves the introduction of new variables, namely $d_{i,h,t}$, which represents the absolute difference between $p_{i,t}$ and $p_{h,t}$, and two binary variables, $b_{i,h,t}$ and $c_{i,h,t}$, which indicate whether $p_{i,t}$ is greater than or equal to $p_{h,t}$ or vice versa, respectively. The constraints required to achieve this linearization are expressed as follows:

$$d_{i,h,t} = |p_{i,t}-p_{h,t}|$$

Next, two new binary variables are introduced, $b_{i,h,t}$ and $c_{i,h,t}$, to represent whether $p_{i,t}$ is greater than or equal to $p_{h,t}$ or vice versa, respectively. These constraints can be written as:

$$p_{i,t} = p_{h,t} - M \cdot (1 - b_{i,h,t})$$

$$p_{h,t} = p_{i,t} - M \cdot (1 - c_{i,h,t})$$

where M is a sufficiently large constant. The complete set of constraints for this linearized objective function is as follows:

$$\text{Maximize} \quad \sum_{i\in I}\sum_{t\in T}\sum_{k\in K} mfactor_{k,i,t} \cdot y_{i,t} - \sum_{i\in I}\sum_{h\in I, h\neq i}\sum_{t\in T} d_{i,h,t} \cdot y_{i,t} \tag{3.17}$$

$$\text{subject to} \quad \sum_{i\in I} y_{i,t} \leq 1 \qquad \forall t \in T \qquad \text{(Not at the same time)} \tag{3.18}$$

$$\sum_{t\in T} y_{i,t} \leq 1 \qquad \forall i \in I \qquad \text{(Trucks fueled once)} \tag{3.19}$$

$$\sum_{i\in I}\sum_{t\in T} y_{i,t} \geq N \qquad \forall i \in I \qquad \text{(All trucks should fuel)} \tag{3.20}$$

$$\sum_{t\in T} fuelratio_{i,t} \cdot y_{i,t} \geq 0\% \qquad \forall i \in I \qquad \text{(No running out of fuel)} \tag{3.21}$$

$$d_{i,h,t} = p_{i,t} - p_{h,t} + M(1 - b_{i,h,t} + c_{i,h,t}), \forall i \in I, h \in I, h \neq i, t \in T \tag{3.22}$$

$$p_{i,t} \geq p_{h,t} - M(1 - b_{i,h,t}), \forall i \in I, h \in I, h \neq i, t \in T \tag{3.23}$$

$$p_{h,t} \geq p_{i,t} - M(1 - c_{i,h,t}), \forall i \in I, h \in I, h \neq i, t \in T \tag{3.24}$$

and

$$y_{i,t} \in \{0,1\} \qquad \forall (i,t) \in (I \times T) \tag{3.25}$$

$$b_{i,h,t}, c_{i,h,t} \in \{0,1\}, \qquad \forall i \in I, h \in I, h \neq i, t \in T \tag{3.26}$$

$$d_{i,h,t} \geq 0, \qquad \forall i \in I, h \in I, h \neq i, t \in T \tag{3.27}$$

Before modifying the code provided in Appendix 14, the complexity of the model and the interpretability of its parameters were considered as a concern. Therefore, the proposed modification introduces two additional decision variables and three sets of constraints, which may lead to a slowdown in the computation time of the solver. However, by simplifying the model, a better understanding of its parameters can be achieved, as exemplified in the subsequent step.

3.2.1.2 Second Formulation - No Fuel Time Window

Minimizing the overlapping times at the fuel station between trucks that require refueling is desirable. These fueling delays cannot be controlled as they are inherent to the haul cycles of trucks, but they can impact the match factor of truck-shovel allocations. Therefore, the number of trucks assigned to a shovel should be better prioritized rather than complicating the model with unnecessary constraints.

The final optimization model, which is feasible, is presented below (find Python code in Appendix 15):

$$
\begin{aligned}
\text{Maximize} \quad & \sum_{i\in I}\sum_{t\in T}\sum_{k\in K} m_{k,i,t}\cdot y_{i,t} && && (3.28)\\
\text{subject to} \quad & \sum_{i\in I} y_{i,t} \le 1 && \forall t\in T && \text{(Not at the same time)} \quad (3.29)\\
& \sum_{t\in T} y_{i,t} \le 1 && \forall i\in I && \text{(Trucks fueled once)} \quad (3.30)\\
& \sum_{i\in I}\sum_{t\in T} y_{i,t} \ge N && \forall i\in I && \text{(All trucks should fuel)} \quad (3.31)\\
& \sum_{t\in T} fuelratio_{i,t}\cdot y_{i,t} \ge 0\% && \forall i\in I && \text{(No running out of fuel)} \quad (3.32)\\
\text{and} \quad & && && \\
& y_{i,t}\in\{0,1\} && \forall (i,t)\in (I\times T) && (3.33)
\end{aligned}
$$

3.2.2 Sensitivity Analysis

According to Schrage and Wolsey (1985), sensitivity analysis can be performed on an integer programming model by modifying the right-hand side coefficients of the constraints. However, this process can become complex and cumbersome. The reader may observe this complexity when generating a sensitivity analysis report using the Solver plug-in in Microsoft Excel. Moreover, altering the right-hand side values of the proposed optimization models is not advisable, as it could increase the likelihood of obtaining infeasible solutions. Consequently, this research will refrain from providing any sensitivity analysis for the obtained results.

3.3 Coding Environment

3.3.1 Machine Learning

The coding environment for this research is built using Python 3 as the programming language. Python is an interpreted language widely used in scientific computing and machine learning due to its ease of use, flexibility, and extensive libraries (Van Rossum and Drake, 2009).

The scikit-learn (`sklearn`) library is utilized to model various regression algorithms, offering a wide range of machine-learning algorithms and tools for data preprocessing, dimensionality reduction, model selection, and evaluation (Pedregosa et al., 2011). Specifically, the following regression algorithms are implemented using `sklearn`: multi-linear regression, support vector regression, decision tree regression, gradient boosting regression, and random forest regression. To optimize the performance of these algorithms, the grid search method is applied to identify the optimal hyperparameters. The grid search method is an exhaustive search that evaluates all possible combinations of hyperparameters specified in a grid, allowing for the identification of the best combination that maximizes the chosen performance metric (Pedregosa et al., 2011).

For modeling artificial neural networks, the researchers are utilizing the `TensorFlow` and `Keras` libraries. `TensorFlow` is an open-source software library that facilitates dataflow and differentiable programming for a variety of tasks, including machine learning, deep learning, and neural networks (Abadi et al., 2015). On the other hand, `Keras` serves as a high-level neural networks API written in Python, which operates on top of `TensorFlow` (Chollet et al., 2015).

During the development process, the researchers are utilizing Jupyter Lab, an interactive development environment that allows the creation of interactive notebooks incorporating code, text, and multimedia elements. Additionally, Jupyter Lab offers a wide array of tools for visualization, data manipulation, and collaborative work (Kluyver et al., 2016).

To consolidate all the scripts and create a website application, Visual Studio Code is employed. Visual Studio Code, developed by Microsoft, is a free source code editor compatible with Windows, Linux, and macOS. Providing an integrated development environment, it supports multiple programming languages and platforms, thus offering a powerful and flexible workflow for the implementation and evaluation of machine learning algorithms in this research project.

3.3.2 Optimization Model

The mathematical formulations described in previous sections will be run using the Gurobi framework in Python. It is one of the most popular solvers for academia and industry in the world (Gurobi Optimization, LLC, 2022). Then, multiple loops will be programmed to decide when to trigger the optimization model by ensuring that the optimization model does not provide infeasible solutions and that trucks only reach the fueling station with fuel.

Table 3.1 shows a summary of the tools used.

Table 3.1: Overview of the Coding Environment

Language	Python 3
Libraries - Machine Learning	Scikit-learn, TensorFlow, Keras
Libraries - Optimization	Gurobi
Hyperparameter Tuning	Grid Search
Development Environment	Jupyter Lab
Source-Code Editor	Visual Studio Code

3.3.3 Computer Resources

Table 3.2 shows the specifications for training and testing the machine learning algorithms, developing the optimization model, and deploying the website application. It is wise to mention that running time might vary across different computer's specifications.

Table 3.2: Computer Resource Specifications

System Model	HP Z1 Tower G4 Workstation
BIOS	A50 Ver. 01.06.07
Processor	Intel(R) Xeon (R) E-2136 CPU @ 3.30 GHz (12 CPUs), 3.3GHz
Memory	32,768 RAM

3.4 Data Collection and Preparation

Data for this research was collected from a surface metal mine operation, referred to here as Mine Z for 18 shifts (9 days). Furthermore, Mine Z uses a fleet management system called FMS F, which records data automatically from sensors on loading and haulage equipment (see Table 3.3 for details), operator input, and dispatchers. Material mined is split into three types: (1) ore, (2) waste, and (3) topsoil. Ore is sent to the crusher, stockpiles, and waste to waste dumps; topsoil is stored for reclamation. Mine Z has 14 loading areas and ten dumping areas (inc. crusher) on average, which constantly change as the mine evolves; see the number of loading and dumping areas for 2 days (4 shifts) in Table 3.4 below.

Table 3.3: Mine Z Haulage Fleet
Source: FMS F

Truck Model	Fleet	Nominal Payload (tons)	Fuel Capacity (l)	Haul Cycles
KOMATSU 930E	57	290	4200[a]	10,285

Note: [a] Limited trucks have up to 5,000 liters of fuel capacity based on Mine Z

Table 3.4: Number of Loading and Dumping Areas of Mine Z per Shift

Day	Shift	Loading Areas	Dumping Areas
2022-Nov-07	Day	15	11
2022-Nov-07	Night	12	9
2022-Nov-08	Day	18	15
2022-Nov-08	Night	12	9

Data collected from FMS F is saved and stored in an SQL Server Management system, from which the following tables have been queried to perform this research:

- Nodes Location: contains information on each node in the road network
- Road Network: stores the relationship between pairs of nodes in the road network
- Load haul cycles: provides timestamp, duration, location of trucks, and distances (incl. effective flat haul) of loading haul cycles, since a truck travels empty until it starts loading.

- Dump haul cycles: provide similar information to loading haul cycles, including the hauled tonnage. Dumping cycles are measured from a truck that has started loading until it dumped the material at the assigned destination.
- Fueling information: stores truck timestamps, duration, and tank capacities when fueling.

A summary of each table queried from FMS F is described in Table 3.5. In addition, Figure 3.1 shows a plan view of Mine Z built using the Road Network and Nodes location tables. The road network shows the extension of the mine site, and the connections between every node dispatchers have drawn in FMS F. These connections are shown as gray arrows below, and the directions of each follow the order of the Road Network.

Table 3.5: Data collected from FMS F for Mine Z

Tables queried	Content (units):
Nodes Location:	1. Name of node (text) 2. Coordinates of node's location (float numbers)
Road Network: Connection between nodes	1. First node (text) 2. Last node (text) 3. Distance between nodes (float numbers in meters)
Load haul cycles: Description of each truck's loading cycle	1. Id of loading cycle (whole number) 2. Truck name (text) 3. Load location (text) 4. Duration and timestamp for each status, e.g., spotting. 5. Effective flat haul distance (float numbers in meters)
Dump haul cycles: Description of each truck's dumping cycle	1. Previous load cycle id (whole number) 2. Truck name (text) 3. Dump location (Text) 4. Tonnage (whole number in tons) 5. Duration and time for each status, e.g., loading 6. Effective flat haul distance (float numbers in meters) 7. Estimated traveling time (float numbers in seconds)
Fueling information: Information on fueling for haulage trucks	1. Truck name (text) 2. Fuel start time (timestamp) 3. Fuel end time (timestamp) 4. Fuel quantity (liters) 5. Fuel capacity before fueling (liters) 6. Fuel capacity after fueling (liters)

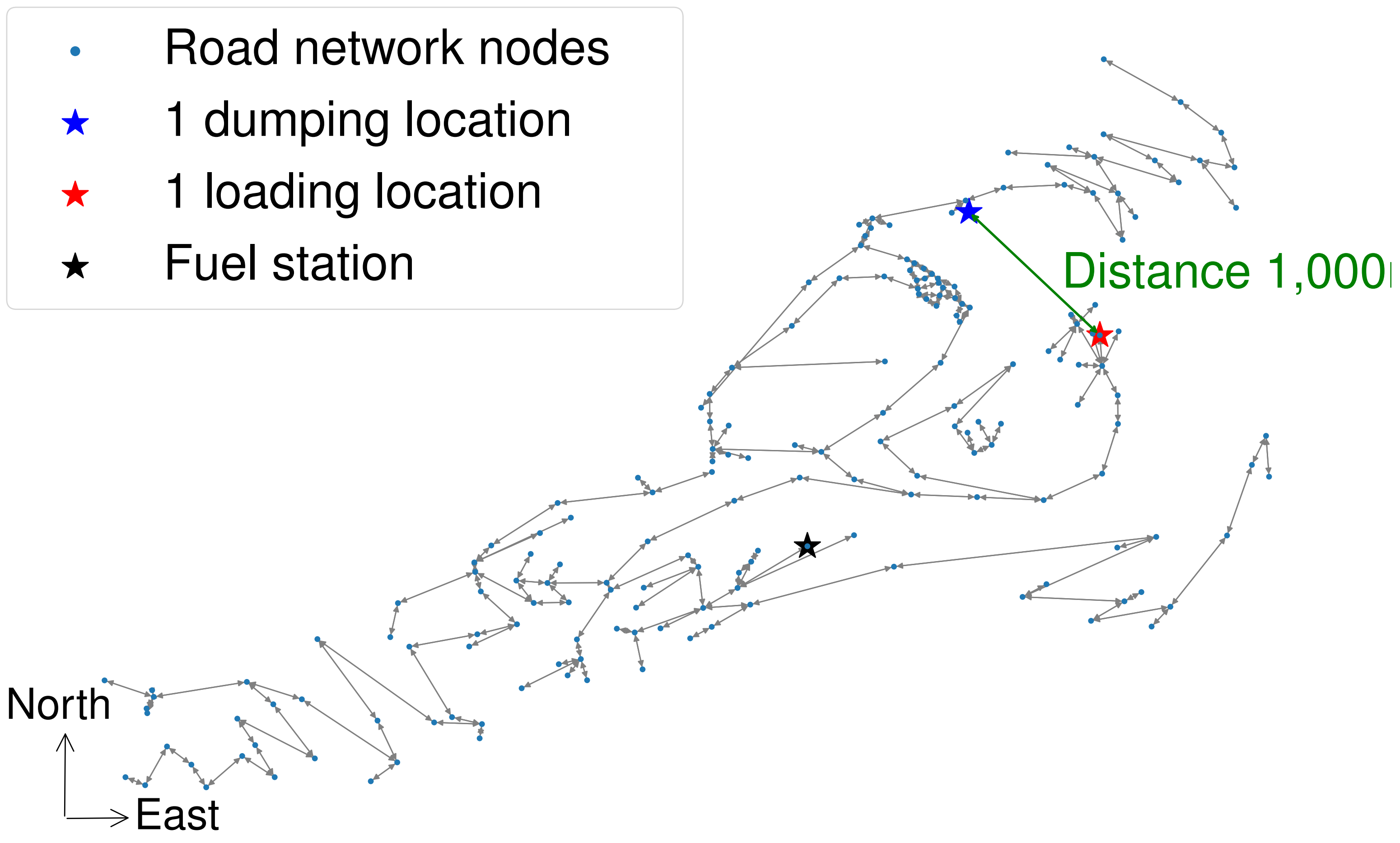


Figure 3.1: Plan View of the Road Network - Mine Z

3.4.1 Data Preparation

3.4.1.1 Distances from/to Fuel Stations

Loading and dumping haul cycles contain information about distances, (1) from a dumping location, when a truck is empty, to a loading location, where a truck will load, and (2) from a loading location, where a truck got loaded to a dumping location, where a truck will dump. However, FMS F does not record any information on distances when:

- Trucks are going to fuel after dumping at an assigned destination,
- And when trucks go to their assigned loading area after fueling.

To compute these distances, the built-in shortest-path function of the `NetworkX` framework in Python (Hagberg et al., 2008) was used to query the road network and get the distances (1) from dumping locations to the fuel stations and (2) from fuel stations to the next loading assigned area of trucks. Table 3.6 and Table 3.7 show a summary sample of these distances after analyzing the 662 times the haulage fleet fueled. For instance, Table 3.6 begins with the time assigned to the fuel stations, and Table 3.7 when the truck travels to the load area and when it arrives there.

Table 3.6: Distances from Dump Locations to the Fuel Station

Truck	Timestamp when empty	Dump locations	Distance to fuel station (m)
CDH66	11/7/2022 9:13	BSF_3570_EST_SUR	1,956
CDH63	11/7/2022 9:22	BSF_3590_LIX	2,294
CDH62	11/7/2022 10:09	BSF_3570_EST	2,132
CDH70	11/7/2022 10:19	BSF_3590_LIX	2,294

Table 3.7: Distances From the Fuel Station to Load Locations

Truck	Distance from fuel station (m)	Load Location	Arrival timestamp
CDH81	5,301	I7A-3205-310	11/7/2022 15:41
CDH57	5,661	D02-3730-312	11/7/2022 17:57
CDH71	5,042	STK_3600_PRIM	11/7/2022 18:22
CDH93	5,078	I7A-3205-300	11/7/2022 22:24

These distances are computed to provide more accurate information to the machine learning models as loading and dumping cycles do not store distance when (1) trucks go to fuel and (2) when they return from the fuel station to their assigned load location.

3.4.1.2 Match Factor Estimation

(Burt and Caccetta, 2007) proposed a formulation to calculate the match factor of truck-shovel allocations by considering the loading time of shovels and the cycle time of trucks (the equation is reiterated below). For this research, the database employed contains recorded information on load and haul cycles (refer to Table 3.5). By merging these two database tables using a primary key, the cycle times of trucks are extracted, encompassing the duration from when they began traveling empty until they completed material dumping.

$$MF = \frac{\sum_{i=1}^{P}(trucks_i \times truck\ loading\ time_i)}{(number\ of\ loaders) \times (truck\ cycle\ time)} \tag{3.34}$$

If a truck is going to fuel, a shovel will have one truck less to load ($P-1$, in Equation 3.34) because the truck will go into a delay; hence the match factor is decreased. This is how the third formulation of the optimization model, in Section 3.2.1.2, embeds the delays brought up by trucks' fueling.

3.4.1.3 Fuel Level Ratio Estimation

Once the best machine learning model is selected, its structure will be saved (e.g., `HDF5` files for neural networks models run with `TensorFlow` (Xing et al., 2018). The main reason behind keeping the machine learning model in a file is that it can be used easily by application programming interfaces (APIs) when developing applications (e.g., website, mobile).

Next, the fuel level ratio for all trucks is computed by querying the databases using the procedure depicted in Figure 3.2. The `Start` point in the flowchart represents the date at which the fuel level ratio is computed. Subsequently, the last fueling time and the corresponding amount fueled are retrieved for all trucks. Additionally, the distances traveled by the trucks when empty and loaded (EFH distances) and the tonnage hauled before the `Start` date are obtained as input for the machine learning (ML) model. The ML model then calculates the burnt fuel, and in conjunction with the amount fueled, the fuel level ratio is determined.

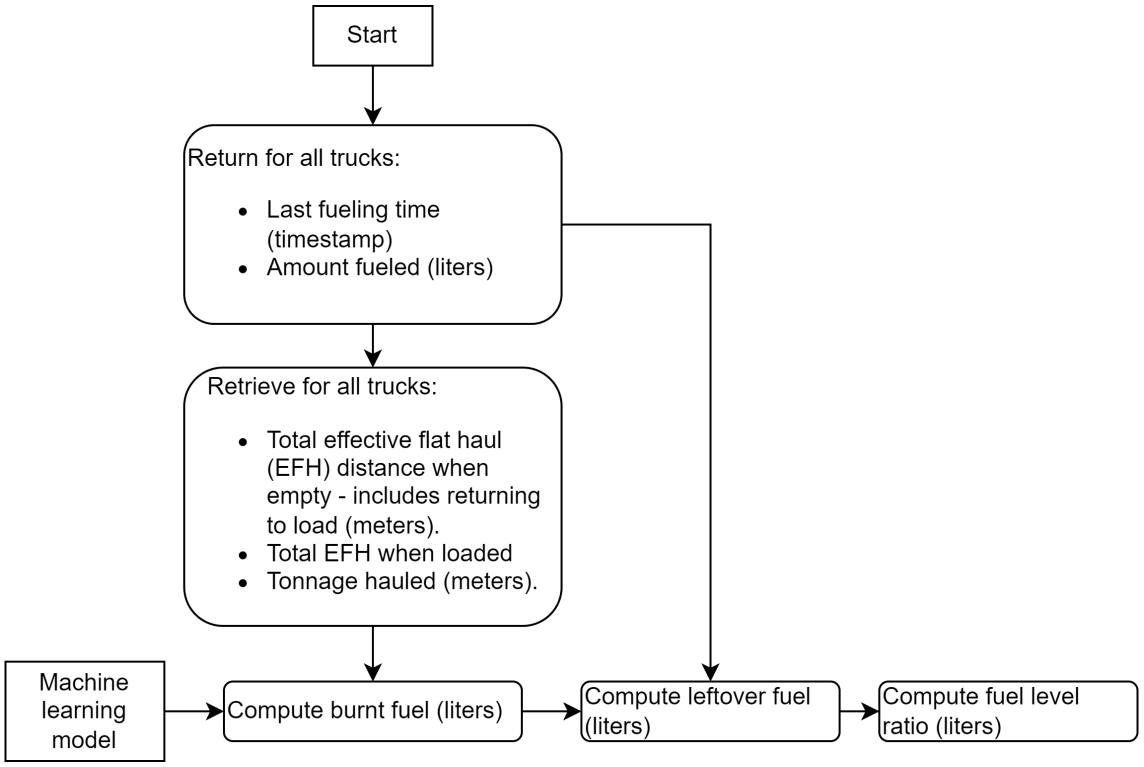

Start
Return for all trucks:
Last fueling time (timestamp)
Amount fueled (liters)
Retrieve for all trucks:
Total effective flat haul (EFH) distance when empty - includes returning to load (meters).
Total EFH when loaded
Tonnage hauled (meters).
Machine learning model
Compute burnt fuel (liters)
Compute leftover fuel (liters)
Compute fuel level ratio (liters)

Figure 3.2: Process of Estimating Fuel Level Ratio

3.4.1.4 Estimated Time of Arrival at the Fuel Station

The estimated time of arrival at the fuel station is a crucial parameter required for the optimization models discussed in previous sections (refer to Section 3.3.2). Figure 3.3 outlines the steps involved in computing this parameter. After calculating the fuel level ratio (as previously done in subsection 3.4.1.3), a threshold is applied to select trucks that are in need of refueling. Subsequently, if at least one truck requires fueling, the expected arrival time will be retrieved from the `Dumpcycles` database table, which contains this data for each truck. Then, the ETA at the fuel station will be determined using an average speed when the truck is empty.

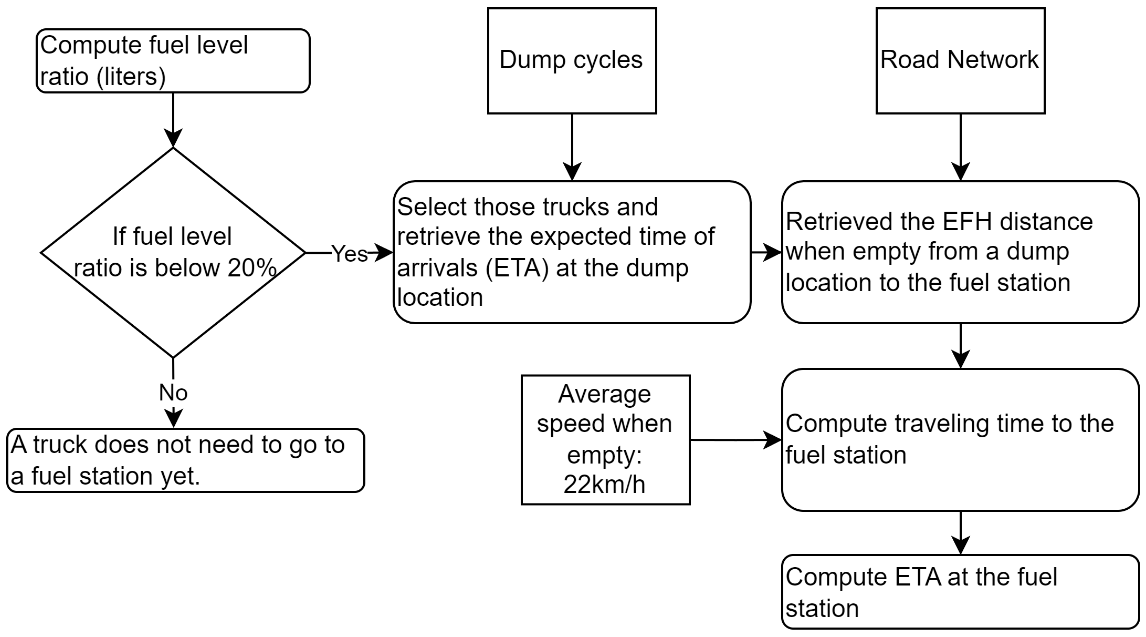

Compute fuel level ratio (liters)
If fuel level ratio is below 20%
Yes
No
A truck does not need to go to a fuel station yet.
Dump cycles
Select those trucks and retrieve the expected time of arrivals (ETA) at the dump location
Road Network
Retrieved the EFH distance when empty from a dump location to the fuel station
Average speed when empty: 22km/h
Compute traveling time to the fuel station
Compute ETA at the fuel station

Figure 3.3: Process of Comparing Current Assignments Against Results of the Optimization Models

3.4.1.5 Comparison with Current Fuel Assignments

Once the proposed optimization models yield optimal results, they will be compared against the current manual assignments based on the match factor of truck-shovel allocations. For instance, as shown in Figure 3.4, when the optimal schedules are computed, the latest ETA of all trucks that need fueling will be retrieved to query the `Fuelinginformation` database and acquire the current assignments. Subsequently, the match factor of the current assignments will be compared against that of the proposed optimization model, thereby highlighting the contributions of the optimization model.

3.4.2 Data to Train Machine Learning Models

The fleet management system F has a fuel module feature that calculates fuel consumption rates based on payload, truck model, and road gradient factors. This calculation aligns with previous research by Caceres and Wells (2017), which utilized truck status, model, and uphill and downhill travel to estimate the remaining fuel hours for trucks. Therefore, this work will analyze the effective flat haul (EFH) distances for loading and dumping cycles, EFH distance when going to fuel and returning to load, and payloads to estimate fuel consumption rates,

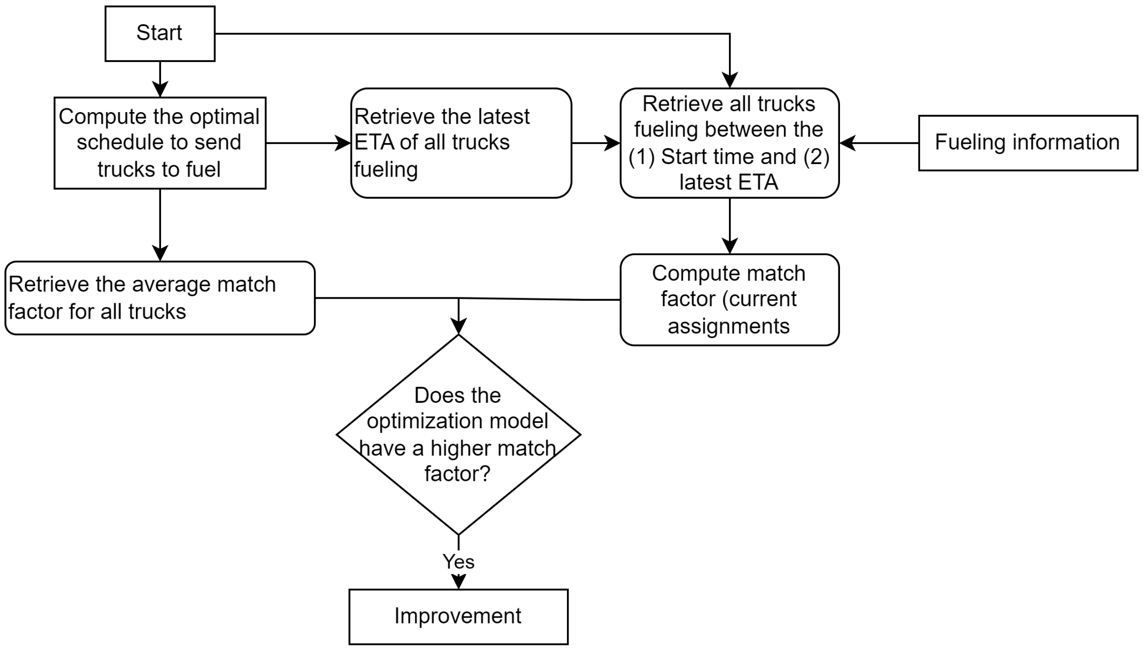

Start
Compute the optimal schedule to send trucks to fuel
Retrieve the latest ETA of all trucks fueling
Retrieve all trucks fueling between the (1) Start time and (2) latest ETA
Fueling information
Retrieve the average match factor for all trucks
Compute match factor (current assignments
Does the optimization model have a higher match factor?
Yes
Improvement

Figure 3.4: Process of Comparing Current Assignments with the Optimization Model

as depicted in Table 3.8. For instance, it was observed that a truck consumed 2,883 liters of fuel while hauling 1,530 tons over 31,451 m (empty) and 40,831 m (full) in effective flat haul distances for the first data point in Table 3.7.

Table 3.8: Dataset to Train Machine Learning Models

Model	EFH[a] empty (m)	EFH[a] full (m)	Tonnage (tons)	Fuel Cons. (l)
KOM 930E	40,831	31,451	1,530	2,883
KOM 930E	68,566	96,651	4,281	3,217
KOM 930E	67,029	90,419	4,011	3,172
KOM 930E	74,157	100,067	4,573	3,205
KOM 930E	97,647	127,924	4,254	3,345
KOM 930E	93,640	94,466	4,272	3,143

Note[a]: Effects of rolling resistance and gradient are embedded in EFH distances (Goris Cervantes, 2018)

3.4.3 Data to Run Optimization Models

After the machine learning model is selected and fuel consumption is estimated, a working shift (day shift from 2022-11-11) from the dataset collected will be used to run the optimization models. An example of how the parameters will be arranged is shown in Table 3.9 as if the optimization models were run at `2022-11-11 10:00:00`.

Table 3.9: Parameters for the Optimization Models

Truck	ETA at fuel station (st.)	Time to st. (min)	Fuel ratio at st. (%)	Next Shovel	Match factor
CDH48	11/11/2022 11:34	14.4	8.5	PAB05	1.5
CDH48	11/11/2022 12:14	7.2	4.8	PAB10	0.5
CDH60	11/11/2022 10:19	6.2	12.4	PAB12	0.5
CDH63	11/11/2022 10:26	14.2	11.5	PHK15	0.9
CDH63	11/11/2022 11:00	14.3	2.6	PAB10	0.6
CDH79	11/11/2022 10:06	5.5	11.8	PAB06	0.1
CDH79	11/11/2022 11:03	14.8	0.9	PAB05	1.1
CDH81	11/11/2022 11:21	7	10.1	PAB12	0.4

In order to incorporate all completed time periods for T in the optimization models, the lower and upper bounds will be derived from the minimum and maximum ETA at the fuel station for all trucks. For example, considering the ETA for truck `CHD79` as `2022-11-1110:06` and for truck `CHD48` as `2022-11-1112:14:33`, the minimum and maximum timestamps will be established. To simplify and round the timestamps to the minute, the seconds recordings are omitted. As a result, Table 3.9 provides five trucks to be fueled, with a lower bound of `2022-11-1110:07:00` and an upper bound of `2022-11-1112:14:00`, representing a duration of 128 minutes. Consequently, the optimization model will encompass 5×128 decision variables.

Additionally, the assigned values will function as follows for the fuel ratio and match factor. If a truck is not expected to arrive during a specific time period (e.g., `CHD79` at `2022-11-1110:06:00`), its fuel ratio will be assigned as -100. Similarly, for the match factor, if a truck is not arriving at a particular time period or not associated with a specific shovel (e.g., `CHD79` at `2022-11-1110:06:00` with shovel `PHK15`), its match factor is considered to be -100.

Finally, a website application has been developed to showcase the results of the optimization model and the computation of the selected machine learning algorithm. You can access the application through the following link. Appendix 16 provides an overview of the features available in this application.

3.4.4 Assumptions and Limitations

- Data described in Table 3.5 is assumed to be accurate. For instance, the road network is drawn by dispatchers in FMS F. In addition, timestamps for fueling, haul cycles, and status are input from operators

- Fueling and haul cycles information for 57 trucks during 18 shifts will represent how the mine site operates during a year.

- There will be a relationship (linear or non-linear) between EFHs, payload, and truck model with fuel consumption rates. These relationships are needed from machine learning algorithms to operate as they look for patterns in the data to improve predictions. Furthermore, given the limited access to all databases in FMS F, e.g., lack of access to fuel consumption rates, this research assumes that machine learning will predict fuel consumption accurately.

- Computation of match factors is accurate. Even though this is not an automatic input from FMS F, Mine Z, and literature have provided information on how to compute them.

- Shortest paths are taken when trucks need to drive from dumping locations to the fuel station and from fuel stations to the assigned loading location.

- It is expected to have a binary integer programming model to improve the match factor of truck-shovel allocation when fueling.

In addition, the following assumption applies to the optimization model

- It is assumed that all trucks require a 15-minute time window to fuel. However, this assumption may lead to infeasible solutions.

- It is believed that computations on the match factor consider the delay of the truck when fueling. The only procedure should be substracting trucks to the trucks allocated to a shovel if the truck goes to fuel.

- The average speed for trucks going empty is assumed to be deterministic - the speed average for trucks in dump cycles. Some researchers argue that this follows a log-normal distribution (Chaowasakoo et al., 2017). However, this research considers a deterministic optimization model, meaning all parameters are deterministic, from which more researchers can build complex models.

Finally, this project presents a snapshot of how trucks should be allocated for refueling at a specific time. However, it does not consider the implications of queues that may arise when an assignment is made (for example, when truck 2 is waiting to refuel while truck 1 is already refueling). Additionally, the assignments made at a particular time point will not affect the outcomes of other time points within a specific period. For instance, if the model is executed at time 1, the results will not influence the outcomes of the model at time 2.

Chapter 4

Results

4.1 Machine Learning Algorithms

4.1.1 Linear Regression

Table 4.1 shows the standardized values for the features used to predict fuel consumption. As mentioned in the methodology section, it is advised to use machine learning algorithms on top of standardized values to avoid weight biased. Furthermore, this linear regression with a closed-form solution will be used as a base model to compare against complex machine learning algorithms.

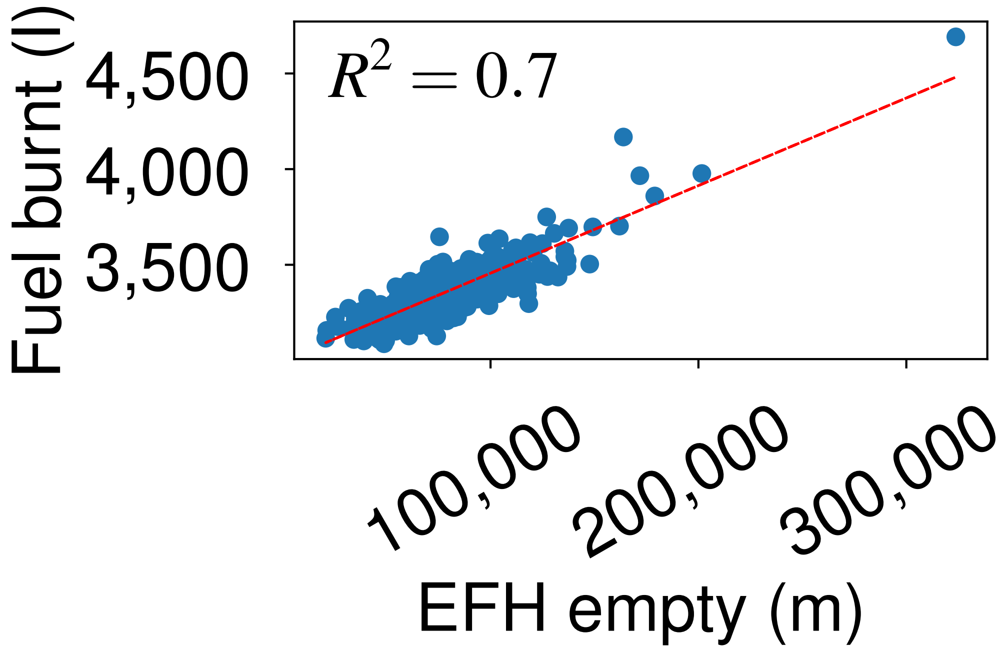
Table 4.1: Standardization of Features' Values

Feature	Mean (*mu*)	Standard Deviation (*sigma*)
EFH empty (m)	76,160.2	25,522.2
EFH full (m)	93,887.1	27,549.6
Tonnage (tons)	4,456.4	1,238.5

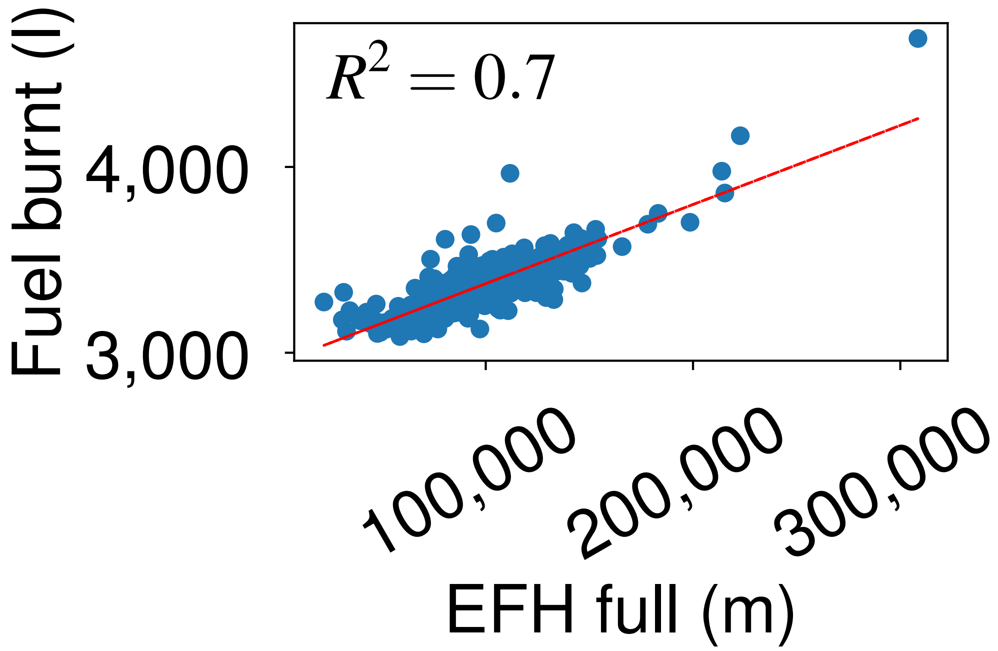
4.1.1.1 Linear Relationship Between Features and Fuel Consumed

Linear regression, in this case, multi-linear regression, assumes features correlate to the predicted variable. For instance, there must be a linear relationship between the EFH distances going empty, full, and payload to the fuel consumption. The following figures show this relationship. For both EFH distances empty and loaded, there is a coefficient of determination, R^2, of 0.7 for both, which indicates a reasonable fit if predicted only with each variable, while for tonnage, the R^2 is 0.4.

There is a higher chance that when combining these three features, the R^2, increases as the linear relationship is still kept.



(a) Fuel consumed vs EFH empty



(b) Fuel consumed vs EFH empty

Figure 4.1: Correlation Between Features and the Predicted Variable

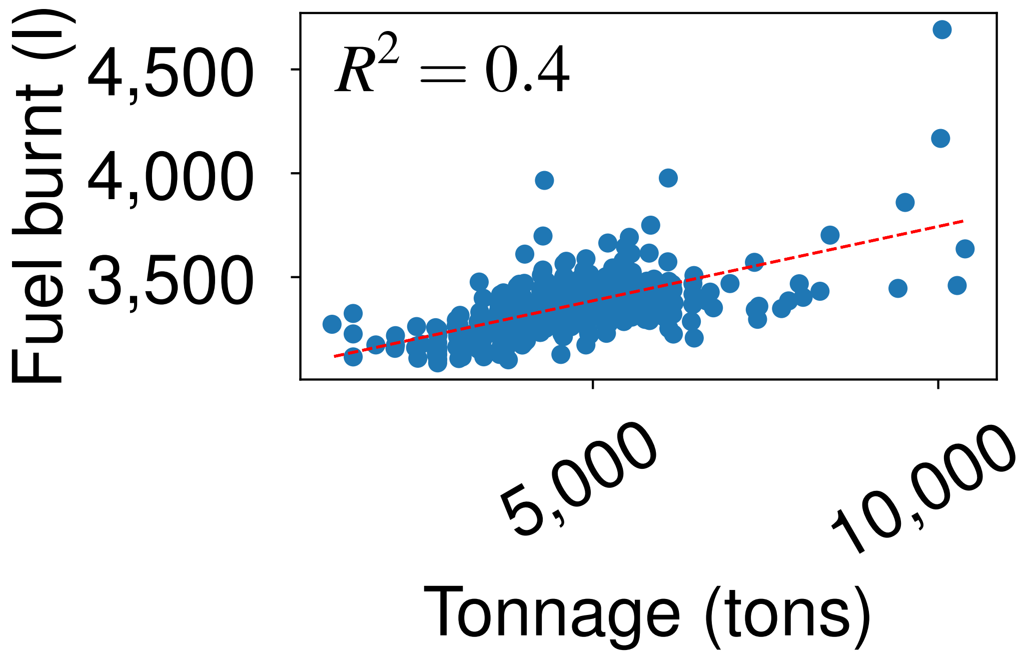


Figure 4.2: Correlation Between Fuel Consumed and Payload

4.1.2 Multi-linear Regression Model

After applying standardization, the values for (1) EFH when empty, (2) EFH when loaded, and (3) tonnage were obtained. These values can be seen in the Python code provided in Appendix 7 and the corresponding values in Table 4.2. Utilizing the sklearn framework and employing the multi-linear regression form as depicted in Equation 3.28, we obtained the following weights and R^2 values. Finally, this model is compared against six other machine learning algorithms to demonstrate that utilizing more complex algorithms leads to an improvement in accuracy.

If the reader would like to predict fuel consumption with new data, the reader should run it on top of Equation 4.1. However, new data points should be standardized before using Table 4.1.

$$Fuelconsumed = Incercept + W_{empty} * S_{empty} + W_{full} * S_{full} + W_{Tonnage} * S_{Tonnage}(^3) \quad (4.1)$$

Table 4.2: Results - Multi-linear regression model

Feature	Weight
EFH empty (m)[2]	67.40
EFH full (m)[2]	54.19
Tonnage (tons)[2]	-7.78
Intercept	3350.87
R^2	0.75
Training Time	126 ms

4.1.3 Support Vector Regression (SVR)

Table 4.3 shows the results of a Support Vector Regression model. The hyperparameters used to obtain the results are shown, including the regularization parameter (C), degree, epsilon, and kernel. The last grid shows the range of values tested for each hyperparameter. The R^2 value obtained from the model is 0.75, indicating a reasonable fit. Furthermore. In this case, a regularization parameter (C) of 200, degree of 1, and epsilon of 0.9 were found to produce the best results for this particular dataset and problem. The polynomial kernel was selected, which can capture nonlinear relationships between the features and target variable. However, the degree suggests that the best model should be linear for `SVR`. The resulting model reasonably fits the test (cross-validated) data, as evidenced by the relatively high R-squared value. The tuning time of 5 minutes and 21 seconds suggests that the model may be computationally intensive, but this will depend on how the upcoming models (4 machine learning algorithms to be trained) perform.

[2] Value of features have been standardized (scaled) to fit the closed-form of the multi-linear regression model

[3] W refers to the weight of each feature and S refers to the standardized value of each feature, e.g., $S_{empty} = (Value - Mean(empty))/StandardDeviation(Empty)$

Table 4.3: Results - Support Vector Regression

Hyper-parameters	Value
Regularization parameter (C)	100
Degree	1
Epsilon	0.9
Kernel	Poly
Last Grid	• C: [1, 20, 50,100, 200] • Degree: [1,2,3,4] • Epsilon: [0.1,0.3,0.6,0.9] • Kernel: ['linear', 'rbf','poly','sigmoid']
R^2	0.75
Tuning Time	5 minutes 21 seconds
Training Time	31 ms

4.1.4 Decision Tree Regression

Table 4.4 shows the hyper-parameters, performance measures, and training time for the decision tree regression model. The hyper-parameters used for this model include the maximum depth of the tree, the number of features, the minimum number of samples, and the minimum weighted fraction of samples for each leaf. The last grid section shows the range of hyperparameter values used for model tuning. The best values for the hyper-parameters were a maximum depth of 5, 3 auto features as shown for EFH empty, full, and tonnage, a minimum number of samples per leaf of 9, and a minimum weighted fraction of 0.1. In addition, the model achieved an R-squared value of 0.55, indicating that the model explained approximately 55% of the variance in the target variable.

The model tuning time was similar to the `SVR`. The results suggest that the selected hyper-parameters of the decision tree model moderately perform in explaining the variance in the target variable. However, a higher value of R-squared may be achievable with more sophisticated algorithms or alternative tuning methods.

Table 4.4: Results - Decision Tree Regression

Hyper-parameters	Value
Maximum depth of the tree (max_depth)	5
Number of features	Auto (3)
Minimum number of samples - leaf (min_samples_leaf)	9
Minimum weighted fraction - leaf (min_weight_fraction_leaf)	0.1
Last Grid	• splitter:[”best”,”random”] • max_depth : [1,3,5,7,9,11,12] • ”min_samples_leaf”:[1,2,3,4,5,6,7,8,9,10] • min_weight_fraction_leaf:[0.1,0.2,...,0.9] • max_features:[”auto”,”log2”,”sqrt”,None] • max_leaf_nodes:[None,10,20,30,40,50,60,70,80,90]
R^2	0.55
Tuning Time	5 minutes 25 seconds
Training Time	107 ms

4.1.5 Gradient Boosting Regression

Table 4.5 reports the results of the Gradient Boosting Regression. The hyperparameters and their respective values are also provided. The model was tuned using a grid search technique with the last grid presented in the table. The reported R^2 value 0.66 indicates that the model can explain 66% of the variability in the dependent variable using the selected independent variables. The tuning time is 3 minutes and 29 seconds, while the training time is 139 ms.

The value of the learning rate is 0.05, which indicates the step size at which the gradient boosting algorithm descends during optimization. The maximum depth of the tree is 4, and it determines the maximum number of levels it can grow. The number of boosting stages is 1, meaning only one decision tree is used. The sample fraction for training is 0.2, indicating that only 20% of the dataset is used for each boosting iteration. The last grid search included the learning rate, subsample, n_estimators, and max_depth hyperparameters. The model's R^2 value indicates moderate performance; however, it is not higher than the base model.

Table 4.5: Results - Gradient Boosting Regression

Hyper-parameters	Value
Learning rate (learning_rate)	0.05
Maximum depth of tree (max_depth)	4
Number of boosting stages (n_estimators)	1
Fraction of samples for training (subsample)	0.2
Last Grid	• learning_rate: [0.01,0.02,0.03,0.04] • subsample : [0.9, 0.5, 0.2, 0.1] • n_estimators : [100,500,1000, 1500] • max_depth: [4,6,8,10]
R^2	0.66
Tuning Time	3 minutes 29 seconds
Training Time	139 ms

4.1.6 Random Forest Regression

Table 4.6 presents the Random Forest Regression algorithm results. The hyper-parameters that were selected based on the tuning process are shown in the last grid column. The tuning process took 1 minute and 7 seconds, less than the Gradient Boosting Regression algorithm's tuning time. The R^2 score for this model was 0.64, slightly less than the R^2 score of the Gradient Boosting Regression algorithm. In addition, even though the training time for this model was 109 milliseconds, which is also less than the training time for the Gradient Boosting Regression algorithm, the accuracy does not improve.

4.1.7 Artificial Neural Networks

Finally, Table 4.7 displays the results of training an Artificial Neural Network (ANN) on a dataset. The hyper-parameters of the model, along with their respective values, are listed in the table. The ANN has four hidden layers with the ReLu activation function and 128 nodes per hidden layer. The output layer has one node with no activation function. The hyper-parameters were tuned using a grid search with a range of values for the number of hidden layers and nodes per hidden layer. The best value of R^2 obtained during the model's test step was 0.90, indicating a strong correlation between the predicted and actual values.

The ANN took a long time to train, with a training time of 65 milliseconds but a tuning time of 21 hours and 17 minutes. This indicates that the grid search for hyper-parameter tuning

Table 4.6: Results - Random Forest Regression

Hyper-parameters	Value
Bootstrap to build tree (bootstrap)	True
Maximum depth of the tree (max_depth)	100
Maximum number of features (max_features)	2
Minimum number of samples to be leaf node (min_samples_leaf)	3
Minimum number of samples required for split (min_samples_split)	10
Number of tree in the forest (n_estimators)	300
Last grid	• bootstrap: [True], • max_depth: [80, 90, 100, 110], • max_features: [2, 3], • min_samples_leaf: [3, 4, 5], • min_samples_split: [8, 10, 12], • n_estimators: [100, 200, 300, 1000]
R^2	0.64
Tuning time	1 minute 7 seconds
Training time	109 ms

took significant time. However, the high value of R^2 suggests that the tuning was successful, and the model will likely perform well on new data. Overall, the results indicate that the ANN is a good model for the given dataset, and the hyper-parameters have been appropriately tuned for optimal performance. Furthermore, with more powerful resources, this tuning time should have a lower computational cost.

Table 4.7: Results - Artificial Neural Network

Hyper-parameters	Value
Number of hidden layers (n_hidden)	4
Activation function for hidden layers	ReLu
Number of nodes per hidden layer (n_nodes_hidden)	128
Number of output layers	1
Activation function for output layer	None
Last grid	• n_hidden: [1, 3, 5, 10, 20] • n_nodes_hidden: [32, 64, ..., 320]
R^2	0.90
Tuning Time	21 hours 17 minutes
Training Time	65 ms

4.1.8 Summary of Results - Machine Learning Algorithms

Table 4.8 displays the results of six machine learning algorithms, including their hyper-parameters, R^2 values, and training times. Multi-linear regression had an R^2 value of 75% in just 126 ms. Support Vector Regression (SVR) achieved the same R^2 value with hyper-parameter tuning taking 5 minutes and 21 seconds. Decision tree regression achieved an R^2 value of 55% with hyper-parameter tuning taking 5 minutes and 25 seconds. Gradient Boosting Regression achieved an R^2 value of 66% with hyper-parameter tuning taking 3 minutes and 29 seconds. Random Forest Regression achieved an R^2 value of 64% with hyper-parameter tuning taking 1 minute and 7 seconds. Artificial Neural Network achieved the highest R^2 value of 90% in just 65 ms of training time with hyper-parameter tuning taking 21 hours and 17 minutes. Overall, Artificial Neural Network performed the best among the algorithms.

Table 4.8: Results of the Machine Learning Algorithms Trained

Algorithm	Hyper-parameters	R^2 (Test)	Hyperparameter tuning and Training Time
Multi-Linear Regression (Base Model)	None	75%	126ms
Support Vector Regression (SVR)	• Regularization parameter(C): 100 • Degree: 1 • Epsilon: 0.9 • Kernel: Poly	75%	• Tuning time: 5m 21 sec. • Training time: 31 ms
Decision Tree Regression	• Maximum depth of tree: 5 • Minimum samples per leaf: 9 • Fraction weight of leaf: 0.1	55%	• Tuning time: 5m 25 sec. • Training time: 107ms
Gradient Boosting Regression	• Learning rate: 0.05 • Maximum depth of tree: 4 • Boosting stages: 1 • Fraction of samples: 0.2 • Strategy to split nodes: best	66%	• Tuning time: 3m 29 sec. • Training time: 139ms
Random Forest Regression	• Bootstrap: True • Maximum depth tree: 100 • Maximum number of features: 2 • Number of samples - leaf: 3 • Number of samples - split: 10 • Number of tree in the forest: 300	64%	• Tuning time: 1m 07 sec. • Training time: 109ms
Artificial Neural Network	• Number of hidden layers: 4 • Activation f.- hidden layer: ReLu • Numer of nodes- hidden layer: 128 • Activation f.- output layer: None	90%	• Tuning time: 21 hr.17 min. • Training time: 65ms

4.2 Optimization Models

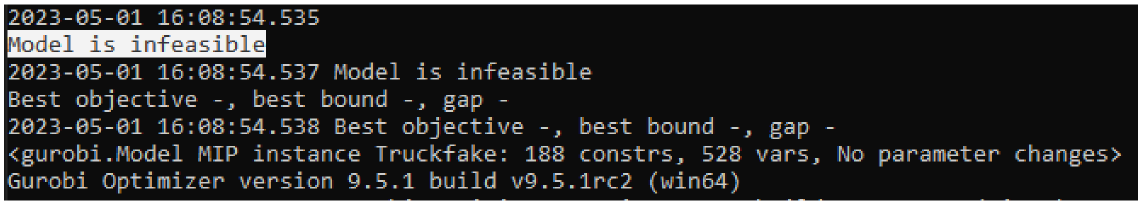
The following describes the results of the optimization model if it were run during the day shift of `2022-11-11` at `14:00:00`.

4.2.1 First Formulation - Fuel Time Window

Table 4.9 shows the parameters used to run the first formulation of the proposed optimization model in Section 3.2.1.1. After running the first formulation, the model was infeasible (see output from Gurobi in Figure 4.3). The infeasibility is caused by the fourth and fifth constraints (Equations 3.5 and 3.6). For example, reducing the time window to one minute eliminates infeasibility. However, we cannot change the current truck-shovel allocations to fit the fuel-dispatching model, which is impractical. For example, as shown in Table 4.9, CDH42 is estimated to arrive at the fuel station four minutes before CDH66, which is the only choice. Therefore, we can not have all trucks fueling while meeting the time window constraint. However, we can fit the fuel dispatching model based on how the current operation is being performed.

Table 4.9: Parameters - Optimization Model Run at 2022-11-11 14:00:00

It.	Truck	ETA at fuel station (st.)	Time to st. (min)	Fuel ratio at st. (%)	Next Shovel	Match factor
1	CDH42	11/11/2022 14:24	11.8	9.9	PAB11	0.8
2	CDH44	11/11/2022 16:12	6.5	14.2	PAB11	0.5
3	CDH44	11/11/2022 16:34	6.3	1	PAB11	0.7
4	CDH46	11/11/2022 14:47	14.4	9.4	PAB11	0.8
5	CDH65	11/11/2022 14:40	14.3	10.2	PAB05	0.8
6	CDH66	11/11/2022 14:28	14.3	9.2	PAB11	0.9
7	CDH67	11/11/2022 14:14	6.8	9.2	PAB11	0.8
8	CDH68	11/11/2022 16:41	11.8	13.9	PAB12	0.5
9	CDH73	11/11/2022 14:41	6.3	11.1	PAB11	0.9
10	CDH73	11/11/2022 16:00	6.8	6	PAB06	0.4
11	CDH89	11/11/2022 14:48	14.1	11.7	PHK13	0.4
12	CDH89	11/11/2022 15:20	6.7	2.4	PHK13	0.1


```
2023-05-01 16:08:54.535
Model is infeasible
2023-05-01 16:08:54.537 Model is infeasible
Best objective -, best bound -, gap -
2023-05-01 16:08:54.538 Best objective -, best bound -, gap -
<gurobi.Model MIP instance Truckfake: 188 constrs, 528 vars, No parameter changes>
Gurobi Optimizer version 9.5.1 build v9.5.1rc2 (win64)
```
Figure 4.3: Gurobi Results From the First Optimization Model

4.2.2 Second formulation - Fuel Time Window

The second formulation deletes the time window constraints while embedding the delay of fueling and overlapping times to the match factor parameter. Once the optimization model was run, it found a feasible and optimal solution (output from Gurobi different from Figure 4.3), as shown in Table 4.10.

It can be shown that the optimization model chose to fuel truck `CDH73` at `14:41:00` instead of `16:00:00` even though it almost overlapped with `CHD65`. This was because of its higher match factor. Finally, if implemented correctly, the run time to obtain optimal solutions is fast, less than 1 second, so that it could provide answers to dispatchers at the mine site.

Table 4.10: Optimal Solution - Second Optimization Model

It.	Truck	ETA at fuel station (st.)	Fuel ratio at st. (%)	Match factor
1	CDH67	11/11/2022 14:14	9.2	0.7
2	CDH42	11/11/2022 14:24	9.9	0.7
3	CDH66	11/11/2022 14:28	9.2	0.8
4	CDH65	11/11/2022 14:40	10.2	0.8
5	CDH73	11/11/2022 14:41	11.1	0.8
6	CDH46	11/11/2022 14:47	9.4	0.9
7	CDH89	11/11/2022 14:48	11.7	0.4
8	CDH44	11/11/2022 16:34	1	0.7
9	CDH68	11/11/2022 16:41	13.9	0.5
			Average Match Factor	0.7
			Run Time	0.01545 sec

Finally, the option of having mobile fuel stations (i.e., fuel trucks) at a mine site presents itself. If such a setup exists, the formulation ”All trucks should fuel” (Equation 3.31) from subsection 3.2.1.2 could be omitted. By doing so, the optimization model gains the flexibility to select the specific trucks that need to travel to a fuel station without compromising the match factor. This arrangement is illustrated in Table 4.11, which depicts a schedule where five out of ten trucks are assigned to a fixed station while the remaining trucks are directed to a mobile fuel station. However, there are two additional considerations that must be taken into account: first, the impact of an increased match factor at the dump location, and second, the availability of the mobile fuel station.

Table 4.11: Optimal Solution - Additional Mobile Fuel Station

It.	Truck	ETA at fuel station (st.)	Fuel ratio at st. (%)	Match factor
1	CDH44	11/11/2022 16:33	1	0.7
2	CDH46	11/11/2022 14:47	9.4	0.9
3	CDH67	11/11/2022 14:14	9.2	0.7
4	CDH73	11/11/2022 16:00	6	0.4
5	CDH89	11/11/2022 15:21	2.4	0.1
			Average Match Factor	0.6
			Run Time	0.01559 sec

4.2.3 Comparison with Current Fuel Assignments

When comparing the performance between the automated fuel assignment program using optimization with the current assignments in terms of match factor, the results run at `2022-11-11-14:00:00` shows that the match factor was increased by 1 point for the optimized model (see results from the current assignment in Table 4.13 and from the optimized model in Table 4.10). In addition, the average difference between the estimated time of arrival of the optimized model and the time that trucks started to fuel is about 10 minutes (see last column of Table 4.13 representing the difference between current - optimized), which provides confidence when using average speed when trucks travel empty to fuel stations.

Table 4.12: Performance of Current Assignments at 2022-11-11 14:00:00

#	Equip.	Current	Opt. Model	MatchFactor	Difference (time)
1	CDH42	11/11/2022 14:38	11/11/2022 14:24	0.5	+0 days 00:14:00
2	CDH44	11/11/2022 16:23	11/11/2022 16:34	0.6	-0 days 00:11:00
3	CDH46	11/11/2022 14:21	11/11/2022 14:47	0.7	-0 days 00:26:00
4	CDH65	11/11/2022 14:47	11/11/2022 14:40	0.6	+0 days 00:07:00
5	CDH66	11/11/2022 14:26	11/11/2022 14:28	0.5	-0 days 00:02:00
6	CDH67	11/11/2022 14:09	11/11/2022 14:14	0.7	-0 days 00:05:00
7	CDH68	11/11/2022 16:51	11/11/2022 16:41	0.5	+0 days 00:10:00
8	CDH73	11/11/2022 14:35	11/11/2022 14:41	0.6	-0 days 00:06:00
9	CDH89	11/11/2022 16:32	11/11/2022 14:48	0.6	+0 days 01:44:00
				Avg. Match Factor	0.6
				Avg. Difference Time	(+/-) 9.44 minutes

Table 4.13: Performance of Current Assignments at 2022-11-11 14:00:00

It.	Equipment	StartTimestamp	MatchFactor	Difference (time)
1	CDH42	11/11/2022 14:38	0.5	+0 days 00:14:00
2	CDH44	11/11/2022 16:23	0.6	-0 days 00:11:00
3	CDH46	11/11/2022 14:21	0.7	-0 days 00:26:00
4	CDH65	11/11/2022 14:47	0.6	+0 days 00:07:00
5	CDH66	11/11/2022 14:26	0.5	-0 days 00:02:00
6	CDH67	11/11/2022 14:09	0.7	-0 days 00:05:00
7	CDH68	11/11/2022 16:51	0.5	+0 days 00:10:00
8	CDH73	11/11/2022 14:35	0.6	-0 days 00:06:00
9	CDH89	11/11/2022 16:32	0.6	+0 days 01:44:00
			Average Match Factor	0.6
			Average Difference Time	(+/-) 9.44 minutes

Chapter 5

Conclusions

The study aimed to identify the best machine learning algorithm for predicting a target variable based on input features. Six algorithms were employed: Multi-Linear Regression, Support Vector Regression (SVR), Decision Tree Regression, Gradient Boosting Regression, Random Forest Regression, and Artificial Neural Network. Performance assessment was based on R^2 scores and training times, with hyperparameter tuning to optimize each model's performance.

Results showed that the Artificial Neural Network achieved the highest R^2 value (90%) with a training time of 65 ms, though hyperparameter tuning took 21 hours and 17 minutes. Multi-Linear Regression demonstrated the fastest training time and a respectable R^2 value of 75% in just 126 ms. SVR also obtained an R^2 value of 75%, requiring 5 minutes and 21 seconds for hyperparameter tuning. Decision Tree Regression achieved an R^2 value of 55% with tuning time of 5 minutes and 25 seconds. Gradient Boosting Regression reached an R^2 value of 66% with tuning time of 3 minutes and 29 seconds. Random Forest Regression achieved an R^2 value of 64% with tuning time of 1 minute and 7 seconds.

Despite the lengthy hyperparameter tuning process, the Artificial Neural Network was the best-performing algorithm, achieving the highest R^2 value with relatively short training time. Nonetheless, Multi-Linear Regression performed well in speed and obtained a respectable R^2 value, making it suitable for simpler models. Other algorithms like SVR, Decision Tree Regression, Gradient Boosting Regression, and Random Forest Regression may be appropriate, depending on specific resource and dataset requirements.

The proposed fuel dispatching model, based on optimization techniques, offers different sets of optimization models to improve haul truck fueling while maximizing the match factor of truck-shovel allocations in mining operations. The initial formulation of the optimization model revealed that current truck-shovel allocations couldn't satisfy the time-window constraint for fueling. It was crucial for the fuel dispatching model to align with the existing truck-shovel allocations, not the other way around. However, the second formulation of the model, which eliminated time window constraints and integrated fueling delays and overlapping times into the match factor parameter, provided a feasible and optimal solution. When running the optimization model at 2022-11-11 14:00:00, it yielded a higher match factor than the current fueling assignments. Additionally, obtaining optimal solutions had a fast runtime, taking less than 1 second for a fleet of 57 trucks, making it suitable for real-time use by dispatchers at the mine site. Therefore, based on these results, it is believed that the optimization model proves highly beneficial for optimizing fuel dispatching in mining operations.

Chapter 6

Future Work

- Investigating the impact of different variables: The research conducted in this study utilized effective flat haul distances for empty and loaded tonnages, considering the data available from the queried databases. However, the model's accuracy could be further enhanced by incorporating additional relevant features (e.g., the type of material being transported - as materials sent to a crusher may result in higher fuel consumption by operators).

- Extending the sample size: This research achieved a 90% accuracy in predicting fuel consumption using EFHs and tonnages with data from 18 shifts (9 days). Nonetheless, it is widely recognized that machine learning algorithms perform better when trained on larger datasets. Thus, leveraging pre-trained models with extensive datasets is advisable to improve their predictive accuracy.

- Implementation: The research has devised a method to predict fuel consumption without requiring access to additional features from FMS providers, potentially circumventing extra costs. Moreover, it sought to mitigate the impact of fueling delays on the match factor of truck-shovel allocations. However, the real challenge lies in practical implementation at mine sites. Determining the optimal execution time of the optimization model and devising effective ways to query databases from FMS are key considerations. The results should be presented in a user-friendly manner, e.g., through the development of user-friendly applications, as dispatchers often deal with multiple tasks simultaneously.

- Including mobile fuel stations in the optimization model: The current research focuses on a fixed fuel station to cater to the fueling needs of all trucks. However, mine sites often have mobile fuel trucks that serve loaders, shovels, and auxiliary equipment. To maximize their utilization and offer greater flexibility to fixed fuel stations, the optimization model can be fine-tuned to incorporate additional fueling locations.

These proposed modifications present avenues for further enhancing the efficiency and accuracy of fuel consumption predictions and optimization of fuel dispatching strategies in mining operations. Their successful implementation can contribute significantly to the overall effectiveness and productivity of mine operations.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Adams, K. and Bansah, K. (2016). Review of operational delays in shovel-truck system of surface mining operations. In *4 th UMaT Biennial International Mining and Mineral Conference*, volume 60, page 65.

Aguirre-Jofré, H., Eyre, M., Valerio, S., and Vogt, D. (2021). Low-cost internet of things (iot) for monitoring and optimising mining small-scale trucks and surface mining shovels. *Automation in Construction*, 131:103918.

Alamdari, S., Basiri, M. H., Mousavi, A., and Soofastaei, A. (2022). Application of machine learning techniques to predict haul truck fuel consumption in open-pit mines. *Journal of Mining and Environment*, 13(1):69–85.

Ali, D. and Frimpong, S. (2021). Deephaul: a deep learning and reinforcement learning-based smart automation framework for dump trucks. *Progress in Artificial Intelligence*, 10:157–180.

Alloghani, M., Al-Jumeily, D., Mustafina, J., Hussain, A., and Aljaaf, A. J. (2020). A systematic review on supervised and unsupervised machine learning algorithms for data science. *Supervised and unsupervised learning for data science*, pages 3–21.

Amoako, R., Jha, A., and Zhong, S. (2022). Rock fragmentation prediction using an artificial neural network and support vector regression hybrid approach. *Mining*, 2(2):233–247.

Anand, R., Aggarwal, D., and Kumar, V. (2017). A comparative analysis of optimization solvers. *Journal of Statistics and Management Systems*, 20(4):623–635.

Anguita, D., Ghelardoni, L., Ghio, A., Oneto, L., and Ridella, S. (2012). The'k'in k-fold cross validation. In *ESANN*, pages 441–446.

Bentéjac, C., Csörgő, A., and Martínez-Muñoz, G. (2021). A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*, 54:1937–1967.

Bhoraskar, A. (2019). Prediction of fuel consumption of long haul heavy duty trucks using machine learning and comparison of the performance of various learning techniques. *Delft University of Technology*.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Breiman, L. (2017). *Classification and regression trees*. Routledge.

Buaba, J. and Brickey, A. (2021). Estimating diesel particulate matter using a predictive technique for use in underground metal mine production scheduling. In *Mine Ventilation*, pages 86–94. CRC Press.

Burt, C. N. and Caccetta, L. (2007). Match factor for heterogeneous truck and loader fleets. *International journal of mining, reclamation and environment*, 21(4):262–270.

Caceres, M. and Wells, D. (2017). Increasing Productivity and Conserving Person-Hours Through Automated Fuel Dispatching at Sierra Gorda. Technical report, Wenco International Mining Systems Ltd.

Carter, R. A. (2012). Fleet management: challenges and choices. *Engineering and Mining Journal*, 213(3):28.

Chaowasakoo, P., Seppälä, H., Koivo, H., and Zhou, Q. (2017). Digitalization of mine operations: Scenarios to benefit in real-time truck dispatching. *International Journal of Mining Science and Technology*, 27(2):229–236.

Chollet, F. (2021). *Deep learning with Python*. Simon and Schuster.

Chollet, F. et al. (2015). Keras. `https://keras.io`.

Chropra, R. and Swart, A. (2022). Tracking the trends 2022 - Redifining Mining. Technical report, Deloitte Touche Tohmatsu Limited, United States.

Dindarloo, S. R. and Siami-Irdemoosa, E. (2016). Determinants of fuel consumption in mining trucks. *Energy*, 112:232–240.

Doan, T. and Kalita, J. (2015). Selecting machine learning algorithms using regression models. In *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, pages 1498–1505. IEEE.

Goris Cervantes, E. (2018). An improved approach to production planning in oil sands mining through detailed analysis and simulation of cycle times.

Gurobi Optimization, LLC (2022). Gurobi Optimizer Reference Manual.

Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182.

Hagberg, A. A., Schult, D. A., and Swart, P. J. (2008). Exploring network structure, dynamics, and function using networkx. In Varoquaux, G., Vaught, T., and Millman, J., editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA.

Hastie, T., Tibshirani, R., Friedman, J. H., and Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer.

Hillier, F. and Lieberman, G. (2010). *Introduction to Operations Research*. McGraw-Hill higher education. McGraw-Hill Higher Education.

Hustrulid, W. A., Kuchta, M., and Martin, R. K. (2013). *Open pit mine planning and design, two volume set & CD-ROM pack*. CRC Press.

Hyder, Z., Siau, K., and Nah, F. (2019). Artificial intelligence, machine learning, and autonomous technologies in mining industry. *Journal of Database Management (JDM)*, 30(2):67–79.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An introduction to statistical learning*. Springer.

Joseph, V. R. and Vakayil, A. (2022). Split: An optimal method for data splitting. *Technometrics*, 64(2):166–176.

Khorasgani, H., Wang, H., and Gupta, C. (2020). Challenges of applying deep reinforcement learning in dynamic dispatching. *arXiv preprint arXiv:2011.05570*.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., and Willing, C. (2016). Jupyter notebooks – a publishing format for reproducible computational workflows. In Loizides, F. and Schmidt, B., editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press.

Krogh, A. (2008). What are artificial neural networks? *Nature biotechnology*, 26(2):195–197.

Lee, I. and Shin, Y. J. (2020). Machine learning for enterprises: Applications, algorithm selection, and challenges. *Business Horizons*, 63(2):157–170.

Leonida, C. (2022). The future fleet forecast - e&mj explores the latest trends in truck-shovel mining. Technical report, Global Mining Guidelines Group.

Li, X. et al. (2013). Using" random forest" for classification and regression. *Chinese Journal of Applied Entomology*, 50(4):1190–1197.

Liu, C., Li, M., Zhang, Y., Han, S., and Zhu, Y. (2019). An enhanced rock mineral recognition method integrating a deep learning model and clustering algorithm. *Minerals*, 9(9):516.

Loh, W.-Y. (2011). Classification and regression trees. *Wiley interdisciplinary reviews: data mining and knowledge discovery*, 1(1):14–23.

Lukacs, Z. (2020). A standardized time classification framework for mobile equipment in surface mining: Operational definitions, time usage model, and key performance indicators. Technical report, Global Mining Guidelines Group.

Mathur, P. and Mathur, P. (2019). Overview of machine learning in retail. *Machine Learning Applications Using Python: Cases Studies from Healthcare, Retail, and Finance*, pages 147–157.

Maulud, D. and Abdulazeez, A. M. (2020). A review on linear regression comprehensive in machine learning. *Journal of Applied Science and Technology Trends*, 1(4):140–147.

Mayr, A., Binder, H., Gefeller, O., and Schmid, M. (2014). The evolution of boosting algorithms. *Methods of information in medicine*, 53(06):419–427.

Modular Mining (2019). Modular Mining's Performance Assurance team helps an Australian coal operation increase truck availability through automated fueling assignments. Technical report, Modular Mining Systems, Inc.

Moradi, A. and Askari-Nasab, H. (2019). Mining fleet management systems: a review of models and algorithms. *International Journal of Mining, Reclamation and Environment*, 33(1):42–60.

Natekin, A. and Knoll, A. (2013). Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, 7:21.

Newman, A., Rubio, E., Caro, R., Weintraub, A., and Eurek, K. (2010). A review of operations research in mine planning. *Interfaces*, 40(3):222–245.

Ngai, E. W. and Wu, Y. (2022). Machine learning in marketing: A literature review, conceptual framework, and research agenda. *Journal of Business Research*, 145:35–48.

Ouanan, H. and Abdelwahed, E. H. (2019). Image processing and machine learning applications in mining industry: Mine 4.0. In *2019 International Conference on Intelligent Systems and Advanced Computing Sciences (ISACS)*, pages 1–5.

Pal, A. (2020). Gradient boosting trees for classification: A beginner's guide. Medium. Accessed: February 15, 2023.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Rardin, R. (2016). *Optimization in Operations Research*. Pearson.

Ray, S. (2019). A quick review of machine learning algorithms. In *2019 International conference on machine learning, big data, cloud and parallel computing (COMITCon)*, pages 35–39. IEEE.

Reichl, C. and Schatz, M. (2022). World mining data 2022. Technical report, Federal Ministry Republic of Austria.

Schmidt, D. (2015). Fueling Automation Technology. Technical report, Coal Age.

Schrage, L. and Wolsey, L. (1985). Sensitivity analysis for branch and bound integer programming. *Operations Research*, 33(5):1008–1023.

Segal, M. R. (2004). Machine learning benchmarks and random forest regression.

Sen, R. P. (2009). *Operations Research: Algorithms And Applications: Algorithms and Applications*. PHI Learning Pvt. Ltd.

Shahhosseini, M., Hu, G., and Pham, H. (2022). Optimizing ensemble weights and hyperparameters of machine learning models for regression problems. *Machine Learning with Applications*, 7:100251.

Shailaja, K., Seetharamulu, B., and Jabbar, M. (2018). Machine learning in healthcare: A review. In *2018 Second international conference on electronics, communication and aerospace technology (ICECA)*, pages 910–914. IEEE.

Siami-Irdemoosa, E. and Dindarloo, S. R. (2015). Prediction of fuel consumption of mining dump trucks: A neural networks approach. *Applied Energy*, 151:77–84.

Soofastaei, A., Aminossadati, S., Kizil, M., and Knights, P. (2016). A comprehensive investigation of loading variance influence on fuel consumption and gas emissions in mine haulage operation. *International Journal of Mining Science and Technology*, 26(6):995–1001.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

Vovk, V. (2013). Kernel ridge regression. *Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik*, pages 105–116.

Wang, Q., Zhang, R., Lv, S., and Wang, Y. (2021). Open-pit mine truck fuel consumption pattern and application based on multi-dimensional features and xgboost. *Sustainable Energy Technologies and Assessments*, 43:100977.

Wenco Mining Systems (2013). Wencomine. `http://wenco.ru/2012/Wenco-information-2012EN.pdf`.

Whittle, D. (2019). *Underground mine plan optimisation.* PhD thesis, University of Melbourne, Parkville, Victoria, Australia.

Wuest, T., Weimer, D., Irgens, C., and Thoben, K.-D. (2016). Machine learning in manufacturing: advantages, challenges, and applications. *Production & Manufacturing Research*, 4(1):23–45.

Xing, H., Floratos, S., Blanas, S., Byna, S., Prabhat, M., Wu, K., and Brown, P. (2018). Arraybridge: Interweaving declarative array processing in scidb with imperative hdf5-based programs. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 977–988. IEEE.

Appendices

7. Python Code - Scaling features

```python
# Importing pandas library for reading CSV file
import pandas as pd
# Importing sklearn methods
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

# Reading only the required columns from the CSV file
data_2 = pd.read_csv(dir_data_ml+'rawDataML_3.csv')[['Empty(m)', 'Full(m)',
                                                     'Tons', 'Fuel(l)']]

# Separating independent variables and dependent variable
x_all = data_2[['Empty(m)', 'Full(m)', 'Tons']]
y_values = data_2['Fuel(l)']

# Scaling the independent variables using StandardScaler
standardScaler = StandardScaler()
x_std = standardScaler.fit_transform(x_all)

# Scaling the independent variables using MinMaxScaler
minmax = MinMaxScaler()
x_minmax = minmax.fit_transform(x_all)

# Setting seed for reproducibility of the train-test split
seed = 4

# Splitting the data into training and testing sets using standard scaling
x_trainstd, x_teststd, y_trainstd, y_teststd = train_test_split(
    x_std, y_values, test_size = 0.2, random_state = seed
)

# Splitting the data into training and testing sets using maximum scaling
x_trainmax, x_testmax, y_trainmax, y_testmax = train_test_split(
    x_minmax, y_values, test_size = 0.2, random_state = seed
)

# Splitting the data into training and testing sets without scaling
x_trainall, x_testall, y_trainall, y_testall = train_test_split(
    x_all, y_values, test_size = 0.2, random_state = seed
)
```

Appendices

8. Python Code - Multi-Linear Regression

```
# Importing necessary libraries
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np

# Creating a Linear Regression object
LR = LinearRegression()

# Fitting the model on the training data
LR.fit(x_trainstd, y_trainstd)

# Predicting the target variable using the testing data
y_prediction = LR.predict(x_teststd)

# Computing the R2 score of the predictions
score = r2_score(y_teststd, y_prediction)

# Printing the R2 score and RMSE of the predictions
print("R2 score  =", round(score, 2))
print("RMSE      =", round(np.sqrt(mean_squared_error(y_teststd,
                                   y_prediction)), 2))
```

Appendices

9. Python Code - Support Vector Regression

```
# Importing necessary libraries
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score
import numpy as np

# Defining the parameter grid for SVR
parametersSVR = {'kernel': ('linear', 'rbf', 'poly', 'sigmoid'),
                 'C': [1, 20, 50, 100, 200, 500],
                 'degree': [1, 2, 3, 4],
                 'gamma': np.logspace(-4, 0, 8),
                 'epsilon': [0.1, 0.3, 0.6, 0.9]}

# Creating a GridSearchCV object with SVR as the estimator and parameter
                                     grid
cv = 5
searchingSVR = GridSearchCV(SVR(), parametersSVR, cv=cv, scoring='r2')

# Fitting the GridSearchCV object on the training data
searchingSVR.fit(x_trainstd, y_trainstd)

# Getting the best parameters and the best R2 score
bestSVR = searchingSVR.best_params_
print(bestSVR)
print(searchingSVR.best_score_)

# Creating an SVR object with the best parameters and fitting it on the
                                     training data
regSVR = SVR(C=bestSVR['C'],
             epsilon=bestSVR['epsilon'],
             kernel=bestSVR['kernel'],
             gamma=bestSVR['gamma'],
             degree=bestSVR['degree'])
regSVR.fit(x_trainstd, y_trainstd)

# Predicting the target variable using the testing data and calculating the
                                      R2 score
y_prediction_svr = regSVR.predict(x_teststd)
score = r2_score(y_teststd, y_prediction_svr)
print("R2 score  =", round(score, 2))
```

Appendices

10. Python Code - Decision Tree Regression

```python
# Import DecisionTreeRegressor from sklearn.tree
from sklearn.tree import DecisionTreeRegressor

# Set the hyperparameters to be searched
parametersDT = {
    "max_depth": [1, 10, 50, 100, 200],
    "min_samples_leaf": [1, 10, 50, 100, 200],
    "min_samples_split": [10, 100, 200, 500],
    "min_weight_fraction_leaf": [0.1, 0.2, 0.3, 0.4, 0.5],
    "max_leaf_nodes": [10, 100, 200, 500]
}

# Create an instance of GridSearchCV and set the hyperparameters
# also set cv and scoring
serchingDT = GridSearchCV(
    DecisionTreeRegressor(),
    param_grid=parametersDT,
    scoring='r2',
    cv=cv
)

# Fit the grid search object with the training data
serchingDT.fit(x_trainall, y_trainall)

# Print the best hyperparameters
bestDT = serchingDT.best_params_
print(bestDT)
print(serchingDT.best_score_)

# Train the model again with the best hyperparameters
DTree = DecisionTreeRegressor(
    max_depth=bestDT['max_depth'],
    max_leaf_nodes=bestDT['max_leaf_nodes'],
    min_samples_leaf=bestDT['min_samples_leaf'],
    min_weight_fraction_leaf=bestDT['min_weight_fraction_leaf'],
    min_samples_split=bestDT['min_samples_split']
)

# Fit the DecisionTreeRegressor model with the training data
DTree.fit(x_trainall, y_trainall)

# Use the trained model to predict fuel consumption for test data
y_prediction_dt = DTree.predict(x_testall)

# Evaluate the performance of the model with the R2 score
```

```
score = r2_score(y_testall, y_prediction_dt)
print("R2 score  =", round(score, 2))
```

Appendices

11. Python Code - Gradient Boosting Regression

```python
# Import the necessary library
from sklearn.ensemble import GradientBoostingRegressor

# Set the parameters to be tested
parametersGB = {'learning_rate': [0.01,0.05, 0.1, 0.5, 1],
              'subsample'     : [0.9, 0.5, 0.2, 0.1],
               'n_estimators' : [100, 400],
               'max_depth'    : [1,5],
               'min_samples_split': [2, 10, 50]}

# Perform grid search to find the best hyperparameters
searchingGB = GridSearchCV(estimator=GradientBoostingRegressor(),
                           param_grid=parametersGB,
                           scoring='r2',
                           cv=cv)
searchingGB.fit(x_trainall, y_trainall)

# Print the best hyperparameters
bestGB = searchingGB.best_params_
print(bestGB)
print(searchingGB.best_score_)

# Train the Gradient Boosting model with the best hyperparameters
GB = GradientBoostingRegressor(
    learning_rate= bestGB['learning_rate'],
    subsample = bestGB['subsample'],
    n_estimators= bestGB['n_estimators'],
    max_depth= bestGB['max_depth'],
    min_samples_split= bestGB['min_samples_split'])
GB.fit(x_trainall, y_trainall)

# Predict using the trained model
y_prediction_gb = GB.predict(x_testall)

# Compute R2 score
scoreGB=r2_score(y_testall,y_prediction_gb)

# Print the R2 score
print("R2 score  =", round(scoreGB,2))
```

Appendices

12. Python Code - Random Forest Regression

```
from sklearn.ensemble import RandomForestRegressor

# define hyperparameters
parametersRF = {
    'n_estimators': [1, 10, 30, 60, 100],
    'max_depth': [1, 10, 50, 100, 200, 500, 1000],
    'min_samples_split': [5, 10, 50],
    'min_samples_leaf': [1, 5, 10]
}

# create a grid search object with the specified parameters and cv
searchingRF = GridSearchCV(estimator=RandomForestRegressor(),
                           param_grid=parametersRF,
                           cv=cv)

# fit the grid search object to the training data
searchingRF.fit(x_trainall, y_trainall)

# Print best hyper
bestRF = searchingRF.best_params_
print(bestRF)
print(searchingRF.best_score_)

# create a new Random Forest Regressor with the best hyperparameters found
RF = RandomForestRegressor(
    n_estimators=bestRF['n_estimators'],
    max_depth=bestRF['max_depth'],
    min_samples_split=bestRF['min_samples_split'],
    min_samples_leaf=bestRF['min_samples_leaf'])

# fit the Random Forest Regressor to the training data
RF.fit(x_trainall, y_trainall)

# use the fitted Random Forest Regressor to make predictions on the test
                                       data
y_prediction_rf = RF.predict(x_testall)

# calculate the R2 score
scoreRF = r2_score(y_testall, y_prediction_rf)

# print the R2 score rounded to 2 decimal places
print("R2 score =", round(scoreRF, 2))
```

Appendices

13. Python Code - Artificial Neural Networks

```
# Import tensorflow and keras frameworks
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import r2_score
from sklearn.model_selection import KFold
import numpy as np

# Set optimizer
opt = 'adam'

# Set callback for early stopping
callback = tf.keras.callbacks.EarlyStopping(monitor='
                                     val_root_mean_squared_error',
                                     patience=3, verbose=1)

# Define a model with one hidden layer
def model1(nodes):
    model = keras.Sequential([
        layers.Dense(nodes, activation="relu"),
        layers.Dense(1)
    ])
    model.compile(optimizer=opt, loss="mse", metrics=[keras.metrics.
                                         RootMeanSquaredError()])
    return model

# Define a model with three hidden layers
def model3(nodes):
    model = keras.Sequential([
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(1)
    ])
    model.compile(optimizer=opt, loss="mse", metrics=[keras.metrics.
                                         RootMeanSquaredError()])
    return model
# Define a model with five hidden layers
def model5(nodes):
    model = keras.Sequential([
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(1)
    ])
    model.compile(optimizer=opt, loss="mse", metrics=[keras.metrics.
                                        RootMeanSquaredError()])
    return model
# Define a model with 10 hidden layers
def model10(nodes):
    model = keras.Sequential([
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(1)
    ])
    model.compile(optimizer=opt, loss="mse", metrics=[keras.metrics.
                                        RootMeanSquaredError()])
    return model
# Define a model with 20 hidden layers
def model20(nodes):
    model = keras.Sequential([
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(nodes, activation="relu"),
        layers.Dense(1)
    ])
    model.compile(optimizer=opt, loss="mse", metrics=[keras.metrics.
                                        RootMeanSquaredError()])
    return model

# Define a function to train a model and return the R-squared score and
                                    number of epochs
```

```
def trainmodel(modelx, xtotaltrain, ytotaltrain, trainmask, valmask, y_test
                                    , x_test):
    y_test = np.array(y_test)
    history = modelx.fit(xtotaltrain[trainmask],
                        np.array(ytotaltrain)[trainmask],
                        epochs=1000,
                        batch_size=16,
                        validation_data=(xtotaltrain[valmask], np.array(
                                                            ytotaltrain)[
                                                            valmask]),
                        callbacks=[callback],
                        verbose=0)
    rscore = r2_score(y_test,modelx.predict(x_test))
    epochf = len(history.history['loss'])
    return rscore, epochf

# Create an empty dictionary to store the results of training the models
results = {}

# Set number of folds for k-fold cross validation
cv = 5

# Split the data into k-folds and train models for different number of
                                    nodes in the hidden layers
kfold = KFold(n_splits=cv, shuffle=True)
for nodes in np.arange(32, 32*10, 32):
    f = 1
    for train, val in kfold.split(x_trainstd, y_trainstd):
        # Add keys to the dictionary if they don't exist
        if 'layers1' not in results:
            results['layers1'] = []
        if 'layers3' not in results:
            results['layers3'] = []
        if 'layers5' not in results:
            results['layers5'] = []
        if 'layers10' not in results:
            results['layers10'] = []
        if 'layers20' not in results:
            results['layers20'] = []
        # nodes, fold, rscore, epochf
        # Train model with one hidden layer
        model_1 = model1(nodes)
        rscore, epochf = trainmodel(model_1, x_trainstd, y_trainstd, train,
                                        val, y_teststd,  x_teststd)
        results['layers1'].append([nodes, f, rscore, epochf])
        # Train model with three hidden layers
        model_3 = model3(nodes)
        rscore, epochf = trainmodel(model_3, x_trainstd, y_trainstd, train,
                                        val, y_teststd,  x_teststd)
        results['layers3'].append([nodes, f, rscore, epochf])
        # Train model with five hidden layers
        model_5 = model5(nodes)
        rscore, epochf = trainmodel(model_5, x_trainstd, y_trainstd, train,
                                        val,y_teststd,  x_teststd)
        results['layers5'].append([nodes, f, rscore, epochf])
        # Train model with ten hidden layers
        model_10 = model10(nodes)
        rscore, epochf = trainmodel(model_10, x_trainstd, y_trainstd, train
                                    , val,y_teststd,  x_teststd)
        results['layers10'].append([nodes,f, rscore, epochf])
        # Train model with 20 hidden layers
        model_20 = model20(nodes)
        rscore, epochf = trainmodel(model_20, x_trainstd, y_trainstd, train
                                    , val,y_teststd,  x_teststd)
        results['layers20'].append([nodes, f, rscore, epochf])

        f+=1
# Prin results
for key in results.keys():
    values = list(results[key])
    print(max(np.array(values)[:,2]))
```

Appendices

14. Python Code - First Optimization Model

```
# Create a new optimization model
model = gp.Model('Truckfake')

# Get the set of all trucks
trucks = frationew.keys()
# Get the set of all time periods
periods = timestamps

# Add binary decision variables for each truck and time period
y_truck_time = model.addVars(trucks, periods, vtype=GRB.BINARY, name='
                                     truckbin')

# Add constraints to the model

## Only one truck can be active at each time period
one_truck_time = model.addConstrs(
    gp.quicksum(y_truck_time[truck, time] for truck in trucks) <= 1 for
                                          time in timestamps)

## Each truck can only fuel once
time_taken = model.addConstrs(
    gp.quicksum(y_truck_time[truck, time] for time in timestamps) <= 1
                                         for truck in trucks)

## Each truck must have enough fuel to complete its assigned time
                                     periods
norunfuel = model.addConstrs(
    gp.quicksum(y_truck_time[truck, time]* fratiopt[truck][time] for
                                         time in timestamps) >=0 for
                                         truck in trucks)

## All trucks should fuel
if all=='Yes':
    model.addConstr(gp.quicksum(
        y_truck_time[truck, time] for truck in trucks for time in
                                            timestamps
            ) >= len(trucks))

## Only 15 minutes allowed for fueling (previously 5)
time_to_fuel = pd.Timedelta(15, 'm')

### Find the earliest and latest time periods
min_t = min(periods)
max_t = max(periods)
```

```
### Add constraints to ensure that no truck is refueling while on duty
for t in periods:
    t_fuel_before = time_to_fuel
    t_fuel_after = time_to_fuel

    if t - min_t < time_to_fuel:
        t_fuel_before = t - min_t
    if max_t - t < time_to_fuel:
        t_fuel_after = max_t - t

    # Define the range of times when the truck cannot refuel
    timerange_a = [pd.to_datetime(dt.strftime('%Y-%m-%d T%H:%M')) for
                                        dt in datetime_range(t-
                                        t_fuel_before, t, timedelta(
                                        minutes=1))]
    timerange_b = [pd.to_datetime(dt.strftime('%Y-%m-%d T%H:%M')) for
                                        dt in datetime_range(t+pd.
                                        Timedelta(1, 'm'), t+
                                        t_fuel_after, timedelta(
                                        minutes=1))]
    if len(timerange_a) > 1 and len(timerange_b) > 1:
        bef = gp.quicksum(y_truck_time[i, tt] for tt in timerange_a for
                                             i in trucks)
        after = gp.quicksum(y_truck_time[i, tt] for tt in timerange_b
                                            for i in trucks)
        model.addConstr(bef+after <=1)
    if len(timerange_a) <= 1 and len(timerange_a) > 0 and len(
                                        timerange_b) > 1:
        current = gp.quicksum(y_truck_time[i,timerange_a[0]] for i in
                                            trucks)
        after = gp.quicksum(y_truck_time[i, tt] for tt in timerange_b
                                            for i in trucks)
        model.addConstr(current+after <=1)
    if len(timerange_b) <= 1 and len(timerange_b) > 0 and len(
                                        timerange_a) > 1:
        bef = gp.quicksum(y_truck_time[i, tt] for tt in timerange_a for
                                             i in trucks)
        current = gp.quicksum(y_truck_time[i,timerange_b[0]] for i in
                                            trucks)
        model.addConstr(bef+current <=1)
# Objective function
max_mf = gp.quicksum(
    mfopt[truck][time]*y_truck_time[truck,time]
    for truck in trucks
    for time in periods)

## Set the objective of the model to maximize the difference between
                                     the match factor and the time
                                     taken
model.setObjective(max_mf, GRB.MAXIMIZE)

# Solve the optimization model
model.optimize()
```

Appendices

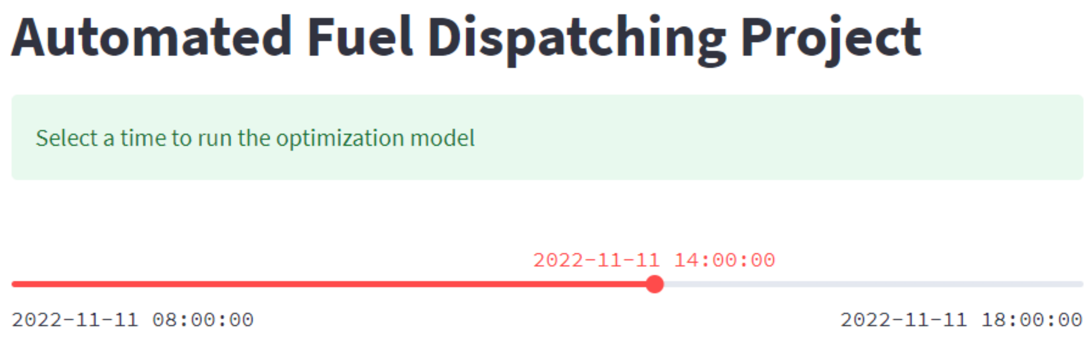
15. Python Code - Second Optimization Model

```
    # Create a Gurobi optimization model instance
    model = gp.Model('Truckfake2')

    # Define the set of all trucks
    trucks = frationew.keys()

    # Define the set of all time periods
    periods = timestamps

    # Add binary decision variables for each truck and time period
    y_truck_time = model.addVars(trucks, periods, vtype=GRB.BINARY, name='
                                                        truckbin')

    # Add constraints to the model
    ## Only one truck can be active at each time period
    one_truck_time = model.addConstrs(
        gp.quicksum(y_truck_time[truck, time] for truck in trucks) <= 1 for
                                                             time in timestamps)

    ## Each truck can only fuel once and all should fuel
    time_taken = model.addConstrs(
        gp.quicksum(y_truck_time[truck, time] for time in timestamps) <= 1
                                                            for truck in trucks)

    ## All trucks should fuel
    model.addConstr(gp.quicksum(y_truck_time[truck, time] for truck in
                                                        trucks for time in timestamps) >=
                                                         len(trucks))

    ## Each truck must have enough fuel to complete its assigned time
                                                        periods
    norunfuel = model.addConstrs(
        gp.quicksum(y_truck_time[truck, time] * fratiopt[truck][time] for
                                                            time in timestamps) >= 0 for
                                                            truck in trucks)

    # Calculate maximum match factor
    max_mf = gp.quicksum(mfopt[truck][time] * y_truck_time[truck, time] for
                                                         truck in trucks for time in
                                                        periods)

    # Set the objective of the model to maximize the difference between the
                                                         match factor and the time taken
    model.setObjective(max_mf, GRB.MAXIMIZE)
    # Solve the optimization model
    model.optimize()
```

Appendices

16. Website features

Refer to the following website application to check the described features below web (link), which has been developed using the Streamlit framework in Python.]

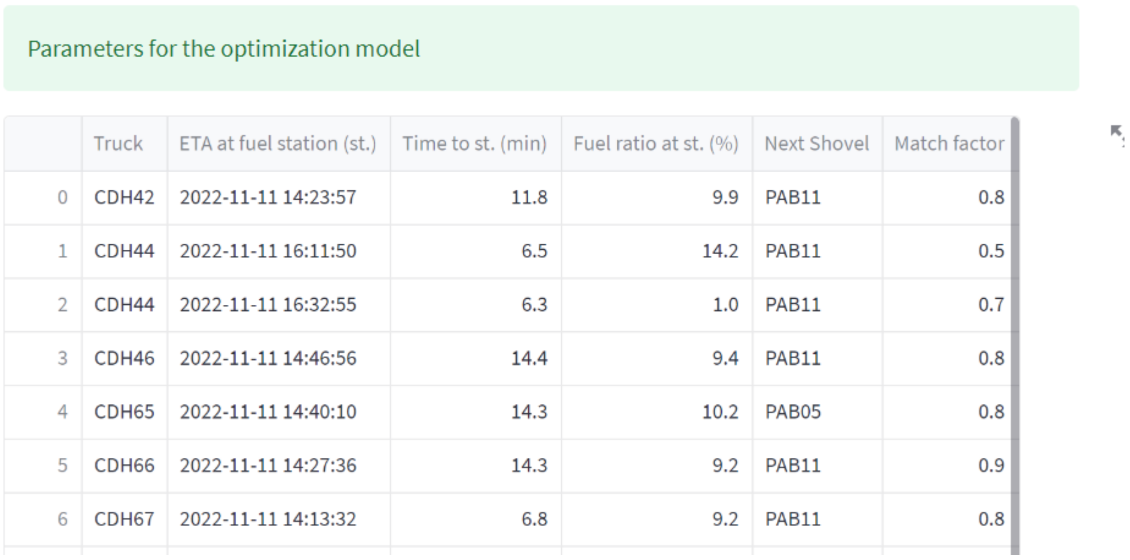

Automated Fuel Dispatching Project

Select a time to run the optimization model

Figure 16.1: 2-hour Time Slider within a Shift

Appendices

17. Website feature - Time Slider



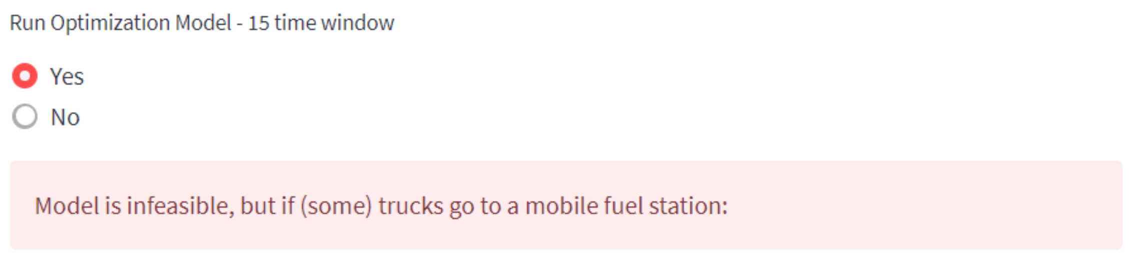
Parameters for the optimization model

	Truck	ETA at fuel station (st.)	Time to st. (min)	Fuel ratio at st. (%)	Next Shovel	Match factor
0	CDH42	2022-11-11 14:23:57	11.8	9.9	PAB11	0.8
1	CDH44	2022-11-11 16:11:50	6.5	14.2	PAB11	0.5
2	CDH44	2022-11-11 16:32:55	6.3	1.0	PAB11	0.7
3	CDH46	2022-11-11 14:46:56	14.4	9.4	PAB11	0.8
4	CDH65	2022-11-11 14:40:10	14.3	10.2	PAB05	0.8
5	CDH66	2022-11-11 14:27:36	14.3	9.2	PAB11	0.9
6	CDH67	2022-11-11 14:13:32	6.8	9.2	PAB11	0.8

Figure 17.1: Parameters for the Optimization Model

Appendices

18. Website feature - Radio Button to Show Results from the First Optimization Model



Run Optimization Model - 15 time window

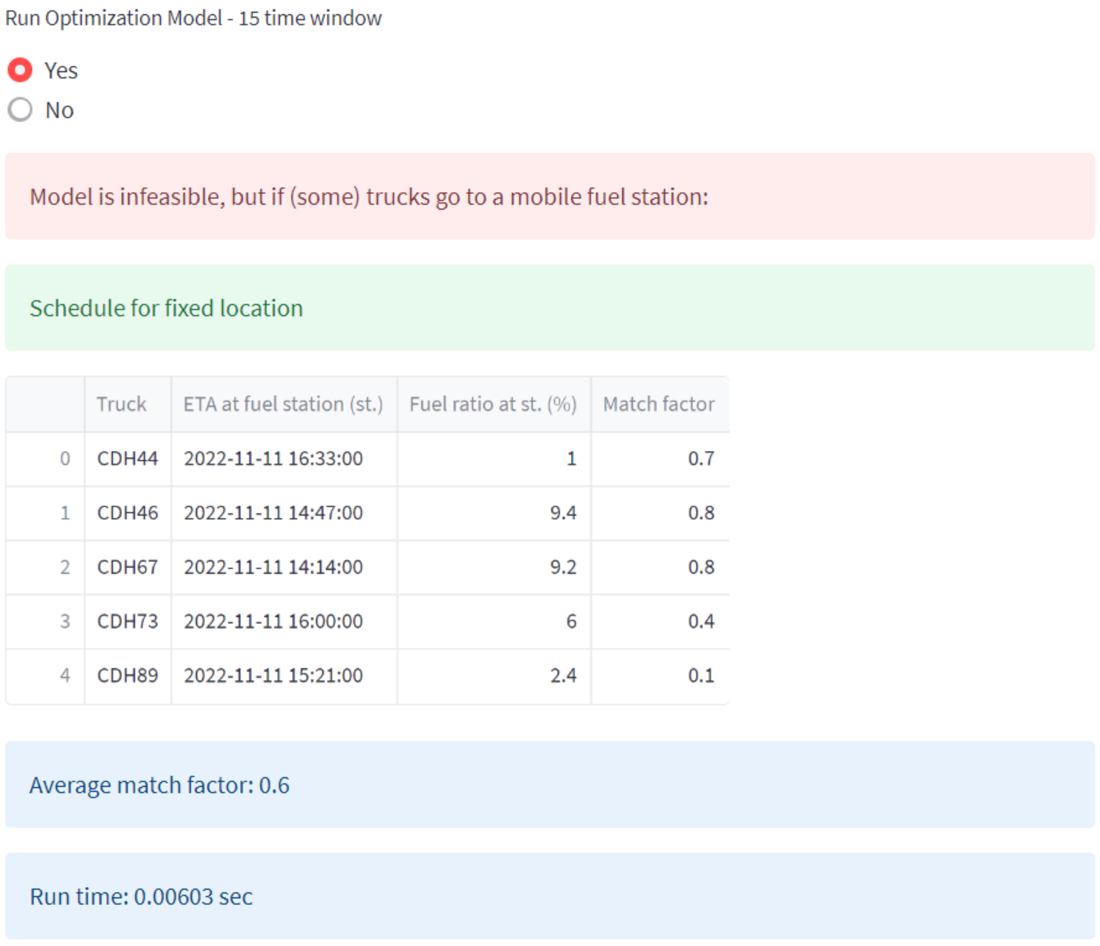
[✓ Yes] No

Model is infeasible, but if (some) trucks go to a mobile fuel station:

Figure 18.1: Radio Button to Show Results from the First Optimization Model

Appendices

19. Website feature - Results from the First Optimization Model (mobile fuel station)



Run Optimization Model - 15 time window

● Yes
○ No

Model is infeasible, but if (some) trucks go to a mobile fuel station:

Schedule for fixed location

	Truck	ETA at fuel station (st.)	Fuel ratio at st. (%)	Match factor
0	CDH44	2022-11-11 16:33:00	1	0.7
1	CDH46	2022-11-11 14:47:00	9.4	0.8
2	CDH67	2022-11-11 14:14:00	9.2	0.8
3	CDH73	2022-11-11 16:00:00	6	0.4
4	CDH89	2022-11-11 15:21:00	2.4	0.1

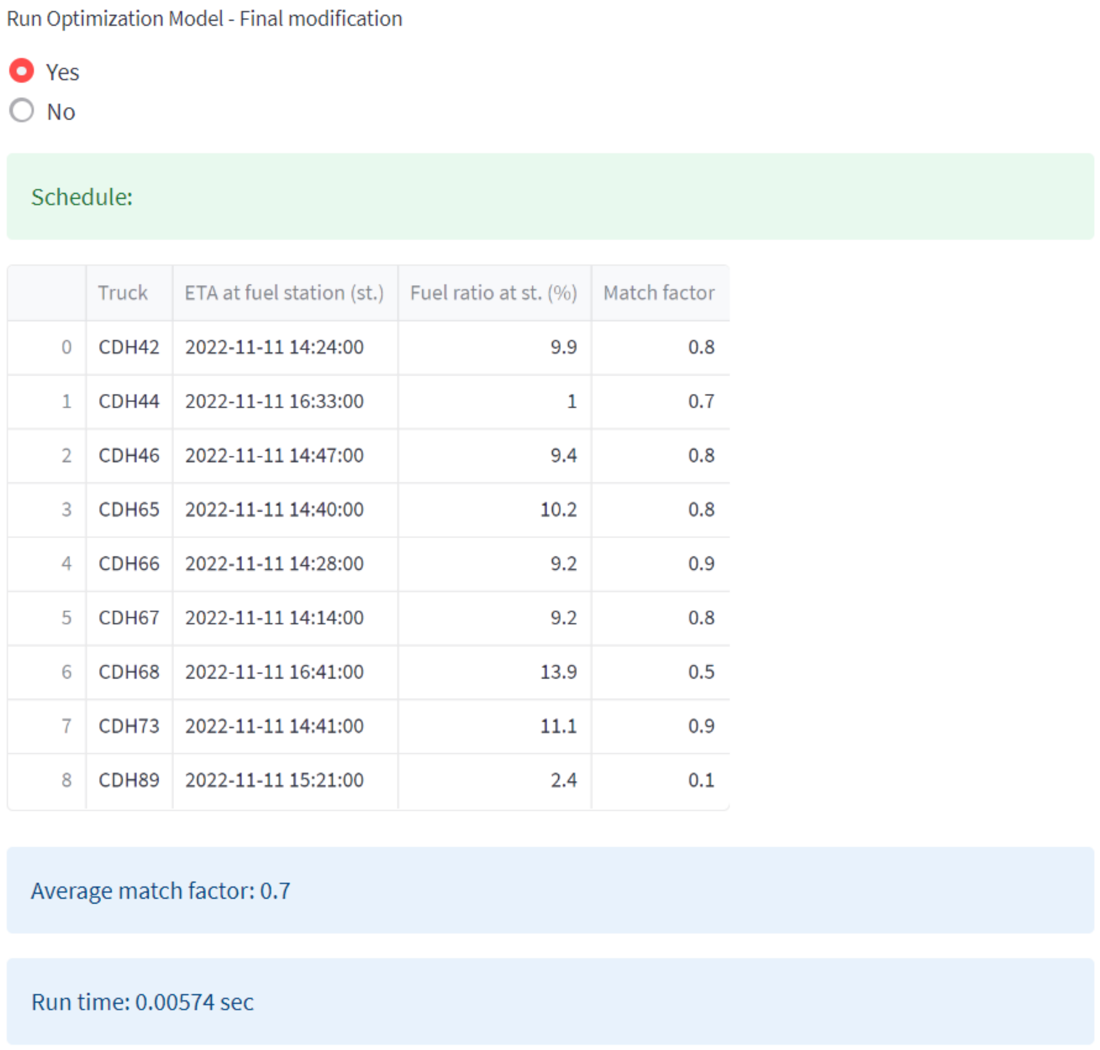
Average match factor: 0.6

Run time: 0.00603 sec

Figure 19.1: Results from the First Optimization Model (mobile fuel station)

Appendices

20. Website feature - Radio Button to Show Results from the Second Optimization Model



Run Optimization Model - Final modification

[✓ Yes] No

Schedule:

	Truck	ETA at fuel station (st.)	Fuel ratio at st. (%)	Match factor
0	CDH42	2022-11-11 14:24:00	9.9	0.8
1	CDH44	2022-11-11 16:33:00	1	0.7
2	CDH46	2022-11-11 14:47:00	9.4	0.8
3	CDH65	2022-11-11 14:40:00	10.2	0.8
4	CDH66	2022-11-11 14:28:00	9.2	0.9
5	CDH67	2022-11-11 14:14:00	9.2	0.8
6	CDH68	2022-11-11 16:41:00	13.9	0.5
7	CDH73	2022-11-11 14:41:00	11.1	0.9
8	CDH89	2022-11-11 15:21:00	2.4	0.1

Average match factor: 0.7

Run time: 0.00574 sec

Figure 20.1: FRadio Button to Show Results from the Second Optimization Model

Appendices

21. Website feature - Download Button to Obtain Lp/ Mps files

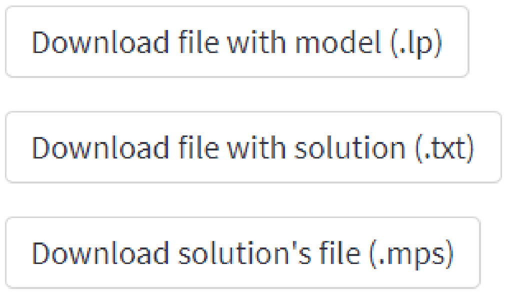
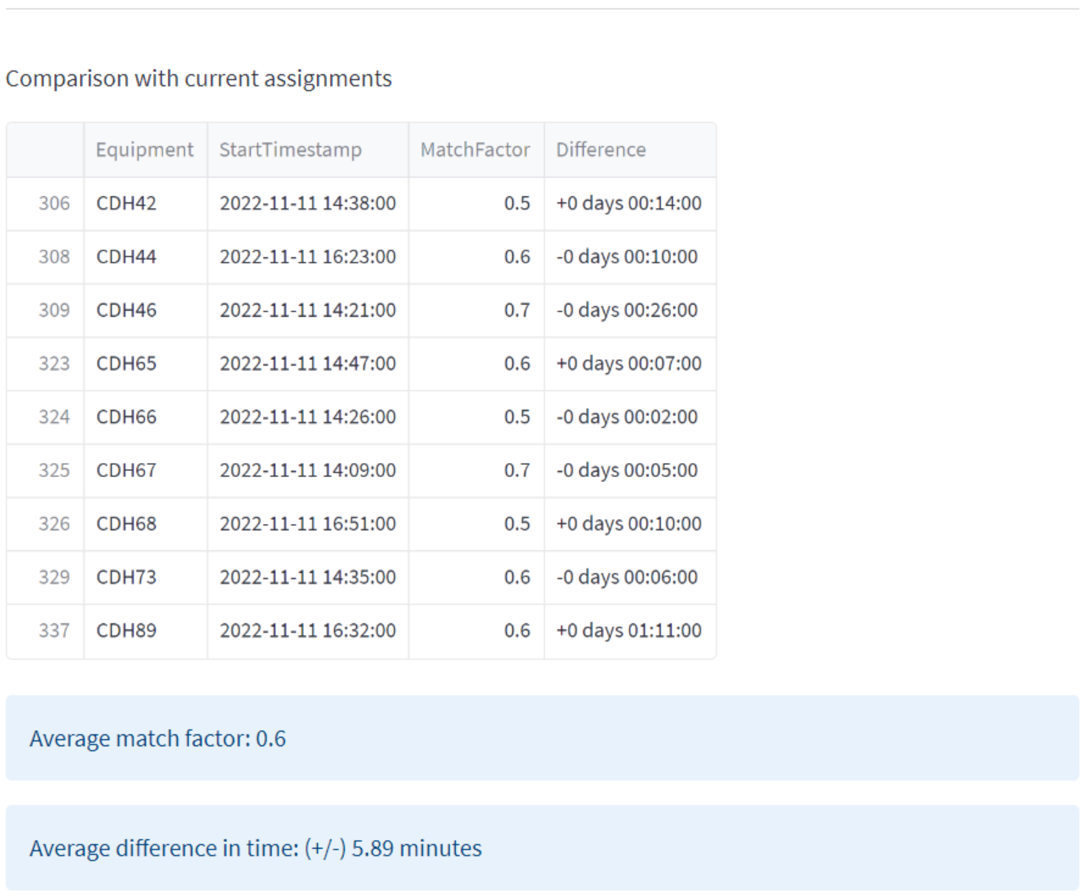

Download file with model (.lp)
Download file with solution (.txt)
Download solution's file (.mps)

Figure 21.1: FDownload Button to Obtain Lp/ Mps Files

Appendices

22. Website feature - Results from the Comparison with Current Fueling Assignments



Comparison with current assignments

	Equipment	StartTimestamp	MatchFactor	Difference
306	CDH42	2022-11-11 14:38:00	0.5	+0 days 00:14:00
308	CDH44	2022-11-11 16:23:00	0.6	-0 days 00:10:00
309	CDH46	2022-11-11 14:21:00	0.7	-0 days 00:26:00
323	CDH65	2022-11-11 14:47:00	0.6	+0 days 00:07:00
324	CDH66	2022-11-11 14:26:00	0.5	-0 days 00:02:00
325	CDH67	2022-11-11 14:09:00	0.7	-0 days 00:05:00
326	CDH68	2022-11-11 16:51:00	0.5	+0 days 00:10:00
329	CDH73	2022-11-11 14:35:00	0.6	-0 days 00:06:00
337	CDH89	2022-11-11 16:32:00	0.6	+0 days 01:11:00

Average match factor: 0.6

Average difference in time: (+/-) 5.89 minutes

Figure 22.1: Results from the Comparison with Current Fueling Assignments

Vita

Luis Fernando Larota Machacca was born on December 1996 in Arequipa, Peru. He obtained his bachelor's degree in Mining Engineering at Universidad Catolica de Santa Maria in Arequipa, Peru, in 2017 after he completed an exchange program for one semester in Pontificia Universidad Catolica de Chile 2017 in Santiago, Chile. He worked as a mining engineer in Peru and Chile from 2018 to 2021. He holds membership with the Society for Mining, Metallurgy Exploration and the International Society of Explosives Engineers.