Western Australia School of Mines

Mine Waste Dump Optimisation Based on Carbon Footprint

Namgyel Sithup Tobgay

0000-0001-9460-0485

This thesis is presented for the Degree of
Master of Philosophy
of
Curtin University

November 2024

Declaration of Authorship

I, Namgyel Sithup Tobgay, declare that this thesis titled, “Mine Waste Dump Optimisation Based on Carbon Footprint” and the work presented in it are my own, I further confirm that:

- To the best of my knowledge and belief this thesis contains no material previously published by another person except where due acknowledgement has been made.
- This thesis contains no material which has been accepted for the award of any other degree or diploma in any university.
- Any citations from prior published works are appropriately referenced.
- For material jointly contributed or any help received, an author contribution statement clearly showing and acknowledging the degree of such assistance is provided.

Signed: _____

Date: 20th January 2025

Publications list

The publications arising from this current research is as follows:

- Tobgay, N., Topal, E., & Asad, W. (2024). Mine waste dump optimisation to minimise carbon footprint. (Under Progress)

Acknowledgements

First and foremost, I would like to express my deep and sincere gratitude to my supervisors, Professor Erkan Topal and Professor Mohammad Waqar Ali Asad for their excellent mentoring during this work.

I would also like to thank all the Curtin University staff for providing such a wonderful atmosphere that promotes academic excellence.

Abstract

The efficiency of modern-day open-pit mining operations largely hinges on optimising profitability while minimising the footprint of environmental impact and carbon emissions. Decarbonisation is becoming a key performance metric as industries gradually turn to cleaner production systems to comply with the increasingly stringent environmental legislation. To achieve this, Mixed Integer Programming (MIP) model, which optimises the net present value (NPV) while minimising the carbon footprint associated with the haulage of mined material (ore or waste) from an open pit mine to designated dump sites and is proposed. By integrating advanced mathematical modelling techniques, the study aims to refine waste dump design and establish optimal haulage routes that minimise haulage cost and carbon emissions.

Three MIP models are proposed for this purpose: The first model maximises the discounted cash flow from production and waste dump scheduling considering only haulage cost, the second model introduces and integrates the carbon cost into the first model minimising the carbon cost taking into consideration the results of the production schedule from the first model. The third model is the combination of the first and second model maximising the discounted cash flow from production and waste dump scheduling considering both haulage and carbon cost. The study introduces a novel approach to calculating actual haulage distances to capture more accurate cost assessments, avoiding traditional flat-distance approximations that are crude and prone to exaggeration of results if used as inputs to downstream optimisation processes.

The primary objective of this research is to establish whether the use of exact haulage distance in the optimisation of production schedule and waste dump schedules in open-pit mines can improve the prediction and mitigation of carbon emissions and related haulage cost but also optimising the mine waste dump design. By recalibrating how haulage costs are calculated and integrating carbon cost assessments, the study provides a comprehensive analysis that could redefine best practices in mine scheduling and waste management.

Contents

List of Figures

List of Tables

1.0 INTRODUCTION

The mining industry faces significant challenges in balancing economic efficiency with environmental sustainability. As global awareness of environmental impacts increases, the sector is under increasing pressure to reduce its carbon footprint and improve sustainability practices. This thesis focuses on optimising waste dump operations in open-pit mining to reduce carbon emissions associated with haulage activities, a major contributor to the environmental impact of mining operations.

The urgency to address these challenges is heightened by the mining industry's substantial contribution to global emissions, primarily through relatively high consumption of diesel fuel for truck haulage operations. Trucks are primarily used for transporting ore and waste materials to processing facilities and waste dumps. Consequently, optimising truck haulage operations present a clear opportunity for reducing carbon emissions. The current industry standard relies on simplified models that often do not accurately reflect the true haulage path distances. As a result, the accuracy of the estimated costs and related environmental impact in the form of carbon emissions is unreliable. This study seeks to close that gap using the proposed optimisation method.

This research proposes the use of development of Mixed Integer Programming (MIP) models to optimize the net present value (NPV) of mined material transport while minimising the carbon emissions associated with these activities. By integrating detailed calculations of haulage distances and employing a more nuanced approach to dump site operations, the study aims to present a methodologically sound and environmentally beneficial strategy for mine waste management.

1.1 Problem statement

In open-pit mining, diesel-powered haul trucks play a crucial prole in transporting ore and waste material to various designated dump sites. Unfortunately, these trucks are also a major source of carbon emissions due to the large volume of diesel consumed in their diesel combusting engines. This contributes heavily to its carbon emissions footprint and points to a significant environmental burden from the mining industry. With the world increasingly focused on sustainability, there's growing pressure on the mining sector to cut these emissions and become more environmentally friendly.

Currently, the methods used for managing mine waste are often too simplistic. Most mathematical models depend on Euclidean flat-distance calculations to schedule haulage and optimise haulage activities for mined material. This approach fails to accurately account for the actual conditions and distances involved, leading to major differences between planned and actual performance. Such

discrepancies compromise the ability to accurately measure and manage the industry's carbon emissions.

Moreover, the design of waste dumps and the management of truck haulage operations often overlook potential efficiencies that could reduce carbon emissions. Despite statutory and regulatory requirements getting stricter and society demanding greener practices, many opportunities to minimise carbon emissions through smarter haulage routes and operational planning are missed by using flat distances estimates instead of more accurate haulage distances.

The use of outdated methods and the industry's slow adoption of innovative technologies contribute to this problem. A reconsideration of how waste dump is managed and optimised is a crucial step towards cleaner production, integrating approaches that can dramatically improve both environmental and economic outcomes. This thesis seeks to address these challenges by proposing mathematical models that refine and optimise mine waste dump and haulage schedules using real haulage distance, aiming to cut down on carbon emissions and improve operational efficiency.

1.2 Objectives

The primary objectives of the study that are outlined as follows:

- To create a Mixed Integer Programming (MIP) model to maximize the discounted cash flow and net present value (NPV) of production and waste dump scheduling of an open pit mine.
- To create a Mixed Integer Programming (MIP) model to minimize the carbon cost during material haulage to designated material dump sites.
- To create a combined Mixed Integer Programming (MIP) model to maximise the discounted cash flow and NPV of an open pit production schedule, including optimisation of haulage and carbon costs for haulage to the designated material dumps sites (processing plant and waste dumps).
- Calculating real haulage distance rather than equivalent flat distance to obtain actual cost values.

1.3 Motivation and scope of research

The primary motivation for this research emerges from the pressing environmental and economic challenges within the mining industry. A significant concern is the environmental impact of diesel-powered haul trucks, which are essential for transporting ore and waste materials in open-pit mining but also major contributors to carbon emissions. As global environmental awareness increases, there is a heightened demand from the mining industry to reduce these emissions and enhance sustainability. Moreover, the industry is facing a growing regulatory pressure that mandates more sustainable and cleaner environmental practices. This research aims to provide solutions that help the industry to align with these growing requirements.

Operational inefficiencies in mining also influence this research, particularly the inaccuracies in cost estimation caused by traditional flat-distance modelling approaches for truck haulage optimisation studies. These inaccuracies can lead to significant financial losses and misaligned resource allocation. The research seeks to improve economic efficiency and waste dump management practices by introducing more precise distance measurements and optimisation techniques.

The scope of this involves key research areas designed to address the objectives outlined earlier. Initially, the research involves creating a new mathematical model that optimises the layout and scheduling of materials to the processing plant, as well as to mine waste dumps and haulage routes. These models aim to decrease carbon emissions and operational haulage costs and will include realistic haulage distance calculations to improve the precision of cost and carbon cost estimations. This marks a significant advancement over traditional methods that rely on Euclidean flat distance calculations.

The application and effectiveness of these models will be tested through simulations using data from actual mining operations. this will validate and adapt the models to accurately reflect real-world conditions.

Finally, the thesis will discuss the potential implications of the research findings and how the proposed models may be used to reduce carbon emissions and enhance operational efficiencies.

1.4 Significance and relevance

The significance of this thesis is rooted in its profound potential to influence environmental sustainability and operational efficiency within the mining and related industries. By developing and applying novel Mixed Integer Programming (MIP) models, this research offers potential solutions to reduce the carbon emissions from open-pit mining operations. the optimisation models proposed in this study offer innovative strategies to minimize diesel fuel consumption through better haulage route selection and waste dump scheduling. This approach not only aligns with international environmental standards but also optimises resource utilisation leading to decreased operational costs and enhanced profitability. Such improvements are vital for the mining industry's commitment to sustainable practices.

Furthermore, the findings of this research have the potential to influence policy and regulatory frameworks. By demonstrating that considerable reductions in carbon emissions can be achieved through optimised waste dumping management strategies, this thesis adds to the growing advocacy for environmental regulations.

Further, the practical tools developed through this thesis can be adopted by mining companies as a basis to enhance their operational and environmental performance. The proposed models are adaptable and have broad applicability across different mining contexts, which opens their use for extensional applications in related industries.

Lastly, by integrating these models developed in this research, mining companies can use these models as a foundation to generate strategies to bolster their public image and meet their corporate social responsibilities. In an industry frequently criticized for its environmental impacts, demonstrating a commitment to reducing carbon footprints can significantly enhance a company's reputation, strengthen stakeholder trust, and potentially improve its market position.

1.5 Thesis overview

Chapter 1 introduces the study by presenting the problem statement, which emphasizes the necessity of establishing an optimised dumping schedule that integrates seamlessly with mine production schedules to reduce both haulage costs and carbon emissions. The chapter outlines the thesis's objectives, which include the development of mixed integer programming (MIP) models focused on maximising net present value (NPV) and minimising carbon costs during transportation of materials to production and waste sites.

Chapter 2 conducts a thorough literature review. It reviews various modelling approaches, such as MIP and linear programming, used to optimise mine production and waste dump scheduling. The discussion extends to the economic and environmental impacts of transportation within mining operations, supported by a wealth of scholarly articles and empirical studies.

Chapter 3 describes the research methodology, detailing the mathematical modelling approach used, from the creation of models to the analysis of solutions. This chapter highlights the importance of a comprehensive framework that integrates mining operations with waste rock dumping systems, showcasing the utility of Mixed Integer Programming (MIP) models in optimising both operational and environmental outcomes. Additionally, the chapter introduces detailed calculations of haulage and carbon costs, which are critical for accurately assessing the economic performance and environmental consequences of mining activities. These calculations involve determining fuel consumption based on the haulage distance, load factor and estimating carbon emissions to translate these into cost assessments through environmental pricing models. The main goal of the methodology is to enhance the NPV by efficiently managing the flow of materials to processing plants and waste dump. This comprehensive framework establishes the core foundation for the investigation and analyses presented in the subsequent chapters.

Chapter 4 details the practical application of the Mixed Integer Programming (MIP) models developed in the thesis. The implementation involves setting up the models with specific parameters and variables, including data inputs such as block dimensions, ore grades, and hauling distances which are crucial for optimisation outcomes. Results from the application of these models to a real gold mine scenario are presented, focusing on the outcomes from the Haulage Cost Model, Carbon Cost Model, and the Combined Model. This comparison highlights the impacts on net present value (NPV), haulage and carbon costs, and the progression of waste dump filling over the operational period.

Chapter 5 synthesizes the findings from the research, offering a summary of significant results and their implications for the mining industry, particularly in terms of minimising carbon emissions and optimising haulage costs. It outlines the contribution of this research to the field, highlighting how it extends existing knowledge in mining operations optimisation with a focus on sustainable practices and carbon footprint reduction. Recommendations for future research are provided, suggesting expansions such as incorporating more realistic scenarios with multiple pits and waste dumps, using non-square dump shapes, and improving model accuracy by adopting stochastic elements. Practical implications for mining companies are discussed, recommending on the possible application of these findings to enhance operational efficiencies and achieve more sustainable mining practices, including potential integration with truck dispatching systems to optimize real-time decisions and transportation logistics.

2.0 LITERATURE REVIEW

Trucks and shovels are commonly employed in mining operations for their ability to transport material over considerable distances. In open-pit mines, utilizing trucks and shovels incurs the highest costs with material transportation accounting for 60% of total mining expenses (Li, 1990).

In open-pit mining, where diesel fuel predominates, the energy consumed by shovels and haul trucks significantly contributes to haulage costs, with haul trucks responsible for about 30% of these costs (Siami-Irdemoosa & Dindarloo, 2015). Enhancing energy efficiency is encouraged as it lowers operational expenses. Research indicates that human actions have played a substantial role in increasing greenhouse gas emissions, which have contributed to the rise in global temperatures (Rosa & Dietz, 2012). Given that fuel consumption during haulage operations emit huge amount of carbon emission in surface mining, various industrial sectors have taken steps to decrease these emissions (Liu et al., 2021). Focusing research on minimizing the carbon footprint helps reduce greenhouse gas emissions while enhancing energy efficiency and improving air quality.

2.1 Production and mine waste dump scheduling optimisation using Mixed integer programming.

Surface mining operations involve a variety of destinations streams such as processing, stockpiling, refining, and waste disposal. A key objective in these operations is to maximize Net present Value while minimising waste rock transportation costs. Strategies to achieve this include reducing haulage distances and heights, increasing truck capacity and equipment reliability, and implementing efficient dispatch systems Li et al. (2014); Lizotte and Bonates (1987). Extensive research using linear programming (LP), integer programming (IP), mixed integer programming (MIP) and dynamic programming (DP) models has been conducted to address production scheduling of ore. (Johnson, 1968) identified the importance of maximising NPV while adhering to the constraints set by mining and processing capacities, ore grade, pit slope, and other critical input parameters. Since Johnson's pioneering research, there has been ongoing progress in mathematical modelling, complemented by developments in hardware and software. This has enabled the routine creation of mine plans and schedules at the mine sites. A few notable example of these models and implementation have been presented on the following references (Caccetta & Hill, 2003; Dagdelen, 1986; Dowd & Onur, 1993; Gershon, 1983; Groeneveld & Topal, 2011; Groeneveld et al., 2019; Johnson, 1968; Mai et al., 2018; Newman et al., 2010; Tolwtnski & Underwood, 1996; Topal & Ramazan, 2012).

Notably, the complexities of waste rock dump management are not often well addressed, typically resulting in the superficial allocation of waste rock to dumps. Noted by Scholars such as (Fathollahzadeh et al., 2021; Osanloo et al., 2008) and (Das et al., 2023), there is a scarcity of research dedicated to optimisation of waste dump, especially those employing in reduction of haulage cost and carbon emissions while maximising NPV using real distance. Given that hauling waste rock accounts for 50-60% of total operational expenses, influenced by the stripping ratio and actual distance, this study will focus on maximising NPV by optimising mine waste dump, emphasizing on haulage cost and carbon cost efficiency.

Early research into mine waste dump planning commenced in the early 2000's. Williams et al. (2008) proposed a model aimed at optimising rock dump scheduling in mining operations with primary focus to minimise haulage costs associated with transporting waste rock from open pits to dumps, while also considering the environmental implications, particularly the management of acid rock drainage (ARD). This is particularly significant as ARD can lead to severe contamination of surface and ground waters, posing threat to both environment and the mining operation itself.

The model presented incorporates a range of operational and environmental parameters into its formulation, including the categorization of waste rock in benign and reactive types, and detailed haulage cost calculations based on distance and material type. The use of mixed-integer linear programming (MILP) allowed for the consideration of multiple scheduling constraints and objectives such as minimising cost and environmental impacts simultaneously, which is a step beyond traditional methods that focus predominantly on economic metrics. The research included a case study using hypothetical data to demonstrate the functionality and potential outcomes of the model. The results highlight the model's capability to effectively schedule waste rock dumping in a manner that optimises haulage route and schedules, significantly reducing cost and mitigating environmental harm. However, the study considered using hypothetical data but also lift-by lift waste dump construction sequencing and Euclidean flat distance and not multi-lift dump construction and actual haulage distance.

Li et al. (2013) formulated two MIP models to optimize waste rock dumping in open pit mines. The study addresses the substantial costs associated with waste rock hauling and dumping, which often comprise a large portion of the operational expense in mining. The research includes two MIP models which are location optimisation (LOP), and Truck Balance (TBA) models designed to minimize hauling costs while ensuring environmental compliance, specifically by encapsulating potential acid-forming waste rocks. These models help in generating detailed plans for where the waste rock should be dumped for each mined block to optimize costs and reduce environmental impact. The paper highlights the importance of integrating the detailed waste dumping plans with mining schedule to

balance economic and environmental objectives efficiently. The models focus on strategically positioning waste work to decrease operational expenses and minimise long-term environmental hazards. However, it does not concurrently optimise the movement of both ore and waste rock.

Li et al. (2014) later combined these models to address the limitations of the initial studies, refining their approach to optimize waste rock placement in open pit mining operations. By integrating the methodology from previous research, they developed a comprehensive MIP model that not only minimised operational costs and environmental risks but also improved the overall efficiency and adaptability of mine operations. The combined model aimed to tackle the challenges identified in previous studies by enhancing the flexibility of waste rock placement strategies and providing a robust solution for managing the environmental impacts, particularly in preventing acid mine drainage. Further, the combined model also revealed that predefined production schedules required by earlier models could lead to inaccuracies in haulage costs, economic worth, and categorisation of ore and waste.

(Shishvan & Sattarvand, 2015) developed a metaheuristic approach to tackle the long-term production planning challenges of open-pit mines. Recognizing the limitation of traditional mathematical programming due to the immense number of decision variable involved, the study proposes the use of Ant Colony Optimisation (ACO), a method modelled after how ants locate the shortest routes to food sources, offering a more efficient alternative. The study demonstrates the application of ACO through case study on a real-scale Copper-Gold deposit, showing how ACO can consider multiple objective functions and complex constraints, and integrate penalties into the objective function from deviating from targets. The research reveals that ACO, particularly through its variants Max-Min Ant System (MMAS) and Ant Colony System (ACS), effectively enhances the net present value of mining operations by up to 12 % significantly outperforming traditional methods such as graph theoretical approach, dynamic programming and network models. However, these models often focus only on scheduling blocks for waste dumps without considering a comprehensive schedule for waste placement.

Li et al. (2016) introduces MIP model designed to optimize waste management and truck logistics in mining operations. Three distinct MIP models were created to maximize truck productivity, minimize truck requirement deviations and combined model of both objectives. While implementing the models on a real-world large-scale open pit mine, these models generated detailed schedules for material placement and predicted the changes and developments in the physical landscape of the mine are over the course of its operational life. The study highlighted the effectiveness of MIP models in improving economic and environmental aspects of mining operations through strategic waste rock management

and logistical optimisation but not consider simultaneous optimisation for ore and waste rock movement and actual haulage distance.

Puell Ortiz (2017) presented a structured approach for enhancing waste dump design in large-scale open pit mining operations, primarily aimed at minimising the total haulage costs. The study integrates multiple optimisation strategies including linear programming (LP) to achieve economical and practical solutions for dump design. The methodology outline consists of three main stages: the formulation of a dump model through LP that minimizes hauling distance and cost; an iterative process to refine this model based on real-world topological constraints; and the application of this method to a waste rock dump case to illustrate its effectiveness. A significant portion of the paper discusses the application of this methodology to a real-world scenario where it demonstrates potential const savings and improved operational efficiency in dump management. The study highlights the importance of integrating economic, environmental and operational factors into the dump design process to optimize resource use and cost-efficiency in open pit mining operations. However, these models did not consider the specific placement of waste rock or environmental constraints and relied on an effective production schedule.

Traditional models such as the Open Pit Mine Production Schedule (OPMPS) utilize economic block models to establish the most financially beneficial sequence of mining operations, aiming to maximize the NPV within various technical and economic limitations. Once an extraction sequence is defined by the OPMPS model, a secondary Waste-Dump Schedule (WDS) is crafted to manage the placement of waste material within designated dumps. However, these models often focus solely on the scheduling of blocks to waste sites and neglect the intricate details of waste placement and the progressive development of waste dumps. The oversight of these details can lead to increased operational costs and environmental impacts, diminishing the models' practical effectiveness. Building on this, (Fu et al., 2019) advanced the research by providing a combined solution for both ore production and waste dump scheduling aimed at maximizing the NPV simultaneously. This approach stands out because it integrates the decision-making processes for both ore extraction and waste rock disposal into a single framework, addressing inefficiencies in traditional methods which typically handle these tasks separately. By calculating optimal sequences and placements that consider material haulage costs and environmental impacts, this model strives to maximize the NPV of mining projects.

The effectiveness of the model is illustrated by a case study on a gold mining operation, where it was shown to outperform existing methods (traditional and two-step MIP models) in terms of NPV and practicality. The study emphasizes the model’s capability to dynamically handle the complexities of mining operations, such as varying haulage costs and environmental constraint related to acid mine

drainage (AMD). This is achieved by sophisticated scheduling that strategically positions potential acid-forming (PAF) and non-acid forming (NAF) material within waste dumps to minimize environmental risk. The integrated approach ensures that the block's economic and environmental attributes are considered in the scheduling process, leading to more efficient resource use and potentially lower overall costs.

(Das et al., 2022) provides insights into optimising mine waste dump management with a focus on minimising environmental impacts. The study introduces a mixed integer programming model that integrates in-pit dumping as part of the mine scheduling process, specially tailored to stratified deposits. The model not only considers the operational efficiency by optimising waste rock disposal via in-pit dumping but also significantly reduces the environmental footprint by decreasing the land that are exposed to mining activities. One of the key aspects of the model is the ability to reduce haulage distances by using in-pit voids deposition, which directly contributes to lower carbon emissions from haulage operations. However, while the research highlights the reduced haulage distances and their implication for lowering carbon emissions, it primarily relies on models that do not differentiate between Euclidean and real distances. This reliance on simplified distance calculations could potentially underrepresent the actual haulage cost and carbon emissions associated with production and waste rock hauling, especially in complex topographical settings where real distances vary significantly from straight-line distances.

(Das et al., 2023) discusses the integration of waste management strategies into open-pit mining to increase net present value (NPV). The review traces the shift from traditional ore-focused approaches to advanced methods that prioritise efficient waste rock management. The authors discuss the adoption of waste rock management in mine scheduling since 2007, highlighting its impact on reducing operational and reclamation costs. The authors explore the use of Mixed Integer Programming (MIP) to optimise ore and waste rock extraction, emphasizing the move towards sophisticated practice like in-pit dumping and selective waste rock placement. These strategies not only decrease haulage distance but also aim to minimise environmental impact by carefully managing potentially acid-forming materials. This integration of advanced computational models facilitates precise material movement scheduling, improving both environmental sustainability and economic viability. While this review thoroughly examines various strategies for integrating waste rock management into mining planning, there remains a gap in specifically addressing the optimisation of waste rock dump strategies based on a comprehensive evaluation of their carbon footprint, especially using real distances instead of Euclidean distances.

(Lin et al., 2024) developed a Mixed Integer Programming (MIP) model that aligns ore production with waste dump management strategies to enhance the sustainability of open-pit mining operations. The model is designed to maximise net present value (NPV) while simultaneously addressing environmental concerns, particularly focusing on encapsulation of potentially acid-forming (PAF) waste rocks to mitigate environmental impacts. The methodology introduced in this study not only seeks to optimise economic outcomes but also significantly reduce haulage distance and greenhouse gas emissions. Specifically, the Simultaneous nixed integer programming (SimMIP) model demonstrated a 1.2% increase in NPV compared to two-step method that generates ore production and waste dump schedules separately. By integrating the management of ore and waste in a unified framework, the model ensures that environmental risks associated with acid mine drainage are efficiently managed. The implementation of this model has shown to yield higher NPV and substantial reductions in waste rock haulage and emissions when compared to traditional planning methods that treat these elements separately. However, the paper discusses the reduction in haulage distance using Euclidean flat distance and not real distance.

(Das et al., 2024) developed a Mixed Integer Programming (MIP) model that integrates ore extraction, waste rock handling, and haul road selection into a unified framework for open-pit mining. Utilising Dijkstra's algorithm, the model efficiently determines the shortest and most cost-effective haulage paths across the mining network by analysing the distances between nodes like pits and dumps. This approach not only streamlines haulage routes in real but also significantly reduces operational cost and environmental impact. The study highlights major operational improvements, including a 39% reduction in haulage distance and 44% decrease in backfilling activities compared to conventional planning methods. Furthermore, the study explores the use of meta-heuristic techniques such as Simulated Annealing to tackle the large-scale computational challenges associated with the complex optimisation problem. However, while the paper discusses the reduction in haulage distance and backfilling, it does not address reductions in carbon costs or the optimization of waste dump design. These elements are crucial for a comprehensive assessment of environmental impacts and operational efficiency in mining practices. The omission suggests that further research could be valuable, particularly to explore how optimized haul road selection could lead to lower carbon emissions and improved waste management strategies.

However, despite these advancements, the studies presented in the literature rarely incorporate measures specifically aimed at reducing carbon emissions or managing carbon costs directly, nor do they consider real-world topological variations in calculating distances, relying instead on Euclidean flat distances which may not accurately reflect actual haulage distances and costs. This gap underscores an area for potential enhancements, where future iterations could include environmental

metrics such as carbon footprint and cost management and more accurate calculations to align with growing sustainability goals in the mining industry.

2.2. Fuel consumption by haul trucks and carbon footprint

In open-pit mining, dump trucks are the primary equipment utilised for hauling materials and account for a large part of operational expenses because of their high fuel usage. Numerous research has been carried out on fuel consumption in surface mining operations.

The primary source of carbon emissions in fleet haulage stems from diesel fuel combustion, with fuel consumption rates varying across different truck models based on the operational load on their engines. Caterpillar Ltd, an organisation which manufactures mining equipment and trucks, has categorized the engine loads experienced by operating haul trucks into three specific load factors.

- "Low (20% - 30%): Continuous operation at an average gross weight less than the recommended level (excellent haul roads, and no overloading).
- Medium (30% - 40%): Continuous operation at an average gross weight approaching the recommended level (minimal overloading and good haul roads).
- High (40% - 50%): Continuous operation at or above maximum recommended gross weight (overloading and poor haul roads)" Dindarloo and Siami-Irdemoosa (2016)

(Kecojevic & Komljenovic, 2010) discovered that lowering the load factor on haul trucks substantially reduces fuel consumption and carbon gas emissions, which, in turn, decreases operational costs. They observed a unique regression between fuel consumption and power across different models of Caterpillar trucks under the same load factor. This relationship enabled them to empirically estimate the CO_2 emissions per hour. By employing a consistent conversion factor for CO_2 emissions resulting from the combustion of diesel fuel, Kecojevic managed to create an empirical model for CO_2 emissions based on a fixed load factor for various truck models.

(Sahoo et al., 2014) created a model to calculate the energy usage of dump trucks in an open pit mine. This model considers various elements, such as vehicle dynamics and engine features to determine the most efficient fuel consumption. Additionally, (Wang et al., 2016) employed a volumetric fuel consumption meter in haulage trucks to accurately and cumulatively measure fuel usage in real-time. Despite these advancements, there remains a lack of comprehensive models for production or waste dump scheduling, with current approaches largely focused on short-term planning.

(Wang et al., 2021) explores the significant fuel costs associated with haulage trucks for open pit mine operations and conclude that fuel costs can constitute up to 22% of the total operational expenses. The researchers developed a predictive model to estimate fuel consumption based on factors such as haulage distance, lifting height, and operational time. The study emphasizes how truck queuing influences fuel consumption, indicating that even seemingly minor operational details can significantly affect efficiency.

(Vukovic, 2013) analysed haul truck operations, including intersection layouts and speeds, using data processed through matrix laboratory (MATLAB) to demonstrate how operational adjustments could reduce fuel consumption and carbon emissions. (Siami-Irdemoosa & Dindarloo, 2015) used a neural network model to study the link between cyclic activity and fuel consumption, though they did not functionally connect various operational phases to fuel usage. Addressing the shortcomings of this initial study. (Dindarloo & Siami-Irdemoosa, 2016) later applied partial least squares regression (PLSR) and autoregressive integrated moving average (ARIMA) techniques to more accurately forecast fuel usage based on cyclic activities, although certain variables like material weight and dumping angle were averaged and not specifically examined, potentially missing precise fuel-saving opportunities.

In a further development, (Bajany, 2017) crafted a mixed -integer linear programming (MILP) model aimed at optimizing production scheduling to minimize fuel consumption. The study focused on an under-trucked mine where the dispatching strategy allocated multiple trucks to multiple shovels. The model is more favourable than traditional fixed dispatch method where each truck is designated to operate with specific shovels throughout a shift demonstrating fuel savings and reduced consumption per tonne of material moved. Fuel consumption is shown to decrease by 8.82% for trucks and 4.49% for the combined truck-shovel system, leading to overall fuel savings of 4.64%. This study showed that the MILP model could predict fuel usage in the truck-shovel system., specifically designed to meet waste dump requirements. However, the model has not considered concurrent optimisation of production and waste dump scheduling, minimising carbon emission or the maximisation of net present value (NPV).

Table 1: Production, waste dump optimisation and fuel consumption models

Study	Year	Considers production	Considers waste dump	solution technique	Highlights
Production and waste dump optimisation models					
Williams et al.	2008	No	Yes	Mixed Integer Linear Programming (MILP)	• First model to made available to the public that combines open pit mining with integrated

					waste dumping strategies. • A 10% increase in gradient waste dump ramp cost five times more than the cost of horizontal haulage. • Selective dumping of non-reactive and reactive rocks, utilising non-reactive rock to cover reactive waste rock, aiming for minimal environmental impact.
Li et al.	2013	No	Yes	Location based (LOP) and truck balanced (TB) model	• LOP and TB models provide automated and optimised dumping plans that details the best dumping location for each mining block. • Presents and compares multi-lift vs lift-by-lift dumping options and concludes multi option offers greater flexibility compared to lift-by-lift dumping option.
Li et al.	2014	No	Yes	Location based (LOP), truck balanced (TB) model and combination of both model	• Location based optimisation (LOP) is effective at minimising overall haulage work, resulting in lowest estimated haulage cost and minimal re-handle of waste rock. • Truck balance (TB) model provides schedule that closely match the available trucking hours, optimising the use of available resources but at a higher cost and with increased waste rock re-handling. • The Combo model achieves result that are intermediate between the LOP and TB models.
Li et al.	2016	No	Yes	Optimum Production (OP), Truck	• Generates long term-optimum rock placement schedules.

				balance (TB) and Combo model	• Various mine pits and dumping sites have been considered. • Three different models are introduced, Optimum production (OP), Truck balance (TB) and Combo model.
Jorge Puell Ortiz	2017	No	Yes	LP (Linear programming)	• Optimises dump design by minimising haulage costs, incorporating linear programming and iterative design processes. • Iterations on variables (waste dump footprint base, number of lifts, haulage distance)
Fu et al.	2019	Yes	Yes	Mixed integer programming (MIP) model	• Calculates the best schedule for production and waste rock disposal concurrently. • Production and waste rock disposal scheduling concurrently significantly enhances the operational efficiency and environmental sustainability of mining operations.
Das et al.	2022	Yes	Yes	Mixed integer programming (MIP) model	• Optimises waste management and scheduling in open-pit mines specifically with stratified deposits • Incorporates in-pit dumping strategies. • Compares in-pit versus external waste rock dumping, considering factors such as haulage costs and available space.
Lin et al.	2024	Yes	Yes	Simultaneous mixed integer programming (SimMIP) model	• Optimises ore production and waste dump scheduling simultaneously. • Considers reduction and preventing acid mine drainage (AMD) by

					encapsulation of reactive blocks with non-reactive blocks. • Considers reduction of carbon footprint.
Das et al.	2024	Yes	Yes	Mixed integer programming (MIP) model	• Optimises open pit mining operations for both ore and waste simultaneously, along with the selection of optimal haul roads. • Employs metaheuristic methods like simulated annealing and Topological sorting to address the computational challenges associated with large-scale optimisation.
Fuel consumption by haul trucks and carbon footprint					
Study	Year	Fuel consumption	Carbon footprint and cost	solution technique	Highlights
Kecojevic and Komljenovic	2010	Yes	Yes	Mathematical relationship	• Develop a mathematical model that correlates truck fuel consumption with power and engine load factor. • Determines CO_2 emission and cost. • Considers power and load factor
Indemoosa and Dindarloo	2016	Yes	No	PLSR (Partial least square regression)	• Longer loading time, idle time, and inefficient layout of mining operations were identified as critical factors to high fuel consumption. • Considers Power and Load factor
Bajany	2017	Yes	No	MILP model	• Model is used heterogenous fleet of shovels and dump trucks. • Demonstrates a MILP model designed to select the most fuel-efficient fleet.

					• Considers power and load factor

Despite the advancements in maximising net present value and measuring fuel consumption shown in Table 1, Recent studies indicate a clear shift toward integrated approaches that consider production, waste disposal, and hauling simultaneously, yet many still handle these elements in isolation, Sustainability is increasingly emphasized, especially regarding carbon emissions, although carbon costs themselves remain underexplored in most existing frameworks. As these models grow to accommodate more components ranging from ore scheduling to multi-lift dump sequencing. The complex of balancing safety, operational flexibility, and computational feasibility rises considerably. Haulage cost minimisation continues to be chief driver, tightly linked to fuel consumption and, by extension, environmental impacts. While multi-lift and lift-by-lift strategies can provide stable structural foundations, they may limit adaptability if unanticipated changes occur. A notable gap persists in connecting these models to long-term production or waste dump scheduling strategies using actual haulage distances to minimize haulage and carbon cost. Moreover, fewer models incorporate carbon costs, which are increasingly relevant under global sustainability mandates.

While the current literature review successfully pinpoints the necessity to include carbon expenses in MIP models for mining, factoring environmental considerations into mine planning not only supports global environmental goals like the United Nations Sustainable Development Goals but also propels the industry towards adopting practices that offer significant economic, social, and environmental advantages (Aghdamigargari et al., 2024). These improvements facilitate major operational shifts, such as implementation of energy-saving technologies and the optimisation of mine designs, leading to notable gains in both economy and ecology. By comparing these practices with those successful in other industries, this research emphasizes the potential for major improvements in mining efficiency and environmental care, stressing the critical role of mining operations in achieving contemporary sustainability objectives. For instance, in the manufacturing sector, companies are adopting green manufacturing practices that significantly reduce carbon footprints and embrace sustainable project designs, which are proving beneficial across various manufacturing sectors(Gholami et al., 2021; Rosen & Kishawy, 2012). This thesis aims to bridge these gaps by creating an integrated optimisation model that includes both economic and environmental costs in mining operations.

3.0 RESEARCH METHODOLOGY

3.1 Mathematical Modelling

Current assessments of software used in mining operations indicate that these tools lack the capability to autonomously generate an optimal schedule for rock dumping. Currently, the process depends heavily on manual input from operators to determine the sequence of dumping, which compromises the optimality of the results. Unlike software that requires human intervention, mathematical modelling offers a systematic approach to derive truly optimal solutions across varying operational scenarios. These methods allow for precise adjustments in response to dynamic conditions.

Mathematical modelling, particularly Mixed Integer Programming (MIP), is widely recognized in various industries for its effectiveness in analysing and enhancing system efficiencies. As detailed by (Taha, 2013), MIP excels in accurately simulating complex processes, making it a valuable tool for mining operations and waste rock management. Given its capabilities, this study leverages the merits of MILP to improve the accuracy and efficiency of optimisation processes involved in mining production schedules and material haulage.

Linear Programming (LP) is a widely utilised method for tackling optimisation problems. The typical structure of an LP model includes a linear objective function, numerous linear mixed integer programming (MIP) constraints, and a set of non-negative conditions:

$$\text{Maximise/Minimise}(Z) = \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + .. \omega_n x_n \text{ with } (\omega_i x_i = 1,2,3..n) \quad (3.1.1)$$

Where Z is the objective function, subject to the following constraints:

$$P_{11}x_1 + P_{12}x_2 + P_{13}x_3 + \ldots\ldots\ldots\ldots + P_{1n}x_n \leq C_1 \quad (3.1.2)$$

$$P_{21}x_1 + P_{22}x_2 + P_{23}x_3 + \ldots\ldots\ldots\ldots + P_{2n}x_n \leq C_2 \quad (3.1.3)$$

. . . .

$P_{m1}x_1 + P_{m2}x_2 + P_{m3}x_3 + \ldots\ldots\ldots\ldots + P_{mn}x_n \leq C_m$ (Topal, 2003).

The objective Z is to maximize or minimize the target variable based on the decision variables x_i with coefficients ω_j. The value Z might represent cost or NPV, serving as a quantitative benchmark for evaluating solutions. The constraints set the limits within which the solutions must fall, defined by the constants ρ_{mn} and c_m.

While multiple solutions may satisfy these constraints, only one set maximizes or minimizes Z, termed the optimal solution, which is established through mathematical proof.

Mixed Integer Programming (MIP) extends LP by requiring some variables to be integers, enhancing the model's realism and precision for representing real-world scenarios. Particularly, binary variables in MIP, which take values of zero or one, enable the model to account for yes-or-no decisions, making MIP an ideal choice for this research."

3.2 Resolving the optimisation issue.

Upon successfully constructing a mathematical model, solving it to find the optimal solution is the next step. Historically, simple LP problems have been solved using a graphical method, as illustrated in the Figure 1. This method defines a feasible region through linear constraints, with any point (X_1, X_2) within this area meeting the specified conditions. The objective function, Z, is graphically represented, and its numerical value is calculated. The optimal solution, characterized by the coordinates (X_1, X_2), is identified at the point where Z is either maximised or minimised, according to specific aim of the problem.

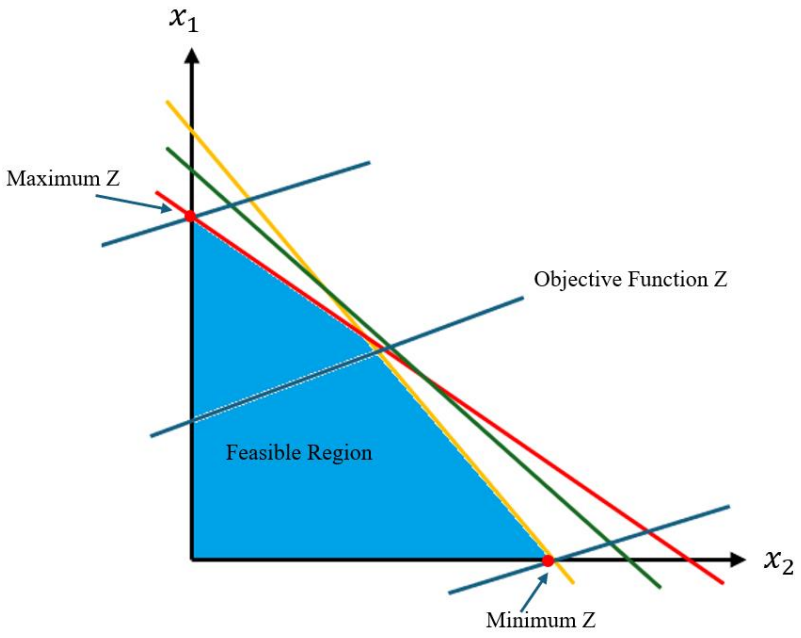

Figure 1: Graphical method concept

However, the graphical method is impractical for addressing problems that involves numerous variables and constraints. The simplex algorithm emerged as the predominant method for tackling complex linear programming problems around the late 1940s (Fourer et al., 2003). This method converts a problem into standard form and then into a tableau form. Optimal solutions are derived through a sequence of tableau row operations, which are thoroughly explained by (Taha, 2013)

With the evolution of computing technologies, these row operations can now be executed more quickly and accurately using computers. Yet, the simplex method cannot resolve problems with binary

variables, which require a branch and bound approach. This technique divides the problem into sub-problems using LP relaxation. If the relaxed problems provide an integer solution for the integer variables, it is considered optimal. Otherwise, the solution undergoes branching, where integer values are assigned close to the non-integer solution to generate two new sub-problems. These sub-problems are subsequently solved using conventional linear programming method. The branching continues until all options at a given stage have been examined and no better objective value can be found. Figure *2* outlines the basics of the branch and bound method.

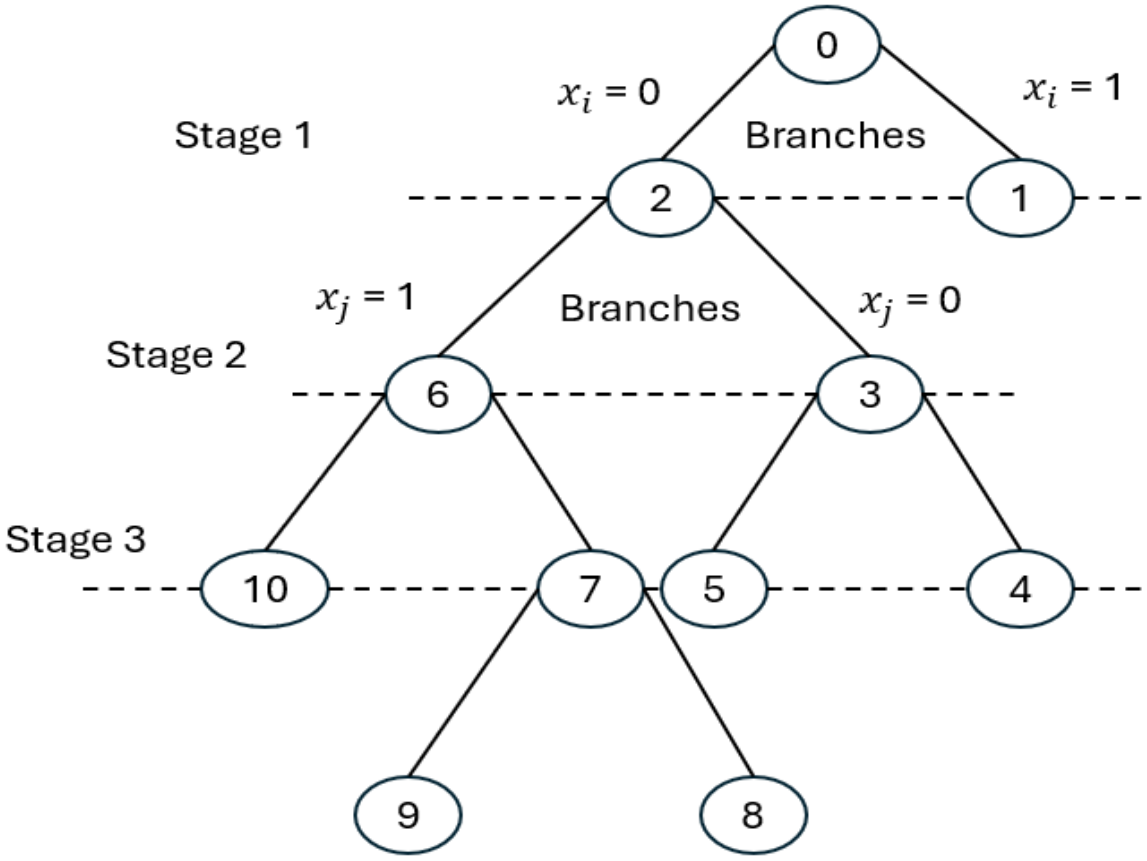

Figure 2: Branch and Bound Tree

GUROBI 10.5 And IBM ILOG CPLEX 22.1.1 which are among the foremost tools in operations research, have integrated these algorithms to tackle large-scale optimisation challenges.

3.3 Framework for coordinating mining operations and waste rock dumping system

Mining waste rock operations, including hauling and dumping, are closely linked, necessitating that a comprehensive dump should schedule be thoroughly consolidated with an optimised mining schedule. This integration should include both numerical output, such as mining block extraction, and visual elements, like the layouts of staged pits and rock dumps.

As an open pit mine develops, multiple stages of pits will form, leading to both temporary and permanent pit exit points. These changes are factored into planning potential haulage routes from the mining block, through various pit entries and exits, and to the dump location. Thus, the rock movement needs to precisely show the journey of the material volume, detailing:

- The timing of material movement

  Every movement of material (ore or waste) should be scheduled with specific times, ensuring that the operation runs smoothly, and that trucks and other equipment are used efficiently.

- The ID of the mining block that was moved

  Each block of mining block moved should be traceable back to its specific location. This helps in managing extraction and transportation more systematic

- The chosen pit exit point

  As the mine develops, it will have various exit points from which the material (ore or waste) can be hauled. These points may change as new areas are mined. The plan should specify which exit points each truck should be used during different phases of the mining operation.

- The exact location of the dump location.

  The destination for each load of waste rock should be clearly defined. This includes not just the general area but exact dump block centroid where the waste rock should be dumped.

3.4 Extraction of Block

The primary objective of a mine production schedule is to optimize the sequence of mining block extraction to maximize Net present value(NPV). Figure 3 depicts a two-dimensional mineral deposit on the left, with the right side displaying the blocks that are mined to achieve the highest NPV during specific time periods.

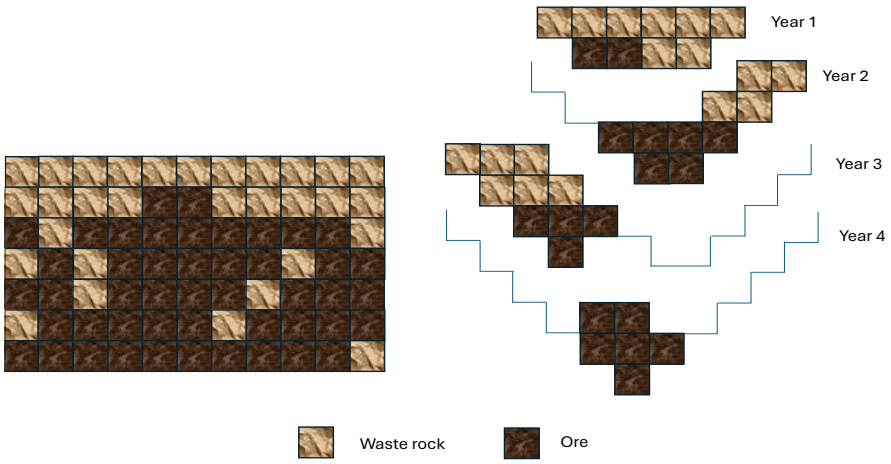

Figure 3: Mineral deposit and mining scheduling

Following extraction, ore and waste rock are usually directed towards different locations such as processing plant, stockpile or waste dump.

3.5 Modelling of dump locations

Presently, many mine software tools represent a waste dump as a single entity or multiple entities when various lifts are considered. This simplification speeds up the solution process but typically results in schedules that omit detailed information on waste rock placement, like exact spatial dumping locations. The lack of detailed data can create mismatches between the planned design and the actual execution at the site, potentially causing failures to achieve design objectives such as specific waste dump heights, capacities, footprints, slope angles and wrong cost estimates.

It is suggested that rock dumps be subdivided into smaller, more practical and precise dump blocks. Each block's centroid would represent a distinct dumping location, each with a defined nominal capacity, enhancing the precision and effectiveness of waste rock management strategies. See Figure 4.

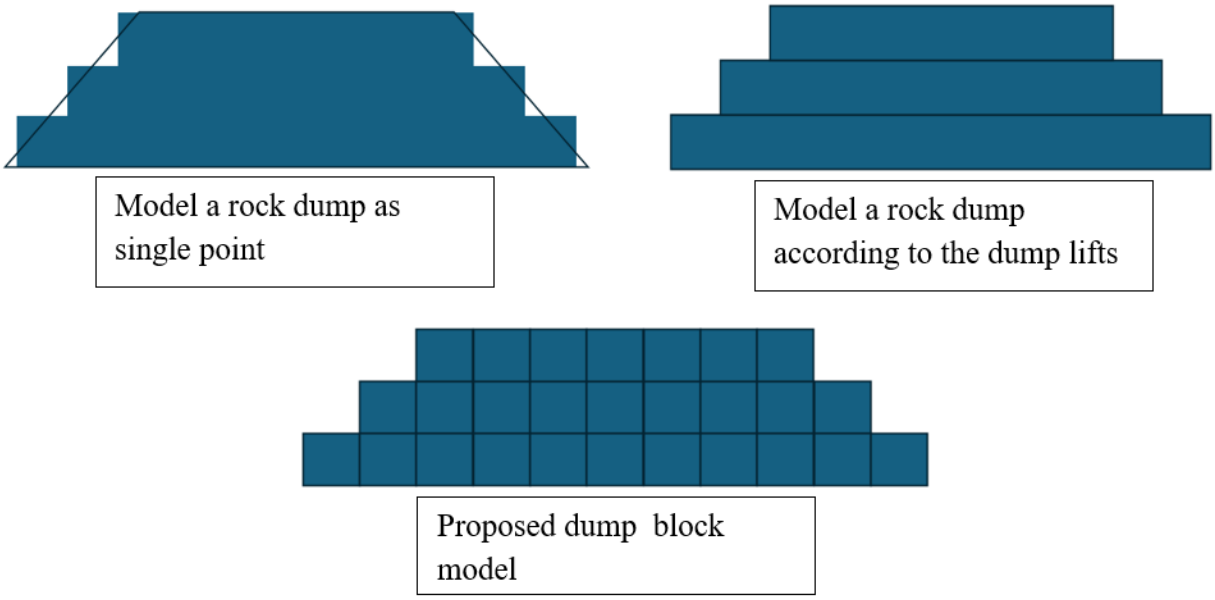

Figure 4:Dump Location Modelling

3.6 Sequence of constructing of waste dumps

To develop a waste dump with a multiple-lift configuration, certain rules need to be adhered to, ensuring that the resulting waste placement schedule is viable in real scenarios. Two sequences are typically considered: lift-by-lift and multi-lift. Each approach has distinct operational implications and must be chosen based on practical considerations for effective implementation.

3.6.1 Lift-by-Lift waste dump construction

The lift-by-lift method of dump construction operates under a dependency condition between lifts, requiring that waste rock be deposited into the lowest available unfilled lift first. Consequently, no dumping is allowed into a higher lift until the one directly beneath it has been filled. Figure 5 illustrates this, showing that only one dump block in the upper level is available, while blocks in higher lifts remain off-limits. This approach restricts the selection of dump blocks, which could hinder operational flexibility and efficiency in several ways:

- Limited routing Option:
  When only one block is available for dumping, trucks lose the freedom to choose among multiple dump locations. This can lead to sub optimal hauling routes if the designated dumping block is farther away or requires more complex manoeuvres.
- Underutilized Equipment and Resources:

By forcing a rigid order of fill, certain fleet of trucks or loading equipment may experience idle times if the only active dumping area is being serviced by another fleet, or if the haulage distance becomes disproportionately long.

- Potential for increased costs and emissions:
  Longer or more convoluted hauling routes can translate directly into increased fuel consumption, which raises both operational costs and carbon emissions.

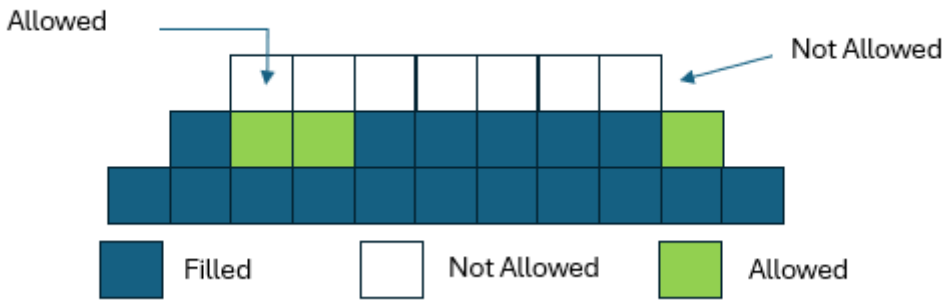

Figure 5: Sequential lift construction for dumps.

3.6.2 Multi-Lift waste dump construction sequence

Multi-lift waste dump construction sequence is a method of building a waste dump in successive horizontal layers, each replying on the layer below for stability. This construction sequence follows a strict inter-block dependency condition, requiring stable slopes maintained by placing one dump block on upon nine underlying blocks. In the Mixed- Integer Programming (MIP) model, a block on an upper lift only becomes eligible for dumping after the nine blocks directly beneath it are filled. The eligibility of the upper block for waste disposal is determined by the collective filled status of these nine lower blocks as shown in Figure 6.

This sequence in multi-lift construction allows for a flexible approach to rock dump scheduling, effectively handling various conditions. Nonetheless, it has a drawback: each dump block, beyond the initial lift, needs the support of nine blocks directly below it. The dependency condition often presumes a relatively uniform, grid-like layout of dump blocks. Some mining sites feature irregular topographies or spatial constraints that may not align neatly with a 3x3 block design. When the geometry or sheer volume of waste does not conform to this structure approach, implementing or scaling the model may require smoothing.

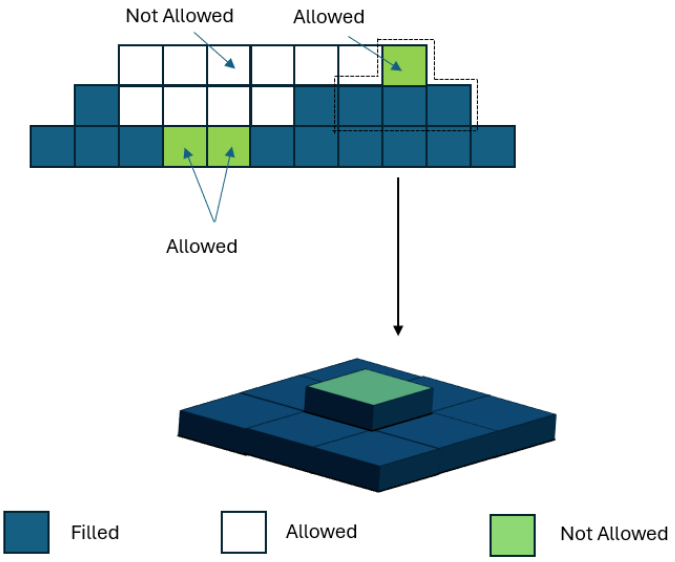

Figure 6: Condition dependent on multi-lift dump construction.

3.7 Haulage distance calculation

The transportation of waste from a mining block to any available dump block locations does not typically involve constraints on material segregation, necessitating an evaluation of all potential haulage paths and distances. These calculations encompass various segments of the haul route:

- From the centroid of the block to the nearest point on the haulage route.
- From the nearest point to the point at the pit exit.
- From the pit exit point to the processing plant.
- From the pit exit point to the dump entry point.
- From the dump entry point to the centroid of a dump block location.

These distances are calculated using the Euclidean distance calculation method to calculate the distance between each point in a haulage route. For example, if the distance from the centroid of the block to the nearest point on a haulage route 1 consists of multiple points, the distance between each point from one point to the other is calculated by using the Euclidean distance calculation method to give the total distance.

The Euclidean distance represents the “straight line” distance between two points in Euclidean space, which is essentially the most direct path connecting them. This concept is a fundamental way to

measure distances where the dimensions are Cartesian, meaning they are defined in terms of a coordinate system based on perpendicular axes.

Euclidean distance, D, between two points in three-dimensional space is determined by:

Distance (D) = $\sqrt{(A_2 - A_1)^2 + (B_2 - B_1)^2 + (C_2 - C_1)^2}$ (3.7.1)

- $A_1,\ B_1,\ C_1$ and $A_2,\ B_2,\ C_2$ are the coordinates of the two points in space. For example, $A_1,\ B_1,\ C_1$ could be the coordinate of the first point, and $A_2,\ B_2,\ C_2$ the coordinate of the second point.
- $(A_2 - A_1), (B_2 - B_1), (C_2 - C_1)$ calculates the difference in each corresponding coordinate between eh two points. This finds the length of the side of the 'box' formed between these two points in each dimension (x, y, and z)
- Squaring each difference eliminates any negative values and emphasizes larger differences
- Adding this squared difference together combines the distances in all three dimensions into a single value
- Taking the square root of the sum brings this combined distance back to the original scale, giving a straight line between the two points.

Figure 7 shows the selection path based on the position of the block in the pit. The block chooses the haulage path based on the distance between the centroid of the block to the closest point on the haulage path. As you can see in the figure below, the block will choose the shortest distance from the block to the closest point on haulage paths 1 or 2 shown by the tick mark. The distance between each point on the haulage path is calculated to determine the total distance of the haulage path.

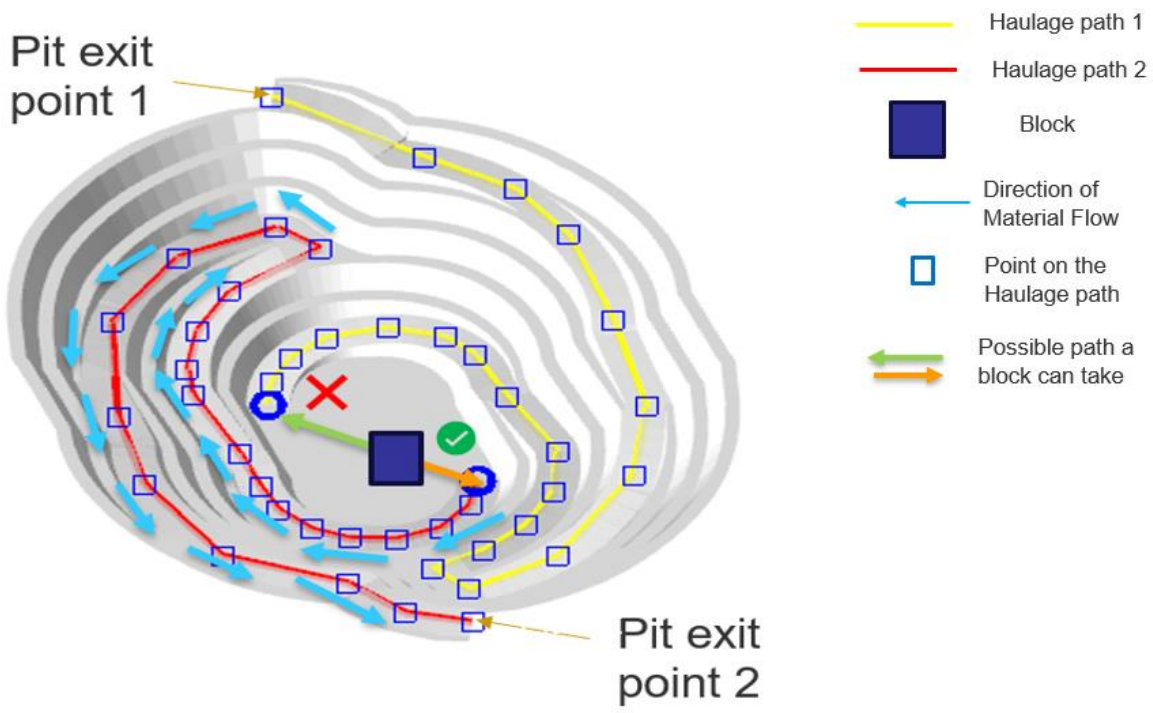

Figure 7: Distance calculation of centroid of block to pit exit

Similarly, the distance from either of the pit exit to the either the processing plant or to the waste dump entry is determined by calculating the distance between each point in a haualge path. From the dump entry to the each centroid of the dump location, the model will select the path based on the total distance but also considering the constraints of the model. This can be seen in Figure 8:

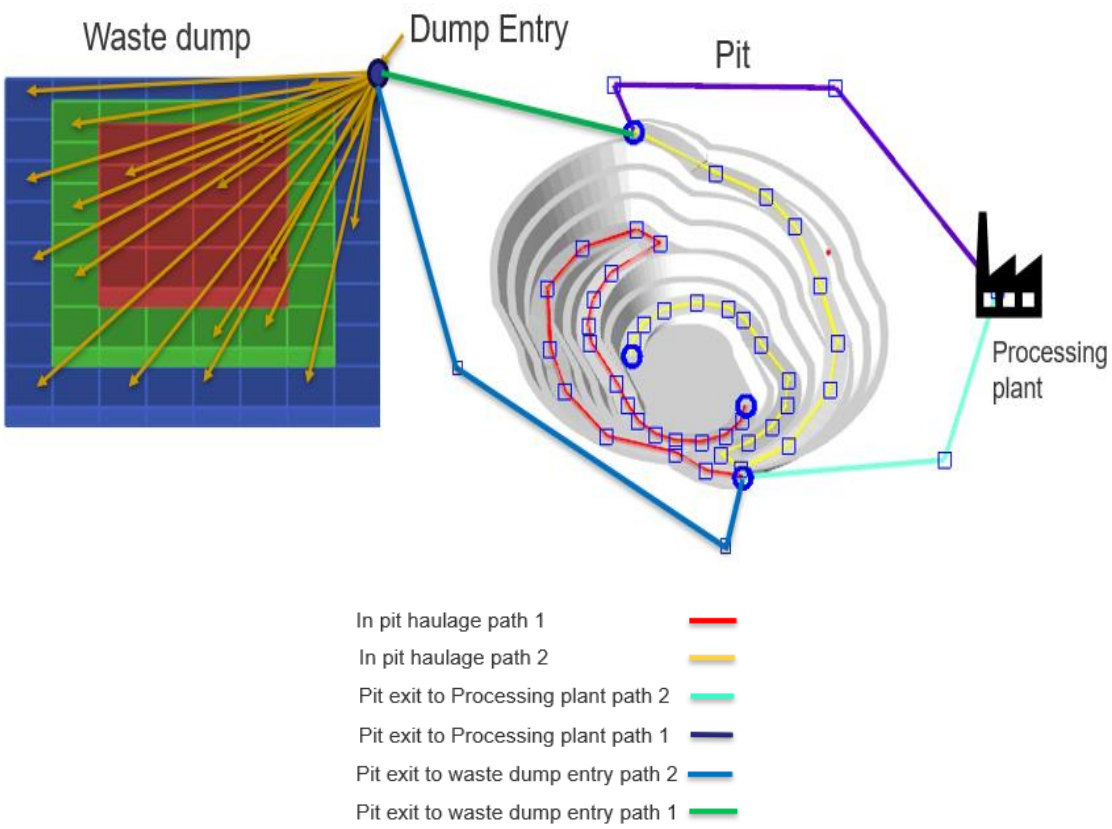

Figure 8: Schematic diagram of haulage routes

The total distance is considered from centroid of the block to the nearest point on the haulate route to the processing plant or from the centroid of the block to the centroid of the dump location. Haualge route are determined and selected based on the total distance from block to either processing plant or waste dump location. The Minimum or the least distance from the block to either processing plant or waste dump is considered as haualge route for that particular block but can also be altered due to differenct constraints such as availability of dump location, mining capacity, processing capacity etc.

3.8 Calculation of haulage cost and carbon cost

3.8.1 Haulage cost calculation

The most precise way to measure fuel consumption is by using data from real mining operations. (Runge, 1998) suggest that fuel consumption per hour can be calculated by:

$$FC = P \times 0.3 \times LF \text{ (Runge, 1998)} \qquad (3.8.1)$$

Where:

P = Engine Power (kW)

LF = Engine Load Factor

and 0.3 is a unit conversion factor

The haulage cost used in the model is calculated as below:

a) For flat distance
   - Fuel consumption (FC) = 455 L/hr at 50% load factor based on the Liebherr T 282 B series truck
   - Where the engine power is 1864 kW(2,500 hp)
   - The fuel consumption per metre is calculated by = $\frac{Fuel\ consumption(FC)\ per\ Hour}{Speed\ of\ truck\ per\ Hour}$
   - Loaded speed of the truck per hour = 20km/hr
   - The fuel consumption per metre (FC/m) = 0.011375 L/metre
   - Haulage cost per metre travelled = FC x Cost of diesel per meter ($1.5)
   - Haulage cost per metre for a single truck type = $0.017

a) Gradient adjustment and recalculation

With a gradient increase factor to fuel : 10% increase in fuel consumption per 1% gradient increase

Increase of 15% gradient :

- Adjusted fuel consumption Factor = 1 + (0.10 x 15) = 2.5
- Fuel consumption rate = 0.011375 x 2.5 = 0.02843 L/metre
- Haulage cost per metre = 0.02843 L/metre x $1.5/L = $ 0.04265 / metre

Similar calculations have been considered for an increase of 30% and 45% gradient where the haulage costs are $0.06825/metre and $0.09384375/ metre respectively.

3.8.2 Carbon cost calculation

CO_2 emissions can be determined by using the equations:

$$C0_2 = FC \times CF \text{ (Runge, 1998)} \qquad (3.8.2)$$

Where:

FC = Fuel Consumption (L/hr)

CF = Conversion Factor

Conversion factor(CF) of CO_2 can be determined as:

$$CF = CC \times 10^{-6} \times 0.99 \times (44/12) \text{ (Runge, 1998)} \qquad (3.8.3)$$

Where CC represents the carbon content for the diesel fuel (g/L), and 0.99 is the oxidation factor.

The conversion factor (CF) of CO_2 emission for diesel fuel is 0.00268. This factor is calculated based on the carbon residue in one litre of diesel.

The carbon content for the diesel is CC = 733 g/L

The carbon cost used in the model is calculated as below :

a) For flat distance
   - Fuel consumption per metre = 0.011375 L/metre
   - CO_2 Emission factor per metre = $FC \times CF$
     Where:

     FC = Fuel Consumption (L/hr)

     CF = Conversion Factor
   - CO_2 Emission factor per metre = 0.011375 x 0.00268 = 0.000030485 tonnes/ metre
   - Carbon cost per metre = $FC \times CF \times Cp$
     Where:
     $Cp = Carbon\ price$ ($50/tonne$)
   - Carbon Cost per metre = 0.000030485 tonnes/ metre x $50/ tonne = $0.00152425/metre

b) Gradient adjustment and recalculation

   As the fuel consumption per meter has already been calculated, the fuel consumption from the haulage cost is used to calculate the carbon cost for different gradient.

   For an increase of 15% gradient:

   - Fuel consumption per metre = 0.02843 L/metre
   - CO_2 emission factor per metre = $FC \times CF$
   - CO_2 emission factor per metre = 0.02843 x 0.00268 = 0.000076233 tonnes/ metre
   - Carbon cost per metre = $FC \times CF \times Cp$
   - Carbon cost per metre = 0.000076233 tonnes/ metre x $50/ tonne = $0.00381/metre

Similar calculations have been considered for an increase of 30% and 45% gradient where the carbon costs are $0.006/metre and $0.0083/metre respectively.

3.9 Mathematical model formulations

Three MIP based mathematical models are proposed for this study:

1. Haulage cost model:
    - Purpose: This model is primarily designed to maximize the Net Present Value(NPV) of the production schedule.
    - Features: it operates without considering the carbon costs. The main objective is to achieve the highest possible economic return by optimizing the haulage routes and schedules to reduce haulage costs.
2. Carbon cost model:
    - Purpose: This model serves to minimise both the haulage and carbon costs of the mine production schedule.
    - Features: it utilizes the results from the Haulage Cost Models as inputs. Blocks that are sent to the processing plant in Haulage Cost model are the blocks used to be sent to the processing plant in this model. The blocks that are sent to the waste dump locations in the Haulage Cost Model is considered as waste rock material for transport. The total volume of material transported, both ore and waste, is identical to that in the Haulage Cost Model. This approach allows for evaluating how the waste dumps progression is affected when carbon costs are considered, providing insights into potential efficiencies and savings in carbon emissions.
3. Combined model:
    - Purpose: This model maximises the NPV of the mining schedule while simultaneously considering both haulage cost and carbon costs.
    - Features: By integrating the cost considerations from both haulage cost and Carbon cost models, this model provides a balanced approach to achieving economic efficiency while also promoting environmental sustainability. The combined model represents a comprehensive strategy that addresses the dual objectives of cost minimisation and reduced carbon emissions.

The structure of material flow for optimal ore production and waste dump scheduling incorporates specific indices, sets, parameters, and variables as presented below that are crucial in the MIP formulations.

Indices

- t ∈ T Time period
- b ∈ B Mining blocks
- b′ ∈ B Precedence blocks over mining block b
- n ∈ N Pit exits
- q ∈ Q Processing plant
- o ∈ O Waste dump entries
- w ∈ W Waste Dump locations
- w′ ∈ W Precedence dump locations to dump location w

Sets

- B = {1,…..|*B*|} Set of mining blocks
- N = {1,…..|*N*|} Set of pit exits
- Q = {1,…..|*Q*|} Set of processing plant
- W ={1,…...|*W*|} Set of waste dump locations
- O = {1,…...|*O*|} Set of dump entries
- T = {1,……|*T*|} Set of all scheduling periods
- $P_b(J)$ Set of predecessor blocks b′ for block b. This set includes all blocks that must be extracted before block b can be extracted, where J is the total number of blocks in the set P_b
- $P_w(U)$ Set of predecessor waste dump locations w′ for waste dump location w where each predecessor waste dump location w′ must be filled before waste dump location w can be filled, where U is the total number of dump locations in the set P_w.

Parameters

- p_t = Selling price of metal in grams in time t
- r_t = Unit refining cost of metal in grams in time t
- M_t= Mining capacity in time t
- P_t= Processing capacity in time t
- q_b = Quantity of material in tonne in block b

- m_t = Mining cost per tonne in time t
- c_t = Processing cost per tonne in time t
- H_S = Haulage cost per tonne per meter per section of haulage path
- CC_S = Carbon cost per meter per section of the haulage path
- r = Recovery rate
- d = Discount rate
- C_w= Waste dump location capacity at each w location
- g_b= Grade of mining block
- ρ = Swell factor
- d_b = Density of mining block b
- y_{p2}^{p1}= the total distance calculated between

  all the points along the haulage route within the system (all the points from block to pit exits, pit exits to processing plant, pit exits to waste dump entry, and waste dump entry to dump location)

Discounted profit calculation for haulage cost model

- K_{bnqt} = Discounted profit per tonne for extracting material from block b through pit exit n, processing plant q in time period t, where $K_{bnqt} = (p_t - r_t) \times g_b \times r_c - m_t - c_t - H_S \times (y_b^n + y_n^q)$,∀ b ∈ B, n ∈ N, q ∈ Q,t ∈ T
- K_{bnowt} = Discounted profit per tonne from extracting material from block b through pit exit n, waste dump entry o, dump location w in time period t, where $K_{bnowt} = -m_t - H_S \times (y_b^n + y_n^o + y_o^w)$,∀ b ∈ B, n ∈ N, o ∈ O,w ∈ W, t ∈ T

Cost calculation for carbon cost model

- L_{bnowt} = Cost per tonne from extracting material from block b through pit exit n, waste dump entry o, dump location w in time period t, where $L_{bnowt} = -m_t - c_t - (H_s + CC_s) \times (y_b^n + y_n^o + y_o^w)$,∀ b ∈ B, n ∈ N, o ∈ O,w ∈ W, t ∈

Discounted profit calculation for combined model

The difference between the haulage cost model and the combined model is the inclusion of the carbon cost (CC) as shown Below:

- C_{bnqt} = Discounted profit per tonne from extracting material from block b through pit exit n, processing plant q in time period t, where $C_{bnqt} = (p_t - r_t) \times g_b \times r_c - m_t - c_t - (H_S + CC_S) \times (y_b^n + y_n^q)$,∀ b ∈ B, n ∈ N, q ∈ Q, t ∈ T
- C_{bnowt} = Discounted profit per tonne from extracting material from block b through pit exit n, waste dump entry o, dump location w in time period t, where $C_{bnodt} = -m_t - c_t - (H_S + CC_S) \times (y_b^n + y_n^o + y_o^w)$,∀ b ∈ B, n ∈ N, o ∈ O,w ∈ W, t ∈ T

Variables

V_{bnqt} Continuous variable, representing the quantity (tonne) of materials extracted from mining block b, through pit exit n, processing plant q at period t.

V_{bnowt} Continuous variable, representing the quantity (tonne) of materials in tonnes extracted from mining block b through pit exit n, waste dump entry o, dump location w at period t.

Y_{bt} Binary variable, if all precedent blocks of block b are mined out to period t

Y_{wt} Binary variable, if all precedent dump locations of dump location w are filled out in period t

Objective function

Objective function for haulage cost model

Maximize

$$\left(\sum_{b=1}^{|B|} \sum_{n=1}^{|N|} \sum_{q=1}^{|Q|} \sum_{t=1}^{|T|} K_{bnqt} V_{bnqt} + \sum_{b=1}^{|B|} \sum_{n=1}^{|N|} \sum_{o=1}^{|O|} \sum_{w=1}^{|W|} \sum_{t=1}^{|T|} K_{bnowt} V_{bnowt} \right) \quad (3.9.1)$$

Objective function of this model maximises the profits from transporting materials from mining blocks to processing plant and waste dump locations. The discounted profit per tonne K_{bnqt} and K_{bnowt} includes selling price, refining costs, mining and processing plans, and specific haulage costs calculated over real distance. It does not include carbon costs. The decision variables V_{bnqt} and V_{bnowt} represent the quantity of materials moved to processing plants and dump locations, respectively.

Objective function for carbon cost model

Minimize

$$\left(\sum_{b=1}^{|B|} \sum_{n=1}^{|N|} \sum_{o=1}^{|O|} \sum_{w=1}^{|W|} \sum_{t=1}^{|T|} L_{bnowt} V_{bnowt} \right) \tag{3.9.2}$$

Objective function of this model minimises the cost associated with transporting materials to dump locations, including both haulage and carbon costs. The focus here shifts from maximizing NPV to minimizing carbon costs using the results of Haulage Cost model as inputs (blocks that are sent to processing plant, waste dump and its respective data such as volume, mining cost, processing cost, recovery cost and grade of mining block) particularly emphasizing the reduction of environmental impacts through lower carbon emissions. This approach allows for evaluating how the waste dumps progression is affected when carbon costs are considered, providing insights into potential efficiencies and savings in carbon emissions.

Objective function for combined model

Maximize

$$\left(\sum_{b=1}^{|B|} \sum_{n=1}^{|N|} \sum_{q=1}^{|Q|} \sum_{t=1}^{|T|} C_{bnqt} V_{bnqt} + \sum_{b=1}^{|B|} \sum_{n=1}^{|N|} \sum_{o=1}^{|O|} \sum_{w=1}^{|W|} \sum_{t=1}^{|T|} C_{bnowt} V_{bnowt} \right) \tag{3.9.3}$$

Objective function of this model aims to maximise NPV and incorporates both the haulage and carbon costs into the objective function, addressing environmental sustainability. The discounted profit per tonne C_{bnmt} and C_{bnowt} are adjusted to account for carbon emission costs associated with haulage distances.

Constraints

$$\sum_{n=1}^{|N|}\sum_{q=1}^{|Q|}\sum_{t=1}^{|T|} \mathrm{V}_{bnqt} + \sum_{n=1}^{|N|}\sum_{o=1}^{|O|}\sum_{w=1}^{|W|}\sum_{t=1}^{|T|} V_{bnowt} \leq q_{\mathrm{b}}, \forall\, \mathrm{b} \in \mathrm{B} \quad (3.9.4)$$

Constraint (3.9.4) ensures that the total quantity (tonne) of material extracted from block b does not exceed its available volume of q_{b}. The equation accounts for material sent to both the processing plants and waste dump over all periods.

$$\sum_{b=1}^{|B|}\sum_{n=1}^{|N|}\sum_{q=1}^{|Q|} V_{bnqt} + \sum_{b=1}^{|B|}\sum_{n=1}^{|N|}\sum_{o=1}^{|O|}\sum_{w=1}^{|W|} V_{bnowt} \leq M_t, \ \forall t \in T \quad (3.9.5)$$

Constraint (3.9.5) limits the total quantity (tonne) of material does not exceed the mining capacity M_t for each time period t and ensures that the operational capacity constraints are not exceeded.

$$\sum_{b=1}^{|B|}\sum_{n=1}^{|N|} V_{bnqt} \leq P_t, \mathrm{q} \in \mathrm{Q}, \mathrm{t} \in T \quad (3.9.6)$$

Constraint (3.9.6) ensures that the total quantity (tonne) of material sent to processing plants q does not exceed the processing capacity of P_t for each time period t.

$$\sum_{b'=1}^{J}\sum_{n=1}^{|N|}\sum_{q=1}^{|Q|}\sum_{t'=1}^{t} \mathrm{V}_{b'nqt'} + \sum_{b'\epsilon P_b}\sum_{n=1}^{|N|}\sum_{o=1}^{|O|}\sum_{w=1}^{|W|}\sum_{t'=1}^{t} V_{b'nowt'} - Y_{\mathrm{bt}} \sum_{b'\epsilon P_b} q_{b'} \geq 0, \forall\, \mathrm{b} \in \mathrm{B}, b' \in P_b(J), \mathrm{t} \in T \quad (3.9.7)$$

Constraint (3.9.7) enforces that block b can only be extracted at time t, if all the predecessor block b' has been extracted in the current or previous time periods up to time t.

$$\sum_{i=n}^{|N|}\sum_{q=1}^{|Q|}\sum_{t=1}^{|T|} \mathrm{V}_{bnqt} + \sum_{i=1}^{|N|}\sum_{o=1}^{|O|}\sum_{w=1}^{|W|}\sum_{t=1}^{|T|} V_{bnowt} \leq \mathrm{Y}_{\mathrm{bt}} \cdot q_{\mathrm{b}} \quad ,\forall\, \mathrm{b} \in \mathrm{B}, \mathrm{t} \in T \tag{3.9.8}$$

Constraint (3.9.8) restricts that material can only be extracted or transported from a given block, if the predecessor block b' is fully mined in previous period.

$$\sum_{b=1}^{|B|}\sum_{n=1}^{|N|}\sum_{o=1}^{|O|}\sum_{t=1}^{|T|}\left(\frac{\rho}{d_b}\right) V_{bnowt} \leq C_w \quad ,w \in W \tag{3.9.9}$$

Constraint (3.9.9) ensures that the total swelled volume of waste material sent to dump location w does not exceed the dump location capacity

$$\sum_{b=1}^{|B|}\sum_{n=1}^{|N|}\sum_{o=1}^{|O|}\sum_{w' \epsilon P_w}\sum_{t'=1}^{t}\left(\frac{\rho}{d_b}\right) V_{bnow't'} - Y_{wt} \sum_{w' \epsilon P_w} C_{w'} \geq 0, \forall\, w \in W, \mathrm{t} \in T \tag{3.9.10}$$

$$\sum_{b=1}^{|B|}\sum_{n=1}^{|N|}\sum_{o=1}^{|O|}\sum_{w \epsilon W}\sum_{t'=1}^{t}\left(\frac{\rho}{d_b}\right) V_{bnow't'} \leq Y_{wt} \cdot \mathrm{C}_w, \quad \forall\, w \in W, \mathrm{t} \in T \tag{3.9.11}$$

Constraint (3.9.10-3.9.11) ensures that dump location w can only be filled at time t if all its predecessors dump locations w' have been filled in previous time periods.

$$Y_{bt} = 0 \text{ or } 1, \quad \forall\, \eta, \mathrm{t} \tag{3.9.12}$$

$$Y_{wt} = 0 \text{ or } 1, \quad \forall\, \mathrm{p}, \mathrm{t} \tag{3.9.13}$$

$$V_{bnqt}, V_{bnowt} \geq 0 \quad \forall \mathrm{b}, \mathrm{n}, \mathrm{q}, \mathrm{o}, \mathrm{w}, \mathrm{t} \tag{3.9.14}$$

Constraint (3.9.12 -3.9.14) enforces the integrity and non- negativity of the variables.

Having detailed the theoretical framework of the models, the next chapter applies these concepts to practical scenarios, highlighting their implementation and real-world impact.

4.0 MIP MODEL IMPLEMENTATION

The models were applied to an open pit gold mine in Western Australia. A total of 1219 blocks of size of 20metres x 20metres x 10 metres was chosen for the study. The area spans operations across a 5-year mine life operation. The ore is moved to the processing plant, while the waste is sent to the waste dump, which consists of 116 dump location blocks distributed over three lifts. The open pit mine has 2 in-pit haulage routes with two haulage paths connected to the processing plant and two haulage paths connected to the waste dump entry through the two pit exits as shown in Figure 9:

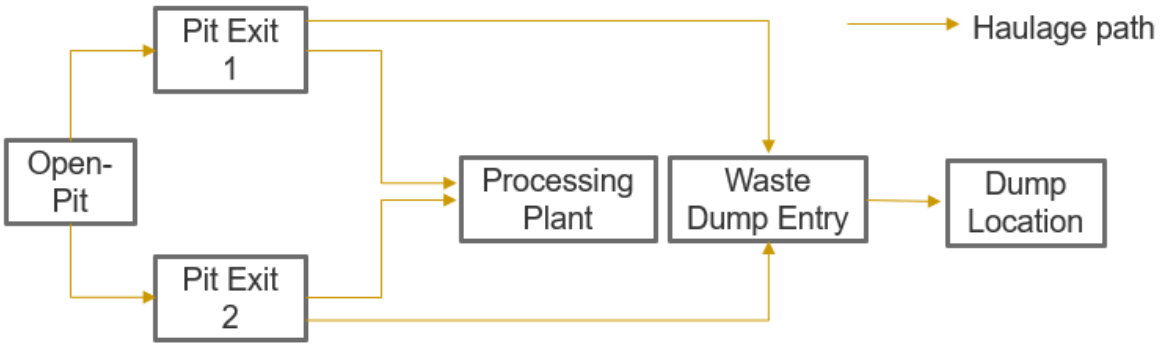

Figure 9: Schematic diagram of the implementation of the proposed methodology.

Additional information and assumptions for the model implementation are presented in Table 2:

Table 2: Input parameters for the optimisation model for gold deposit

Input Parameters	Variable name	Maximum	Unit
Mining Capacity	M_t	7000000	Tonne/ Year
Processing Capacity	P_t	4000000	Tonne/ Year
Ore sale Price	p_t	74	$ per gram
Mining Cost	m_t	4.045	$ per tonne
Refining Cost	r_t	0.05	$ per gram
Processing Cost	c_t	22.4	$ per tonne
Recovery Rate	r	93	%
Discount Rate	d	10	%
Swell Factor	ρ	1.25	Volume to tonne ratio

The haulage cost and the carbon costs for every haulage path is different due to the difference in gradient and fuel consumed as shown in Table 3.

Table 3:Input parameters for the optimisation model for Haulage cost as well as the Carbon costs used in the models.

Input Parameters	Haulage cost/metre	Carbon cost/metre
Block to pit exit	0.042	0.006
Pit exit to waste dump entry	0.017	0.0015
Pit exit to processing plant	0.017	0.0015
Waste dump entry to first lift	0.042	0.003
Waste dump entry to second lift	0.068	0.006
Waste dump entry to third lift	0.093	0.0083

The proposed model was developed using Python programming language (see Figure 10) and solved using the GUROBI solver on a computer with AMD Ryzen 7 5800H with 40.0 GB RAM. GUROBI is a commercial solver recognized for its speed and robustness in handling large-scale, complex optimization tasks and supports a range of problem types—from basic Linear Programming (LP) and Mixed Integer Linear Programming (MILP) to Quadratic Programming (QP), Quadratically Constrained Programming (QCP), and Mixed Integer Quadratic Programming (MIQP).

The pre-processing time for the haulage cost model and the Combined Model took around 15 hours with the carbon cost model taking approximately 5 hours. See appendix A for Haulage Cost Model; appendix B for Carbon Cost Model; appendix C for Combined Model.

```python
def calculate_discounted_profit_per_unit(block_ID, path_key, DumpLocationID, destination, year,
                                         block_model_df, waste_dump_df,
                                         haulage_paths, haulage_costs, carbon_costs):
    block_row = block_model_df.loc[block_ID]
    grade = block_row['grade']
    mining_cost_per_unit = block_row['mining_cost']  # Assuming this is per unit
    discount_rate = 0.1
    quantity = block_row['quantity']

    # Calculate the cost including both haulage and carbon cost based on the destination
    if destination == "plant":
        # Unpack both haulage cost and carbon cost
        haulage_cost_per_unit, carbon_cost_per_unit = haulage_cost_to_processing_plant(
            block_ID, block_model_df, processing_plant_coords,
            haulage_paths,  # Only pass haulage_paths here
            haulage_costs, carbon_costs, quantity
        )
        processing_cost_per_unit = block_row.get('processing_cost_p1', 0)
        recovery_rate = block_row.get('recovery_rate_p1', 1)  # Default recovery rate for processing
    else:
        # Unpack both haulage cost and carbon cost
        haulage_cost_per_unit, carbon_cost_per_unit = total_haulage_cost_per_unit(
            block_ID, block_model_df, DumpLocationID, waste_dump_df,
            haulage_paths,  # Only pass haulage_paths here
            haulage_costs, carbon_costs, quantity
        )
        processing_cost_per_unit = 0  # No processing cost for waste dump
        recovery_rate = 0  # No recovery as the material is treated as waste

    # Include both haulage and carbon costs in the total operational cost
    total_operational_cost_per_unit = (
        mining_cost_per_unit
        + processing_cost_per_unit
        + haulage_cost_per_unit
        + carbon_cost_per_unit  # Carbon cost is now part of the operational cost
    )

    # Calculate the revenue per unit based on grade and recovery rate
    revenue_per_unit = grade * recovery_rate * calc_gold_price(year)

    # Calculate the net value per unit
    net_value_per_unit = revenue_per_unit - total_operational_cost_per_unit

    # Apply the discount factor to get the discounted net value
    discounted_net_value_per_unit = net_value_per_unit / ((1 + discount_rate) ** (year - 1))
```

Figure 10: Example of python code

Table 4 summarizes the characteristics of the three different optimisation models used for evaluating haulage costs, carbon costs, and a combined approach for the case study mining operation. Each model is described by the number of lines in the python code, the solution time required to run the model , the number of linear and binary integer variables involved, and the total number of constraints. The haulage cost model and the Combined model are larger and more complex, both requiring more computational time and involving a higher number of variables compared to the carbon cost model.

Table 4:Optimisation model characteristics

Model Type	Code lines	Solution time(Min)	Linear variables	Binary integer variables	Constraints
Haulage Cost Model	528	2	1,425,060	7,458	1,368
Carbon Cost Model	340	1	534,760	3,673	1,368
Combined Model	560	2	1,425,060	7,458	1,368

4.1 Result Analysis

The material flow can be seen in Table 5 which compares the movement of material (in tonnes) to both the processing plant and the waste dumps. According to the table, the amount of material sent to the processing plant is higher than in the Haulage Cost Model and Carbon Cost Model, while more material is directed to the waste dumps in the Combined Model. This difference arises from the addition of carbon costs in the Combined Model. The material movement for the Carbon Cost Model is same as the Haulage Cost Model as it relies on the output data from the Haulage Cost Model.

Table 5: Results obtained from the models for the movement of ore and waste tonnes to the processing plant and waste dump on annual basis.

Period (Years)	Haulage Cost Model and Carbon Cost Model (in tonnes)		Combined Model (in tonnes)	
	Processing Plant	Waste Dump	Processing Plant	Waste Dump
1	1,688,000	4,312,,000	1,508,000	4,400,000
2	1,010,000	3,430,000	1,010,000	3,700,000
3	2,500,000	1,960,000	2,500,000	2,200,000
4	2,382,000	1,078,000	2,332,000	1,002,000
5	1,712,000	588,000	1,602,000	678,000
Total	**9,292,000**	**11,297,800**	**8,952,000**	**11,980,000**

The cut of grade of the block sent to the processing plant every year in both haulage cost model, Carbon Cost Model and the combined model can be seen in Table 6 below with higher cut of grade requirements in the combined model due to extra cost incurred which is the carbon cost.

Table 6: Cut of grade of blocks sent to processing plant yearly

Period	Haulage Cost Model and Carbon Cost Model	Combined Model
1	0.09	0.1
2	0.5	0.6
3	0.6	0.8
4	1.1	1.2
5	1.23	1.3

The total haulage cost and carbon cost for the Haulage Cost Model, Carbon Cost Model and the Combined Model can be seen in Table *7*. The haulage cost model consistently incurs higher haulage cost each year compared to the carbon cost model and the combined model. This is because the

haulage cost model prioritises reducing distance without necessarily considering carbon emissions whereas the Carbon Cost Model and combined model shows lower carbon costs annually, reflecting its focus on reducing emissions. The carbon emission for the Haulage Cost Model is 14,750.55 tonnes of CO_2 and 12,943.89 tonnes of CO_2 for the Carbon Cost Model with a reduction of 12.24 % reduction in carbon emission with Carbon Cost Model with respect to Haulage Cost Model and the carbon emissions is 12,334.44 tonnes of CO_2 for the Combined Model with a reduction of 16.37% reduction in carbon emissions with respect to Haulage Cost Model. This suggests that focusing on carbon emissions can lead to overall lower haulage cost due to factors such as avoiding carbon taxes and penalties.

Table 7: Annual haulage cost and carbon cost comparison for the Haulage Cost Model, Carbon Cost Model and Combined Model

Year	Haulage Cost Model		Carbon Cost Model		Combined Model	
	Haulage Cost	Carbon cost	Haulage cost	Carbon cost	Haulage Cost	Carbon Cost
1	$ 1,865,741.69	$ 166,773.20	$ 1,550,800.58	$ 140,950.88	$ 1,770,337.98	$ 140,950.88
2	$ 1,941,574.65	$ 173,548.38	$ 1,762,067.06	$ 147,861.80	$ 1,767,603.73	$ 147,861.80
3	$ 2,119,548.41	$ 189,396.42	$ 2,001,820.69	$ 180,605.88	$ 2,042,208.15	$ 180,605.88
4	$ 1,461,516.71	$ 130,562.14	$ 1,371,867.51	$ 125,933.97	$ 1,119,781.99	$ 125,933.97
5	$ 863,813.00	$ 77,247.59	$ 620,418.40	$ 51,841.92	$ 593,884.77	$ 51,841.92
Total	**$ 8,252,194.47**	**$ 737,527.73**	**$ 7,306,974.24**	**$ 647,194.45**	**$ 7,293,816.62**	**$ 616,722.48**

Table 8 evaluates the NPV under three distinct models: the Haulage Cost Model, the Carbon Cost Model, and the Combined Model. The Haulage Cost Model, which focuses on maximising NPV while minimising haulage cost, achieves a high NPV of approximately $511 million, underscoring its effectiveness in maximising NPV. In contrast, the Combined Model, which incorporates both haulage and carbon cost reductions, shows a slightly lower NPV of about $505 million-a 1.20% decrease from the Haulage Cost Model. This suggests that the additional costs involved in reducing carbon emissions can slightly reduce economic gains. The Carbon Cost Model, which prioritises minimising carbon emissions and integrates outcomes from the Haulage Cost Model, reports the highest NPV of $512 million. This model benefits from optimising the placement of blocks at dump locations that offer the lowest combined haulage and carbon costs. Despite using the same number of blocks as determined by the Haulage Cost Model, strategic redistribution to minimise costs enhances its economic performance. However, the Combined Model processes less ore due to inclusion of carbon costs, which negatively impacts the economic value of the extracted ore blocks, resulting in a lower overall financial return compared to the other models. This comparison

illustrates how different operational strategies impact the financial outcomes, highlighting the trade-offs between environmental sustainability and economic performance in mining operations.

Table 8: NPV (Net Present Value) for each model

Haulage Cost Model	Carbon Cost Model	Combined Model
$ 510,983,675.14	$ 511,948,121.63	$ 504,914,388.07

The Progression of waste dump for the Haulage Cost Model can be seen in Figure 11 below. The development of the waste dump is managed through a lift-by-lift sequence where each lift is filled sequentially. During the first year, the progression of waste dumping operations focused on the lowest lift of the dump. By the second year, the first lift is completely filled , and the operations transition to the second lift and similarly for year 3,4 and 5.

The optimisation model aims to minimise the haulage cost relies on the constraints that govern the order and feasibility of the waste rock placement. All dump locations in the lower lift are filled before advancing to the lift above ensuring slope stability and safe construction practices. However, this constraint also influences haul routes as trucks must travel to newly activated blocks in an orderly sequence.

Each phase in the figure shows where waste can be deposited during a particular year, reflecting the outputs of the optimisation solver under given constraints (capacity limits, availability of dump location and precedence). As the dump expands upwards, haul distance can increase, impacting the total operational costs and emissions.

Lower hauling distance reduce costs and carbon emissions but limiting the upper lifts until the lower ones are filled can reduce operational flexibility. The constraints encode these trade-offs, dictating which dump location must be filled first and how to do so to minimise cost.

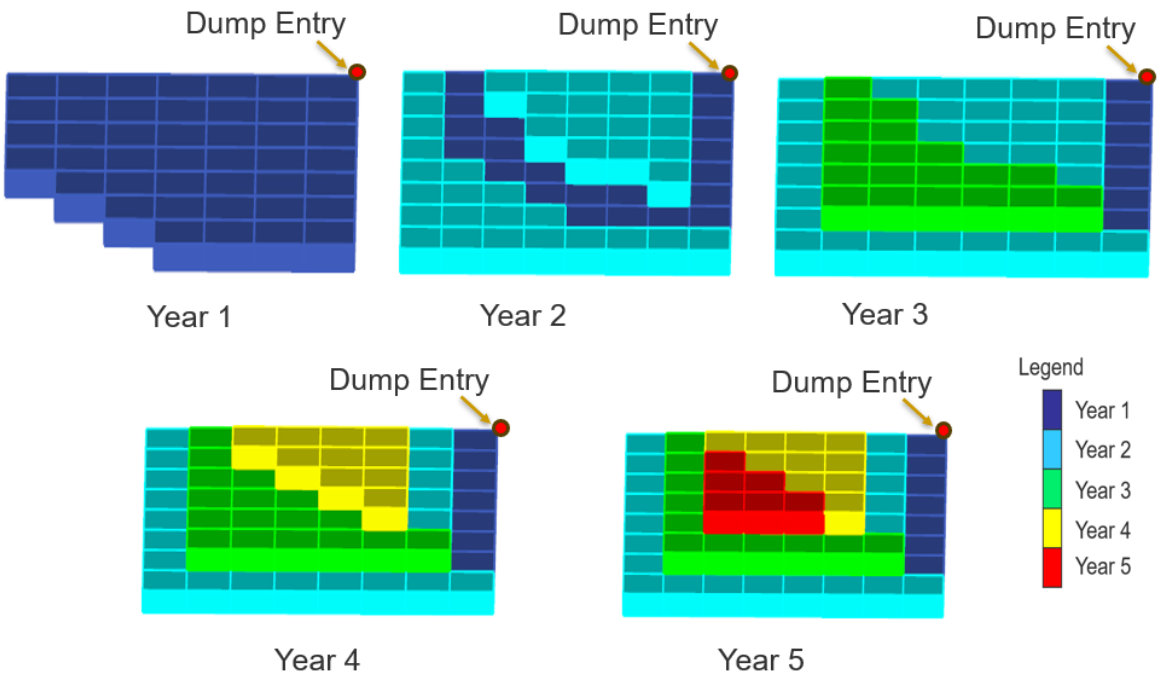

Figure 11:Progression of waste dump for the Haulage Cost Model

The Progression of waste dump for the Carbon Cost Model can be seen in Figure 12 below. The development of the waste dump follows a multi-lift sequence. During the first two years, operations concentrate on filling the lower lift. Starting from the third year, there is a shift to sequential filling, targeting both the first and second lifts simultaneously. This pattern becomes more pronounced in year four, as dump locations in the first , second and third lifts are filled concurrently. This multi-lift sequence allows for a more dynamic and flexible approach to waste management, accommodating simultaneous operations across different levels of the waste dump.

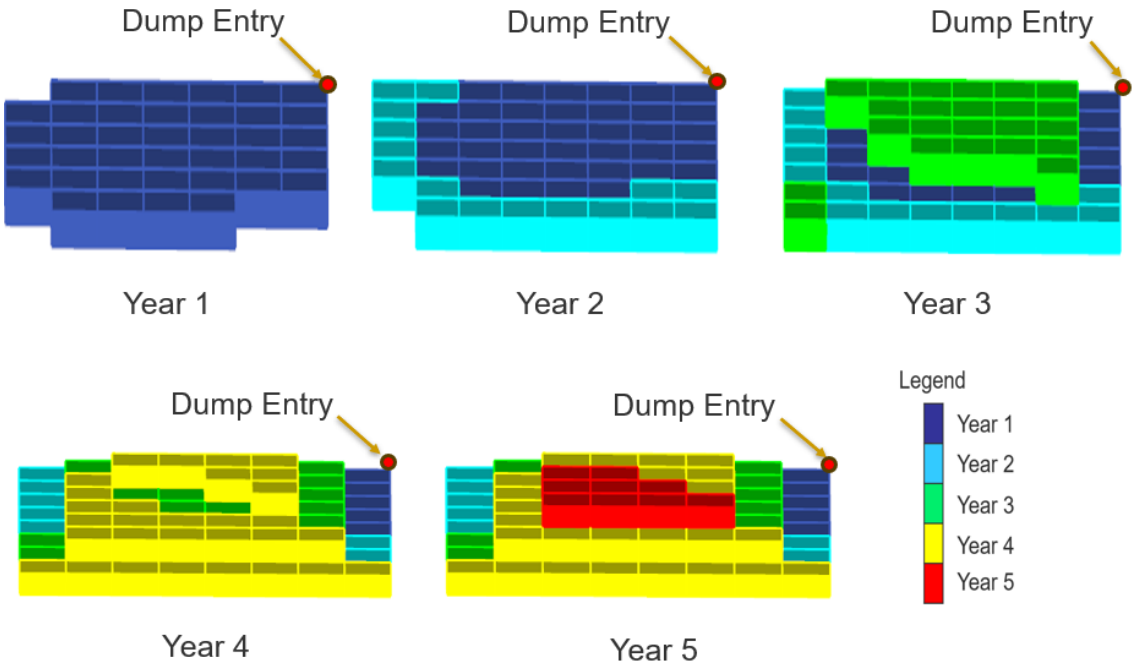

Figure 12: Progression of waste dump for the Carbon Cost Model

The Progression of waste dump for the Combined Model can be seen in Figure 13 below. Like carbon cost model, the development follows a multi-lift sequence. During the first year, operations concentrate on filling the lower lift first. Starting from the second year, there is a shift to sequential filling, targeting both the first and second lifts simultaneously.

The figure offers a visual timeline of how the multi-lift dump evolves under the combined model, where both economic (haulage cost) and environmental (carbon pricing and emissions) factors are integrated into a single optimisation framework. Each dump location block indicates where and when waste rock is dumped, illustrating the dynamic shift in operational priorities from one year to the next. By observing the distribution of waste rock across different lifts over time, stakeholders can quickly gauge whether the optimisation approach leans more towards cost minimisation (favouring the blocks closer to the dump entry) or carbon emission strategies (spreading out the waste to avoid emissions penalties).

The result helps decision makers see which blocks are activated earlier and why. For example, a block may be prioritised in Year 2 instead of Year 3 if it reduces overall haulage distance (lower cost) or meets carbon-related objectives (reduced emissions). The spatial progression thus makes trade-offs between economics and environmental impact more transparent.

The spatial layout of dumped material and the visualisation of timing (year-by-year) can pinpoint areas where scheduling or routing adjustments could yield greater cost saving or carbon cost reduction. This visual serves as a communication tool for interdisciplinary teams- allowing engineers, environmental officers, and financial analysts to collaborate more effectively.

The progressive view of dump also highlights future constraints. For instance, if certain lifts fill up early, subsequent lifts might involve longer haul distances or higher carbon fees. Such forward-looking insight supports proactive planning, ensuring the strategy viable both financially and environmentally over the life of the mine.

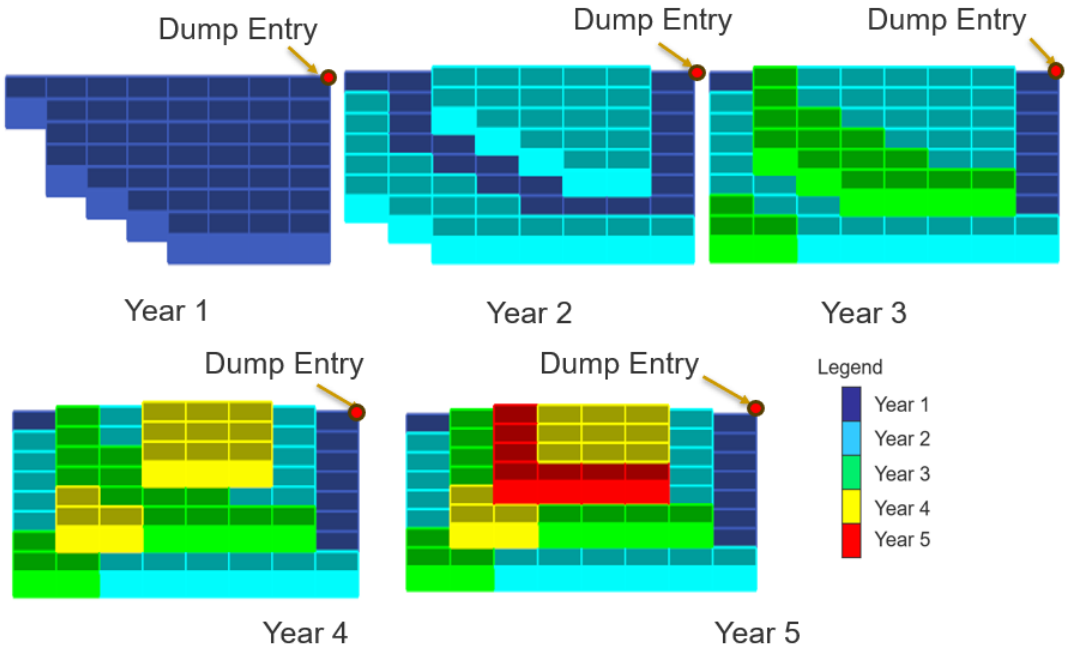

Figure 13: Progression of Waste Dump for the Combined Model

5.0 CONCLUSIONS AND RECOMMENDATIONS

This chapter provides a concise overview of the research conducted throughout this thesis, highlighting the key findings and conclusions. Additionally, it outlines potential avenues for future research and development in the area.

5.1 Conclusions

This research aimed to create three mathematical model: The first model maximises the discounted cash flow from production and waste dump scheduling considering only haulage cost, the second model introduces and integrates the carbon cost into the first model minimising the carbon cost taking into consideration the results of the production schedule from the first model. The third model is the combination of the first and second model maximising the discounted cash flow from production and waste dump scheduling considering both haulage and carbon cost. The three models are based on calculating actual haulage distances rather than using equivalent flat distances, thus providing more accurate cost estimates.

Most existing mine planning studies have traditionally focused on maximizing NPV and reducing haulage costs, but few have considered carbon emissions into account. This model addresses that gap by optimizing carbon emissions alongside haulage costs, while also using actual haulage distances. It allows for the optimal allocation of waste to dump locations and determining the best mining sequence, adding significant operational value to mining projects. The proposed model supports the mining industry's efforts towards achieving a lower carbon footprint while also potentially reducing haulage costs and emissions, and still maximising the production operations NPV.

The model was successfully implemented at an open pit gold mine. Results indicated differences in waste dump design when comparing the Haulage Cost Model, Carbon Cost Model and the Combined Model. The progression of waste dump in the haulage cost model follows a lift-by-lift waste dump construction sequence whereas the Carbon Cost Model and The Combined Model follows a multi-lift waste dump construction sequence proving that following a multi-lift waste dump construction sequence reduces haulage and carbon cost resulting in reduced carbon emissions.

The Combined Model resulted in a reduction of NPV of 1.2 % with respect to the Haulage Cost Model. The Carbon Cost Model resulted in an increase of NPV of 0.19% with respect to the Haulage Cost Model. The combined model resulted with reduced carbon emissions by 16.37% compared to

the haulage cost model and similarly a reduction of carbon emission of 12.24% with the Carbon cost model with respect to the Haulage cost model.

5.2 Recommendations

A valuable next step is to broaden the comparative analysis between the proposed approach and existing or alternative methods. This includes a more in-depth evaluation of net present value (NPV), carbon emissions and operational costs associated with the solution. Specifically, a comprehensive cost-benefit analysis would provide clearer evidence on the research. Due to limited data and scope, numerical comparison is not feasible but incorporating such quantitative and qualitative comparisons will enable stakeholders to better appreciate the practical advantages of the proposed approach, along with its contribution to sustainability, financial viability, and operational efficiency.

Currently the research focuses on a single pit and a single waste dump, while most mines have multiple pits and waste dumps. Expanding the model to account for multiple pits and dumps would yield more realistic results and solutions. Additionally, the current research assumes a perfectly square waste dump, but future studies could incorporate more realistic waste dump shapes, as no dump is a perfect square.

Integrating the model with a truck dispatching system in mining operations could substantially enhance operational efficiency. Such system optimizes the flow of material by making real-time decisions on truck assignments and routes, which could be aligned with the model to dynamically manage waste transport based on current conditions, thereby reducing haulage costs and minimizing carbon emissions.

Furthermore, while the proposed model is deterministic, transitioning to a stochastic model by incorporating probabilistic elements into the inputs would better reflect the inherent uncertainties of mining operations. This shift would allow for a more nuanced approach to decision-making, providing robust solutions under various scenarios, such as fluctuations in fuel prices or variable ore grades.

This research may potentially provide significant value to the mining industry, particularly in minimizing carbon emissions and haulage costs while maximizing NPV. The models, especially if evolved to include these recommendations, are highly recommended for companies seeking to adapt to more sustainable mining practices. These enhancements will ensure that the model addresses current operational efficiencies which can easily be adapted to future challenges, thereby providing a comprehensive and flexible tool to drive sustainable mining practices.

References

Aghdamigargari, M., Avane, S., Anani, A., & Adewuyi, S. O. (2024). Sustainability in Long-Term Surface Mine Planning: A Systematic Review of Operations Research Applications. *Sustainability*, *16*(22), 9769. https://www.mdpi.com/2071-1050/16/22/9769

Bajany, M. D. (2017). *A mixed integer linear programming model for truck-shovel scheduling to minimize fuel consumption* University of Pretoria].

Caccetta, L., & Hill, S. P. (2003). An application of branch and cut to open pit mine scheduling. *Journal of global optimization*, *27*, 349-365.

Dagdelen, K. (1986). Optimum open pit mine production scheduling by Lagrangian parameterization. *Proc. of the 19th APCOM*, 127-142.

Das, R., Topal, E., & Mardaneh, E. (2022). Improved optimised scheduling in stratified deposits in open pit mines; using in-pit dumping. *International journal of mining, reclamation and environment*, *36*(4), 287-304. https://doi.org/10.1080/17480930.2022.2036559

Das, R., Topal, E., & Mardaneh, E. (2023). A review of open pit mine and waste dump schedule planning. *Resources policy*, *85*, 104064. https://doi.org/10.1016/j.resourpol.2023.104064

Das, R., Topal, E., & Mardaneh, E. (2024). Concurrent optimisation of open pit ore and waste movement with optimal haul road selection. *Resources policy*, *91*, 104834. https://doi.org/10.1016/j.resourpol.2024.104834

Dindarloo, S. R., & Siami-Irdemoosa, E. (2016). Determinants of fuel consumption in mining trucks. *Energy (Oxford)*, *112*, 232-240. https://doi.org/10.1016/j.energy.2016.06.085

Dowd, P., & Onur, A. (1993). Open-pit optimization. 1. Optimal open-pit design. *Transactions of the Institution of Mining and Metallurgy Section A-Mining Industry*, *102*.

Fathollahzadeh, K., Asad, M. W. A., Mardaneh, E., & Cigla, M. (2021). Review of Solution Methodologies for Open Pit Mine Production Scheduling Problem. *International journal of mining, reclamation and environment*, *35*(8), 564-599. https://doi.org/10.1080/17480930.2021.1888395

Fourer, R., Gay, D. M., & Kernighan, B. W. (2003). AMPL. A modeling language for mathematical programming.

Fu, Z., Asad, M. W. A., & Topal, E. (2019). A new model for open-pit production and waste-dump scheduling. *Engineering optimization*, *51*(4), 718-732. https://doi.org/10.1080/0305215X.2018.1476501

Gershon, M. E. (1983). Optimal mine production scheduling: evaluation of large scale mathematical programming approaches. *International journal of mining engineering*, *1*, 315-329.

Gholami, H., Abu, F., Lee, J. K. Y., Karganroudi, S. S., & Sharif, S. (2021). Sustainable Manufacturing 4.0—Pathways and Practices. *Sustainability*, *13*(24), 13956. https://www.mdpi.com/2071-1050/13/24/13956

Groeneveld, B., & Topal, E. (2011). Flexible open-pit mine design under uncertainty. *Journal of mining science*, *47*(2), 212-226. https://doi.org/10.1134/S1062739147020080

Groeneveld, B., Topal, E., & Leenders, B. (2019). Examining system configuration in an open pit mine design. *Resources policy*, *63*, 101438.

Johnson, T. B. (1968). *Optimum open pit mine production scheduling*. University of California, Berkeley.

Kecojevic, V., & Komljenovic, D. (2010). Haul truck fuel consumption and CO2 emission under various engine load conditions. *Mining Engineering*, *62*(12), 44-48.

Li, Y., Topal, E., & Ramazan, S. (2016). Optimising the long-term mine waste management and truck schedule in a large-scale open pit mine. *Transactions of the Institution of Mining and Metallurgy. Section A, Mining technology*, *125*(1), 35-46. https://doi.org/10.1080/14749009.2015.1107343

Li, Y., Topal, E., & Williams, D. (2013). Waste rock dumping optimisation using mixed integer programming (MIP). *International journal of mining, reclamation and environment*, *27*(6), 425-436. https://doi.org/10.1080/17480930.2013.794513

Li, Y., Topal, E., & Williams, D. J. (2014). Optimisation of waste rock placement using mixed integer programming. *Transactions of the Institution of Mining and Metallurgy. Section A, Mining technology*, *123*(4), 220-229. https://doi.org/10.1179/1743286314Y.0000000070

Li, Z. (1990). A methodology for the optimum control of shovel and truck operations in open-pit mining. *Mining science & technology*, *10*(3), 337-340. https://doi.org/10.1016/0167-9031(90)90543-2

Lin, J., Asad, M. W. A., Topal, E., Chang, P., Huang, J., & Lin, W. (2024). A novel model for sustainable production scheduling of an open-pit mining complex considering waste encapsulation. *Resources policy*, *91*, 104949.

Liu, L.-y., Ji, H.-g., Lü, X.-f., Wang, T., Zhi, S., Pei, F., & Quan, D.-l. (2021). Mitigation of greenhouse gases released from mining activities: A review. *International Journal of Minerals, Metallurgy and Materials*, *28*, 513-521.

Lizotte, Y., & Bonates, E. (1987). Truck and shovel dispatching rules assessment using simulation. *Mining science & technology*, *5*(1), 45-58. https://doi.org/10.1016/S0167-9031(87)90910-8

Mai, N., Topalt, E., & Ertent, O. (2018). A new open-pit mine planning optimization method using block aggregation and integer programming. *Journal of the Southern African Institute of Mining and Metallurgy*, *118*(7), 705-714.

Newman, A. M., Rubio, E., Caro, R., Weintraub, A., & Eurek, K. (2010). A Review of Operations Research in Mine Planning. *Interfaces (Providence)*, *40*(3), 222-245. https://doi.org/10.1287/inte.1090.0492

Osanloo, M., Gholamnejad, J., & Karimi, B. (2008). Long-term open pit mine production planning: a review of models and algorithms. *International journal of mining, reclamation and environment*, *22*(1), 3-35. https://doi.org/10.1080/17480930601118947

Puell Ortiz, J. (2017). Methodology for a dump design optimization in large-scale open pit mines. *Cogent engineering*, *4*(1), 1387955. https://doi.org/10.1080/23311916.2017.1387955

Rosa, E. A., & Dietz, T. (2012). Human drivers of national greenhouse-gas emissions. *Nature climate change*, *2*(8), 581-586. https://doi.org/10.1038/nclimate1506

Rosen, M. A., & Kishawy, H. A. (2012). Sustainable Manufacturing and Design: Concepts, Practices and Needs. *Sustainability*, *4*(2), 154-174. https://www.mdpi.com/2071-1050/4/2/154

Runge, I. C. (1998). *Mining economics and strategy*. SME.

Sahoo, L. K., Bandyopadhyay, S., & Banerjee, R. (2014). Benchmarking energy consumption for dump trucks in mines. *Applied energy*, *113*, 1382-1396. https://doi.org/10.1016/j.apenergy.2013.08.058

Shishvan, M. S., & Sattarvand, J. (2015). Long term production planning of open pit mines by ant colony optimization. *European journal of operational research*, *240*(3), 825-836. https://doi.org/10.1016/j.ejor.2014.07.040

Siami-Irdemoosa, E., & Dindarloo, S. R. (2015). Prediction of fuel consumption of mining dump trucks: A neural networks approach. *Applied energy*, *151*, 77-84. https://doi.org/10.1016/j.apenergy.2015.04.064

Taha, H. A. (2013). *Operations research: an introduction*. Pearson Education India.

Tolwtnski, B., & Underwood, R. (1996). A scheduling algorithm for open pit mines. *IMA Journal of Management Mathematics*, *7*(3), 247-270.

Topal, E. (2003). *Advanced Underground Mine Scheduling Using Mixed Integer Programming* ProQuest Dissertations & Theses].

Topal, E., & Ramazan, S. (2012). Mining truck scheduling with stochastic maintenance cost. *Journal of Coal Science and Engineering (China)*, *18*, 313-319.

Vukovic, V. (2013). *Reducing Haul Truck Fuel Consumption in Open Pit Mines by Strategic Changes to the Haulage Cycle* ProQuest Dissertations Publishing].

Wang, X., Chow, J. C., Kohl, S. D., Percy, K. E., Legge, A. H., & Watson, J. G. (2016). Real-world emission factors for Caterpillar 797B heavy haulers during mining operations. *Particuology*, *28*(5), 22-30. https://doi.org/10.1016/j.partic.2015.07.001

Williams, D., Topal, E., Zhang, N., & Scott, P. (2008). Development of a rock dump scheduling model using linear programming. Rock Dumps 2008: Proceedings of the First International Seminar on the Management of Rock Dumps, Stockpiles and Heap Leach Pads,

APPENDIX A

Haulage Cost Model using python

```
import pandas as pd

import math

from gurobipy import Model, GRB, quicksum

from gurobipy import min_

import numpy as np


# Load the haulage path data files

pit_exit1_to_processing_plant_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\Processing road 1.csv")

pit_exit2_to_processing_plant_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktoptest 8\Processing road 2.csv")

block_to_pit_exit1_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\block to pit exit 1.csv")

block_to_pit_exit2_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\block to pit exit 2.csv")

pit_exit1_to_waste_dump_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\pit exit 1 to waste dump entry.csv")

pit_exit2_to_waste_dump_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\pit exit 2 to waste dump entry.csv")



def calculate_haulage_distance(df):
    # Convert DataFrame to numpy array if not already
    if isinstance(df, pd.DataFrame):
        points = df[['X', 'Y', 'Z']].to_numpy()
    elif isinstance(df, np.ndarray):
        points = df
    else:
        raise ValueError("Input must be a pandas DataFrame or a numpy array")


    # Check if there are enough points to calculate distances
    if points.size == 0 or len(points) < 2:
```

```
        return 0

    return sum(math.sqrt((x2 - x1)**2 + (y2 - y1)**2 + (z2 - z1)**2)
               for (x1, y1, z1), (x2, y2, z2) in zip(points[:-1], points[1:]))

distance_block_to_pit_exit1 = calculate_haulage_distance(block_to_pit_exit1_df)
distance_block_to_pit_exit2 = calculate_haulage_distance(block_to_pit_exit2_df)
distance_pit_exit1_to_processing_plant = calculate_haulage_distance(pit_exit1_to_processing_plant_df[['X', 'Y', 'Z']].values)
distance_pit_exit2_to_processing_plant = calculate_haulage_distance(pit_exit2_to_processing_plant_df[['X', 'Y', 'Z']].values)
distance_pit_exit1_to_waste_dump = calculate_haulage_distance(pit_exit1_to_waste_dump_df)
distance_pit_exit2_to_waste_dump = calculate_haulage_distance(pit_exit2_to_waste_dump_df)

haulage_paths = {
    'block_to_pit_exit1': distance_block_to_pit_exit1,
    'block_to_pit_exit2': distance_block_to_pit_exit2,
    'pit_exit1_to_processing_plant': distance_pit_exit1_to_processing_plant,
    'pit_exit2_to_processing_plant': distance_pit_exit2_to_processing_plant,
    'pit_exit1_to_waste_dump': distance_pit_exit1_to_waste_dump,
    'pit_exit2_to_waste_dump': distance_pit_exit2_to_waste_dump
}
#block model and precedence data
block_model_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\VALENS 6.csv")
precedence_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\block_precedence2.csv")
waste_dump_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\waste dump 26092024.csv")
waste_precedence_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\waste_precedence1.csv")
```

```
# Dictionary for each dump mapping to its predecessors
waste_dump_precedence_dict = {dump: [] for dump in waste_dump_df['DumpLocationID'].unique()}  # Initialize all dumps with an empty list
for index, row in waste_precedence_df.iterrows():
    dump_id = row['DumpLocationID']
    precedent_id = row['PrecedentDumpLocationID']
    if dump_id in waste_dump_precedence_dict:
        waste_dump_precedence_dict[dump_id].append(precedent_id)

# Precedence dictionary for mining blocks
precedences_dict = {}
for index, row in precedence_df.iterrows():
    block_id = row['blockID']
    precedent_id = row['precedenceID']
    if block_id not in precedences_dict:
        precedences_dict[block_id] = []
    precedences_dict[block_id].append(precedent_id)

# Dictionary for waste dump capacities for easy access
waste_dump_capacities = pd.Series(waste_dump_df.volume.values, index=waste_dump_df.DumpLocationID).to_dict()

# Set 'blockID' as the DataFrame index if it's not already
block_model_df.set_index('blockID', inplace=True)
waste_dump_df.set_index('DumpLocationID', inplace=True)

# preprocessing to create the model parameters
blocks = block_model_df.index.tolist()
dumps = waste_dump_df.index.tolist()
processing_plant_coords = {
    'processing_plant': (4592.805, 3430.879, 510.000)
}
```

```
# Initialize the dictionary for waste dump locations after setting 'DumpLocationID' as index
waste_dump_coords = {}
for dump_id, row in waste_dump_df.iterrows():
    waste_dump_coords[f'waste_dump_{dump_id}'] = (row['X'], row['Y'], row['Z'])

# Parameters
def calc_gold_price(t):
    # Assuming gold price increases by 2% every year starting from a base price of 1500
    base_price = 200
    annual_growth_rate = 0.05
    return base_price * (1 + annual_growth_rate) ** (t-1)

def mining_capacity(t):
    # Assuming mining capacity is fixed at 100000 tons per year
    base_capacity = 6000000
    final_capacity = 7000000
    return base_capacity + ((final_capacity - base_capacity) / (T-1)) * (t-1)

def processing_capacity(t):
    # Assuming processing capacity increases from 30000 to 50000 tons over 5 years linearly
    base_capacity = 3000000
    final_capacity = 4000000
    return base_capacity + ((final_capacity - base_capacity) / (T-1)) * (t-1)

MC = MC = dict(zip(block_model_df.index, block_model_df['mining_cost']))

T = 5  # Time periods (years)

r = 0.1  # Discount rate
```

```
MCap_t = {t: mining_capacity(t) for t in range(1, T+1)}


PCap_t = {t: processing_capacity(t) for t in range(1, T+1)}
Swell_factor = 1.25
Density = 2.75
# Define the Euclidean distance function
def euclidean_distance(point1, point2):
    return math.sqrt(sum((p1 - p2) ** 2 for p1, p2 in zip(point1, point2)))


# Function to extract haulage paths from the Excel data
def extract_haulage_paths(haulage_data_df):
    haulage_paths = {}
    for sheet_name in haulage_data_df.sheet_names:
        df = pd.read_excel(haulage_data_df, sheet_name=sheet_name)
        points = df.sort_values(by='PointID')[['X', 'Y', 'Z']].values
        # Calculate path distance
        path_distance = sum(euclidean_distance(points[i], points[i + 1]) for i in range(len(points) - 1))
        # Store path info
        haulage_paths[sheet_name] = {'points': points, 'path_distance': path_distance}
    return haulage_paths


# Haulage cost multipliers for different segments
haulage_costs = {
    'block_to_pit': 0.042,   # Cost per unit distance from block to pit
    'pit_to_entry': 0.017,  # Cost per unit distance from pit to waste dump entry
    'entry_to_dump_lower': 0.042,  # Cost per unit distance from waste dump entry to the lower dump level
    'entry_to_dump_middle': 0.068, # Cost per unit distance from waste dump entry to the middle dump level
    'entry_to_dump_upper': 0.093,  # Cost per unit distance from waste dump entry to the upper dump level
    'pit_exit1_to_plant': 0.017,  # Cost per unit distance from pit exit 1 to processing plant
    'pit_exit2_to_plant': 0.017  # Cost per unit distance from pit exit 2 to processing plant
```

```
}
carbon_costs = {
    'block_to_pit': 0.0,  # Carbon cost per tonne-meter for block to pit
    'pit_to_entry': 0.0,  # Carbon cost per tonne-meter for pit to waste dump entry
    'entry_to_dump_lower': 0.0,  # Carbon cost per tonne-meter for lower dump level
    'entry_to_dump_middle': 0.0, # Carbon cost per tonne-meter for middle dump level
    'entry_to_dump_upper': 0.0,  # Cost per unit distance from waste dump entry to the upper dump level
    'pit_exit1_to_plant': 0.0,   # Carbon cost per tonne-meter for pit exit 2 to processing plant
    'pit_exit2_to_plant': 0.0   # Carbon cost per tonne-meter for pit exit 2 to processing plant
}


def closest_point_distance(block_coords, path_df):
    # Calculate distances from block to all points in the DataFrame
    distances = path_df.apply(lambda row: euclidean_distance(block_coords, row[['X', 'Y', 'Z']].values), axis=1)
    min_index = distances.idxmin()
    min_distance = distances.loc[min_index]
    closest_point_coords = path_df.loc[min_index, ['X', 'Y', 'Z']].values
    return min_distance, closest_point_coords


def haulage_cost_to_processing_plant(
    block_ID, block_model_df, processing_plant_coords,
    haulage_paths, haulage_costs, carbon_costs, quantity):


    block_coords = block_model_df.loc[block_ID, ['X', 'Y', 'Z']].values
    destination_key = 'processing_plant'
    if destination_key not in processing_plant_coords:
        raise ValueError("Processing plant coordinates not found in destination coordinates.")


    # Access pre-calculated distances from the haulage_paths dictionary
    distance_block_to_pit_exit1 = haulage_paths['block_to_pit_exit1']
```

```
    distance_block_to_pit_exit2 = haulage_paths['block_to_pit_exit2']
    distance_pit_exit1_to_processing_plant = haulage_paths['pit_exit1_to_processing_plant']
    distance_pit_exit2_to_processing_plant = haulage_paths['pit_exit2_to_processing_plant']


    # Calculate distances from the block to the closest points of the pit exits
    distance_to_closest_point_exit1, closest_point_exit1 = closest_point_distance(block_coords, block_to_pit_exit1_df)
    distance_to_closest_point_exit2, closest_point_exit2 = closest_point_distance(block_coords, block_to_pit_exit2_df)


    # Calculate total distances for the paths from the block to the processing plant via each pit exit
    total_distance_via_exit1 = distance_to_closest_point_exit1 + distance_block_to_pit_exit1 + distance_pit_exit1_to_processing_plant
    total_distance_via_exit2 = distance_to_closest_point_exit2 + distance_block_to_pit_exit2 + distance_pit_exit2_to_processing_plant


    # Calculate haulage costs for each route (excluding carbon costs)
    haulage_cost_via_exit1 = (
        (distance_to_closest_point_exit1 + distance_block_to_pit_exit1) * haulage_costs['block_to_pit'] +
        distance_pit_exit1_to_processing_plant * haulage_costs['pit_exit1_to_plant']
    ) / quantity


    haulage_cost_via_exit2 = (
        (distance_to_closest_point_exit2 + distance_block_to_pit_exit2) * haulage_costs['block_to_pit'] +
        distance_pit_exit2_to_processing_plant * haulage_costs['pit_exit2_to_plant']
    ) / quantity


    # Calculate carbon costs for each route
    carbon_cost_via_exit1 = (
```

```
        (distance_to_closest_point_exit1 + distance_block_to_pit_exit1) * carbon_costs['block_to_pit']
+
        distance_pit_exit1_to_processing_plant * carbon_costs['pit_exit1_to_plant']
    ) / quantity

    carbon_cost_via_exit2 = (
        (distance_to_closest_point_exit2 + distance_block_to_pit_exit2) * carbon_costs['block_to_pit']
+
        distance_pit_exit2_to_processing_plant * carbon_costs['pit_exit2_to_plant']
    ) / quantity

    # Aggregate haulage and carbon costs for each route
    total_costs_per_unit = {
        'exit1': (haulage_cost_via_exit1, carbon_cost_via_exit1),
        'exit2': (haulage_cost_via_exit2, carbon_cost_via_exit2)
    }

    # Select the minimum cost path based on the total of haulage and carbon costs
    selected_path_key = min(total_costs_per_unit, key=lambda x: sum(total_costs_per_unit[x]))

    # Return both the haulage cost and the carbon cost for the selected path
    return total_costs_per_unit[selected_path_key]



def closest_point_distance(block_coords, path_df):
    # Calculate distances from block to all points in the DataFrame
    distances = path_df.apply(lambda row: euclidean_distance(block_coords, row[['X', 'Y',
'Z']].values), axis=1)
    # Find the index of the minimum distance
    min_index = distances.idxmin()
    # Return the minimum distance and the coordinates of the closest point
    min_distance = distances.loc[min_index]
    closest_point_coords = path_df.loc[min_index, ['X', 'Y', 'Z']].values
```

```
    return min_distance, closest_point_coords

def total_haulage_cost_per_unit(block_ID, block_model_df, DumpLocationID, waste_dump_df,
                                haulage_paths, haulage_costs, carbon_costs, quantity):

    # Extract the block coordinates from the block model DataFrame
    block_coords = block_model_df.loc[block_ID, ['X', 'Y', 'Z']].values



    if DumpLocationID not in waste_dump_df.index:
        raise ValueError(f"DumpLocationID {DumpLocationID} not found in waste_dump_ddddf.")

    waste_dump_coords = waste_dump_df.loc[DumpLocationID, ['X', 'Y', 'Z']].values
    waste_dump_z = waste_dump_df.loc[DumpLocationID, 'Z']

    # Determine dump level cost key based on elevation
    if waste_dump_z <= 550:
        dump_level_cost_key = 'entry_to_dump_lower'
    elif waste_dump_z <= 570:
        dump_level_cost_key = 'entry_to_dump_middle'  # New middle level
    else:
        dump_level_cost_key = 'entry_to_dump_upper'

    # Access pre-calculated distances from the haulage_paths dictionary
    distance_block_to_pit_exit1 = haulage_paths['block_to_pit_exit1']
    distance_block_to_pit_exit2 = haulage_paths['block_to_pit_exit2']
    distance_pit_exit1_to_waste_dump = haulage_paths['pit_exit1_to_waste_dump']
    distance_pit_exit2_to_waste_dump = haulage_paths['pit_exit2_to_waste_dump']

    total_costs = {}
    total_carbon = {}
```

```
    for exit_key, path_components in {
        'exit1': (block_to_pit_exit1_df, pit_exit1_to_waste_dump_df),
        'exit2': (block_to_pit_exit2_df, pit_exit2_to_waste_dump_df)
    }.items():
        # Calculate the closest points and distances for the components of the selected path
        distance_to_closest_point, closest_first_point = closest_point_distance(block_coords, path_components[0])
        # Calculate the distance from the last point on the pit exit path to the waste dump
        last_point_to_dump_distance = euclidean_distance(path_components[1].iloc[-1][['X', 'Y', 'Z']].values, waste_dump_coords)

        # Total distance for the haulage route includes segments from block to pit, pit to last path point, and last point to dump
        total_distance = (distance_to_closest_point + distance_block_to_pit_exit1 if exit_key == 'exit1' else distance_block_to_pit_exit2) \
                         + (distance_pit_exit1_to_waste_dump if exit_key == 'exit1' else distance_pit_exit2_to_waste_dump) \
                         + last_point_to_dump_distance

        # Calculate total haulage cost based on distances and costs for each segment
        total_cost = (
            (distance_to_closest_point + distance_block_to_pit_exit1 if exit_key == 'exit1' else distance_block_to_pit_exit2) * (haulage_costs['block_to_pit'] + carbon_costs['block_to_pit']) +
            (distance_pit_exit1_to_waste_dump if exit_key == 'exit1' else distance_pit_exit2_to_waste_dump) * (haulage_costs['pit_to_entry'] + carbon_costs['pit_to_entry']) +
            last_point_to_dump_distance * (haulage_costs[dump_level_cost_key] + carbon_costs[dump_level_cost_key])
        )

        # Carbon cost calculation for each segment
        total_carbon_emissions = (
            (distance_to_closest_point + distance_block_to_pit_exit1 if exit_key == 'exit1' else distance_block_to_pit_exit2) * carbon_costs['block_to_pit'] +
```

```
        (distance_pit_exit1_to_waste_dump if exit_key == 'exit1' else distance_pit_exit2_to_waste_dump) * carbon_costs['pit_to_entry'] +
        last_point_to_dump_distance * carbon_costs[dump_level_cost_key]
    )

    # Normalize the total cost and carbon emissions by the quantity to get the cost and carbon per unit
    total_costs_per_unit = total_cost / quantity
    total_carbon_per_unit = total_carbon_emissions / quantity

    total_costs[exit_key] = total_costs_per_unit
    total_carbon[exit_key] = total_carbon_per_unit

# Select the minimum cost path per unit and return both cost and carbon per unit
selected_path_key = min(total_costs, key=total_costs.get)
return total_costs[selected_path_key], total_carbon[selected_path_key]



def calculate_discounted_profit_per_unit(block_ID, path_key, DumpLocationID, destination, year,
                                         block_model_df, waste_dump_df,
                                         haulage_paths, haulage_costs, carbon_costs):
    block_row = block_model_df.loc[block_ID]
    grade = block_row['grade']
    mining_cost_per_unit = block_row['mining_cost']  # Assuming this is per unit
    discount_rate = 0.1
    quantity = block_row['quantity']

    # Calculate the cost including both haulage and carbon cost based on the destination
    if destination == "plant":
        # Unpack both haulage cost and carbon cost
        haulage_cost_per_unit, carbon_cost_per_unit = haulage_cost_to_processing_plant(
```

```
        block_ID, block_model_df, processing_plant_coords,
        haulage_paths,  # Only pass haulage_paths here
        haulage_costs, carbon_costs, quantity
    )
    processing_cost_per_unit = block_row.get('processing_cost_p1', 0)
    recovery_rate = block_row.get('recovery_rate_p1', 1)  # Default recovery rate for processing
else:
    # Unpack both haulage cost and carbon cost
    haulage_cost_per_unit, carbon_cost_per_unit = total_haulage_cost_per_unit(
        block_ID, block_model_df, DumpLocationID, waste_dump_df,
        haulage_paths,  # Only pass haulage_paths here
        haulage_costs, carbon_costs, quantity
    )
    processing_cost_per_unit = 0  # No processing cost for waste dump
    recovery_rate = 0  # No recovery as the material is treated as waste

# Include both haulage and carbon costs in the total operational cost
total_operational_cost_per_unit = (
    mining_cost_per_unit
    + processing_cost_per_unit
    + haulage_cost_per_unit
    + carbon_cost_per_unit  # Carbon cost is now part of the operational cost
)



# Calculate the revenue per unit based on grade and recovery rate
revenue_per_unit = grade * recovery_rate * calc_gold_price(year)

# Calculate the net value per unit
net_value_per_unit = revenue_per_unit - total_operational_cost_per_unit
```

```
    # Apply the discount factor to get the discounted net value
    discounted_net_value_per_unit = net_value_per_unit / ((1 + discount_rate) ** (year - 1))

    return discounted_net_value_per_unit

# Create a new model
m = Model("maximize_npv")

# Decision Variables
# Decision Variables for routing to the processing plant and waste dumps
to_processing_plant = m.addVars(blocks, ['exit1', 'exit2'], range(1, T+1), vtype=GRB.CONTINUOUS, name="to_processing_plant", lb=0)
to_waste_dump = m.addVars(blocks, dumps, ['exit1', 'exit2'], range(1, T+1), vtype=GRB.CONTINUOUS, name="to_waste_dump", lb=0)
x = m.addVars(blocks, range(1, T+1), vtype=GRB.BINARY, name="x")
# Variables for dump activation and fullness
is_full = m.addVars(dumps, range(1, T+1), vtype=GRB.BINARY, name="is_full")

# Objective Function: Minimize the cost of transporting materials to the processing plant and waste dumps
m.setObjective(
    quicksum(
        to_processing_plant[i, path, t] * calculate_discounted_profit_per_unit(
            i, path, None, 'plant', t, block_model_df, waste_dump_df,
            haulage_paths, haulage_costs, carbon_costs
        ) for i in blocks for path in ['exit1', 'exit2'] for t in range(1, T+1)
    ) + quicksum(
        to_waste_dump[i, d, path, t] * calculate_discounted_profit_per_unit(
            i, path, d, 'dump', t, block_model_df, waste_dump_df,
            haulage_paths, haulage_costs, carbon_costs
        ) for i in blocks for d in dumps for path in ['exit1', 'exit2'] for t in range(1, T+1)
    ), GRB.MAXIMIZE
)
```

```
#Contraint
# Constraints to ensure the volume sent via each path does not exceed the total quantity mined
for i in blocks:
    for t in range(1, T+1):
        m.addConstr(
            quicksum(to_processing_plant[i, path, t] for path in ['exit1', 'exit2']) +
            quicksum(to_waste_dump[i, d, path, t] for d in dumps for path in ['exit1', 'exit2'])
            ≤ block_model_df.loc[i, 'quantity'],
            name=f"allocation_{i}_{t}"
        )

# Mining Capacity Constraint
for t in range(1, T+1):
    m.addConstr(
        quicksum(quicksum(to_processing_plant[i, path, t] for path in ['exit1', 'exit2']) for i in blocks) +
        quicksum(quicksum(to_waste_dump[i, d, path, t] for path in ['exit1', 'exit2']) for i in blocks for
d in dumps)
        <= MCap_t[t],
        name=f"mining_cap_{t}"
    )
# Processing Capacity Constraint
    m.addConstr(
        quicksum(quicksum(to_processing_plant[i, path, t] for path in ['exit1', 'exit2']) for i in blocks)
        <= PCap_t[t],
        name=f"processing_capacity_{t}"
    )

# Precedence constraints for blocks
for block_id, predecessors in precedences_dict.items():
    for t in range(1, T+1):  # T is your total number of time periods
        for precedent_id in predecessors:
```

```
            m.addConstr(
                x[block_id, t] <= quicksum(x[precedent_id, tp] for tp in range(1, t)),
                name=f"precedence_{block_id}_{precedent_id}_{t}"
            )


# Waste dump precedence constraints
for d in dumps:
    for t in range(1, T+1):
        # Cumulative waste calculation updated to new path keys
        cumulative_waste = quicksum((swell_factor [i] / density [i] ) * to_waste_dump[i, d, path, tp]
                                    for i in blocks for path in ['exit1', 'exit2']
                                    for tp in range(1, t+1))



        # Capacity and fullness constraints remain the same
        m.addConstr(cumulative_waste <= waste_dump_capacities[d], name=f"capacity_{d}_{t}")
        m.addGenConstrIndicator(is_full[d, t], True, cumulative_waste >= waste_dump_capacities[d],
name=f"full_{d}_{t}")
        m.addGenConstrIndicator(is_full[d, t], False, cumulative_waste <= waste_dump_capacities[d]
- 1e-5, name=f"not_full_{d}_{t}")


        # Refined activation based on predecessors being full
all_preds_full = m.addVar(vtype=GRB.BINARY, name=f"all_preds_full_{d}_{t}")
dump_active = m.addVars(dumps, range(1, T+1), vtype=GRB.BINARY, name="dump_active")


        if t > 1:
            if waste_dump_precedence_dict[d]:  # Check if there are predecessors
                m.addGenConstrAnd(all_preds_full, [is_full[pred, t-1] for pred in
waste_dump_precedence_dict[d]])
                m.addConstr(dump_active[d, t] == all_preds_full, name=f"activate_{d}_{t}")
            else:
                m.addConstr(dump_active[d, t] == 1, name=f"activate_{d}_{t}_no_predecessors")
        else:
```

```
        m.addConstr(dump_active[d, t] == (1 if not waste_dump_precedence_dict[d] else 0), name=f"activate_{d}_{t}_initial")


# constraints linking waste disposal to Active dumps
for i in blocks:
    for d in dumps:
        for path in ['exit1', 'exit2']:  # Use the new path keys
            for t in range(1, T+1):
                m.addConstr(
                    to_waste_dump[i, d, path, t] <= dump_active[d, t] * block_model_df.loc[i, 'quantity'],
                    name=f"link_waste_{i}_{d}_{path}_{t}"
                )

# Solve the model
m.setParam('MIPGap', 0.1)

m.optimize()  # Optimize the model
if m.status == GRB.OPTIMAL or m.status == GRB.SUBOPTIMAL:
    print("Optimal or suboptimal solution found:\n")
    # Initialize dictionaries for analysis
    total_to_plant_by_year = {t: 0 for t in range(1, T+1)}
    total_to_dump_by_year = {t: 0 for t in range(1, T+1)}
    mined_blocks_by_year = {t: set() for t in range(1, T+1)}
    full_dumps_by_year = {t: set() for t in range(1, T+1)}
    grade_to_plant_by_year = {t: 0 for t in range(1, T+1)}
    quantity_to_plant_by_year = {t: 0 for t in range(1, T+1)}
    blocks_to_plant_by_year = {t: set() for t in range(1, T+1)}
    blocks_to_dump_by_year = {t: {} for t in range(1, T+1)}
    yearly_haulage_costs = {year: 0 for year in range(1, T+1)}
    yearly_carbon_costs = {year: 0 for year in range(1, T+1)}
    yearly_haulage_costs_to_plant = {year: 0 for year in range(1, T+1)}
    yearly_haulage_costs_to_dump = {year: 0 for year in range(1, T+1)}
```

```
    data_records = []
    total_haulage_cost = 0
    total_carbon_cost = 0

    for t in range(1, T+1):
        for d in dumps:
            blocks_to_dump_by_year[t][d] = set()

    # Iterate over all blocks and time periods
    for i in blocks:
        for t in range(1, T+1):
            if x[i, t].X > 0.5:
                mined_blocks_by_year[t].add(i)

            for path_key in ['exit1', 'exit2']:
                quantity_to_plant = to_processing_plant[i, path_key, t].X
                if quantity_to_plant > 0:
                    total_to_plant_by_year[t] += quantity_to_plant
                    grade_to_plant_by_year[t] += block_model_df.loc[i, 'grade'] * quantity_to_plant
                    quantity_to_plant_by_year[t] += quantity_to_plant
                    blocks_to_plant_by_year[t].add(i)
                    cost_per_unit, carbon_per_unit = haulage_cost_to_processing_plant(i, block_model_df, processing_plant_coords, haulage_paths, haulage_costs, carbon_costs, quantity_to_plant)
                    yearly_haulage_costs[t] += cost_per_unit * quantity_to_plant
                    yearly_haulage_costs_to_plant[t] += cost_per_unit * quantity_to_plant  # Accumulate separately
                    yearly_carbon_costs[t] += carbon_per_unit * quantity_to_plant
                    data_records.append({
                        'Block': i,
                        'Year': t,
                        'Path': path_key,
                        'To_Plant': quantity_to_plant,
```

```
                'Dump_ID': '',
                'To_Dump': 0,
                'Haulage_Cost': cost_per_unit,
                'Carbon_Cost': carbon_per_unit
            })

        for d in dumps:
            quantity_to_dump = to_waste_dump[i, d, path_key, t].X
            if quantity_to_dump > 0:
                total_to_dump_by_year[t] += quantity_to_dump
                blocks_to_dump_by_year[t][d].add(i)
                cost_per_dump, carbon_per_dump = total_haulage_cost_per_unit(i, block_model_df, d, waste_dump_df, haulage_paths, haulage_costs, carbon_costs, quantity_to_dump)
                yearly_haulage_costs[t] += cost_per_dump * quantity_to_dump
                yearly_haulage_costs_to_dump[t] += cost_per_dump * quantity_to_dump  # Accumulate separately
                yearly_carbon_costs[t] += carbon_per_dump * quantity_to_dump
                data_records.append({
                    'Block': i,
                    'Year': t,
                    'Path': path_key,
                    'To_Plant': 0,
                    'Dump_ID': d,
                    'To_Dump': quantity_to_dump,
                    'Haulage_Cost': cost_per_dump,
                    'Carbon_Cost': carbon_per_dump
                })
                if is_full[d, t].X > 0.5:
                    full_dumps_by_year[t].add(d)
    # Compute average grade for blocks sent to the plant each year
    average_grade_to_plant_by_year = {t: (grade_to_plant_by_year[t] / quantity_to_plant_by_year[t]) if quantity_to_plant_by_year[t] > 0 else None for t in range(1, T+1)}
```

```
# Output the summaries
print("\nSummary of total material sent to processing plants and waste dumps by year:")
print("Total material sent to the processing plant by year:")
for year, total in total_to_plant_by_year.items():
    print(f"Year {year}: {total} units")
    print(f"Blocks sent to plant: {sorted(blocks_to_plant_by_year[year])}")

print("\nTotal material sent to waste dumps by year:")
for year, total in total_to_dump_by_year.items():
    print(f"Year {year}: {total} units")
    for d in dumps:
        if blocks_to_dump_by_year[year][d]:
            print(f"  Dump {d}: {sorted(blocks_to_dump_by_year[year][d])}")

print("\nAverage Grade of Blocks Sent to the Processing Plant by Year:")
for year, avg_grade in average_grade_to_plant_by_year.items():
    print(f"Year {year}: Average Grade = {avg_grade if avg_grade is not None else 'No Processing'}")

print("\nWaste dump locations filled by year:")
for year, dumps_set in full_dumps_by_year.items():
    print(f"Year {year}: Dump Locations {sorted(dumps_set)}")

print("\nMined blocks by year:")
for year, blocks_set in mined_blocks_by_year.items():
    print(f"Year {year}: Mined Blocks {sorted(blocks_set)}")

# Output the separate haulage costs
print("\nAnnual Haulage Costs to Processing Plant:")
for year in range(1, T+1):
    print(f"Year {year}: {yearly_haulage_costs_to_plant[year]}")
```

```
    print("\nAnnual Haulage Costs to Waste Dumps:")
    for year in range(1, T+1):
        print(f"Year {year}: {yearly_haulage_costs_to_dump[year]}")

    print("\nTotal Haulage Costs Over All Years:")
    print(f"Total Haulage Cost to Processing Plant: {sum(yearly_haulage_costs_to_plant.values())}")
    print(f"Total Haulage Cost to Waste Dumps: {sum(yearly_haulage_costs_to_dump.values())}")

    print("\nAnnual and Total Carbon Costs:")
    for year in range(1, T+1):
        print(f"Year {year} - Carbon Cost: {yearly_carbon_costs[year]}")
    print(f"Total 5-Year Carbon Cost: {sum(yearly_carbon_costs.values())}")

    # Save to CSV
    model_output_df = pd.DataFrame(data_records)
    model_output_df.to_csv('model_output_all_blocks1.csv', index=False)
    print("Data for all blocks has been saved to 'model_output_all_blocks1.csv'.")
else:
    print("Failed to find an optimal solution. Status code:", m.status)
```

APPENDIX B

Carbon Cost Model using python

```
import pandas as pd

import math

from gurobipy import Model, GRB, quicksum

import numpy as np


# Load the new haulage path data files


block_to_pit_exit1_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\block to pit exit 1.csv")

block_to_pit_exit2_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\block to pit exit 2.csv")

pit_exit1_to_waste_dump_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\pit exit 1 to waste dump entry.csv")

pit_exit2_to_waste_dump_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\pit exit 2 to waste dump entry.csv")

def calculate_haulage_distance(df):
    # Convert DataFrame to numpy array if not already
    if isinstance(df, pd.DataFrame):
        points = df[['X', 'Y', 'Z']].to_numpy()
    elif isinstance(df, np.ndarray):
        points = df
    else:
        raise ValueError("Input must be a pandas DataFrame or a numpy array")


    # Check if there are enough points to calculate distances
    if points.size == 0 or len(points) < 2:
        return 0


    return sum(math.sqrt((x2 - x1)**2 + (y2 - y1)**2 + (z2 - z1)**2)
               for (x1, y1, z1), (x2, y2, z2) in zip(points[:-1], points[1:]))
```

```
distance_block_to_pit_exit1 = calculate_haulage_distance(block_to_pit_exit1_df)
distance_block_to_pit_exit2 = calculate_haulage_distance(block_to_pit_exit2_df)
# Calculate distances from each pit exit to the processing plant
distance_pit_exit1_to_waste_dump = calculate_haulage_distance(pit_exit1_to_waste_dump_df)
distance_pit_exit2_to_waste_dump = calculate_haulage_distance(pit_exit2_to_waste_dump_df)
print(distance_block_to_pit_exit1)
haulage_paths = {
    'block_to_pit_exit1': distance_block_to_pit_exit1,
    'block_to_pit_exit2': distance_block_to_pit_exit2,
    'pit_exit1_to_waste_dump': distance_pit_exit1_to_waste_dump,
    'pit_exit2_to_waste_dump': distance_pit_exit2_to_waste_dump
}
#block model and precedence data
block_model_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\VALENS 7.csv")
waste_dump_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\waste dump 26092024.csv")
waste_precedence_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\waste_precedence1.csv")


# Set 'DumpLocationID' as the DataFrame index and ensure it's integer
waste_dump_df.set_index('DumpLocationID', inplace=True)
waste_dump_df.index = waste_dump_df.index.astype(int)  # Ensure indices are integers


# Create a dictionary for waste dump capacities with integer keys
waste_dump_capacities = {int(k): v for k, v in waste_dump_df['volume'].to_dict().items()}


# Create a dictionary for each dump mapping to its predecessors
waste_dump_precedence_dict = {int(dump): [] for dump in waste_dump_df.index.unique()}  # Ensure keys are integers


for index, row in waste_precedence_df.iterrows():
    # Convert 'DumpLocationID' and 'PrecedentDumpLocationID' to integers
```

```
    dump_id = int(row['DumpLocationID'])
    precedent_id = int(row['PrecedentDumpLocationID'])

    if dump_id in waste_dump_precedence_dict:  # Ensure the reference is correct
        waste_dump_precedence_dict[dump_id].append(precedent_id)

# Verify that dumps without predecessors are correctly identified
print("Waste Dump Precedence Dictionary:")
for dump_id, predecessors in waste_dump_precedence_dict.items():
    print(f"Dump {dump_id}: Predecessors - {predecessors}")

# List of dumps without predecessors
dumps_without_predecessors = [d for d, preds in waste_dump_precedence_dict.items() if not preds]
print("Dumps without predecessors:", dumps_without_predecessors)

# Set 'blockID' as the DataFrame index and ensure it's integer
block_model_df.set_index('blockID', inplace=True)
block_model_df.index = block_model_df.index.astype(int)  # Ensure indices are integers

# Create lists of blocks and dumps as integers
blocks = [int(i) for i in block_model_df.index.tolist()]
dumps = [int(d) for d in waste_dump_df.index.tolist()]

# Print to verify data types and keys (print only first 5 keys)
print("Sample keys in waste_dump_capacities:")
for key in list(waste_dump_capacities.keys())[:5]:
    print(f"Key: {key}, Type: {type(key)}")

print("\nSample dumps list:")
for dump in dumps[:5]:
    print(f"Dump: {dump}, Type: {type(dump)}")
```

```
waste_dump_coords = {}
for dump_id, row in waste_dump_df.iterrows():
    waste_dump_coords[f'waste_dump_{dump_id}'] = (row['X'], row['Y'], row['Z'])

# Define the Euclidean distance function
def euclidean_distance(point1, point2):
    return math.sqrt(sum((p1 - p2) ** 2 for p1, p2 in zip(point1, point2)))
# Haulage cost multipliers
haulage_costs = {
    'block_to_pit': 0.042,
    'pit_to_entry': 0.017,
    'entry_to_dump_lower': 0.042,
    'entry_to_dump_middle': 0.068,
    'entry_to_dump_upper': 0.093,
}

carbon_costs = {
    'block_to_pit': 0.006,
    'pit_to_entry': 0.0015,
    'entry_to_dump_lower': 0.003,
    'entry_to_dump_middle': 0.006,
    'entry_to_dump_upper': 0.0083,
}

def closest_point_distance(block_coords, path_df):
    distances = path_df.apply(lambda row: euclidean_distance(block_coords, row[['X', 'Y', 'Z']].values), axis=1)
    min_index = distances.idxmin()
    min_distance = distances.loc[min_index]
    closest_point_coords = path_df.loc[min_index, ['X', 'Y', 'Z']].values
    return min_distance, closest_point_coords
```

```
def total_haulage_cost_per_unit(block_ID, block_model_df, DumpLocationID, waste_dump_df,
                                haulage_paths, haulage_costs, carbon_costs, path_key):
    block_coords = block_model_df.loc[block_ID, ['X', 'Y', 'Z']].values
    quantity = block_model_df.loc[block_ID, 'quantity']

    if DumpLocationID not in waste_dump_df.index:
        raise ValueError(f"DumpLocationID {DumpLocationID} not found in waste_dump_df.")

    waste_dump_coords = waste_dump_df.loc[DumpLocationID, ['X', 'Y', 'Z']].values
    waste_dump_z = waste_dump_df.loc[DumpLocationID, 'Z']

    if waste_dump_z <= 550:
        dump_level_cost_key = 'entry_to_dump_lower'
    elif waste_dump_z <= 570:
        dump_level_cost_key = 'entry_to_dump_middle'
    else:
        dump_level_cost_key = 'entry_to_dump_upper'

    if path_key == 'exit1':
        path_components = (block_to_pit_exit1_df, pit_exit1_to_waste_dump_df)
        distance_block_to_pit = haulage_paths['block_to_pit_exit1']
        distance_pit_exit_to_waste_dump = haulage_paths['pit_exit1_to_waste_dump']
    else:
        path_components = (block_to_pit_exit2_df, pit_exit2_to_waste_dump_df)
        distance_block_to_pit = haulage_paths['block_to_pit_exit2']
        distance_pit_exit_to_waste_dump = haulage_paths['pit_exit2_to_waste_dump']

    distance_to_closest_point, _ = closest_point_distance(block_coords, path_components[0])
    last_point_to_dump_distance = euclidean_distance(path_components[1].iloc[-1][['X', 'Y',
'Z']].values, waste_dump_coords)
```

```
    # Haulage cost calculation per unit
    haulage_cost_per_unit = (
        (distance_to_closest_point + distance_block_to_pit) * haulage_costs['block_to_pit'] +
        distance_pit_exit_to_waste_dump * haulage_costs['pit_to_entry'] +
        last_point_to_dump_distance * haulage_costs[dump_level_cost_key]
    )

    # Carbon cost calculation per unit
    carbon_cost_per_unit = (
        (distance_to_closest_point + distance_block_to_pit) * carbon_costs['block_to_pit'] +
        distance_pit_exit_to_waste_dump * carbon_costs['pit_to_entry'] +
        last_point_to_dump_distance * carbon_costs[dump_level_cost_key]
    )

    return haulage_cost_per_unit, carbon_cost_per_unit

def calculate_cost_per_unit(block_ID, path_key, DumpLocationID,
                            block_model_df, waste_dump_df,
                            haulage_paths, haulage_costs, carbon_costs):
    block_row = block_model_df.loc[block_ID]
    quantity = block_row['quantity']

    # Unpack both haulage cost and carbon cost with matching variable names
    haulage_cost_per_unit, carbon_cost_per_unit = total_haulage_cost_per_unit(
        block_ID, block_model_df, DumpLocationID, waste_dump_df,
        haulage_paths, haulage_costs, carbon_costs, path_key
    )

    # Include both haulage and carbon costs in the total operational cost
    total_operational_cost_per_unit = haulage_cost_per_unit + carbon_cost_per_unit
```

```
    return total_operational_cost_per_unit

# Create a new model
m = Model("minimize_carbon_cost")

# Parameters
T = 5  # Time periods (years)
Swell_factor = 1.25
Density = 2.75

# Total waste sent to dumps per year from the first model
total_waste_to_dumps_by_year = {
    1: 4312000.0,
    2: 3430000.0,
    3: 1959999.99995,
    4: 1078000.00005,
    5: 588000.0
}
# Decision Variables
to_waste_dump = m.addVars(blocks, dumps, ['exit1', 'exit2'], range(1, T+1),
                          vtype=GRB.CONTINUOUS, name="to_waste_dump", lb=0)

# Variables to indicate allocation period for each block
y = m.addVars(blocks, range(1, T+1), vtype=GRB.BINARY, name="y")

# Objective Function: Minimize the operational cost (haulage + carbon)
m.setObjective(
    quicksum(
        to_waste_dump[i, d, path, t] * calculate_cost_per_unit(
            i, path, d, block_model_df, waste_dump_df,
            haulage_paths, haulage_costs, carbon_costs
        ) for i in blocks for d in dumps for path in ['exit1', 'exit2'] for t in range(1, T+1)
```

```
    ), GRB.MINIMIZE
)

# Each block must be allocated in exactly one period
for i in blocks:
    m.addConstr(
        quicksum(y[i, t] for t in range(1, T+1)) == 1,
        name=f"block_mined_once_{i}"
    )

# Allocation Constraints
for i in blocks:
    for t in range(1, T+1):
        m.addConstr(
            quicksum(to_waste_dump[i, d, path, t] for d in dumps for path in ['exit1', 'exit2']) == block_model_df.loc[i, 'quantity'] * y[i, t],
            name=f"allocation_{i}_{t}"
        )

# Waste dump precedence constraints
epsilon = 1e-6  # Small positive number to enforce activation when waste is sent

for d in dumps:
    for t in range(1, T+1):
        # Cumulative waste calculation
        for d in dumps:
            for t in range(1, T+1):
        # Cumulative waste calculation updated to new path keys
        cumulative_waste = quicksum((swell_factor [i] / density [i] ) * to_waste_dump[i, d, path, tp]
                                    for i in blocks for path in ['exit1', 'exit2']
                                    for tp in range(1, t+1))
```

```
        # Capacity constraints
        m.addConstr(cumulative_waste <= waste_dump_capacities[d], name=f"capacity_{d}_{t}")

        # Fullness indicator constraints
        m.addGenConstrIndicator(is_full[d, t], True, cumulative_waste >= waste_dump_capacities[d] - epsilon, name=f"full_{d}_{t}")
        m.addGenConstrIndicator(is_full[d, t], False, cumulative_waste <= waste_dump_capacities[d] - epsilon, name=f"not_full_{d}_{t}")

        # Activation constraints based on predecessors being full
        if t > 1:
            if waste_dump_precedence_dict[d]:  # Check if there are predecessors
                all_preds_full = m.addVar(vtype=GRB.BINARY, name=f"all_preds_full_{d}_{t}")
                dump_active = m.addVars(dumps, range(1, T+1), vtype=GRB.BINARY, name="dump_active")
                is_full = m.addVars(dumps, range(1, T+1), vtype=GRB.BINARY, name="is_full")
                m.addGenConstrAnd(all_preds_full, [is_full[pred, t-1] for pred in waste_dump_precedence_dict[d]])
                m.addConstr(dump_active[d, t] == all_preds_full, name=f"activate_{d}_{t}")
            else:
                m.addConstr(dump_active[d, t] == 1, name=f"activate_{d}_{t}_no_predecessors")
        else:
            if not waste_dump_precedence_dict[d]:
                m.addConstr(dump_active[d, t] == 1, name=f"activate_{d}_{t}_initial")
            else:
                m.addConstr(dump_active[d, t] == 0, name=f"activate_{d}_{t}_initial")

# Constraints linking waste disposal to active dumps
for i in blocks:
    for d in dumps:
        for path in ['exit1', 'exit2']:
            for t in range(1, T+1):
                m.addConstr(
```

```
                to_waste_dump[i, d, path, t] <= dump_active[d, t] * block_model_df.loc[i, 'quantity'],
                name=f"link_waste_{i}_{d}_{path}_{t}"
            )

# Ensure that waste is only sent to active dumps
for d in dumps:
    for t in range(1, T+1):
        m.addConstr(
            quicksum(to_waste_dump[i, d, path, t] for i in blocks for path in ['exit1', 'exit2']) >= epsilon
* dump_active[d, t],
            name=f"waste_sent_if_active_{d}_{t}"
        )

# Waste generated constraints per period
for t in range(1, T+1):
    m.addConstr(
        quicksum(
            block_model_df.loc[i, 'quantity'] * y[i, t]
            for i in blocks
        ) <= total_waste_to_dumps_by_year[t],
        name=f"waste_generated_limit_{t}"
    )

# Solve the model
m.setParam('MIPGap', 0.1)
m.optimize()

# Check if the model found an optimal or suboptimal solution
if m.status == GRB.OPTIMAL or m.status == GRB.SUBOPTIMAL:
    print("Optimal or suboptimal solution found:\n")

    # Initialize dictionaries for analysis
```

```
    total_material_sent = 0  # Total material sent to waste dumps over all periods
    total_to_dump_by_year = {t: 0 for t in range(1, T+1)}  # Total material sent to waste dumps each year
    dumps_used_by_year = {t: set() for t in range(1, T+1)}  # Dumps that received material each year
    dumps_full_time = {}  # The period when each dump becomes full
    yearly_haulage_costs = {t: 0 for t in range(1, T+1)}
    yearly_carbon_costs = {t: 0 for t in range(1, T+1)}
    data_records = []

    # Iterate over dumps to determine when they become full and collect data
    for d in dumps:
        cumulative_waste = 0
        dump_full = False
        for t in range(1, T+1):
            period_waste = sum(
                to_waste_dump[i, d, path, t].X
                for i in blocks for path in ['exit1', 'exit2']
                if to_waste_dump[i, d, path, t].X > 0
            )
            if period_waste > 0:
                dumps_used_by_year[t].add(d)
                total_to_dump_by_year[t] += period_waste
                total_material_sent += period_waste

                # Collect cost data
                for i in blocks:
                    for path in ['exit1', 'exit2']:
                        quantity_to_dump = to_waste_dump[i, d, path, t].X
                        if quantity_to_dump > 0:
                            # Use quantity_to_dump as the quantity parameter
                            haulage_cost_per_unit, carbon_cost_per_unit = total_haulage_cost_per_unit(
                                i, block_model_df, d, waste_dump_df,
```

```
                    haulage_paths, haulage_costs, carbon_costs,
                    path
                )
                haulage_cost = haulage_cost_per_unit * quantity_to_dump
                carbon_cost = carbon_cost_per_unit * quantity_to_dump
                total_operational_cost = haulage_cost + carbon_cost

                yearly_haulage_costs[t] += haulage_cost
                yearly_carbon_costs[t] += carbon_cost

                data_records.append({
                    'Block': i,
                    'Year': t,
                    'Path': path,
                    'Dump_ID': d,
                    'To_Dump': quantity_to_dump,
                    'Haulage_Cost': haulage_cost,
                    'Carbon_Cost': carbon_cost,
                    'Total_Operational_Cost': total_operational_cost
                })

        cumulative_waste += period_waste

        # Check if dump becomes full in this period
        if not dump_full and cumulative_waste >= waste_dump_capacities[d] - 1e-6:
            dumps_full_time[d] = t
            dump_full = True  # Mark that the dump is full

# Output the summaries
print("\nTotal material sent to waste dumps over all periods: {} units".format(total_material_sent))
```

```
    print("\nTotal material sent to waste dumps by year:")
    for t in range(1, T+1):
        print(f"Year {t}: {total_to_dump_by_year[t]} units")

    print("\nWaste dump locations filled by year:")
    for t in range(1, T+1):
        dumps_filled_this_year = [d for d, t_full in dumps_full_time.items() if t_full == t]
        print(f"Year {t}: Dump Locations {sorted(dumps_filled_this_year)}")

    print("\nAnnual and Total Haulage and Carbon Costs:")
    for t in range(1, T+1):
        print(f"Year {t} - Haulage Cost: {yearly_haulage_costs[t]}, Carbon Cost: {yearly_carbon_costs[t]}")

    total_haulage_cost = sum(yearly_haulage_costs.values())
    total_carbon_cost = sum(yearly_carbon_costs.values())
    print(f"Total Haulage Cost over {T} years: {total_haulage_cost}")
    print(f"Total Carbon Cost over {T} years: {total_carbon_cost}")

    # Save detailed data to CSV
    model_output_df = pd.DataFrame(data_records)
    model_output_df.to_csv('model_output_all_blocks.csv', index=False)
    print("Data for all blocks has been saved to 'model_output_all_blocks.csv'.")

elif m.status == GRB.INFEASIBLE:
    print("The model is infeasible; computing IIS")
    m.computeIIS()
    m.write("model.ilp")
    print("IIS written to file 'model.ilp'")
else:
    print("Failed to find an optimal solution. Status code:", m.status)
```

APPENDIX C

Combined Model using python

```
import pandas as pd

import math

from gurobipy import Model, GRB, quicksum

from gurobipy import min_

import numpy as np


# Load the haulage path data files

pit_exit1_to_processing_plant_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\Processing road 1.csv")

pit_exit2_to_processing_plant_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktoptest 8\Processing road 2.csv")

block_to_pit_exit1_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\block to pit exit 1.csv")

block_to_pit_exit2_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\block to pit exit 2.csv")

pit_exit1_to_waste_dump_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\pit exit 1 to waste dump entry.csv")

pit_exit2_to_waste_dump_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\pit exit 2 to waste dump entry.csv")



def calculate_haulage_distance(df):
    # Convert DataFrame to numpy array if not already
    if isinstance(df, pd.DataFrame):
        points = df[['X', 'Y', 'Z']].to_numpy()
    elif isinstance(df, np.ndarray):
        points = df
    else:
        raise ValueError("Input must be a pandas DataFrame or a numpy array")


    # Check if there are enough points to calculate distances
    if points.size == 0 or len(points) < 2:
```

```
        return 0

    return sum(math.sqrt((x2 - x1)**2 + (y2 - y1)**2 + (z2 - z1)**2)
               for (x1, y1, z1), (x2, y2, z2) in zip(points[:-1], points[1:]))

distance_block_to_pit_exit1 = calculate_haulage_distance(block_to_pit_exit1_df)
distance_block_to_pit_exit2 = calculate_haulage_distance(block_to_pit_exit2_df)
distance_pit_exit1_to_processing_plant = calculate_haulage_distance(pit_exit1_to_processing_plant_df[['X', 'Y', 'Z']].values)
distance_pit_exit2_to_processing_plant = calculate_haulage_distance(pit_exit2_to_processing_plant_df[['X', 'Y', 'Z']].values)
distance_pit_exit1_to_waste_dump = calculate_haulage_distance(pit_exit1_to_waste_dump_df)
distance_pit_exit2_to_waste_dump = calculate_haulage_distance(pit_exit2_to_waste_dump_df)

haulage_paths = {
    'block_to_pit_exit1': distance_block_to_pit_exit1,
    'block_to_pit_exit2': distance_block_to_pit_exit2,
    'pit_exit1_to_processing_plant': distance_pit_exit1_to_processing_plant,
    'pit_exit2_to_processing_plant': distance_pit_exit2_to_processing_plant,
    'pit_exit1_to_waste_dump': distance_pit_exit1_to_waste_dump,
    'pit_exit2_to_waste_dump': distance_pit_exit2_to_waste_dump
}
#block model and precedence data
block_model_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\VALENS 6.csv")
precedence_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\block_precedence2.csv")
waste_dump_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\waste dump 26092024.csv")
waste_precedence_df = pd.read_csv(r"C:\Users\Alienware\OneDrive - Curtin University of Technology Australia\Desktop \test 8\waste_precedence1.csv")
```

```
# Create a dictionary for each dump mapping to its predecessors
waste_dump_precedence_dict = {dump: [] for dump in waste_dump_df['DumpLocationID'].unique()}  # Initialize all dumps with an empty list
for index, row in waste_precedence_df.iterrows():
    dump_id = row['DumpLocationID']
    precedent_id = row['PrecedentDumpLocationID']
    if dump_id in waste_dump_precedence_dict:
        waste_dump_precedence_dict[dump_id].append(precedent_id)

# Precedence dictionary for mining blocks
precedences_dict = {}
for index, row in precedence_df.iterrows():
    block_id = row['blockID']
    precedent_id = row['precedenceID']
    if block_id not in precedences_dict:
        precedences_dict[block_id] = []
    precedences_dict[block_id].append(precedent_id)

# Create a dictionary for waste dump capacities for easy access
waste_dump_capacities = pd.Series(waste_dump_df.volume.values, index=waste_dump_df.DumpLocationID).to_dict()

# Set 'blockID' as the DataFrame index if it's not already
block_model_df.set_index('blockID', inplace=True)
waste_dump_df.set_index('DumpLocationID', inplace=True)

# preprocessing to create the model parameters
blocks = block_model_df.index.tolist()
dumps = waste_dump_df.index.tolist()
processing_plant_coords = {
    'processing_plant': (4592.805, 3430.879, 510.000)
}
```

```
# Initialize the dictionary for waste dump locations after setting 'DumpLocationID' as index
waste_dump_coords = {}
for dump_id, row in waste_dump_df.iterrows():
    waste_dump_coords[f'waste_dump_{dump_id}'] = (row['X'], row['Y'], row['Z'])

# Parameters
def calc_gold_price(t):
    # Assuming gold price increases by 2% every year starting from a base price of 1500
    base_price = 200
    annual_growth_rate = 0.05
    return base_price * (1 + annual_growth_rate) ** (t-1)

def mining_capacity(t):
    # Assuming mining capacity is fixed at 100000 tons per year
    base_capacity = 6000000
    final_capacity = 7000000
    return base_capacity + ((final_capacity - base_capacity) / (T-1)) * (t-1)

def processing_capacity(t):
    # Assuming processing capacity increases from 30000 to 50000 tons over 5 years linearly
    base_capacity = 3000000
    final_capacity = 4000000
    return base_capacity + ((final_capacity - base_capacity) / (T-1)) * (t-1)

MC = MC = dict(zip(block_model_df.index, block_model_df['mining_cost']))

T = 5  # Time periods (years)

r = 0.1  # Discount rate
```

```
MCap_t = {t: mining_capacity(t) for t in range(1, T+1)}


PCap_t = {t: processing_capacity(t) for t in range(1, T+1)}
Swell_factor = 1.25
Density = 2.75


# Define the Euclidean distance function
def euclidean_distance(point1, point2):
    return math.sqrt(sum((p1 - p2) ** 2 for p1, p2 in zip(point1, point2)))


# Function to extract haulage paths from the Excel data
def extract_haulage_paths(haulage_data_df):
    haulage_paths = {}
    for sheet_name in haulage_data_df.sheet_names:
        df = pd.read_excel(haulage_data_df, sheet_name=sheet_name)
        points = df.sort_values(by='PointID')[['X', 'Y', 'Z']].values
        # Calculate path distance
        path_distance = sum(euclidean_distance(points[i], points[i + 1]) for i in range(len(points) - 1))
        # Store path info
        haulage_paths[sheet_name] = {'points': points, 'path_distance': path_distance}
    return haulage_paths


# Haulage cost multipliers for different segments
haulage_costs = {
    'block_to_pit': 0.042,   # Cost per unit distance from block to pit
    'pit_to_entry': 0.017,  # Cost per unit distance from pit to waste dump entry
    'entry_to_dump_lower': 0.042,  # Cost per unit distance from waste dump entry to the lower dump level
    'entry_to_dump_middle': 0.068, # Cost per unit distance from waste dump entry to the middle dump level
    'entry_to_dump_upper': 0.093,  # Cost per unit distance from waste dump entry to the upper dump level
    'pit_exit1_to_plant': 0.017,  # Cost per unit distance from pit exit 1 to processing plant
```

```
    'pit_exit2_to_plant': 0.017  # Cost per unit distance from pit exit 2 to processing plant
}
carbon_costs = {
    'block_to_pit': 0.006,  # Carbon cost per tonne-meter for block to pit
    'pit_to_entry': 0.0015,  # Carbon cost per tonne-meter for pit to waste dump entry
    'entry_to_dump_lower': 0.003,  # Carbon cost per tonne-meter for lower dump level
    'entry_to_dump_middle': 0.006, # Carbon cost per tonne-meter for middle dump level
    'entry_to_dump_upper': 0.0083,  # Cost per unit distance from waste dump entry to the upper dump level
    'pit_exit1_to_plant': 0.0015,   # Carbon cost per tonne-meter for pit exit 2 to processing plant
    'pit_exit2_to_plant': 0.0015   # Carbon cost per tonne-meter for pit exit 2 to processing plant
}

def closest_point_distance(block_coords, path_df):
    # Calculate distances from block to all points in the DataFrame
    distances = path_df.apply(lambda row: euclidean_distance(block_coords, row[['X', 'Y', 'Z']].values), axis=1)
    min_index = distances.idxmin()
    min_distance = distances.loc[min_index]
    closest_point_coords = path_df.loc[min_index, ['X', 'Y', 'Z']].values
    return min_distance, closest_point_coords

def haulage_cost_to_processing_plant(
    block_ID, block_model_df, processing_plant_coords,
    haulage_paths, haulage_costs, carbon_costs, quantity):

    block_coords = block_model_df.loc[block_ID, ['X', 'Y', 'Z']].values
    destination_key = 'processing_plant'
    if destination_key not in processing_plant_coords:
        raise ValueError("Processing plant coordinates not found in destination coordinates.")

    # Access pre-calculated distances from the haulage_paths dictionary
```

```
    distance_block_to_pit_exit1 = haulage_paths['block_to_pit_exit1']
    distance_block_to_pit_exit2 = haulage_paths['block_to_pit_exit2']
    distance_pit_exit1_to_processing_plant = haulage_paths['pit_exit1_to_processing_plant']
    distance_pit_exit2_to_processing_plant = haulage_paths['pit_exit2_to_processing_plant']



    # Calculate distances from the block to the closest points of the pit exits
    distance_to_closest_point_exit1, closest_point_exit1 = closest_point_distance(block_coords, block_to_pit_exit1_df)
    distance_to_closest_point_exit2, closest_point_exit2 = closest_point_distance(block_coords, block_to_pit_exit2_df)


    # Calculate total distances for the paths from the block to the processing plant via each pit exit
    total_distance_via_exit1 = distance_to_closest_point_exit1 + distance_block_to_pit_exit1 + distance_pit_exit1_to_processing_plant
    total_distance_via_exit2 = distance_to_closest_point_exit2 + distance_block_to_pit_exit2 + distance_pit_exit2_to_processing_plant


    # Calculate haulage costs for each route (excluding carbon costs)
    haulage_cost_via_exit1 = (
        (distance_to_closest_point_exit1 + distance_block_to_pit_exit1) * haulage_costs['block_to_pit'] +
        distance_pit_exit1_to_processing_plant * haulage_costs['pit_exit1_to_plant']
    ) / quantity


    haulage_cost_via_exit2 = (
        (distance_to_closest_point_exit2 + distance_block_to_pit_exit2) * haulage_costs['block_to_pit'] +
        distance_pit_exit2_to_processing_plant * haulage_costs['pit_exit2_to_plant']
    ) / quantity


    # Calculate carbon costs for each route
    carbon_cost_via_exit1 = (
```

```
        (distance_to_closest_point_exit1 + distance_block_to_pit_exit1) * carbon_costs['block_to_pit']
+
        distance_pit_exit1_to_processing_plant * carbon_costs['pit_exit1_to_plant']
    ) / quantity

    carbon_cost_via_exit2 = (
        (distance_to_closest_point_exit2 + distance_block_to_pit_exit2) * carbon_costs['block_to_pit']
+
        distance_pit_exit2_to_processing_plant * carbon_costs['pit_exit2_to_plant']
    ) / quantity

    # Aggregate haulage and carbon costs for each route
    total_costs_per_unit = {
        'exit1': (haulage_cost_via_exit1, carbon_cost_via_exit1),
        'exit2': (haulage_cost_via_exit2, carbon_cost_via_exit2)
    }

    # Select the minimum cost path based on the total of haulage and carbon costs
    selected_path_key = min(total_costs_per_unit, key=lambda x: sum(total_costs_per_unit[x]))

    # Return both the haulage cost and the carbon cost for the selected path
    return total_costs_per_unit[selected_path_key]



def closest_point_distance(block_coords, path_df):
    # Calculate distances from block to all points in the DataFrame
    distances = path_df.apply(lambda row: euclidean_distance(block_coords, row[['X', 'Y',
'Z']].values), axis=1)
    # Find the index of the minimum distance
    min_index = distances.idxmin()
    # Return the minimum distance and the coordinates of the closest point
    min_distance = distances.loc[min_index]
    closest_point_coords = path_df.loc[min_index, ['X', 'Y', 'Z']].values
```

```
    return min_distance, closest_point_coords

def total_haulage_cost_per_unit(block_ID, block_model_df, DumpLocationID, waste_dump_df,
                                haulage_paths, haulage_costs, carbon_costs, quantity):

    # Extract the block coordinates from the block model DataFrame
    block_coords = block_model_df.loc[block_ID, ['X', 'Y', 'Z']].values



    if DumpLocationID not in waste_dump_df.index:
        raise ValueError(f"DumpLocationID {DumpLocationID} not found in waste_dump_ddddf.")

    waste_dump_coords = waste_dump_df.loc[DumpLocationID, ['X', 'Y', 'Z']].values
    waste_dump_z = waste_dump_df.loc[DumpLocationID, 'Z']

    # Determine dump level cost key based on elevation
    if waste_dump_z <= 550:
        dump_level_cost_key = 'entry_to_dump_lower'
    elif waste_dump_z <= 570:
        dump_level_cost_key = 'entry_to_dump_middle'  # New middle level
    else:
        dump_level_cost_key = 'entry_to_dump_upper'

    # Access pre-calculated distances from the haulage_paths dictionary
    distance_block_to_pit_exit1 = haulage_paths['block_to_pit_exit1']
    distance_block_to_pit_exit2 = haulage_paths['block_to_pit_exit2']
    distance_pit_exit1_to_waste_dump = haulage_paths['pit_exit1_to_waste_dump']
    distance_pit_exit2_to_waste_dump = haulage_paths['pit_exit2_to_waste_dump']

    total_costs = {}
    total_carbon = {}
```

```
    for exit_key, path_components in {
        'exit1': (block_to_pit_exit1_df, pit_exit1_to_waste_dump_df),
        'exit2': (block_to_pit_exit2_df, pit_exit2_to_waste_dump_df)
    }.items():
        # Calculate the closest points and distances for the components of the selected path
        distance_to_closest_point, closest_first_point = closest_point_distance(block_coords, path_components[0])
        # Calculate the distance from the last point on the pit exit path to the waste dump
        last_point_to_dump_distance = euclidean_distance(path_components[1].iloc[-1][['X', 'Y', 'Z']].values, waste_dump_coords)

        # Total distance for the haulage route includes segments from block to pit, pit to last path point, and last point to dump
        total_distance = (distance_to_closest_point + distance_block_to_pit_exit1 if exit_key == 'exit1' else distance_block_to_pit_exit2) \
                         + (distance_pit_exit1_to_waste_dump if exit_key == 'exit1' else distance_pit_exit2_to_waste_dump) \
                         + last_point_to_dump_distance

        # Calculate total haulage cost based on distances and costs for each segment
        total_cost = (
            (distance_to_closest_point + distance_block_to_pit_exit1 if exit_key == 'exit1' else distance_block_to_pit_exit2) * (haulage_costs['block_to_pit'] + carbon_costs['block_to_pit']) +
            (distance_pit_exit1_to_waste_dump if exit_key == 'exit1' else distance_pit_exit2_to_waste_dump) * (haulage_costs['pit_to_entry'] + carbon_costs['pit_to_entry']) +
            last_point_to_dump_distance * (haulage_costs[dump_level_cost_key] + carbon_costs[dump_level_cost_key])
        )

        # Carbon cost calculation for each segment
        total_carbon_emissions = (
            (distance_to_closest_point + distance_block_to_pit_exit1 if exit_key == 'exit1' else distance_block_to_pit_exit2) * carbon_costs['block_to_pit'] +
```

```
        (distance_pit_exit1_to_waste_dump if exit_key == 'exit1' else distance_pit_exit2_to_waste_dump) * carbon_costs['pit_to_entry'] +
        last_point_to_dump_distance * carbon_costs[dump_level_cost_key]
    )

    # Normalize the total cost and carbon emissions by the quantity to get the cost and carbon per unit
    total_costs_per_unit = total_cost / quantity
    total_carbon_per_unit = total_carbon_emissions / quantity

    total_costs[exit_key] = total_costs_per_unit
    total_carbon[exit_key] = total_carbon_per_unit

# Select the minimum cost path per unit and return both cost and carbon per unit
selected_path_key = min(total_costs, key=total_costs.get)
return total_costs[selected_path_key], total_carbon[selected_path_key]



def calculate_discounted_profit_per_unit(block_ID, path_key, DumpLocationID, destination, year,
                                         block_model_df, waste_dump_df,
                                         haulage_paths, haulage_costs, carbon_costs):
    block_row = block_model_df.loc[block_ID]
    grade = block_row['grade']
    mining_cost_per_unit = block_row['mining_cost']  # Assuming this is per unit
    discount_rate = 0.1
    quantity = block_row['quantity']

    # Calculate the cost including both haulage and carbon cost based on the destination
    if destination == "plant":
        # Unpack both haulage cost and carbon cost
        haulage_cost_per_unit, carbon_cost_per_unit = haulage_cost_to_processing_plant(
```

```
        block_ID, block_model_df, processing_plant_coords,
        haulage_paths,  # Only pass haulage_paths here
        haulage_costs, carbon_costs, quantity
    )
    processing_cost_per_unit = block_row.get('processing_cost_p1', 0)
    recovery_rate = block_row.get('recovery_rate_p1', 1)  # Default recovery rate for processing
else:
    # Unpack both haulage cost and carbon cost
    haulage_cost_per_unit, carbon_cost_per_unit = total_haulage_cost_per_unit(
        block_ID, block_model_df, DumpLocationID, waste_dump_df,
        haulage_paths,  # Only pass haulage_paths here
        haulage_costs, carbon_costs, quantity
    )
    processing_cost_per_unit = 0  # No processing cost for waste dump
    recovery_rate = 0  # No recovery as the material is treated as waste

# Include both haulage and carbon costs in the total operational cost
total_operational_cost_per_unit = (
    mining_cost_per_unit
    + processing_cost_per_unit
    + haulage_cost_per_unit
    + carbon_cost_per_unit  # Carbon cost is now part of the operational cost
)

# Calculate the revenue per unit based on grade and recovery rate
revenue_per_unit = grade * recovery_rate * calc_gold_price(year)

# Calculate the net value per unit
net_value_per_unit = revenue_per_unit - total_operational_cost_per_unit

# Apply the discount factor to get the discounted net value
```

```
    discounted_net_value_per_unit = net_value_per_unit / ((1 + discount_rate) ** (year - 1))

    return discounted_net_value_per_unit

# Create a new model
m = Model("maximize_npv")

# Decision Variables
# Decision Variables for routing to the processing plant and waste dumps
to_processing_plant = m.addVars(blocks, ['exit1', 'exit2'], range(1, T+1), vtype=GRB.CONTINUOUS, name="to_processing_plant", lb=0)
to_waste_dump = m.addVars(blocks, dumps, ['exit1', 'exit2'], range(1, T+1), vtype=GRB.CONTINUOUS, name="to_waste_dump", lb=0)
x = m.addVars(blocks, range(1, T+1), vtype=GRB.BINARY, name="x")
# Variables for dump activation and fullness
is_full = m.addVars(dumps, range(1, T+1), vtype=GRB.BINARY, name="is_full")


# Objective Function: Minimize the cost of transporting materials to the processing plant and waste dumps
m.setObjective(
    quicksum(
        to_processing_plant[i, path, t] * calculate_discounted_profit_per_unit(
            i, path, None, 'plant', t, block_model_df, waste_dump_df,
            haulage_paths, haulage_costs, carbon_costs
        ) for i in blocks for path in ['exit1', 'exit2'] for t in range(1, T+1)
    ) + quicksum(
        to_waste_dump[i, d, path, t] * calculate_discounted_profit_per_unit(
            i, path, d, 'dump', t, block_model_df, waste_dump_df,
            haulage_paths, haulage_costs, carbon_costs
        ) for i in blocks for d in dumps for path in ['exit1', 'exit2'] for t in range(1, T+1)
    ), GRB.MAXIMIZE
)
```

```
#Contraint
# Constraints to ensure the volume sent via each path does not exceed the total quantity mined
for i in blocks:
    for t in range(1, T+1):
        m.addConstr(
            quicksum(to_processing_plant[i, path, t] for path in ['exit1', 'exit2']) +
            quicksum(to_waste_dump[i, d, path, t] for d in dumps for path in ['exit1', 'exit2'])
             ≤ x[i, t] * block_model_df.loc[i, 'quantity'],
            name=f"allocation_{i}_{t}"
        )

# Mining Capacity Constraint
for t in range(1, T+1):
    m.addConstr(
        quicksum(quicksum(to_processing_plant[i, path, t] for path in ['exit1', 'exit2']) for i in blocks) +
        quicksum(quicksum(to_waste_dump[i, d, path, t] for path in ['exit1', 'exit2']) for i in blocks for
d in dumps)
        <= MCap_t[t],
        name=f"mining_cap_{t}"
    )
# Processing Capacity Constraint
    m.addConstr(
        quicksum(quicksum(to_processing_plant[i, path, t] for path in ['exit1', 'exit2']) for i in blocks)
        <= PCap_t[t],
        name=f"processing_capacity_{t}"
    )

# Precedence constraints for blocks
for block_id, predecessors in precedences_dict.items():
    for t in range(1, T+1):  # T is your total number of time periods
        for precedent_id in predecessors:
```

```
        m.addConstr(
            x[block_id, t] <= quicksum(x[precedent_id, tp] for tp in range(1, t)),
            name=f"precedence_{block_id}_{precedent_id}_{t}"
        )


# Waste dump precedence constraints
for d in dumps:
    for t in range(1, T+1):
        # Cumulative waste calculation updated to new path keys
        cumulative_waste = quicksum((swell_factor [i]/ density[i]) to_waste_dump[i, d, path, tp]
                                    for i in blocks for path in ['exit1', 'exit2']
                                    for tp in range(1, t+1))



        # Capacity and fullness constraints remain the same
        m.addConstr(cumulative_waste <= waste_dump_capacities[d], name=f"capacity_{d}_{t}")
        m.addGenConstrIndicator(is_full[d, t], True, cumulative_waste >= waste_dump_capacities[d], name=f"full_{d}_{t}")
        m.addGenConstrIndicator(is_full[d, t], False, cumulative_waste <= waste_dump_capacities[d] - 1e-5, name=f"not_full_{d}_{t}")


        # Refined activation based on predecessors being full
        if t > 1:
            if waste_dump_precedence_dict[d]:  # Check if there are predecessors
                all_preds_full = m.addVar(vtype=GRB.BINARY, name=f"all_preds_full_{d}_{t}")
              dump_active = m.addVars(dumps, range(1, T+1), vtype=GRB.BINARY, name="dump_active")
                m.addGenConstrAnd(all_preds_full, [is_full[pred, t-1] for pred in waste_dump_precedence_dict[d]])
                m.addConstr(dump_active[d, t] == all_preds_full, name=f"activate_{d}_{t}")
            else:
                m.addConstr(dump_active[d, t] == 1, name=f"activate_{d}_{t}_no_predecessors")
        else:
```

```
        m.addConstr(dump_active[d, t] == (1 if not waste_dump_precedence_dict[d] else 0), name=f"activate_{d}_{t}_initial")


# constraints linking waste disposal to Active dumps
for i in blocks:
    for d in dumps:
        for path in ['exit1', 'exit2']:  # Use the new path keys
            for t in range(1, T+1):
                m.addConstr(
                    to_waste_dump[i, d, path, t] <= dump_active[d, t] * block_model_df.loc[i, 'quantity'],
                    name=f"link_waste_{i}_{d}_{path}_{t}"
                )

# Solve the model
m.setParam('MIPGap', 0.1)

m.optimize()  # Optimize the model
if m.status == GRB.OPTIMAL or m.status == GRB.SUBOPTIMAL:
    print("Optimal or suboptimal solution found:\n")
    # Initialize dictionaries for analysis
    total_to_plant_by_year = {t: 0 for t in range(1, T+1)}
    total_to_dump_by_year = {t: 0 for t in range(1, T+1)}
    mined_blocks_by_year = {t: set() for t in range(1, T+1)}
    full_dumps_by_year = {t: set() for t in range(1, T+1)}
    grade_to_plant_by_year = {t: 0 for t in range(1, T+1)}
    quantity_to_plant_by_year = {t: 0 for t in range(1, T+1)}
    blocks_to_plant_by_year = {t: set() for t in range(1, T+1)}
    blocks_to_dump_by_year = {t: {} for t in range(1, T+1)}
    yearly_haulage_costs = {year: 0 for year in range(1, T+1)}
    yearly_carbon_costs = {year: 0 for year in range(1, T+1)}
    yearly_haulage_costs_to_plant = {year: 0 for year in range(1, T+1)}
    yearly_haulage_costs_to_dump = {year: 0 for year in range(1, T+1)}
```

```
    data_records = []
    total_haulage_cost = 0
    total_carbon_cost = 0

    for t in range(1, T+1):
        for d in dumps:
            blocks_to_dump_by_year[t][d] = set()

    # Iterate over all blocks and time periods
    for i in blocks:
        for t in range(1, T+1):
            if x[i, t].X > 0.5:
                mined_blocks_by_year[t].add(i)

            for path_key in ['exit1', 'exit2']:
                quantity_to_plant = to_processing_plant[i, path_key, t].X
                if quantity_to_plant > 0:
                    total_to_plant_by_year[t] += quantity_to_plant
                    grade_to_plant_by_year[t] += block_model_df.loc[i, 'grade'] * quantity_to_plant
                    quantity_to_plant_by_year[t] += quantity_to_plant
                    blocks_to_plant_by_year[t].add(i)
                    cost_per_unit, carbon_per_unit = haulage_cost_to_processing_plant(i, block_model_df, processing_plant_coords, haulage_paths, haulage_costs, carbon_costs, quantity_to_plant)
                    yearly_haulage_costs[t] += cost_per_unit * quantity_to_plant
                    yearly_haulage_costs_to_plant[t] += cost_per_unit * quantity_to_plant  # Accumulate separately
                    yearly_carbon_costs[t] += carbon_per_unit * quantity_to_plant
                    data_records.append({
                        'Block': i,
                        'Year': t,
                        'Path': path_key,
                        'To_Plant': quantity_to_plant,
```

```
                'Dump_ID': '',
                'To_Dump': 0,
                'Haulage_Cost': cost_per_unit,
                'Carbon_Cost': carbon_per_unit
            })

        for d in dumps:
            quantity_to_dump = to_waste_dump[i, d, path_key, t].X
            if quantity_to_dump > 0:
                total_to_dump_by_year[t] += quantity_to_dump
                blocks_to_dump_by_year[t][d].add(i)
                cost_per_dump, carbon_per_dump = total_haulage_cost_per_unit(i, block_model_df, d, waste_dump_df, haulage_paths, haulage_costs, carbon_costs, quantity_to_dump)
                yearly_haulage_costs[t] += cost_per_dump * quantity_to_dump
                yearly_haulage_costs_to_dump[t] += cost_per_dump * quantity_to_dump  # Accumulate separately
                yearly_carbon_costs[t] += carbon_per_dump * quantity_to_dump
                data_records.append({
                    'Block': i,
                    'Year': t,
                    'Path': path_key,
                    'To_Plant': 0,
                    'Dump_ID': d,
                    'To_Dump': quantity_to_dump,
                    'Haulage_Cost': cost_per_dump,
                    'Carbon_Cost': carbon_per_dump
                })
                if is_full[d, t].X > 0.5:
                    full_dumps_by_year[t].add(d)
    # Compute average grade for blocks sent to the plant each year
    average_grade_to_plant_by_year = {t: (grade_to_plant_by_year[t] / quantity_to_plant_by_year[t]) if quantity_to_plant_by_year[t] > 0 else None for t in range(1, T+1)}
```

```
# Output the summaries
print("\nSummary of total material sent to processing plants and waste dumps by year:")
print("Total material sent to the processing plant by year:")
for year, total in total_to_plant_by_year.items():
    print(f"Year {year}: {total} units")
    print(f"Blocks sent to plant: {sorted(blocks_to_plant_by_year[year])}")

print("\nTotal material sent to waste dumps by year:")
for year, total in total_to_dump_by_year.items():
    print(f"Year {year}: {total} units")
    for d in dumps:
        if blocks_to_dump_by_year[year][d]:
            print(f"  Dump {d}: {sorted(blocks_to_dump_by_year[year][d])}")

print("\nAverage Grade of Blocks Sent to the Processing Plant by Year:")
for year, avg_grade in average_grade_to_plant_by_year.items():
    print(f"Year {year}: Average Grade = {avg_grade if avg_grade is not None else 'No Processing'}")

print("\nWaste dump locations filled by year:")
for year, dumps_set in full_dumps_by_year.items():
    print(f"Year {year}: Dump Locations {sorted(dumps_set)}")

print("\nMined blocks by year:")
for year, blocks_set in mined_blocks_by_year.items():
    print(f"Year {year}: Mined Blocks {sorted(blocks_set)}")

# Output the separate haulage costs
print("\nAnnual Haulage Costs to Processing Plant:")
for year in range(1, T+1):
    print(f"Year {year}: {yearly_haulage_costs_to_plant[year]}")
```

```
    print("\nAnnual Haulage Costs to Waste Dumps:")
    for year in range(1, T+1):
        print(f"Year {year}: {yearly_haulage_costs_to_dump[year]}")

    print("\nTotal Haulage Costs Over All Years:")
    print(f"Total Haulage Cost to Processing Plant: {sum(yearly_haulage_costs_to_plant.values())}")
    print(f"Total Haulage Cost to Waste Dumps: {sum(yearly_haulage_costs_to_dump.values())}")

    print("\nAnnual and Total Carbon Costs:")
    for year in range(1, T+1):
        print(f"Year {year} - Carbon Cost: {yearly_carbon_costs[year]}")
    print(f"Total 5-Year Carbon Cost: {sum(yearly_carbon_costs.values())}")

    # Save to CSV
    model_output_df = pd.DataFrame(data_records)
    model_output_df.to_csv('model_output_all_blocks1.csv', index=False)
    print("Data for all blocks has been saved to 'model_output_all_blocks1.csv'.")
else:
    print("Failed to find an optimal solution. Status code:", m.status)
```