The University of the Witwatersrand

Masters Research Dissertation

Optimisation in Open Pit Mining

Author:
Raymond Phillips

Supervisors:
Professor Montaz Ali
Professor Cuthbert Musingwini

A dissertation submitted in fulfilment of the requirements for the degree of Masters of Science

in the

School of Computational and Applied Mathematics

July 2018

Contents

List of Figures

List of Algorithms

List of Tables

Abstract

The mining industry forms an integral part of South Africa - its society, culture, history and of course, its economy.

This research dissertation focuses on the Open Pit Mine Production Scheduling Problem, a cornerstone in the design and planning of an open pit mining venture and its profitability thereafter.

The accompanying optimisation problem is usually both complex and large. We investigate existing initial solutions as well as two existing metaheuristic algorithms that have been used to solve this problem, improving upon them and introducing a pseudo greedy approach that seeks production schedule improvement in the immediate solution space neighbourhood. This addition greatly improves initial solutions to the problem.

Through analysis on a smaller and larger mining instance we reveal the perceived advantages and disadvantages of two existing metaheuristics in producing optimal production schedules. We then propose a parent algorithm that interchangeably selects either of these algorithms based on probabilities determined by their observed performances during computation periods. The parent algorithm produces a strong production schedule that surpasses the current best found solution for the larger mining instance.

With these findings we propose a probabilistic selection method parent algorithm that interchanges between both algorithms in an effort to achieve a better solution.

Dedication

I dedicate this to aspiring academics and practitioners in the fields of Optimisation, Operations Research and Mining Research in the hope that they find use of this research in their ambitions and further research.

To my grandparents for their love and support during their lives. Without them I would never have gotten any of the opportunities I am so grateful for today...

Declaration

I, Raymond Aaron Phillips, declare that this dissertation titled, *Optimisation in Open Pit Mining* and the work presented in it are my own. I also confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

_____

Date:

_____

Acknowledgements

I would firstly like to acknowledge the University of the Witwatersrand for giving me a foundation and the opportunity to conduct this Masters in Science research.

I would also like to thank the National Research Foundation. I am immensely grateful for their assistance in completing this research.

I would like to thank my family and friends for keeping me motivated, supporting me and keeping me driven during the completion of this dissertation. A special thank you to Ben, Mom, Dad, Sonali and Matthew.

Most importantly, I would like to express my sincere gratitude and appreciation to my supervisors, Professor Montaz Ali and Professor Cuthbert Musingwini. Their support, patience and wisdom was paramount to me completing this research. I am especially thankful for my sittings with Professor Ali. He has been my mentor and guiding hand in my adventure through the intriguing world of Optimisation.

Chapter 1

Introduction

This chapter is intended to give readers a brief introduction to the content of this dissertation. Section 1.1 gives a description of what defines open-pit mining. This is then followed by Section 1.2, an explanation of what characterises open-pit mines. In Section 1.3 the phases of an open-pit mine are described. It is here that we start to see the focal point of this dissertation - how an open-pit mine's development should be optimised. Section 1.4 details the important role that mining plays in South Africa, both in the economy and society. This section then concludes with a motivation for this research. Lastly, Section 1.5 provides an outline of the subsequent chapters of this dissertation.

1.1 Open-pit Mining: Definition and Profile

Definition 1.1 (Mining [Newman *et al.* 2010])**.**

Mining refers to the removal of naturally occurring material from the earth for profitable gain.

Open-pit mining refers to the extraction of rocks or minerals that are near the earth's surface. It also goes by the names : *opencast mining or open-cut mining.*

Open-pit mining is characterised by the cone-like appearance due to the manner in which the earth material is excavated. This shape can be attributed to the requirement of *safe wall slopes*, important for safety reasons (eg. averting pit slope failures) and to aid in manageable excavation, transport of material and equipment, in and out of the pit. More often than not, it is a more productive and profitable venture than underground mining

projects [Newman *et al.* 2010]. Figure 1.1 shows a cross-sectional view of the general excavation process of an open-pit mine exhibiting this cone-like shape.

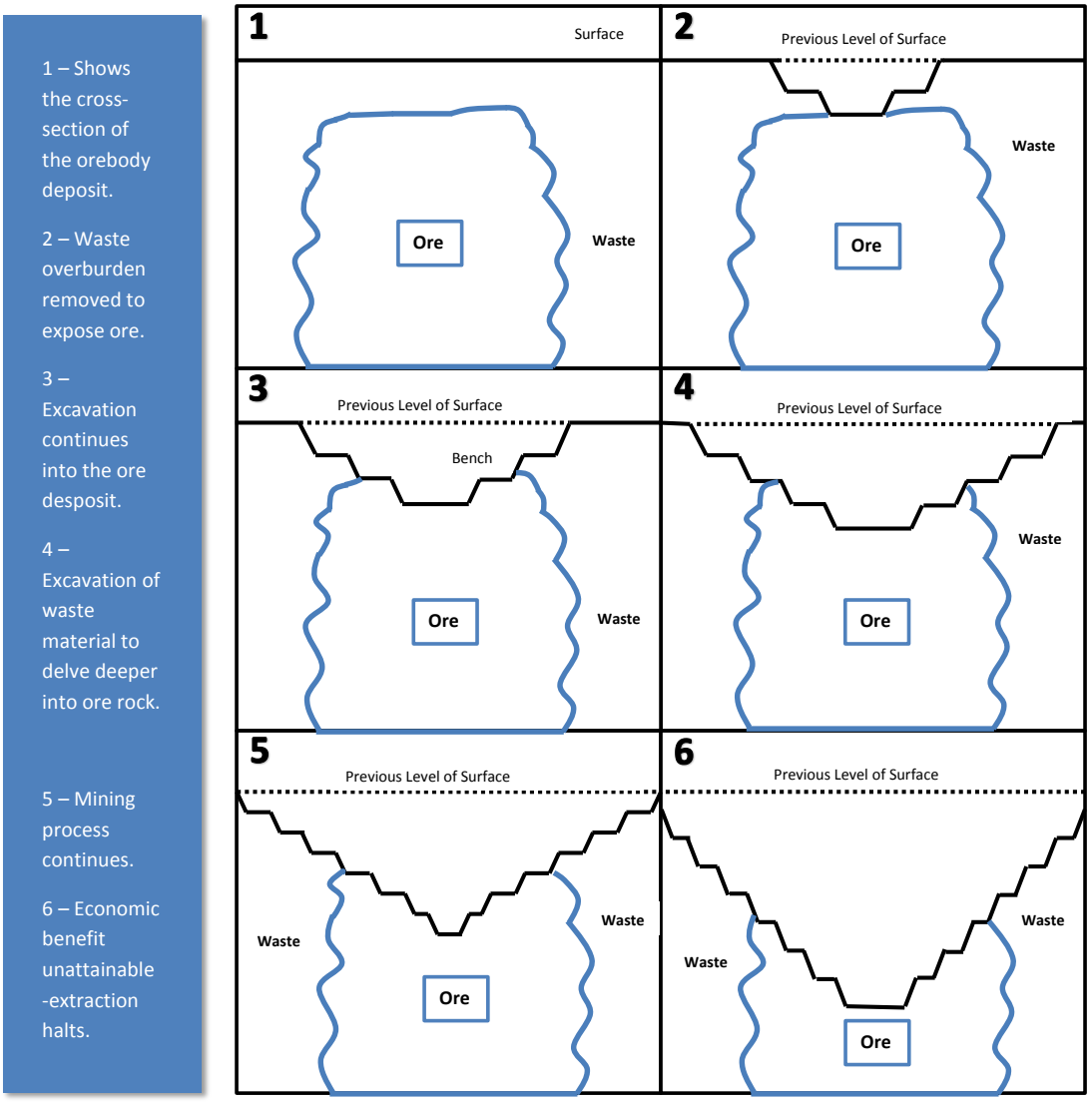

FIGURE 1.1: Cross-sectional View of General Extraction Progression

1.2 Features of an Open-pit Mine

Referring to Figure 1.2, we can identify the primary features of an open-pit mine. One might observe the *overburden* label in Figure 1.2, essentially a waste layer that needs to be removed before extraction of rock and minerals can begin. The *benches* are the areas of the open-pit mine where extraction takes place on exposed rock *faces*. The extraction process results in the formation of a *crater* in the *ground surface*. Also take note of the label designating *bench heights*, delineating the distance between two bench levels. *Ore* is referred to as the material that contains one or more minerals that can be mined, processed and sold for profit [Fricke 2006]. Material is designated as *waste* if it is believed that its extraction cannot provide a meaningful economic gain upon further processing. Waste is usually dumped outside the confines of the pit due to lack of space within the pit. This dumping is done as close as possible to the pit in an effort to reduce transportation costs. Haul roads run from the bottom of the pit to the surface and provide a route for transportation of materials (both ore and waste) out of the mine, usually via truck loads. Haulage *ramps* bring ore and material to the ground surface [Newman *et al.* 2010]. The *sump* is a depression at the bottom of the pit that collects ground water from mining processes.

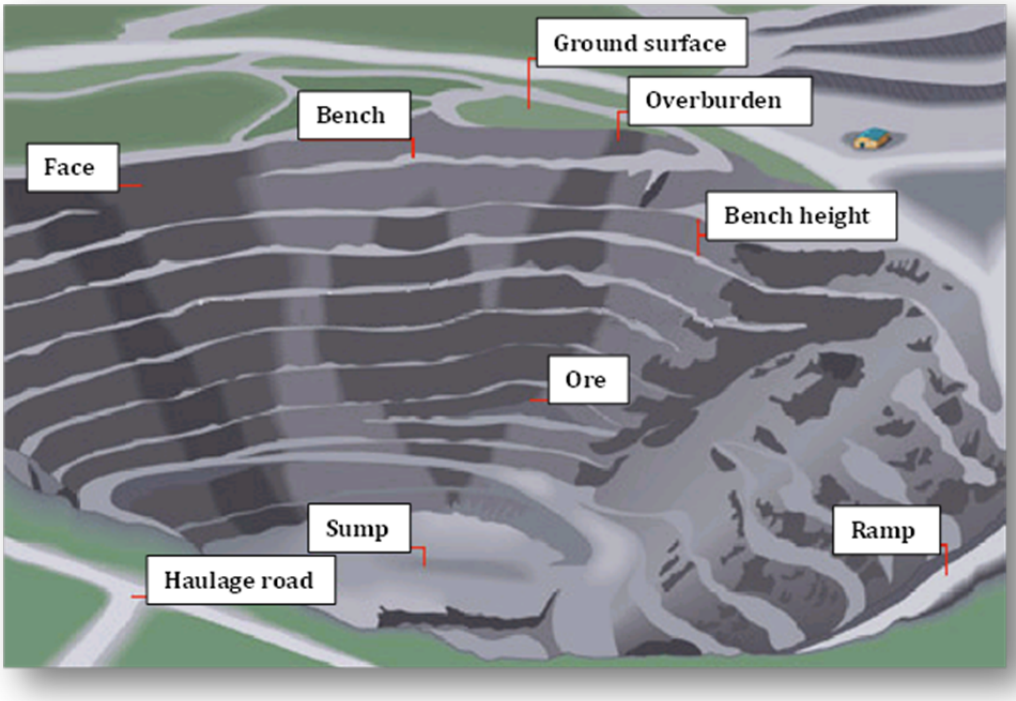

FIGURE 1.2: An Open-pit Mine [Espinoza et al. 2012]

1.3 The Life of an Open-pit Mining Venture

An open-pit mining project, along with its accompanying activities, is a complex and ambitious undertaking. It is, however, possible to segregate the operation into a number of distinct phases as outlined in [Fricke 2006] and [Newman *et al.* 2010]:

1. **Prospecting Phase**
   - The detection and identification of mineral deposits.
2. **Exploration Phase**
   - Evaluation of the discovered mineral deposits.
3. **Development Phase**
   - Administrative procedures(eg. acquisition of mining rights).
   - Mine design.
   - Excavation of primary access to ore.
4. **Exploitation Phase**
   - Excavation and transportation of extracted minerals and materials.
5. **Reclamation Phase**
   - The attempt to restore the region back to its original state once the mining project has concluded.

Phases 1 and 2 see a lot of activities involving geologists. Using visual inspection as well as physical measurements of earth properties, they attempt to identify mineral deposits [Newman *et al.* 2010]. Geologists then set about trying to determine the value of the discovered deposit using various simulation and interpolation techniques, such as *kriging*.

Definition 1.2 (Kriging)**.**

An interpolation technique used in geostatistics which utilises sampled data from measured locations to predict a value at an unmeasured location.

Phase 3 incorporates some fundamental design information and important decisions. This includes the mining technique, infrastructure requirements and getting an idea on the production capacities for the project.

When it comes to open pit mining, *Operations Research* has mostly been focussed on the development and exploitation phases. It is in these phases that mining engineers ply their trade. These phases involve answering two pivotal questions:

- Phase 3: What method(s) should be used to procure the material(s)?
- Phase 4: What should be done with these recovered material(s)?

1.4 The South African Mining Industry: A Compelling Influence on the Economy and Society

The South African Mining Industry is the fifth largest mining industry worldwide. Historically, South Africa's mineral wealth has resulted in gold rushes (the Witwatersrand Gold Rush of 1886), the foundation of cities (eg. Johannesburg), wars (the Second Boer War) and the expeditious influx of immigrants seeking fortune. Today, mining also plays an important role in South Africa. It continues to be an imperative contributing factor to South Africa's economy, society, policies and development [Jackson 1999].

1.4.1 GDP and Investment

The Chamber of Mines of South Africa [Chamber of Mines F&F 2016] publication states that mining is directly responsible for 7.1% of South Africa's GDP(Gross Domestic Product). Fixed investment in mining amounts to 89.4 billion ZAR. The mining industry also remains a vital element in the JSE (Johannesburg Securities Exchange) and comprises a large portion of the JSE All Share Index. This translates to a value of 1.4 trillion ZAR. Figure 1.3 reveals the extent of the financial contribution of the mining sector in South Africa circa 2012.

1.4.2 Further Influences

The [Chamber of Mines AR 2012] publication states several other compelling influences that the mining industry has on South Africa. The mining sector was involved in creating approximately 1.3 million jobs. According to [Chamber of Mines F&F 2016], the most recent number is $457,698$ direct jobs, which indicates a decline in jobs directly created by the mining sector. The mining sector has a *dependency ratio* of 10:1. This is an economic term that implies that about 13.5 million people are dependent on the 1.3 million jobs created in the mining sector for their daily food.

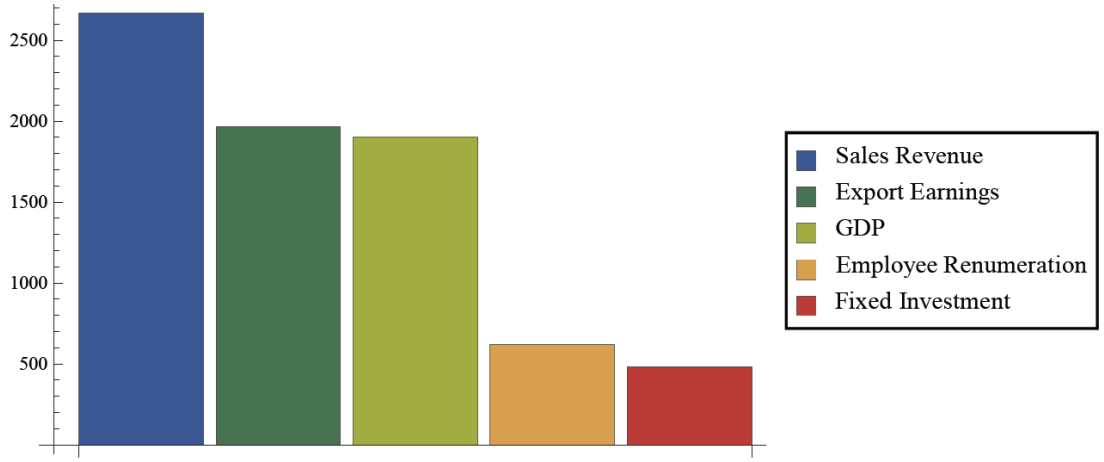

FIGURE 1.3: The Financial Contribution of the Mining Industry in Billions of Rands. Data obtained from [Chamber of Mines F&F 2012]

1.4.3 The Discernible Impact of the Global Financial Crisis

Figure 1.5 and Figure 1.4 reveal the growth rates of mining GDP and fixed investment in the mining sector respectively. The global financial crisis of 2008 did not spare South Africa's mining sector from its infectious global aura. As is most evident from Figure 1.5, there was a significant impact on the mining sector surrounding the harsh financial year of 2009. Also, we see a decline in fixed investment in the time domain around the financial crisis, as is often a symptom of a financial recession. However, one must note that some other factors have also affected investors' opinions, for example the call for the nationalisation of mines in South Africa. Investment by mining companies was rather high prior to 2009, with values of 31.3% and 27.5% in 2007 and 2008 respectively, more recent years have seen a sharp decline.

Not shown in the figures is that latest data from [Chamber of Mines AR 2015] is that mining, although showing signs of recovery continues to struggle. Investment in mining shrank by 0.6% in 2015. Private sector investment also dropped from 20% to 17% in the period from 2012 to 2015.

1.4.4 Research Motivation: A Potential Way Forward for the Mining Sector

Underground mining is more prevalent in South Africa than open-pit mining however, there are still a large number of open-pit mines. Open-pit mines form an integral part of many mining company ventures in South Africa. Some important South African mining companies that have significant surface mining projects include De Beers, Anglo American, Exxaro Resources, Gold Fields, Palabora Mining and African Rainbow Minerals.

FIGURE 1.4: RSA Mining Fixed Investment Growth Rate

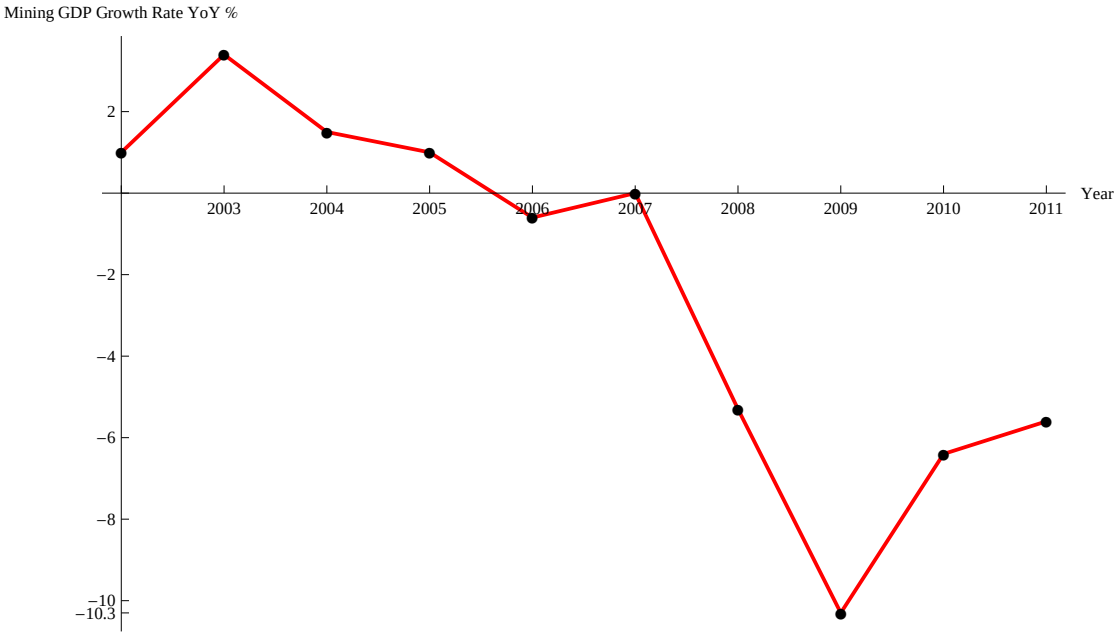

FIGURE 1.5: RSA Mining GDP Growth Rate

An open-pit mining project is a costly, lengthy and intensive endeavour. Optimising its operations and design will greatly benefit the country, economies, companies or persons with vested interests in the mining project. Open-pit mine design operations have received a lot of attention from academic and company research as far back as the 1960s in an attempt to increase efficiency. This is especially important in the industry's current climate with often wavering investor confidence, an increase in competitiveness, the emergence of financial instruments tailored for mining projects, new technologies, declining mining GDP values and erratic mineral prices. This is due, in large part, to the

Global Financial Crisis but also regulations, constraints, unrest, nationalisation calls and instability. No where is better reflected than in the performance of the *RESI20* Index, *JGOLD* Index and *JPLAT* Index seen in Figures 1.6,1.7 and 1.8.

These figures display the volumes (the bars at the bottom of each figure) as well as the value of these indices (the line chart). From the beginning of 2007 until the start of 2018 one can identify that the mining and resources sector in South Africa has struggled to make much progress. As we can see from these figures, the resources, gold and platinum sectors within South Africa have fallen away over the past decade.

Definition 1.3 (Market Capitalisation)**.**

Market Capitalisation, or market cap, refers to the total monetary market value of a company's outstanding shares. It can be calculated as follows:

$$MK_t = P_t s_t$$

where MK_t, P_t and s_t refers to the market cap, share price and number of outstanding shares at a time t.

Definition 1.4 (Market Capitalisation Weighted Index)**.**

This is an index where individual constituents hold a weighting in the index's value based on their market capitalisation.

Definition 1.5 (RESI20 Index)**.**

The FTSE/JSE Resource Index is a market capitalisation weighted index. It includes the top 10 largest companies by free float market capitalisation within the resources economic group of South Africa.

Definition 1.6 (JGOLD Index)**.**

The FTSE/JSE Africa Gold Mining Index is a market capitalisation weighted index. It includes companies involved in gold mining ventures, listed on the JSE.

Definition 1.7 (JPLAT Index)**.**

The FTSE/JSE Africa Platinum Mining Index is a market capitalisation weighted index. It includes companies involved in platinum mining ventures, listed on the JSE.

FIGURE 1.6: RESI20 Index [Financial Times]

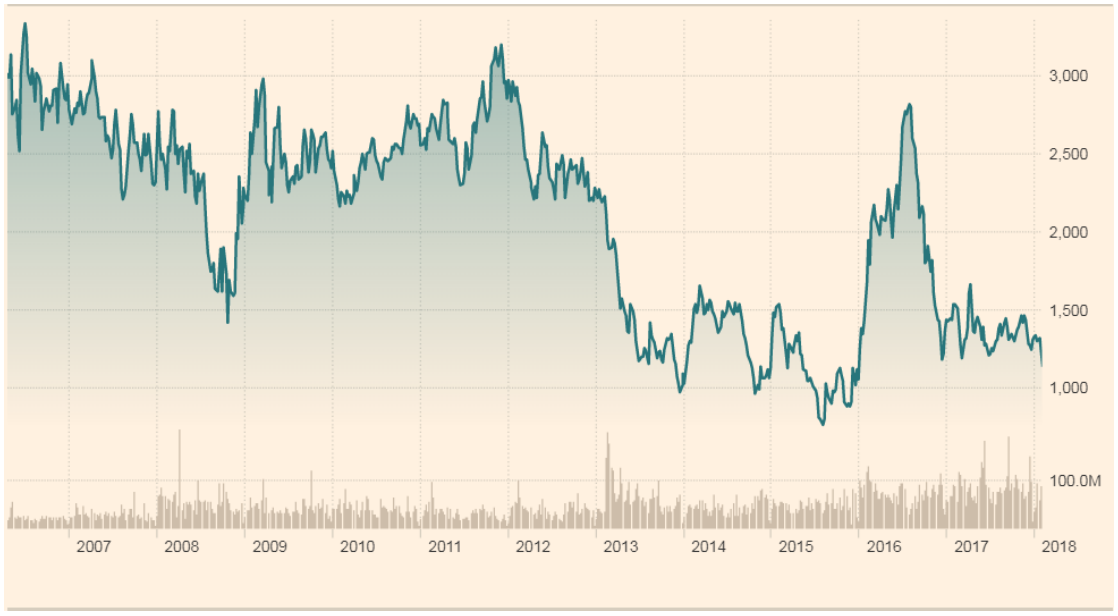

FIGURE 1.7: JGOLD Index [Financial Times]

FIGURE 1.8: JPLAT Index [Financial Times]

Concerns have been heightened globally because of the dramatic drop that the gold price has seen over the past 2 years, revealed in Figure 1.9. Figure 1.10 also paints a difficult picture of the gold price from 1967 to the collapse in 2012. Gold is not the only precious metal that has seen a drop in value. Figure 1.11 portrays the shared deterioration in the platinum price.

FIGURE 1.9: Gold Price (USD) since 2007 [Financial Times]

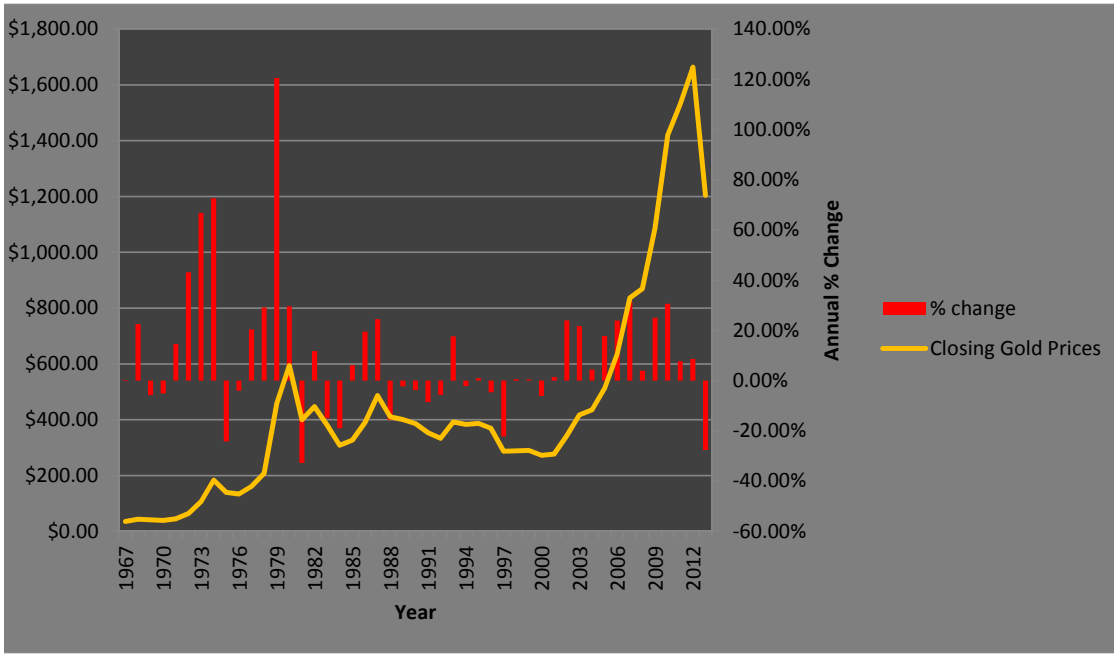

FIGURE 1.10: The Volatility of the Gold Price [onlygold.com]

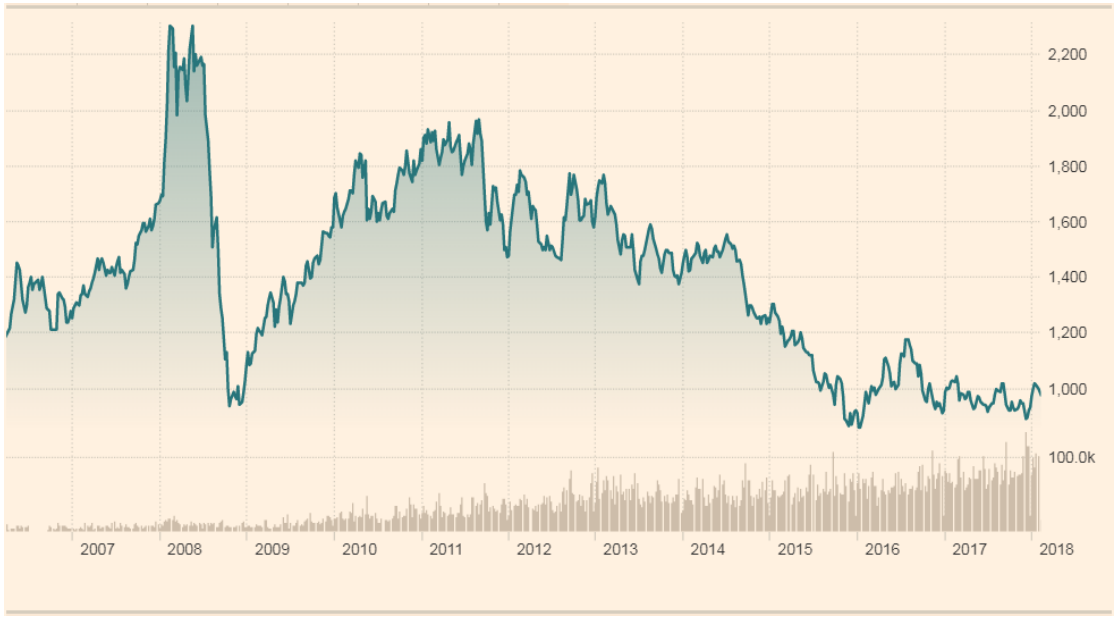

FIGURE 1.11: Declining Platinum Price (USD) [Financial Times]

The mining sector needs to adapt to this recent harsh climate it finds itself in and exploit any opportunities that will aid in a recovery.

It is clear that the mining sector is an important player worldwide, as well as a considerable influence on the lives of South Africans. If one could improve, possibly even revolutionise,

the way in which we answer critical questions in open-pit mine design it may provide that injection of rejuvenation to the mining sector.

An open-pit mining venture has a number of imperative mine design questions, many of these are portrayed as mathematical problems. *Mathematical Programming* and *computational science* are two powerful tools we have in our repertoire that will prove vital to solving such problems.

The next chapter will introduce Mathematical Programming, explaining the discipline's origins, terminologies, usages and forms that will feature heavily in this research.

1.5 Structure of Dissertation

1.5.1 Layout

This research is segregated into chapters, each with a defined purpose that is outlined in the lines that follow.

Chapter 2 will introduce Mathematical Programming, a tool we will use to model open-pit mine design. This forms the cornerstone of optimisation problems.

Chapter 3 describes relevant methods to this research that are explicitly or inherently founded in the optimisation mechanisms of this dissertation, literature and computation software packages.

Chapter 4 describes the heuristics and algorithms that are employed in this research and form the grounding for finding solutions to the mathematical programming problems that we will encounter in open pit mine planning and design.

Chapter 5 delves into the modelling of open-pit mine planning with mathematics, giving one an insight to the procedure for this phase of a mining venture. It introduces how the planning is moulded into a mathematical program to form a model that assists in producing mine plans and production schedules.

This is then followed by Chapter 6 which is a literature review on the Open-pit Mine Production Scheduling Problem (OPMPSP), a vital constituent in mine design and the focus of this dissertation. This chapter is partitioned into sections based on what methodology each research employed in an effort to solve the Open-pit Mine Production Scheduling Problem.

Chapter 7 introduces the evolution of the model formulations to solve the Open-pit Mine Production Scheduling Problem. After taking these into account we then present the particular model scheme utilised in this research.

The model formulations are then followed by Chapter 8, which introduces the Precedence Constrained Knapsack Problem. This problem is synonymous in its structure to the Open-pit Mine Production Scheduling Problem. The solution methods and inferences from this problem are applicable to the focus of this dissertation's research. We use these methodologies to find initial solutions for the Open-pit Mine Production Scheduling Problem.

Chapter 9 outlines the Tabu Search metaheuristic's design in the scope of the Open-pit Mine Production Scheduling Problem. It explains the rationale behind the metaheuristic in solving this problem.

Chapter 10 explains the framework for the Simulated Annealing metaheuristic in its application to the Open-pit Mine Production Scheduling Problem.

The implementation, performance, analysis and results of the aforementioned methods is detailed in Chapter 11. It is in this chapter that we also introduce the contributions of this dissertation, namely the *Pseudo Greedy Search*, the *Dual Interchange Algorithm* as well as some analysis on the parameters in existing methods.

Finally, Chapter 12 provides a conclusion on the research and the Open-pit Mine Production Scheduling Problem. It also includes a section on suggested further research on this interesting and complex problem in mine planning and design.

1.5.2 Contribution

Within the implementation section in Chapter 11 we introduce a new algorithm, whose behaviour is likened to that of a local search. We name this algorithm *Pseudo-Greedy Search* (PGS). PGS improves upon the initial solutions to the Open-pit Mine Production Scheduling Problem, which is the primary focus of this research. It can find application during other stages, as well as other algorithms, in solving the Open-pit Mine Production Scheduling Problem. In short, it serves as a computationally efficient way to improve one's current solution.

Literature on choice of parameters for metaheuristics is sometimes rather sparse. In the metaheuristics that we utilised, namely Tabu Search and Simulated Annealing, we offer some guidance on these parameter values, indicating which of these showed promise. This was largely done empirically.

Strong coding practices, such as vectorisation, were employed within the coding frameworks of the metaheuristics. We are of the belief that these improved the computational efficiency of our methods as well as that of previous methods.

We introduce a novel algorithm that we name the Dual Interchange Algorithm (DIA). It makes use of both the Simulated Annealing and Tabu Search algorithms, hoping to draw on the strengths of both of these metaheuristics.

Finally, there is an analytical component of this research where the performance and features of the respective algorithms are highlighted, analysed and scrutinised. This analysis allows us to draw conclusions on the respective methods and suggest further research.

Chapter 2

Introduction to Mathematical Programming

"In a mathematical optimization (or programming) problem, one seeks to minimize or maximize a real function of real or integer variables, subject to constraints on the variables. The term mathematical optimization refers to the study of these problems: their mathematical properties, the development and implementation of algorithms to solve these problems, and the application of these algorithms to real world problems. Note in particular the word "programming" does not specifically refer to computer programming. In fact, the term "mathematical programming" was coined before the word "programming" became closely associated with computer software. This confusion is sometimes avoided by using the term **optimization** *as an approximate synonym for mathematical programming."*

An extract from the [Mathematical Optimization Society]

2.1 The Mathematical Program

A mathematical program has the following presentation:

$$\max f(x) : x \in X, \tag{2.1}$$

subject to:

$$g(x) \leq 0, \tag{2.2}$$
$$h(x) = 0, \tag{2.3}$$

where X represents a subset of $\mathbb{R}^n$ within the domain of f, g and h. x is the decision variable(s) for the mathematical program.

Definition 2.1 (Decision Variables)**.**

In a mathematical program, decision variables are a set of variables whose values can be controlled in an effort to maximise or minimise the problem.

(2.1) is known as the *Objective Function*. It is this expression that is desired to be maximised or minimised in the optimisation problem. Equations (2.2) and (2.3) are known as the *constraints*, being the inequality and equality constraints respectively.

There are formulations that deviate from the above however, for model convention purposes, one might desire to express the original formulation in the standard framework above ie. equations (2.1)–(2.3).

Definition 2.2 (A Feasible Point)**.**

A point x is feasible $\iff$ $x \in X$ and it satisfies the constraints equations (2.2) and (2.3).

Definition 2.3 (An Optimal Point)**.**

A point x^* is classified as optimal if it is **feasible** and $f(x^*) \geq f(x), \quad \forall x \in X.$

Of course, the optimisation problem may be one of minimisation rather than maximisation. In such cases, Definition 2.3 would have the opposite inequality criterion $f(x^*) \leq f(x), \quad \forall x \in X$.

2.2 The Mathematical Programming Discipline

Mathematical Programming can refer to the study of one or more components in the spectrum of a mathematical program. This can include theoretical pursuits regarding the existence of a solution for the mathematical program. Undertakings in this discipline may not be theoretical but rather more applicative, such as the development of new algorithms or exploitation of known algorithms to find the optimal solution for the mathematical program. Sometimes these algorithms can also inform us of the existence of an optimal solution. Mathematical Programming, when talking about the discipline as a whole, can also entail the process of devising a mathematical expression for a given problem. This is known as formulation construction and has its own set of analysis, comparisons and protocols. Studies in mathematical programming often extend into the aftermath of solving the actual problem. The results are analysed, scrutinised or substantiated at this stage. Finally, an area that receives much attention is harnessing the potential of computing power within optimisation practices. Being able to express a mathematical program in a computational environment is of great benefit given that majority of problems today are complex.

2.3 The Linear Program (LP)

There are several forms of a mathematical program. Perhaps the most well-known of these is the *linear program*. Linear programming refers to the study of mathematical programs that have a linear objective function subject to linear inequality and linear equality constraints. These constraints describe a *polyhedron*.

Definition 2.4 (Polyhedron)**.**

A polyhedron is a set in Euclidean space defined by a finite set of linear inequalities and linear equations.

A linear program can be expressed mathematically as follows:

$$\max c^T x, \tag{2.4}$$

subject to:

$$Ax \leq b, \tag{2.5}$$
$$Cx = d, \tag{2.6}$$
$$x \in \mathbb{R}^n_+. \tag{2.7}$$

In the above formulation, c and x are $(n \times 1)$ vectors, A and C are $(m \times n)$ matrices and b and d are $(m \times 1)$ vectors. Equation (2.4) is the objective function of the linear program. The equations (2.5)–(2.7) are the constraints, the constructs of the polyhedron.

Figure 2.1 shows an example of a polyhedron for a 3-dimensional linear programming problem. The constraints in the linear programming problem determines the geometry of the polyhedron. The area enclosed by the polyhedron is known as the *feasible region*. It is in here that the feasible and optimal solutions to the linear program reside.

The *simplex method* and *interior point* methods are common approaches to solving linear programming problems [Dantzig and Thapa 2003].

2.4 The Integer Program (IP)

Integer Programming shares some similarities with Linear Programming in that the objective function and the constraints are linear however, there is the further constraint that all variables must be integers. The equations (2.8)–(2.11) describe a formulation for an integer program:

$$\max c^T x, \tag{2.8}$$

subject to:

$$Ax \leq b, \tag{2.9}$$
$$Cx = d, \tag{2.10}$$
$$x \in \mathbb{Z}^n_+. \tag{2.11}$$

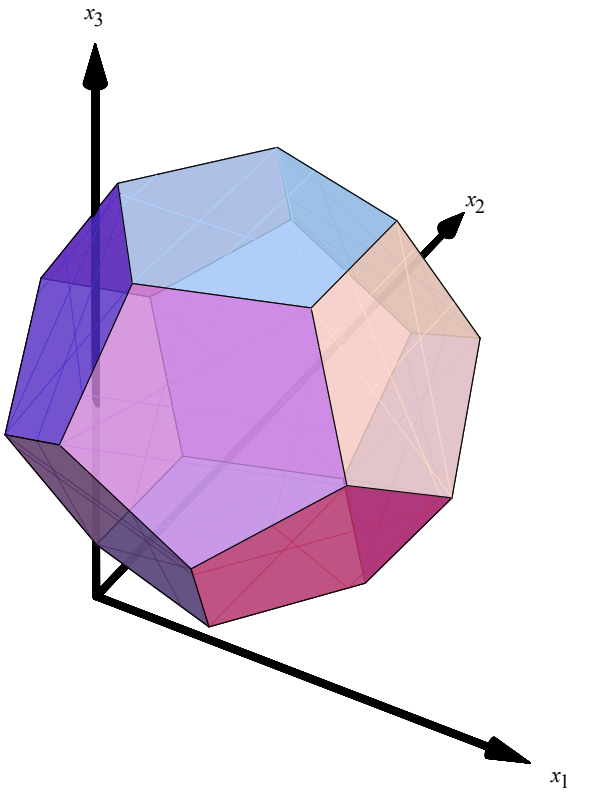

FIGURE 2.1: Polyhedron in Euclidean Space for a Mathematical Program

Many mathematical problems can be modelled with integer programming. Often, these models will include *decision variables* that must be/are integers.

2.4.1 Binary Integer Program (BIP)

In the above model of an integer program, the decision variables x can be any positive integer. Sometimes, a problem may call for a model that affords the ability to express decisions of a "yes-or-no" nature. For such cases one would make use of what is referred to as a *binary integer program* (BIP). The formulation of a BIP is as follows:

$$\max c^T x, \tag{2.12}$$

subject to:

$$Ax \leq b, \tag{2.13}$$

$$Cx = d, \tag{2.14}$$

$$x \in \{0,1\}^n. \tag{2.15}$$

Notice in the formulation of the binary integer program above, equations (2.12)–(2.15), that the decision variables can only take on 2 values, namely 0 or 1. It is this equation (2.15) that simulates the yes-or-no decision setup, 0 for no and 1 for yes.

2.4.2 The Mixed Integer Program (MIP)

The mixed integer program can be thought of as a variant of the integer program and linear program. With this mathematical program not all decision variables have to be positive integers, as seen in the formulation of an integer program:

$$\max c^T x + \underline{e}^T y, \tag{2.16}$$

subject to:

$$Ax + By \leq b, \tag{2.17}$$

$$Cx + Dy = d, \tag{2.18}$$

$$x \in \mathbb{Z}_+^n \quad \text{and} \quad y \in \mathbb{R}_+^p. \tag{2.19}$$

In the above formulation, $\underline{e}$ and y are $(p \times 1)$ vectors and B and D are $(m \times p)$ matrices. The other variables involved in this formulation retain their dimensions as defined previously.

Notice in the integrality constraint, equation (2.19), that decision variable x is of the positive integers but y can take positive values from the reals. It is this constraint that characterises mixed integer programs.

2.5 Program Formulation Consequences and Considerations

In a linear program, an optimal solution is found at the vertices of the polyhedron as highlighted in Figure 2.2. Unfortunately, in integer programming this is not the case. The integrality constraints in the formulations of the IP, BIP and MIP problems make them significantly more difficult to solve than an LP. A number of methods have been developed to solve these integer programs. Several of these will be described in the subsequent chapter of this dissertation.

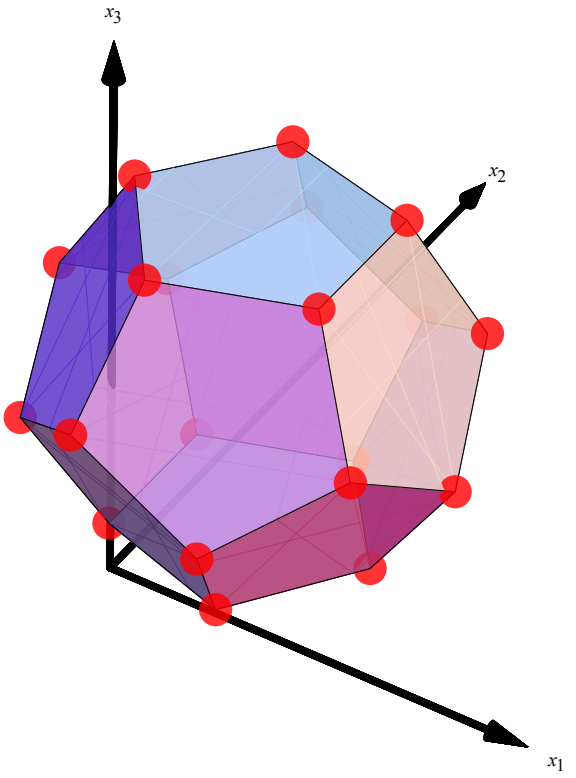

FIGURE 2.2: An LP Formulation with Potential Optimal Solution Points

Consider Figure 2.3, a feasible region for a 2-dimensional integer programming problem. A set of integer points, X, highlighted within the polyhedron are the feasible points. It is evident that a linear programming problem is much easier to solve because of the characteristic position of possible optimal solutions at the vertices in its feasible region. Indeed, the simplex method makes full use of this knowledge [Dantzig and Thapa 2003]. Conversely, the polyhedral feasible region in Figure 2.3 for the IP does not provide much assistance with regards to finding the optimal solution. Vertices of the polyhedron cannot be considered because they violate the integrality constraint as they are not integers.

Evidently, formulations play a major role in mathematical programming and the way problems can be solved. There is some study into improving formulations, particularly with integer programs. Certainly, there can be several alternative IP formulations for a specific problem. [Wolsey 1998] investigated some differing formulations of integer programs and attempted to reveal how some formulations may be better than others. [Wolsey 1998] also presented the idea of an ideal formulation. The next set of definitions and explanations will attempt convey this concept and highlight the attraction of improving a formulation.

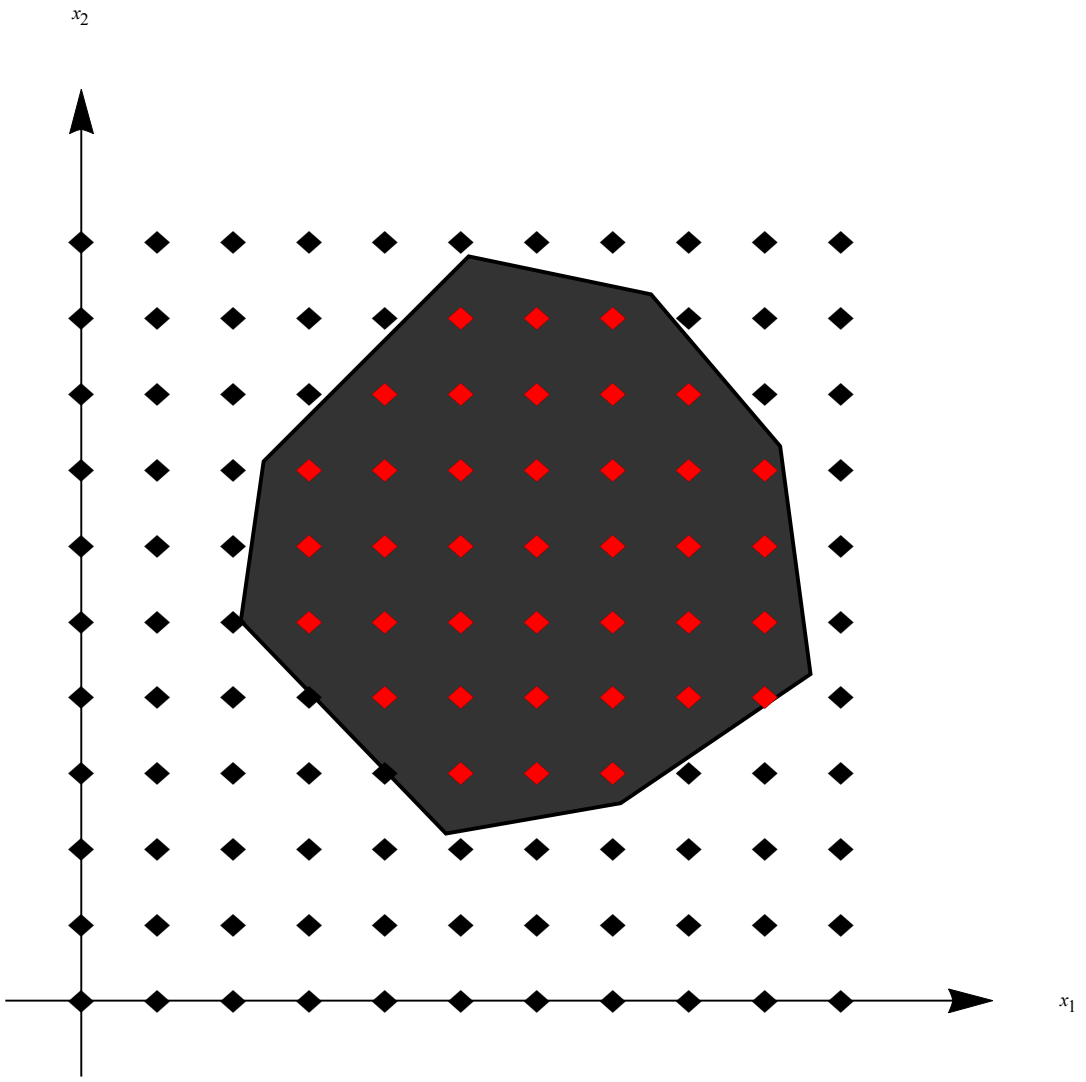

FIGURE 2.3: A Graphical Depiction of a Integer Program Formulation

2.5.1 The Ideal Formulation - Convex Hull

Consider a polyhedron defined by a finite set of linear constraints, $P = \{x \in \mathbb{R}^n : Ax \leq b\}$. A polyhedron is sometimes referred to as a *formulation*.

Definition 2.5 (Formulation [Wolsey 1998])**.**

A polyhedron $P \subseteq \mathbb{R}^{n+p}$ is a formulation for a set $X \subseteq \mathbb{Z}^n \times \mathbb{R}^p$ *iff* $X = P \cap (\mathbb{Z}^n \times \mathbb{R}^p)$.

This is essentially the mathematical notation for the proverbial feasible region. An **ideal formulation** would be one where the solution of the linear program also gives the solution to the integer program. This "ideal" polyhedron, denoted by P^*, can be defined as the *convex hull* of the set X.

Definition 2.6 (Convex Hull [Wolsey 1998])**.**

Given a set $X \subseteq \mathbb{R}^n$, the convex hull, $conv(X)$, is: $conv(X) = \{x : x = \sum_{i=1}^{t} \lambda_i x^i, \sum_{i=1}^{t} \lambda_i = 1, \lambda_i \geq 0, \quad i = \{1, ..., t\}$ and $\{x^1, ..., x^t\}$ a finite subset of $X\}$.

Referring to Figure 2.4, one can see that graphically $P^* = conv(X)$ is a polyhedron with vertex points within the set X. Therefore, the integer program, $\max\{c^T x : x \in X\}$, with $X = \{Ax \leq b, x \in \mathbb{Z}_+^n\}$, can also be formulated as a corresponding LP, $\{\max c^T x : x \in conv(X)\}$. It should be noted that it may not always be possible to find the convex hull of a set, the ideal formulation to the integer program. However, one could still reduce the polyhedron by the addition of beneficial *valid inequalities*.

Definition 2.7 (Valid Inequality [Wolsey 1998])**.**

An inequality, $\pi x \leq \pi_0$, is known as a valid inequality for the set X if $\pi x \leq \pi_0$ $\forall\ x \in X$.

If P_1 was the original formulation of an IP and valid inequalities were added to this formulation, P_1 could be reduced to a new polyhedron P_2. P_2 is a better formulation than P_1 since $P_2 \subset P_1$ and any solution methods applied to search this polyhedron will likely be more efficient [Fricke 2006].

For further applications and theory on integer program formulations the reader can refer to [Wolsey 1998], [Fricke 2006] and [Nemhauser and Wolsey 1988].

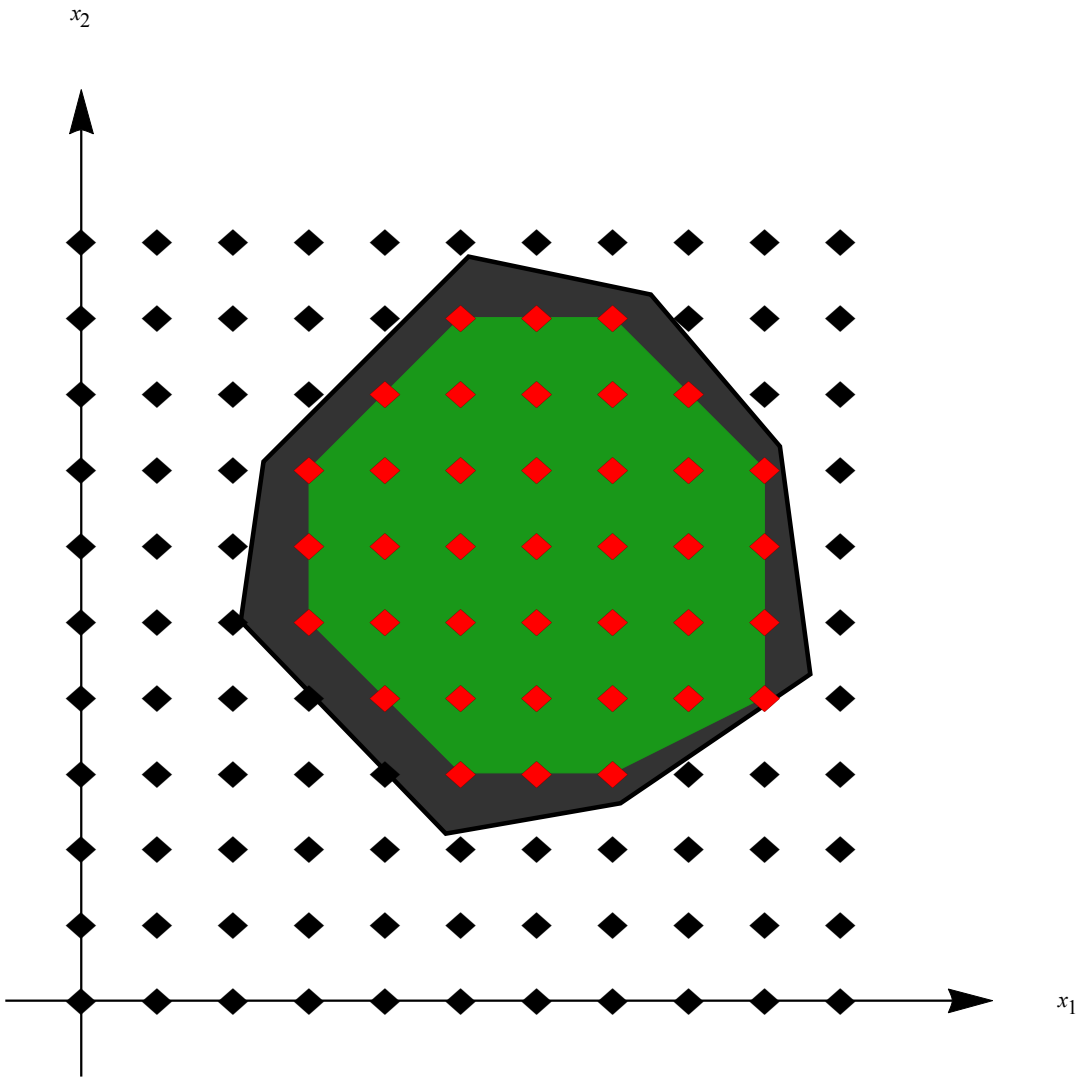

FIGURE 2.4: The Ideal Formulation - Convex Hull

The study of the types, structures and nature of mathematical programs is evidently an invaluable component in the quest to optimise. Without these fundamentals there would be no medium for the modelling of optimisation problems.

A set of problems that frequently interplays with the mathematical programming is *combinatorial optimisation*. The next section will briefly describe how it is defined and where it resides in the broader domain of applied mathematics.

2.6 Combinatorial Optimisation

2.6.1 Introduction

Although not a universal classification for combinatorial optimisation problems, there are 3 properties that the majority of these problems exhibit:

1. They are defined as optimisation problems, see Section 2.1.
2. They are easy to state.
3. There may be a very large, yet finite, number of feasible solutions.

Combinatorial optimisation can be thought of as particular brand of problems that involves selecting an optimal item from a set of feasible items. Looking at the definition provided by [Dahl and Mannino 2012], we can get a mathematical interpretation of these problems.

Definition 2.8 (Combinatorial Optimisation Problem)**.**

Let Ω be a finite set and $\mathcal{F}$ a family of feasible subsets of Ω. Let f be a real valued weight function, the objective function, defined on the elements of Ω:

$$f(F) = \sum_{\omega \in F} f(\omega), \quad F \in \mathcal{F}. \tag{2.20}$$

A minimisation combinatorial optimisation problem is then to find a $F^* \in \mathcal{F}$ such that,

$$f(F^*) = \min_{F \in \mathcal{F}} f(F) \tag{2.21}$$

When comparing combinatorial optimisation to other mathematical areas, such as algebra and geometry, it appears relatively young. After all, the beginnings of algebra can be traced as far back as the times of the ancient Babylonians [Struik 1987]. Combinatorial optimisation only achieved some resemblance of a structure in the 1950's when there was more of an awareness of *operations research* [Schrijver 2005].

Definition 2.9 (Operations Research)**.**

Operations research, or operational research (in the UK), is the science of utilising advanced analytical methods to enable better management of complex systems that typically have interactions of money, materials, people, equipment and other entities.

Operations research, colloquially referred to as the "science of better", features a number of advanced analytical methods, combinatorial optimisation being but one of many.

2.6.2 Prominent Problems and Complexities

The *travelling salesman problem*, *optimum assignment*, *shortest spanning tree* and transportation problems are examples of some discussion material that one might encounter in the combinatorial optimisation fraternity. These are but a few problems amongst a plethora of other real-world problems in the field. Some of these have *polynomial time complexity* algorithms. For example, the shortest path problem. However, the majority of problems in combinatorial optimisation are *NP Hard*, no polynomial method for their solution is known. Some examples of these more complex problems are vehicle routing, crew scheduling and production planning [Schrijver 2005].

2.6.3 Origins

There is a vast expanse of combinatorial optimisation problems. The diverse nature of this field can be affirmed by investigating, rather philosophically, the day-to-day problems, questions and decisions that systems, organisations, institutions and humans face. A valid, yet primitive, example is considering that tasks such as searching for food or finding shortest routes can be thought of as combinatorial optimisation problems. Consider now that planning a day to do some shopping is alike to solving a travelling salesman problem. Similarly, a manager designating duties to workers can be seen as an assignment problem. It is evident that combinatorial optimisation is not exclusively for the academic arena [Schrijver 2005].

2.6.4 A Question of Tractability - Exact or Heuristic Methods

Section 2.6.2 alluded to some of the more popular material in combinatorial optimisation and the notion of complexity in these problems. It is imperative that there is some amount

of deliberation on a problem's complexity before solving it can commence. Problem complexity greatly influences the methods one would employ to solve a problem.

If the problem in question is big enough it may prove futile to attempt to solve it using exact methods because the problem becomes *intractable*.

Definition 2.10 (Intractable Problems)**.**

Problems that are deemed solvable in theory, but are not solvable in practice are deemed intractable.

Instead, it is better to concede that the luxury of finding the global optimal solution to the problem may not be attainable. Despite the powerful computational resources available in these times, it may simply take too long to find the best solution. To illustrate the immensity that problems can have, or develop, consider the following frivolous example - imagine *Black Friday*, the unofficial start of the holiday shopping season in the United States, is approaching and you have to plan which shops you are going to visit. Consider that you have $n = 100$ shops to choose from and you can only go to k of these during the course of the day. Table 2.1 reveals how the problem size increases as the number of choices increases to $k = 7$.

TABLE 2.1: Example of an ever Increasing Solution Space

k	2	3	4	5	6	7
$\binom{n}{k}$	4950	161700	3.92×10^6	7.52×10^7	1.19×10^9	1.68×10^{10}

Of course, it is highly unlikely that a shopper will think of this in this rather mathematical format however, the point is clear - large solution spaces can render problems intractable to solve with exact solution approaches. This is especially true if the exact solution approach searches every possible solution in the problem space. In the above example, this would mean that approximately 16 billion shop combination choices would have to be evaluated according to what the shopper identifies as "optimal" if $k = 7$ choices can be made. Thankfully, exact solution approaches are not exclusively "brute force" searches through an entire solution space. There are a number of more efficient exact methods that still guarantee the optimal solution upon completion. However, they too may succumb to large problem instances.

Should it not be practical to use exact solution methods, one might utilise heuristic approaches. Heuristics are methods that offer no assurance of the optimal solution

however, they work towards acquiring a near-optimal solution in a reasonable amount of time. Depending on the problem and circumstances, it may even be possible to guarantee an optimal solution using a heuristic method.

Chapter 3

Background on Relevant Exact Optimisation Methods

The last chapter concluded with an explanation of exact solution methods and heuristics in solving optimisation problems. During the course of this research a number of exact methods were implemented or encountered. This chapter seeks to describe the concepts and workings of a number of pertinent exact methods involved in this research. These methods find application, both explicitly or implicitly, within future methods and software deployed in this research.

3.1 Dynamic Programming

Although not a universal exact solution method for all problems, the dynamic programming approach has been used as an exact solution method in many problems in operations research.

Dynamic programming is generally used when it is thought that a full solution to a problem can be recursively defined by solutions to its subproblems. One is essentially solving subsets of the problem, progressively moving closer to solving the full problem. This approach can lead to finding quicker solutions however, at the cost of extra memory usage. After all, we are storing each subproblem during the course of this method. An example of its usage as an exact solution approach is in the solving of the *All-Pairs Shortest Path Problem*. We will use the dynamic programming solution approach to this problem to reveal the inner workings of this algorithmic paradigm.

3.1.1 All-Pairs Shortest Path Algorithm

Consider a graph network, $G = (V, E)$, with vertices labelled from 1 to n and weighted edges (Figure 3.1). The All-Pairs Shortest Path Problem asks to find the shortest distance between any two pairs of vertices in the graph network.

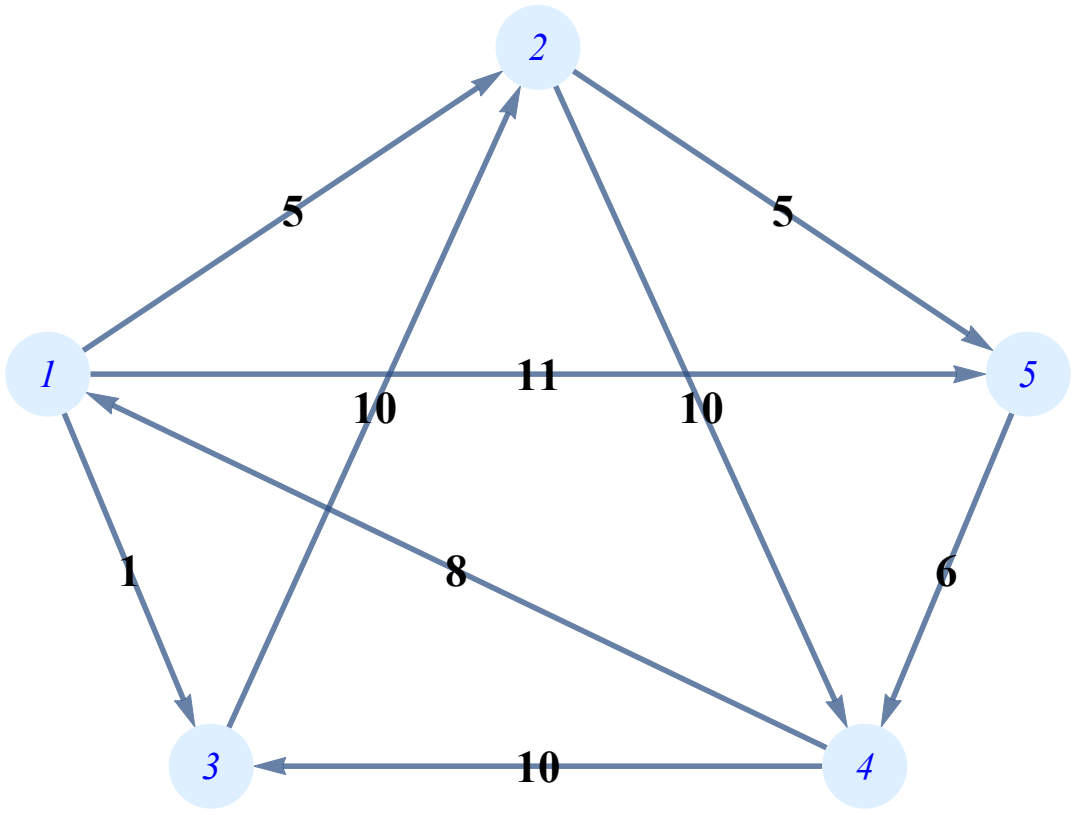

FIGURE 3.1: An Example Graph Network

One could always just use Dijkstra's algorithm on every single vertex pair. However, there is a dynamic programming approach to solving this problem that is in some cases more efficient than using Dijkstra to find the shortest path between all vertices.

In his works Dijkstra observed that the shortest distance from a vertex i to another vertex j, denoted d_{ij}, is either the shortest distance to j found so far **or** the shortest distance from i to some other vertex k, denoted d_{ik}, followed by the direct distance from vertex k to vertex j, denoted w_{kj}. This is known as the *relaxation property* and can be expressed in the following formula:

$$d_{ij} = \min\{d_{ij}, d_{ik} + w_{kj}\} \tag{3.1}$$

This formula plays an important role in the dynamic programming method for this problem.

These values would be stored in a *distance array*. For the example in this section the distance array will have the following format, with $n = 5$.

$$D = \begin{pmatrix} d_{11} & d_{12} & \cdots & d_{1n} \\ d_{21} & d_{22} & \cdots & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n1} & d_{n2} & \cdots & d_{nn} \end{pmatrix}$$

Subproblems are constructed by considering the shortest path from a vertex i to a vertex j only using vertices labelled $\{1, .., k\}$. We now add another dimension to the d variable. The shortest distances for each pair in these subproblems are then stored as d_{kij}. If $k = 0$ then $d_{ij0} = w_{ij}$, ie. the direct distance from i to j. Of course, should there be no direct link between i and j then $w_{ij} = \infty$. Otherwise, it is the weight of the edge in question.

$$w_{ij} = \begin{cases} 0 & \text{if } i = j, \\ w(i,j) & \text{if } i \neq j \text{ and } (i,j) \in E, \\ \infty & \text{if } i \neq j \text{ and } (i,j) \notin E. \end{cases}$$

The shortest distances for each of the subproblems is updated as follows:

$$d_{k,i,j} = \min\{d_{k-1,i,j}, d_{k-1,i,k} + d_{k-1,k,j}\} \tag{3.2}$$

The concept of equation (3.1) is reflected in equation (3.2). The shortest distance from i to j is the previous shortest distance from i to j **or** the previous shortest distance from i to k and then the addition of the shortest distance from k to j. With the increase in k we are successively solving subproblems, checking to see if allowing k as an intermediary between i and j will result in a shorter distance.

This dynamic programming approach is known as the *Floyd-Warshall Algorithm*. The pseudo code in algorithm 1 outlines its procedure to finding the shortest distances between pairs.

The *path* data structure in the algorithm 1 stores the next vertex that must be visited in the path for the shortest distance from i to j. This enables the tracing of all vertices that will be visited in the traversing of this shortest path. This algorithm checks the distances between pairs by utilising different intermediary vertices and updates shortest distances

based on previously acquired knowledge from subproblems. It has a time complexity of $O(n^3)$.

Algorithm 1 Outline of the Floyd-Warshall Algorithm - Shortest Pairs Problem

```
1  Initialise d(i,j) ← w(i,j)        % initialise shortest distances as direct distances.
2  % Initialise path data structure to store shortest paths between vertices.
3  for all (i,j) ∈ E do
4     path(i,j) = j
5  end for
6  for all (i,j) ∉ E do
7     path(i,j) = ∞
8  end for
9  for k ← 1 to n do
10    for i ← 1 to n do
11       for j ← 1 to n do
12          if d(i,k) + d(k,j) < d(i,j) then
13             % update shortest distance and path.
14             d(i,j) ← d(i,k) + d(k,j)
15             path(i,j) ← k
16          end if
17       end for
18    end for
19 end for
```

The conclusion of the algorithm produces the following shortest distances,

$$D = \begin{pmatrix} 0 & 5 & 1 & 15 & 10 \\ 18 & 0 & 19 & 10 & 5 \\ 28 & 10 & 0 & 20 & 15 \\ 8 & 13 & 9 & 0 & 18 \\ 14 & 19 & 15 & 6 & 0 \end{pmatrix}$$

and the *path* array that allows us to trace out the route.

$$path = \begin{pmatrix} \infty & 2 & 3 & 2 & 2 \\ 4 & \infty & 4 & 4 & 5 \\ 4 & 2 & \infty & 2 & 2 \\ 1 & 1 & 1 & \infty & 2 \\ 4 & 4 & 4 & 4 & \infty \end{pmatrix}$$

As an example, consider we wanted the shortest distance from vertex 1 to vertex 5. This value is stated as 10 units, as seen in the distance array, D. For the route we would read $path_{15} = 2$. This means one should traverse to vertex 2. Thereafter, reading $path_{25} = 5$ means we should move to vertex 5, the target vertex and conclusion of the path. Looking at figure 3.1, this is the shortest distance from 1 to 5, 10 units. Certainly, it is less than the direct distance of $w_{15} = 11$.

3.2 Branch-and-bound

The branch-and-bound method is an exact solution approach that has been extensively utilised in the solving of combinatorial optimisation problems of larger sizes and complexities [Clausen 2003]. Having a generalised framework, the algorithm has to be adapted for particular problem types. Many of these problems exhibit NP-Hard complexity, meaning that they often require efficient techniques to reduce the solution space that needs to be explored in the quest to find an optimal solution, least the problem remain highly intractable. The fundamentals of the branch-and-bound method allow it to alleviate the immensity of some of these problems.

The branch-and-bound method, as it is seen in this research, is geared towards solving integer programming problems.

3.2.1 The Search Tree Data Structure

Branch-and-bound has as its bedrock the *search tree*.

Definition 3.1 (Tree)**.**

A tree is a data structure that is a defined by a collection of nodes/vertices linked in a hierarchical (ordered) fashion. The first of these nodes is known as the root node.

Definition 3.2 (Search Tree)**.**

A search tree is a tree data structure that acts as a mechanism to store and retrieve ordered data.

Search trees allow the efficient searching or traversal through a solution space. Tracing out a specific set of nodes in the tree is analogous to systematically constructing a candidate solution to a problem. The nodes can be thought of as parts of a solution. The inclusion of a new node in the solution, thus "growing" the solution, is the act of *branching* from a node. Each node can have several different *branches*. Branching occurs in a *depth-first search* manner.

3.2.1.1 Depth-first Search (DFS)

Depth-first Search (DFS) is an algorithm used in the searching of graphs. Starting at the root node, exploration occurs along a particular branch until a *leaf node* is reached.

Definition 3.3 (Leaf Node)**.**

A leaf node is a node in a tree data structure that has no child nodes. That is, no further branching can occur.

From a leaf node, backtracking occurs until a node is reached that has the potential to branch again. A DFS terminates when all nodes of the tree have been searched. The pseudocode for DFS is presented by Algorithm 2 :

Algorithm 2 The Recursive Depth-first Search

```
1 function DFS(G, i, visited)
2 Inputs : A graph, G = (V, E), a current node, i, a boolean array visited.
3 visited[i] ← TRUE.      % mark current node as visited.
4 for all (i, j) ∈ E do
5   if visited[j] == FALSE then
6     DFS(G, j, visited)      % recursively call DFS.
7   end if
8 end for
9 end function
```

3.2.1.2 Breadth-first Search (BFS)

Breadth-first Search (BFS) is another algorithm for graph searching. It begins at the root node and proceeds to explore all the neighbouring nodes. Then, for each of these nodes, it searches their neighbour nodes. This process is repeated until the goal has been found. Typically, the method makes use of a data structure known as a *queue*.

Definition 3.4 (Queue)**.**

A queue is a first-in-first-out (FIFO) data structure that stores ordered entities. Typical queue operations include **enqueue**, addition of an entity to the end of a queue, and **dequeue**, removing an entity from the front of a queue.

The pseudocode for BFS is presented in Algorithm 3 below:

Algorithm 3 Breadth-first Search

```
1  function BFS(G, i)
2  Inputs : A graph, G = (V, E), a current node, i.
3  Initialise a queue data structure, Q.
4  Initialise a set, Closed.
5  Q ← Q ∪ {i}      Enqueue i onto Q.
6  Closed ← Closed ∪ {i}
7  while Q is not empty do
8     v ← Q.dequeue()      v is taken out of queue.
9     if v is goal then
10       return v
11    else
12       for all edges (v, u) ∈ E do
13          if u ∉ Closed then
14             Closed ← Closed ∪ {u}
15             Q ← Q ∪ {u}
16          end if
17       end for
18    end if
19 end while
```

3.2.2 Bounds and Pruning

Occasionally, a depth-first search branching procedure ensues from the onset of the algorithm, the goal being to find an *incumbent solution*. Assuming we are dealing with a minimisation problem, this would be an *upper bound*. This solution takes its place as the best solution found thus far. Later branching in the algorithm will result in encountering further leaf nodes and thus other solutions. If a discovered solution is better than the incumbent solution it becomes the new incumbent solution. This updating of the upper bound forms a fundamental mechanism in the operation of branch-and-bound. The

status of a branch can be assessed at a node by comparing the upper bound and a *lower bound*. The lower bound is an estimate into the unexplored subset, usually by means of a linear programming relaxation. Together, these bounds provide an idea on a branch's potential.

The bounds on any point in a solution path allow us to employ a feature of branch-and-bound that separates it from raw, inefficient enumeration that comes with solely using DFS. This feature is known as *pruning*, a rather descriptive term when considering how this tool is utilised within a search tree. Rather than exploring the entire solution space, a mammoth and time consuming task for larger problems, certain solution path explorations are cut-off, or rather some branches in the tree are "pruned." Entire subspaces of the problem are essentially castaway. Assuming a minimisation problem, the incumbent solution takes on the form of an upper bound and the linear relaxation solution, at a node in the tree, the lower bound. The solution of the linear relaxation at a particular node is the best solution that can possibly be obtained for the sub-problem. Thus, if this best possible solution is not better than the upper bound then there is no benefit in branching from the current node. Branching in that subspace is halted and exploration occurs from other nodes, saving us from searching branches of the tree that will never have a better solution. The algorithm terminates when there are no unexplored regions in the solution space. At the conclusion of the algorithm, the upper bound is considered the optimal solution value.

The methodology outlined above is most synonymous with the *Lazy Branch-and-Bound* algorithm. The pseudocode in Algorithm 4 provides a blueprint for its modus operandi. X^* represents the optimal solution found during the procedure.

Algorithm 4 Outline of Branch-and-bound Algorithm - Minimisation Problem

```
1  UB ← ∞      % Initialise upper bound.
2  list ← {R}                          % Initialise depth-first search at the root node.
3  while list is not empty do
4    Select the first element, n, from list to be analysed.
5    Calculate a lower bound for this branch at node n, LB(n).
6    if  LB(n) > UB  then
7      % Prune subspace at node n, halting further exploration from this node.
8      list ← list \ {n}                                  % Pop n from list.
9    else
10     % Expand on this node, exploring the subspace further.
11     list ← list \ {n}                                  % Pop n from list.
12     list ← {n_1, ..., n_c} ∪ list                      % Prepend n's children to list.
13     if  {n_1, ..., n_c} = ∅  then
14       % if n has no children we are at a leaf node.
15       Evaluate valid solution f(X_{R,...,n}).
16       if  f(X_{R,...,n}) < UB then
17         UB ← f(X_{R,...,n})                % Update upper bound value.
18         X* ← X_{R,...,n}                   % Capture new incumbent solution.
19       end if
20     end if
21   end if
22 end while
```

Figure 3.2 shows a graphical representation of a solution space being searched during various phases of the algorithm. The images on the left are an illustration of a solution space being gradually partitioned and analysed. The corresponding search tree for the space state is displayed on the right showing the branching (partitioning), searching and pruning.

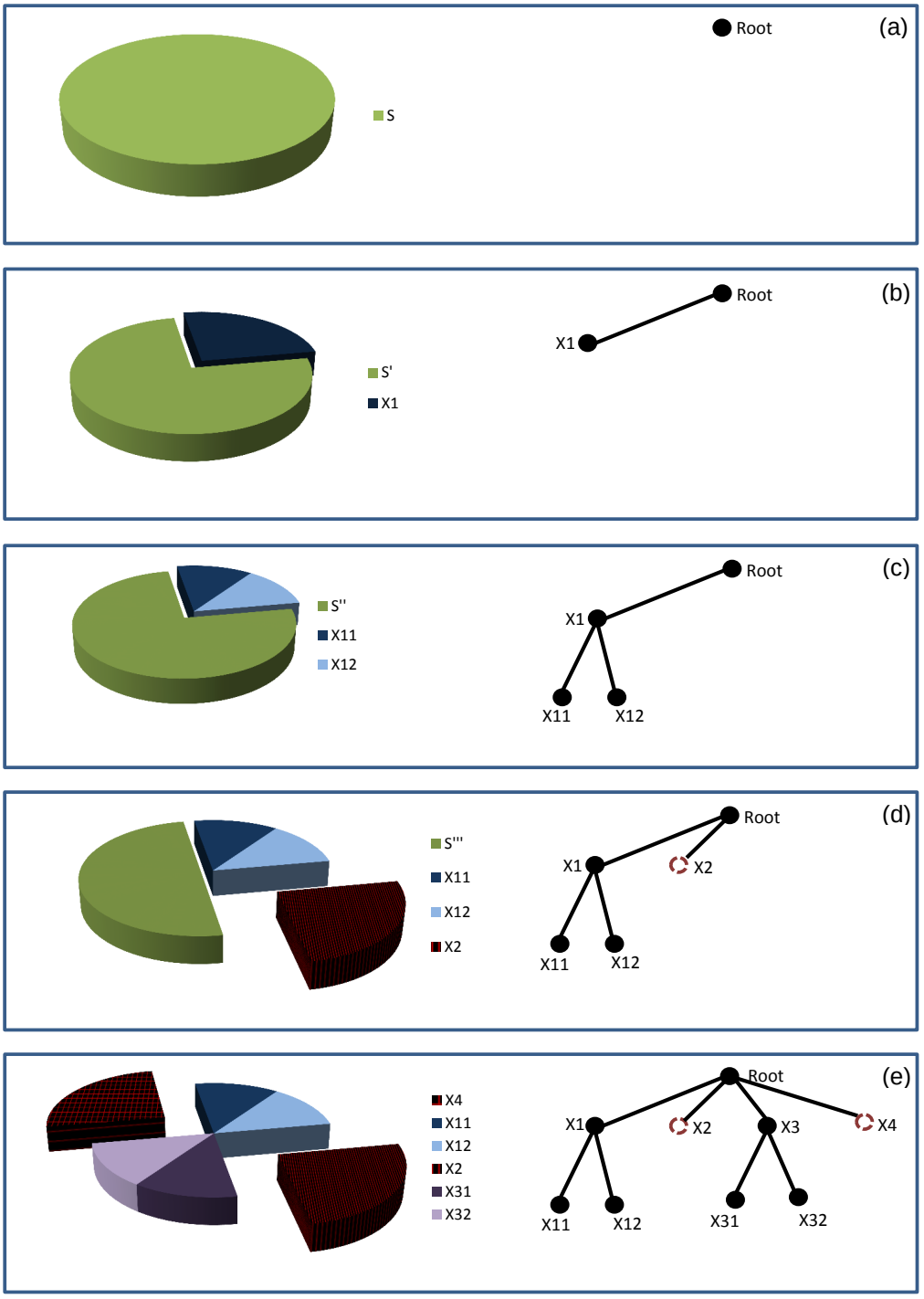

FIGURE 3.2: Example branch-and-bound Progression within a Solution Space

Figure 3.2(a) shows the first state of the algorithm. No branching has commenced and, with regards to the search tree, we only have the root node, depicting the entire, "untouched" solution space S. Figure 3.2(b) reveals an initial branching from the root

node. A subspace X_1 is now being considered by the tree search. Figure 3.2(c) is the conclusion of the depth-first expansion in the X_1 subspace. There are 2 valid solution paths within the exploration of X_1, namely X_{11} and X_{12}. One of these solution paths represent the new incumbent solution. Figure 3.2(d) displays the pruning concept. The exploration into subspace X_2 has halted because it was discovered that further expansion would definitely not yield a better solution than the current incumbent solution. This could possibly have been decided by evaluating a *linear programming relaxation* at the node, X_2. Figure 3.2(e) shows the branch-and-bound procedure at its conclusion. The entire space S has been explored. Notice further pruning at X_4 and expansion at X_3. The optimal solution is the final incumbent which could be X_{11}, X_{12}, X_{31} or X_{32}.

3.3 Branch-and-cut

Branch-and-cut is a combination of utilising *cutting plane methods* along with branch-and-bound. It forms the basis of the IBM ILOG CPLEX Studio Optimisation Software, which will be introduced in later sections of this dissertation.

Definition 3.5 (Cutting Plane Methods)**.**

Cutting plane methods are a broad set of techniques that reduces a feasible solution space by the addition of valid inequalities, often referred to as cuts.

The addition of these "cuts" allows for tighter linear programming relaxations in a branch-and-bound procedure. The relaxations will be closer approximates of the integer program which is being solved. This makes effective pruning of branches more likely and can greatly increase the efficiency of searching through a tree. Less nodes have to be explored. This combined strategy ensures that branch-and-cut will be faster than pure branch-and-bound [Albert 2006]. The pseudocode for a general branch-and-cut algorithm is presented in Algorithm 5.

Algorithm 5 Outline of Branch-and-cut Algorithm - Minimisation Problem

1 $UB \leftarrow \infty$
2 $list \leftarrow \{R\}$ % initialise at root node.
3 **while** *list* is not empty **do**
4 Select first element in *list*, n.
5 $list \leftarrow list \setminus \{n\}$ % Pop n from *list*.
6 Solve LP relaxation at node n, X.
7 $LB(n) \leftarrow f(X)$
8 **if** $LB(n) \leq UB$ **then**
9 **if** $X \in \mathbb{Z}$ **then**
10 $X^* \leftarrow X$
11 $UB \leftarrow LB(n)$
12 **else**
13 If desired, add cutting planes that are violated by X. Then return to line 6.
14 **end if**
15 $list \leftarrow \{n_1, ..., n_c\} \cup list$ % Prepend n's children to *list*.
16 **if** $\{n_1, ..., n_c\} = \emptyset$ **then**
17 % if n has no children we are at a leaf node.
18 Evaluate valid solution $f(X_{R,...,n})$.
19 **if** $f(X_{R,...,n}) < UB$ **then**
20 $UB \leftarrow f(X_{R,...,n})$ % Update upper bound value.
21 $X^* \leftarrow X_{R,...,n}$ % Capture new incumbent solution.
22 **end if**
23 **end if**
24 **end if**
25 **end while**

Chapter 4

Background on Relevant Heuristics

The previous chapter dealt with the relevant exact methods pertaining to this research. These approaches are concerned with finding the optimal solution of a mathematical program without compromise.

This chapter will introduce inexact methods that were utilised in this dissertation. Such approaches are grouped underneath the unifying banner that is **heuristics**.

4.1 Simulated Annealing

Finding the optimal solution of combinatorial optimisation problems can sometimes prove quite a precarious task. The size and/or structure of these problems can render them very difficult to solve practically, particularly if one is confined to only utilising exact solution methods. The early 1980s saw the advent of a new approximate solution method to mitigate such concerns - *simulated annealing*.

4.1.1 Inspiration for Simulated Annealing

[Kirkpatrick et al. 1983] proposed the simulated annealing method based on principles from statistical mechanics. Inspiration was drawn from the behaviour of physical systems in the annealing process found in metallurgy.

The first step to creating a crystal is to heat the raw materials to a molten state. The temperature is then reduced until a crystal structure is formed. If the temperature is

decreased rapidly the procedure fails. This is known as *rapid quenching*. The resultant crystal often has extensive irregularities and an energy level higher than that of a perfectly structured crystal. However, if the temperature is gradually lowered the likelihood of an imperfect crystal is lowered. By ensuring that the temperature does not vastly outpace the current equilibrium energy level we decrease the possibility of achieving sub-optimal crystal structures. Simulated Annealing can be likened to an algorithmic version of this process.

4.1.2 An Enhanced Local Optimisation Method

For those not well-versed in the physics of thermodynamics, simulated annealing can be thought of as an improved form of *local optimisation*. Before delving into the workings of this heuristic we need to understand the mathematical definition of a *neighbourhood*.

Definition 4.1 (δ - Neighbourhood)**.**

A δ-neighbourhood of a point $x \in \mathbb{R}^n$, denoted $N_\delta(x)$, is the set of all points x' such that:

$$N_\delta(x) : ||x' - x|| \leq \delta \tag{4.1}$$

Definition 4.2 (Local Optimisation [Johnson et al. 1989])**.**

Given an initial solution to a combinatorial optimisation problem, local optimisation refers to methods aimed at incrementally improving this initial solution through a sequence of local changes, in the neighbourhood of the initial solution.

Algorithm 6 provides an algorithmic description of local optimisation methods.

Algorithm 6 The Local Optimisation Concept

```
1 Assume an initial solution, x
2 while the neighbourhood of x is not fully explored do
3    Let x' be an unexplored neighbour of x
4    if f(x') < f(x) then
5       x ← x'
6    end if
7 end while
8 return x
```

Simulated annealing deviates from local optimisation principles by sometimes allowing changes that produce a worse solution. This is done in an attempt to try and diminish the probability of the method becoming stranded in *local optima*.

Definition 4.3 (Local optimum)**.**

A local optimum is the best solution, x^*, within a neighbourhood, $N_\delta(x^*)$. That is, in terms of a minimisation problem:

$$f(x^*) \leq f(x), \quad x \in N_\delta(x^*). \tag{4.2}$$

Local optimisation heuristics often return a local optimum that may be significantly worse than the global optimum. Figure 4.1 shows a 3-dimensional plot of a function that has several local maxima and a global maximum. Assuming the objective is to maximise, exclusively using local optimisation would heighten the risk of finding one of the local maxima (the smaller, yellow "hills") as opposed to the global maximum (the highest, red "hill"). Methods that solely rely on the fundamentals of local optimisation are more susceptible to becoming trapped in certain neighbourhoods of a solution space.

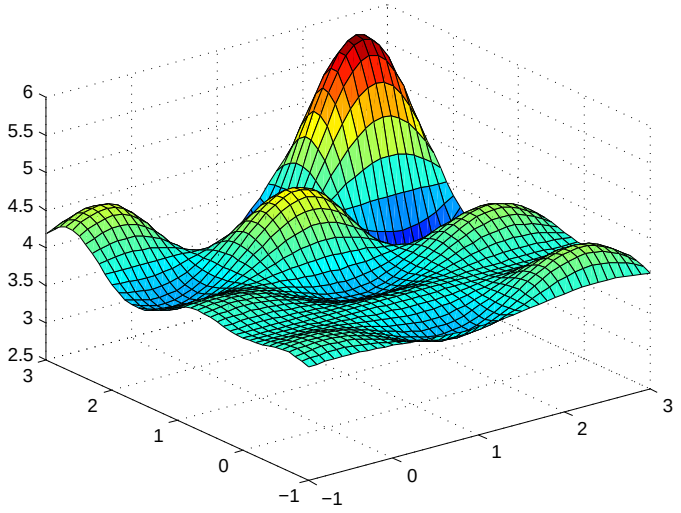

FIGURE 4.1: A Function with Local Optima Traps

4.1.3 The Analogy with Physical Annealing

Simulated annealing attempts to escape these local optima "traps" through the utilisation of a random number generator and the *temperature* parameter, modelled on the control of temperature in the physical annealing procedure.

Two further controls in the simulated annealing algorithm are *cooling* and the length of the Markov Chain, L. The Markov Chain represents the stochastic paths for the local searches in simulated annealing algorithms. We see the analogy of a physical system expressed as a mathematical system, one which aims to maximise or minimise a mathematical program. Table 4.1 paints the analogy between the physical system and the mathematical problem [Johnson et al. 1989]:

TABLE 4.1: Simulated Annealing - The Analogy

Physical System	Mathematical Program
A state	A feasible solution
Energy	Objective function value
Ground state	Optimal solution
Rapid quenching	Local search
Careful Annealing	Simulated annealing

4.1.4 Description of Algorithm

Algorithm 7 reveals the basic operation of the simulated annealing algorithm:

Algorithm 7 The General Form of Simulated Annealing

```
1  Acquire initial solution, x
2  Set initial temperature, Temp
3  while Temp ≥ ϵ do
4    for l = 1 to L do
5      Generate y, a neighbouring solution to x
6      Δ ← f(y) − f(x)
7      if Δ ≤ 0 then
8        x ← y
9        f(x) ← f(y)
10     else
11       Generate random number r ∈ (0, 1)
12       if r ≤ e^{−Δ/Temp} then
13         x ← y
14         f(x) ← f(y)
15       end if
16     end if
17   end for
18   Cool Temp
19 end while
20 return x
```

At the core of the algorithm is the acceptance criterion which is modelled around the *Boltzmann Distribution*. Assuming a minimisation problem, a new solution is accepted with probability, $\mathbb{P}_B$, according to equation (4.3):

$$\mathbb{P}_B = \begin{cases} 1 & \text{if } f(y) \leq f(x), \\ e^{-\Delta/Temp} & \text{otherwise.} \end{cases} \tag{4.3}$$

where x is the original solution and y is the new solution in the neighbourhood of x. Δ is the difference between the new solution and the current solution ie. $\Delta = f(y) - f(x)$. The *temperature* parameter is given by $Temp$.

If $f(y)$ is better than $f(x)$ then the new solution is accepted. In such a situation, the principles of local optimisation are being adhered to.

For a given temperature, if $f(y)$ is only slightly worse than $f(x)$ (Δ is small) then the probability of acceptance will be greater. Higher $Temp$ values will also encourage more solutions to be accepted rather than rejected. Figure 4.2 is a colourmap plot that reveals the effect that different degrees of Δ and temperature can have on acceptance probabilities. Notice the high probability of acceptance especially for the higher temperature values.

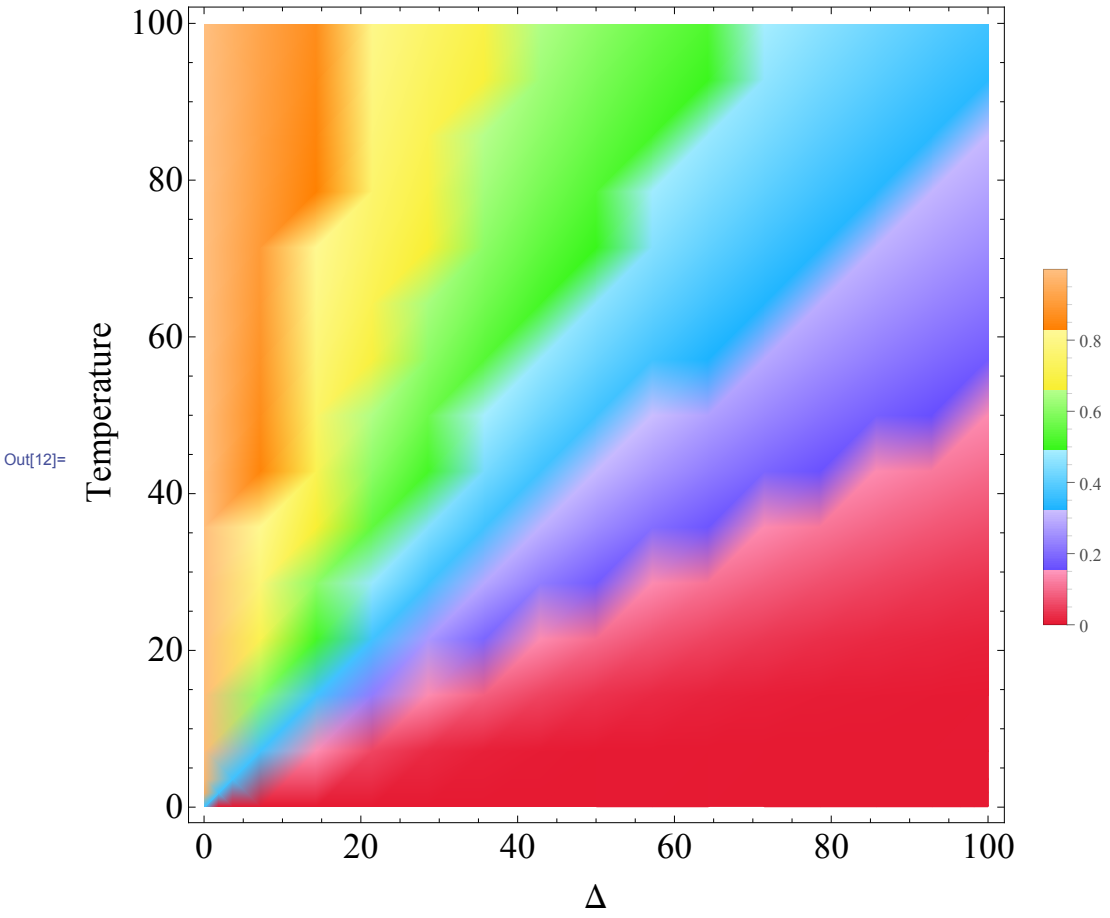

FIGURE 4.2: A Colourmap of the Acceptance Probability

The Metropolis algorithm makes up the inner loop of the Simulated Annealing algorithm. [Metropolis et al. 1953] initially performed Monte Carlo simulations of physical systems. They attempted to simulate the behavior of gases in an external heat bath at a fixed temperature, a rather prudent example considering the analogy simulated annealing shares with physical annealing.

The outer loop can be thought of as the "careful annealing" equivalent in the optimisation problem. The temperature is "cooled" slowly through each iteration. In this way, the chances of the algorithm becoming stranded in a sub-optimal solution are diminished. This cooling continues until the $Temp$ becomes less than some value ϵ. The algorithm then terminates.

Of course, this is the generic form of simulated annealing. There are a number of modern adaptations of it, especially in modern literature in operations research. For example, sometimes this *Temp* parameter may be heated and not just cooled, depending on the quality of solutions found in the algorithm's operation.

4.2 Tabu Search

4.2.1 Introduction

Similar to Simulated Annealing, *Tabu Search* is a metaheuristic procedure that improves upon classic local search methodologies. The heuristic attempts to improve the efficiency of searching a solution space, thus increasing the likelihood of finding optimal solutions, or at least good solutions. The metaheuristic's name gives a clue to the primary feature that it employs in its algorithm.

Definition 4.4 (Tabu)**.**

Tabu is a variant of taboo which refers to something that is prohibited, forbidden or excluded.

The tabu-element of the algorithm is present in the preventative measures that are taken to prohibit redundant searching of portions of a solution space. For a set number of iterations a particular move is "forbidden." The algorithm is less likely to come across regions that have already been explored.

Tabu Search was originally proposed in [Glover 1986]. It has since found widespread use in a number of combinatorial optimisation problems, such as the *job shop scheduling problem* and the *capacitated plant location problem.*

4.2.2 Memory Structures - Usage and an Ancient Analogy

Tabu search makes use of historical memory structures known as *tabu lists.* It is these lists that allay the chances of searching in explored regions. The tabu search starts by proceeding to local optima. Once a local optima is reached the tabu list has a recording of recent steps and prevents the search path from being reversed.

Simulated annealing shares an analogy with physical annealing. Comparisons can be drawn between each aspect of each "process" rather easily, especially since the inspiration

for simulated annealing came from physical annealing. One might think of the story of the ancient Greek hero, Theseus, and his adventures in the labyrinth of the Minotaur as an analogy for tabu search.

Theseus had to search the labyrinth for the monstrous Minotaur, a half bull half man creature. The labyrinth was a maze where heroes or prisoners could easily become lost. Fortunately, a princess called Adriana provides Theseus with a ball of thread which he trails behind him during his search in the labyrinth. This was intended to help Theseus find his way out of the maze but it also availed to him a method to prevent needless backtracking in his search. Figure 4.3 shows an artists impression of the story of Theseus, searching the labyrinth with his ball of thread.

FIGURE 4.3: A Medieval Interpretation of Theseus in the Labyrinth [Burne-Jones 1861]

The ball of thread that Theseus uses is essentially a memory structure. It ensures that he does not revisit areas he has already been to. In the tabu search, the tabu list performs a similar function, preventing needless backtracking.

TABLE 4.2: Tabu Search - An Analogy

Theseus and the Minotaur	Mathematical Program
The search for the Minotaur	The search for the optimal solution
The Labyrinth	The solution space
Ball of thread	Tabu list

4.2.3 Description of Algorithm

As mentioned previously, Tabu search is initially greedy in its approach, much like most local search methods. However, when it comes across a local optimum the tabu list that it has built up along its route allows it to proceed with a series of non-improving steps away from the local optimum it has recently encountered. The most common tabu list scheme is a short-term memory configuration where only the latest steps are recorded and prohibited [Gendreau et al. 2005].

An important consideration in tabu search is that the tabu lists may be too prohibitive. They have the potential to prevent a lucrative move that will result in a very good solution. A *classical aspiration criterion* is a measure put in place in order to try and counteract such an situation. The simplest classical aspiration criterion states that a move that is on the tabu list can be initiated if it results in a solution that is better than any solution found thus far in the search [Gendreau et al. 2005].

Consider an optimisation problem to minimise a function $f(x)$. Let x represent the current solution, x^* the best known solution with objective function value $f(x^*)$, $N(x)$ a neighbourhood of x and $\tilde{N}(x) \subseteq N(x)$ a set of permissible moves in the neighbourhood $N(x)$ ie. those moves that are non-tabu or satisfy the classical aspiration criterion. The simple tabu search algorithm to solve this generic problem is presented in Algorithm 8 below:

Algorithm 8 A Template for Tabu Searches [Gendreau et al. 2005]

```
1  Acquire initial solution, x_0, and evaluate f(x_0)
2  Define a tabu list, tabu ← ∅
3  Deine 𝕋̄ as the maximum number of records tabu keeps
4  x ← x_0
5  f(x) ← f(x_0)
6  x* ← x_0
7  f(x*) ← f(x_0)
8  while termination criterion not satisfied do
9     x ← argmin_{x'∈Ñ(x)} f(x')
10    if f(x) ≤ f(x*) then
11       f(x*) ← f(x)
12       x* ← x
13       Record this move in tabu
14       if size(tabu) > 𝕋̄ then
15          Delete oldest record in tabu
16       end if
17    end if
18 end while
19 return x
```

There are a number of termination criteria that one could specify for tabu search. [Gendreau et al. 2005] specify some example termination criteria which are:

1. After a predefined number of iterations or CPU time the algorithm terminates.
2. Termination after a set number of non-improving iterations.
3. Termination when the objective function value reaches some threshold.

4.2.4 Additional Features to Tabu Search

The basic concepts for tabu search described above can sometimes not be efficient enough to solve some especially complex problems. There are some more features that can be added to the tabu search to make it even more effective.

4.2.4.1 Intensification Strategies

The general idea by intensification strategies is that the most encouraging regions of the solution space are explored more frequently. The hope is that the revisiting portions of these neighbours may present a path to the optimal solution.

Intensification can be achieved by using a "recency" array that will keep track of the number of consecutive iterations certain parts of the solution remained fixed. An alternative is to change the neighbourhood structure of the search allowing for potentially more powerful or exploring steps [Gendreau et al. 2005].

4.2.4.2 Diversification Strategies

The primary concern with tabu search should be that local search methodologies form a core part of its functioning. The majority of the steps in the search will be greedy and although the tabu lists mitigate the ill-fated paths that greedy approaches so often result in, it is sometimes not enough. Diversification further aids searches from becoming stuck in redundant local neighbourhoods. These strategies force the algorithm to search in regions that have not been explored frequently. Frequency memory structures can be employed to restart the search in these unexplored portions. This is known as *restart diversification*. The other form of diversification is known as *continuous diversification*, which takes place throughout the generation of steps in the search but adds in a factor that prefers steps that have been selected less frequently.

4.2.4.3 Traversal of Infeasible Solutions - Constraint Relaxation

A problem may have constraints that restrict the searching process too much resulting in sub-optimal solutions. Relaxing certain constraints synthesises a larger solution space and an environment for less complex neighbourhoods - generating new solutions takes less time.

This approach is implemented by removing the constraints from the polyhedron and adding them to the objective function with penalty weights for solutions in which they are violated. Penalty weightings are sometimes adjusted dynamically during the course of the search. If there is a certain number of successive violations the weightings are increased. Another approach is to set the penalties in such a fashion that it aids the diversification process. The search has the potential to then cross over the feasibility boundary into parts of the solution space that would take ages to reach otherwise. This technique is known as *strategic oscillation* [Gendreau et al. 2005].

4.3 Conclusion

Tabu search and simulated annealing are powerful metaheuristics that can be applied to a great deal of optimisation problems. Furthermore, they offer a certain sense of flexibility by allowing several extensions upon the classic form of the algorithm, further increasing their efficiencies. They are indeed algorithms that are constantly evolving and improving through their application in hybrid methods, industry and research.

Chapter 5

A Background to Open-pit Mine Planning

The discipline of combinatorial optimisation and the devices and algorithms employed within the discipline have been laid out in Chapters 2, 3 and 4. These find application in open-pit mine planning and design. This chapter will provide a background on open-pit mine planning, revealing the extent of the imperious application of the content described in the previous 3 chapters.

5.1 A Model for the Orebody

A fundamental task in the development phase of an open-pit mine is *mine design*. Mine design has at its heart the representation of the *orebody*.

Definition 5.1 (Orebody)**.**

An orebody is a well-defined collective mass of rock that contains amounts of ore that can be mined economically and technically.

Geologists, using various techniques and analysis of acquired samples, are able to construct a 3 dimensional representation of the orebody. This visualisation takes on the form of a 3D array of blocks, the *block model*, thus discretising the orebody (see Figure 5.1). Each block in the model has its own set of attributes. These attributes may differ depending

on the mineral(s) being mined but will typically include characteristics like rock tonnage, impurity level, ore content etc. [Fricke 2006].

The discretization of the orebody has inherently provided mine planners with a more functional depiction of the orebody. There is now a definitive medium in which mine design can progress, opening the doorway to mathematical optimisation. Essentially, each of the blocks in the block model are now elements in the 3D array that are defined as variables in a mathematical program.

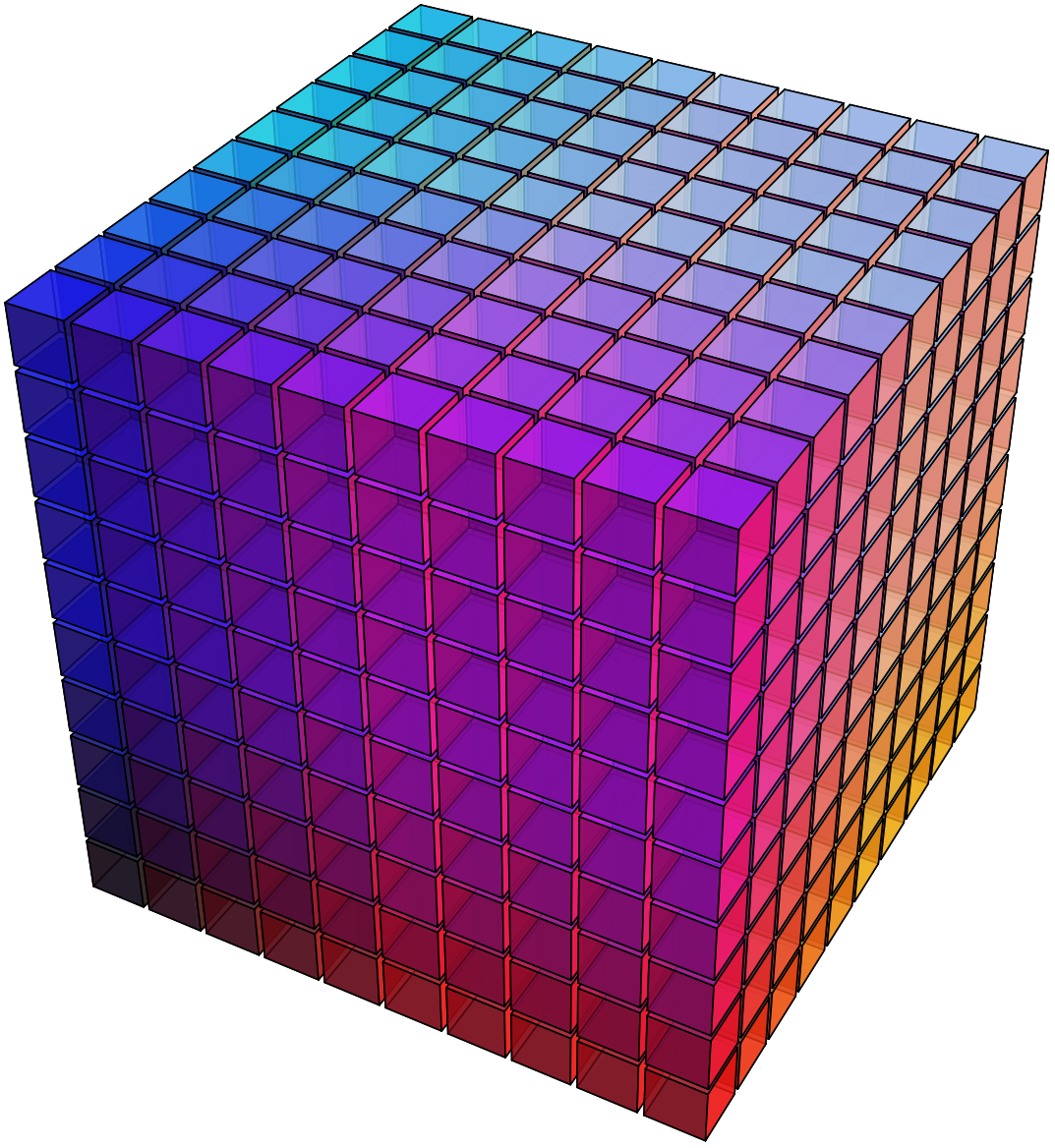

FIGURE 5.1: A Graphical Representation of an Orebody Block Model

In the past, mining projects have generally employed this model within a deterministic framework ie. blocks are given attribute values that are deemed to be certain (eg. the ore content in blocks). However, one cannot ignore the geological uncertainty that comes

with, not only the prospecting phase of mining projects but also the construction of this very model. The attribute values for the blocks are derived from estimative methods, such as interpolation. Therefore, one has to call into question the consistency that a deterministic block model provides in robust mine planning and design. As a result, there are now some models that involve statistical concepts to account for this degree of uncertainty and to allow for multiple realisations of the orebody in the subsequent stages of mine planning. The inclusion of this concept in mine planning would undoubtedly increase the complexity.

The orebody has now been portrayed in a workable medium from which mine planning problems can begin to be solved. Perhaps the first cardinal question that needs to be answered is what part of the orebody should be mined, or in the model's context, what blocks should be mined. In order to solve this we first need to understand the concept of *cutoff grade*.

5.1.1 Cut-off Grade : "The Ore -Waste Discriminator"

In mining, cutoff grade refers to the amount of mineral present in ore in order to deem it economically feasible to mine [Fricke 2006]. Material with a mineral content below this cutoff grade is regarded as waste while material with mineral content above the cutoff grade is distinguished as ore. Cutoff grade is a fundamental concept in allowing the valuation of blocks in the model.

Research in cut-off grade optimisation has stagnated somewhat since 1988 with the work of [Lane 1988]. He segregated a mining operation into 3 stages - the mining, processing and marketing phases. For each of these, a cut-off grade was then determined with the aim to maximise the mining project's cash flow. [Minnit 2004] describes the method of [Lane 1988] and [Lane 1997] as an elegant yet simple approach to cut-off grade optimisation in cases where there are lower grades. Despite the succinct mathematics involved in this method it is not extensively applied in determining cut-off grades.

[Kumral 2012] makes use of a simple formula for calculating cut-off grade:

$$\text{cut-off grade} = \frac{\text{process cost}}{\text{price} \times \text{recovery}} \tag{5.1}$$

5.1.2 Block Valuation

Each block can be given a value based on the net balance realised given its ore content, extraction costs and processing costs. A block's economic value (BEV) can be reduced to the following equation:

$$BEV = \text{revenue} - \text{costs} \tag{5.2}$$

Revenue can typically be stated as:

$$\text{revenue} = \text{metal content} \times \text{recovery} \times \text{metal price} \tag{5.3}$$

There can be more than one revenue stream. For example, if the we are dealing with Gold-Copper ores then there is a gold associated revenue and a copper associated revenue. Recovery is the percentage of the metal content that is salvageable.

Costs can be mainly the cost of mining the material and processing the material:

$$\text{costs} = \text{mining costs} + \text{processing costs} \tag{5.4}$$

Mining costs tend to increase with the hardness of the rock and depth of the mining. Processing costs usually increase with complexity of the ore. For example, gold-copper ores are more costly to process than gold ores. Indeed, polymetallic ores are more costly to process than their single metal ore counterparts.

Waste blocks have a negative value indicating that the extraction cost is greater than the revenue from the recoverable ore content of the block. *Ore blocks* are those that have a revenue greater than the extraction cost. There are also *marginal blocks* which are blocks that have a revenue roughly equal to the cost associated with extracting them.

A co-requisite to the valuation of a block (eg. assigning a ZAR, USD or GBP value) is to have some idea of the expected price of the metal in question. This results in a rather conspicuous risk since metal prices, on world markets, have historically been quite volatile and will most certainly change over the duration of the mining venture, potentially resulting in significant inaccuracies. Despite this, a single price is usually employed to derive the value of each block in the model [Fricke 2006]. This issue is perhaps better emphasised by investigating the gold price since the 1940s. As seen earlier in this dissertation with Figure 1.10, the gold price, like many metal prices, has been volatile. Open-pit gold mining ventures that assume a constant price throughout

the years of its operation are failing to accommodate for the *metal price uncertainty* in solving problems.

5.2 The Ultimate Pit Problem

5.2.1 An Introduction to the Ultimate Pit Problem (UPL)

Open-pit mining projects involve significant amounts of risk and expenditure over a prolonged period of time. Before this ambitious and expensive undertaking can commence, it is sensible to decide **what** should be mined in order to make the open-pit mining project as successful or profitable as possible.

Solving the *Ultimate Pit Problem* (henceforth referred to as UPL), although a small constituent of the mine design and planning process, is a principal component of the entire enterprise [Fricke 2006]. The objective is to find the ultimate contour of the pit that results in the maximum profit from extracting the corresponding material contained within this final shape. Additionally, this contour must satisfy the safe wall constraint. A final pit outline can be obtained by smoothing out the ultimate contour providing the overall shape of the open-pit mine [Caccetta and Hill 2003]. The last stage in Figure 1.1 shows the delineation of the ultimate pit.

[Lerchs and Grossmann 1965] were the first to mathematically describe and attempt to solve the ultimate pit problem. They employed a graph theoretic as well as a dynamic programming approach.

From the time that this initial work was conducted, there has been a plethora of research geared towards finding the optimal solution of the ultimate pit problem. It may be apt to investigate some of the fundamental graphing concepts used to account for important constraints that will arise in the solving of the ultimate pit problem.

5.2.2 The Modelling of Block Precedence Utilising Graph Theory

[Lerchs and Grossmann 1965] first required a mathematical representation of the block model to solve the ultimate pit problem. This was done by portraying the block model, along with its relationships, as a *directed acyclic graph* : $G = (V, E)$.

Definition 5.2 (Directed Graph - Figure 5.2)**.**

A directed graph, or a digraph, is a network of vertices and edges where the edges have an associated direction attribute from one vertex to another vertex.

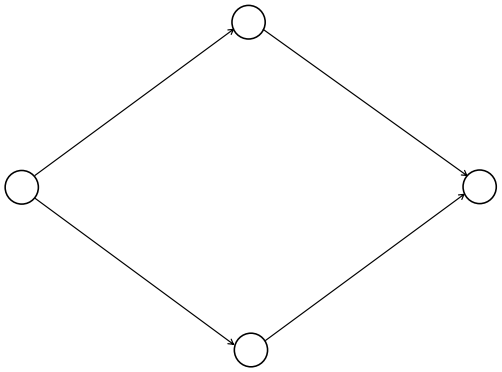

FIGURE 5.2: A Simple 4 Vertex Digraph

Definition 5.3 (Directed Cycle - Figure 5.3)**.**

A directed cycle in a digraph is a permutation of directed edges that start and end at the same vertex.

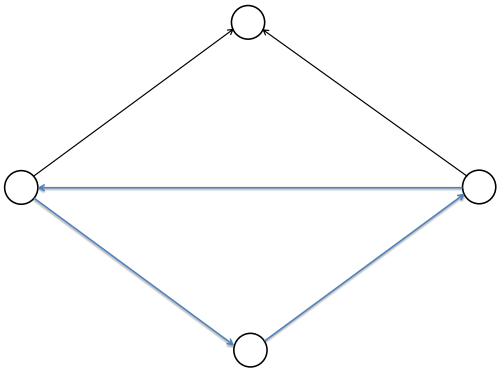

FIGURE 5.3: A Directed Cycle

Definition 5.4 (Directed Acyclic Graph - Figure 5.4)**.**

A directed acyclic graph, or a DAG, is a directed graph that is **devoid** of directed cycles. It is not possible to start at a vertex v and find a route of directed edges that return to v.

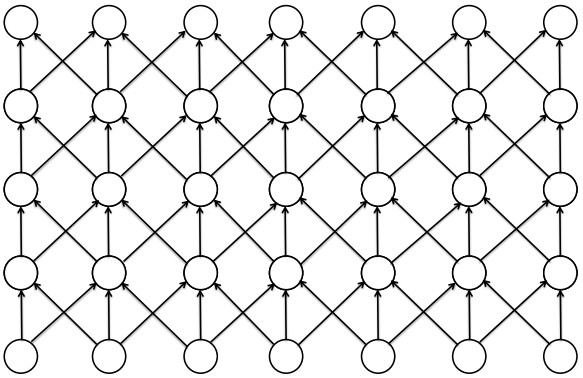

FIGURE 5.4: A Directed Acyclic Graph

V represents the vertices set and contains all the blocks in the block model. E represents the set of edges or arcs that connect vertices, $E \subseteq V \times V$. This edge set is integral to the model because it captures the precedence relationships between vertices ie. between the blocks in the model. Essentially, the set E exists to ensure that a specific block can only be removed if its respective overlying blocks have been extracted and if its removal continues to satisfy safe slope constraints.

Figure 5.5 shows the precedence relationships that can be forced on block extraction. The **left** graphic reveals a sequencing rule where the overlying 5 depicted blocks (in red) need to be extracted in order to be able to feasibly remove the green block. The **right** graphic portrays the sequencing rule where all 9 overlying blocks (in red) need to be removed before the blue block can be excavated.

Taking into account the above description of precedence, there exists a directed edge $(i, j) \in E$ between the blue or green block i and going to each of the red blocks j. This implies that a block i can only be mined when the j blocks have been mined, therefore capturing the precedence constraints. Figure 5.5 is an illustration of *immediate precedence*. Mathematically, this is captured in a set format definition.

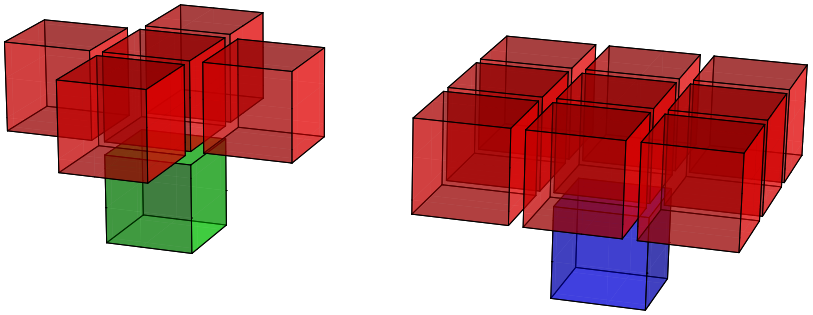

FIGURE 5.5: Immediate Precedence Set Representation

Let $\rho_i = \{j \in V : (i,j) \in E\} \cup \{i\}$ denote the immediate precedence set for block i, blocks j being the predecessor blocks. This set would contain the red blocks as seen in Figure 5.5, depending on whether it is the 9 block rule or the 5 block rule.

Let $C(G)$ represent the *transitive closure* of the graph G.

Definition 5.5 (The Transitive Closure of a Graph)**.**

Consider a graph, $G = (V, E)$, where $(i,j) \in E$ and $(j,k) \in E$: $i, j, k \in V$ ie. $i \to j$, $j \to k$. A graph that has the addition of all edges (i,k) is defined as the transitive closure of the original graph.

This defines a new graph $C(G) = (V, E^+)$, where E^+ is the original edge set with addition of edges (i,j) if there is a directed path from a block i to a block j in the original graph G.

Let $P_i = \{j \in V : (i,j) \in E^+\} \cup \{i\}$ denote the entire precedence set for block i. This set includes **all** blocks j, and the block i, in the entire block model that need to be removed in order to deem block i removable.

Figure 5.6 displays an entire precedence set. The left figure shows a 3-dimensional view of the blocks that need to be extracted so as to mine the blue block. The right figure shows a more front-on-view, effectively portraying the consistent slope angle which in this case is 45°. The slope angle value can differ depending on geotechnical slope parameters. It is important to recall that an entire precedence set A_i includes predecessors of block i as well as block i itself.

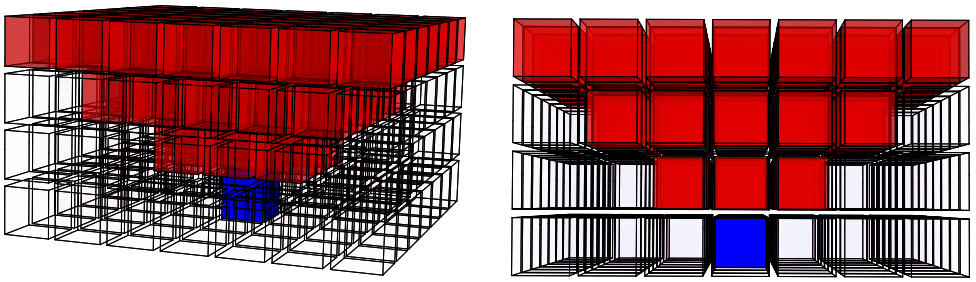

FIGURE 5.6: Example of an Entire Precedence Set

5.2.3 The Mathematical Formulation of the Ultimate Pit Problem

The mathematical formulation of the ultimate pit problem can be presented as follows:

$$\max \sum_{i=1}^{N} p_i x_i \tag{5.5}$$

subject to:

$$x_i \leq x_j \quad \forall\, (i,j) \in E, \tag{5.6}$$

$$x_i \in \{0,1\} \quad i = 1, ..., N. \tag{5.7}$$

$$x_i = \begin{cases} 1 & \text{if block } i \text{ is mined in the solution to the ultimate pit,} \\ 0 & \text{otherwise.} \end{cases}$$

Equation (5.5) is the objective function of the Ultimate Pit Problem. It seeks to maximise the economic benefit from inclusion of blocks in the ultimate pit limit. p_i refers to the economic gain realised from mining block i. $N = |V|$ which denotes the number of blocks in the block model. Equation (5.6) is essentially a mathematical statement of Figure 5.5. It embodies the precedence constraint, ensuring that prerequisite overlying blocks are mined if the block i in question is to be extracted, and the preservation of safe wall slope constraints. The graph network plays a pivotal role in the functioning of equation (5.6). Finally, equation (5.7) is the integrality constraint enforcing binary decision variables.

5.2.4 A Brief Summary on Solution Methods to the Ultimate Pit Problem

The approaches to solving the ultimate pit problem can be segregated into 4 types [Fricke 2006]:

1. Graph Theoretic Approaches
2. Network Flow Approaches
3. Linear Programming Approaches
4. Heuristic Approaches

The method contained within [Lerchs and Grossmann 1965] is a prime example of a graph theoretic approach. They extensively utilise the precedence graph $G = (V, E)$ along with the concepts within the creation of this graph ie. precedence modelling. They show that deciding which blocks to mine in order to maximise profit reduces to finding the *maximal closure* of the graph G.

Definition 5.6 (Closure of a Graph - [Picard 1976])**.**

A closure of a graph is defined as a subset of vertices, η, such that if a vertex is in η then all its successors are also in η.

Definition 5.7 (Maximal Closure of a Graph - [Picard 1976])**.**

Should each vertex i in a graph have an associated weighted-value, p_i, then a maximal closure, η^*, of a graph is defined as that **closure** that has: $\max \sum_{i \in \eta^*} p_i$.

This requires finding the vertices that produce a maximum weight set and ensuring that all successors of these vertices are also included in this maximum weight set.

[Picard 1976] suggested a maximum flow algorithm as an alternative to the graph theoretic approach of [Lerchs and Grossmann 1965]. It was shown that by converting the original network G into a related network $\bar{G}$ that the problem changes from one that searches for the maximum closure of a graph to a maximum flow problem on a related network. [Picard 1976] demonstrated how the maximum closure of G is actually the *minimum cut* of $\bar{G}$.

Definition 5.8 (Minimum Cut of a Graph - [Picard 1976]).

Considering a graph, $G = (V, E)$, a cut is a partitioning of V into 2 disjoint subsets, $(\mathbb{S}_1, \mathbb{S}_2)$. The minimum cut is the cut that results in the smallest sum of weighted edges in the sets $\mathbb{S}_1$ and $\mathbb{S}_2$.

Obtaining this $\bar{G}$ is achieved using sink nodes and dummy source nodes. A comparison between both methods revealed the method of [Picard 1976] to be typically faster than the graph theoretic approach of [Lerchs and Grossmann 1965].

The ultimate pit problem can be formulated as a linear program. This approach has however been deemed intractable due to the poor performances of solving this form on realistically sized instances. Real open-pit mines can have up to 5×10^6 blocks in the model [Fricke 2006].

The algorithm found in [Lerchs and Grossmann 1965] often underperforms on larger problem sizes. This led researchers to exploring the use of heuristics to solve the ultimate pit problem. The *floating cone method* is one of the more popular heuristics. [Kim 1978] provides a compressed explanation of the method and its various modifications and extensions.

5.2.5 The Traditional Approach to Open-pit Mine Production Scheduling

The first step that mine planners take is to determine a set of pushbacks. Pushbacks can be thought of as extraction phases for a production schedule.

Firstly, a decreasing sequence of cost vectors is determined. Denote these cost vectors $\mathbb{C}^1 > \mathbb{C}^2 > ... > \mathbb{C}^k$. Each of these cost vectors represent sequential portions of the block model. Each of these partitions of blocks are solved as an UPL problem, resulting in the solutions $x^1, ..., x^k$. [Lerchs and Grossmann 1965] showed that $\mathbb{C}^1 > \mathbb{C}^2 > ... > \mathbb{C}^k \Rightarrow f(x^1) \leq f(x^2) \leq ... \leq f(x^k)$. The solutions actually correspond to a sequence of pits with $P^1 \subseteq P^2 \subseteq .... \subseteq P^k$. Pushbacks are then determined by taking the difference between ensuing pits ie. $F^1 = P^1, F^2 = P^2 - P^1, ..., F^k = P^k - P^{k-1}$. Now that the pushbacks have been defined, blocks at the same vertical level are grouped together. These are known as bench-phases. The bench-phases are formulated in such a manner that they do not exceed resource and mining capacities per time period. Material in each bench-phase is allocated a destination. This destination is determined by the cutoff grade which allows for the discrimination between ore and waste. The final touches to

the scheduling plan are then applied including smoothing out pushbacks for geometric operational requirements and the designing of ramps and haulage roads.

This established approach to block scheduling has some disadvantages:

1. Processing capacity and the time value of money (NPV) are ignored until only after the bench-phases have been defined.
2. Blocks are independently scheduled from their multiple possible processing options.
3. Block destination decisions are based purely on cutoff grade criteria rather than also considering capacities, time and how other blocks are processed.

5.3 The Open-pit Mine Production Scheduling Problem

5.3.1 A More Comprehensive Model for Open-pit Mine Planning

An open-pit mining project is most assuredly more sophisticated than the relatively simple question that the ultimate pit problem poses. Its model formulation is relatively uncomplicated when comparing it to other operational research problems and considering the enormity of the open-pit mining, see equations (5.5)–(5.7). As listed in the disadvantages of the traditional approach to open-pit mine planning, there is a noticeable absence of a temporal presence in the ultimate pit problem and thereby there is the omission of the decisions of **when** specific blocks should be extracted. There are also various other factors that are not accounted for in the ultimate pit problem. To list some of these:

1. Mining Constraints - The net weight of the blocks extracted during a particular time period should be greater than some minimum limit. This alleviates unbalanced mining flow issues during mining operations. Additionally, the total weight mined should also not exceed an upper limit. This upper limit value accounts for the mining equipment capacity available in that time period.
2. Processing Constraints - The total weight of blocks designated as ore mined during a period should be at least equal to the minimum amount required to supply the processing facility. There is also an upper processing limit that represents the processing plant's capacity to process amounts of ore.
3. Metal Production Constraints - A time period should see that the metal content, realised from the processing of ore, not higher than the amount that can be sold during that period and not less than a minimum amount. It should be noted

however, that metal production constraints do not apply to precious metals such as gold and silver.

The *Open-pit Mine Production Scheduling Problem* (OPMPSP) has the facility to account for such considerations, attempting to answer the question of "when blocks should be mined given a set of constraints so as to maximise economic gain." Constructing a production schedule is an integral and prevailing constituent in open-pit mine planning.

5.3.1.1 Financial Theory in Mine Planning with the OPMPSP

A ubiquitous financial principle in the OPMPSP is the *time value of money*.

Definition 5.9 (The Time Value of Money)**.**

The financial concept that money accessible at the present time is more valuable than an equivalent amount in the future.

The notion being that, provided money can earn interest, an amount received sooner is more valuable than if it was received later.

An intrinsic equation for this financial principle is the *present value* formula:

$$PV = \frac{FV}{(1+r)^t} \tag{5.8}$$

PV and FV are the present and future values respectively. t is the time periods and r is the discount rate.

Definition 5.10 (Present Value)**.**

The current value of a future cash flow at a specified discount rate.

Where the UPL problem sought to maximise the economic gain realised from delineating the final pit shape the OPMPSP seeks to maximise the *Net Present Value* (NPV) of the mining venture. That is, it seeks to maximise the present value of the economic gains realised from the blocks designated for extraction.

Definition 5.11 (Net Present Value)**.**

Often used as a measure of a project's profitability, NPV calculates the net present worth of cash flows by *discounting* them.

$$NPV = \sum_{t=1}^{T} \frac{R_t}{(1+r)^t} \tag{5.9}$$

where R_t is the cash flow at a time t. $(1+r)^t$ is the discounting term of the cash flow R_t with respective discount rate r.

Discounting affords the ability to better value the extracted block as a function of its extraction date. Realising \$40 000 from the extraction of a block today will not have the same value as realising \$40 000 from extracting that block several years from now. A later section on the formulation of the OPMPSP will highlight the utilisation of NPV in this problem.

5.3.2 Formulation of the Open-pit Mine Production Scheduling Problem

It is believed that the first comprehensive attempt to solve the open-pit mine production scheduling problem, utilising exact optimisation techniques, was by [Johnson 1968]. A number of capacity constraints were considered. These included: (i) Hours available for an equipment type; (ii) Waste handling capacity; (iii) Stockpiling capacity; (iv) Capacities of refinery plants, concentrators, maintenance facilities; (v) blasting man-hours and others [Chicoisne et al. 2012].

A general formulation of Johnson's model is presented below:

$$\max \sum_{i=1}^{N} \sum_{d=1}^{D} \sum_{t=1}^{T} v_{idt}(x_{idt} - x_{i,d,t-1}) \tag{5.10}$$

subject to:

$$\sum_{d=1}^{D}\sum_{i=1}^{N} a_{idtr}(x_{idt} - x_{i,d,t-1}) \leq c_{rt} \quad r = 1, .., R, \quad t = 1, ..., T, \tag{5.11}$$

$$\sum_{t=1}^{T}\sum_{d=1}^{D}(x_{idt} - x_{i,d,t-1}) \leq 1 \quad i = 1, ..., N, \tag{5.12}$$

$$x_{idt} \leq x_{jdt} \quad \forall (i,j) \in E, \quad t = 1, ..., T, \quad d = 1, ..., D, \tag{5.13}$$

$$x_{i,d,t-1} \leq x_{idt} \quad i = 1, ..., N, \quad t = 1, ..., T, \quad d = 1, ..., D, \tag{5.14}$$

$$x_{id0} = 0 \quad i = 1, ..., N, \quad d = 1, ..., D, \tag{5.15}$$

$$0 \leq x_{idt} \leq 1 \quad i = 1, ..., N, \quad d = 1, ..., D. \tag{5.16}$$

x_{idt} is the variable that expresses the fraction of block i that is allotted to destination d in a time period t or before. v_{idt} is the profit realised per unit of block i going to destination d in the time period t. a_{idtr} refers to the amount of resource r that is used in a time period t per unit of block i sent to destination d. c_{rt} indicates how much of resource r is available in the time period t [Chicoisne et al. 2012].

Equation (5.10) is the objective function of Johnson's model. It involves maximising the NPV. The resource constraint(s) are captured by equation (5.11). The decision variables are setup in such a way to indicate whether a block has been mined **by** a certain time period. Equation (5.12) accommodates for the reserve constraint which ensures a block can only be entirely mined once. Equation (5.14) maintains the validity of the mined "by" concept. If a block is mined by a time period $t-1$ it must then logically be mined by a time t and beyond. Equation (5.13) incorporates the graph network that accounts for the precedence constraint. No blocks have been mined by the time period $t = 0$ and thus we have equation (5.15). Lastly, equation (5.16) represents the integrality constraint of Johnson's model.

A real world open-pit mine production scheduling problem consists of many blocks, destinations and capacity constraints. The increasing of the resolution of the time divisions only aggravates the resulting complexities from these plentiful variables. In most research and literature a more simplified version of Johnson's model is usually presented. It is a model that decides the destination of blocks a priori, as is customary in today's economic block model constructs. We take note of the omittance of d in the model below as well as the introduction of the binary constraint (5.22):

$$\max \sum_{i=1}^{N}\sum_{t=1}^{T} v_{it}(x_{it} - x_{i,t-1}) \tag{5.17}$$

subject to:

$$\sum_{i=1}^{N} a_{ri}(x_{it} - x_{i,t-1}) \leq c_{rt} \quad t = 1, ...T, \quad r = 1, ..., R. \tag{5.18}$$

$$x_{it} \leq x_{jt} \quad \forall (i,j) \in E, \quad t = 1, ..., T. \tag{5.19}$$

$$x_{it} \leq x_{i,t+1} \quad i = 1, ..., N, \quad t = 1, ..., T-1. \tag{5.20}$$

$$x_{i0} = 0 \quad i = 1, ..., N. \tag{5.21}$$

$$x_{it} \in \{0, 1\} \quad i = 1, ..., N, \quad t = 1, ..., T. \tag{5.22}$$

Chapter 6

Literature on the Open-pit Mine Production Scheduling Problem

There has been a barrage of research conducted on the open-pit mine production scheduling problem since the initial attempt by [Johnson 1968]. Research has been conducted in various facets of the problem including "pure" attempts at finding an optimal block sequencing schedule (eg. [Amaya et al. 2009]), improving the formulation capabilities (eg. [Kumral 2011]), finding better optimality measures, maximising equipment efficiency and others.

In terms of the research done on solution, it is widely recognised that exact methods struggle on realistically sized problems however, there are proponents for both methods [Lamghari et al. 2012]. The subsequent sections will seek to briefly outline some of the types of approaches that have been employed in the open-pit mine production scheduling problem. Many use a variety of techniques but are grouped according to what is believed to be their primary element in terms of the method employed.

6.1 Cutting Planes and Strengthening Formulations

[Caccetta and Hill 2003] utilise a branch-and-cut (see Section 3.3) exact solution approach to solve their mixed integer programming formulation. Their formulation uses the "by" decision variable format, similar to equations (5.17)–(5.22), to indicate if a block is mined by a certain time period. It is a particularly noteworthy research paper because the authors claim to be able to solve a problem instance of up to 250 000 blocks, something that had proven elusive to achieve. It is rather unfortunate that [Caccetta and Hill 2003] are unable to divulge their approach in its entirety due to commercial reasons however,

they do outline some significant features that they employ in the algorithm. Their implementation consists of 17 000 lines of `C++` code using a mixture of breadth-first search and depth-first search. Breadth-first search is effective since it allows for the exploring of many branches. Depth-first search is useful in that consecutive child node linear programs are closely related, contributing to finding good solutions earlier. The method also involves the fixing, and therefore elimination, of a number of variables during the process. Scheduling is only considered for those blocks in the optimal pit, thus removing a fair amount of variables from the problem. This promotes the linear programming relaxations to be smaller in size, promoting more branching than is customary to find in branch-and-cut methods. The authors make mention of several extensions that can be applied to the mixed integer program, namely preprocessing of different ore types, maximum limit on vertical depths (haul road accessibility issues), minimum pit bottom width (for equipment ergonomics) and stockpiling.

[Bley *et al.* 2010] and [Fricke 2006] also have cuts, variable eliminations/fixings and other strengthening-techniques as focal points in their research. For this reason, they could be thought of as sequels to [Caccetta and Hill 2003]. [Bley *et al.* 2010] add a number of inequalities to the formulation thereby strengthening it and reducing the amount of time to find an optimal solution. The authors of [Bley *et al.* 2010] make extensive use of the presence another problem that is inherent in the OPMPSP, namely the *precedence constrained knapsack problem*. A large portion of [Fricke 2006] is also devoted to this significant problem, exploiting its structures to better solve the OPMPSP - for every time period and attribute in the problem the relative production constraint and precedence relationships between the blocks link to form a special structure, providing the opportunity to add several additional constraints to the formulation. This leads to tighter linear programming relaxations and a considerable reduction in computation times. This research dissertation will introduce the precedence constrained knapsack problem in a later chapter since we also aim to exploit its relationships with the OPMPSP.

[Bley *et al.* 2010] found some variable eliminations and valid inequalities to be more instrumental to improving computational performances than others. A set of inequalities, known as *multiple block cover inequalities*, sometimes increased computation times.

Future work could include finding more strong facet-defining inequalities for the formulation, possibly by finding facet-defining inequalities for the precedence constrained knapsack polytope. This, once again, demonstrates the advantages to be had from the structure's appearance in the OPMPSP. The authors also believe there may be a more efficient way to solve the *separation problem*, a means to finding valid inequalities.

6.2 Lagrangian Relaxation Based Approaches

[Dagdelen et al. 1986] provide the first glimpse of using Lagrangian relaxation on a problem that involves extracting a set amount of material from a mine with a model expressing precedence constraints. When utilised to solve the OPMPSP, it is generally regarded as an exact solution approach.

[Dagdelen et al. 1986] exploit the network structure that arises from the relaxing of a side constraint, for example, a mining capacity constraint. The constraint is moved to the objective function and its Lagrangian multipliers are then subsequently solved using a subgradient multipliers updating scheme. [Akaike et al. 1999] propose a different method for updating the Lagrangian multipliers but they do not always obtain feasible solutions. [Kawahata 2007] is primarily an extension on the work of [Dagdelen et al. 1986]. His approach does include some thoughtful considerations such as stockpiling and variable cutoff grade however, as is common to these Lagrangian relaxation avenues, [Kawahata 2007] struggles to find feasible solutions on a consistent basis.

A recent utilisation of Lagrangian multipliers is given by [Cullenbine et al. 2011]. The method employed is not exact but is rather a heuristic approach that "recursively defines, solves and partially fixes an approximating model" [Cullenbine et al. 2011]. This is accomplished by partitioning the problem according to time periods. Variables in the early time periods are fixed, the middle periods form part of an exact submodel and the later periods are defined within a relaxed submodel. The heuristic involved in [Cullenbine et al. 2011] recursively uses these 3 "stages."

6.3 Heuristic Algorithms Development

Similar to [Caccetta and Hill 2003], [Amaya et al. 2009] utilise "by" decision variables in their formulation. They use a local search methodology to develop a heuristic algorithm that attempts to improve upon a heuristically generated *incumbent solution.*

Definition 6.1 (Incumbent Solution)**.**

The incumbent solution is the best solution found so far in an algorithm. In tree searching strategies it is often the upper bound, when dealing with a minimisation problem.

This is achieved by iteratively fixing, relaxing and solving segments of the full model [Cullenbine et al. 2011].

If we let $x^{(0)}$ represent the incumbent solution, the heuristic will seek to find different subsets of blocks, Γ, and for each of these subsets Γ find a solution x' in the Γ-neighbourhood of $x^{(0)}$ that achieves the best objective function value. [Amaya et al. 2009] provide a definition of this neighbourhood concept:

Definition 6.2 (Γ-neighbourhood)**.**

"Given a solution $x^{(0)}$ and a subset of blocks Γ we define the Γ-neighbourhood of $x^{(0)}$ as the set of all solutions x' which coincide with $x^{(0)}$ everywhere except, possibly, the blocks in Γ."

The authors make use of 3 strategies to acquire the Γ sets and also aid in breaking out of local optima. One of these strategies is the *cone-above strategy*. In-line with the above descriptions, it operates by randomly selecting a block i and finding the best solution in the P_i-neighbourhood of the incumbent solution $x^{(0)}$. Recall that P_i denotes the entire precedence set of block i. Figure 6.1 is a graphical depiction of the cone-above strategy.

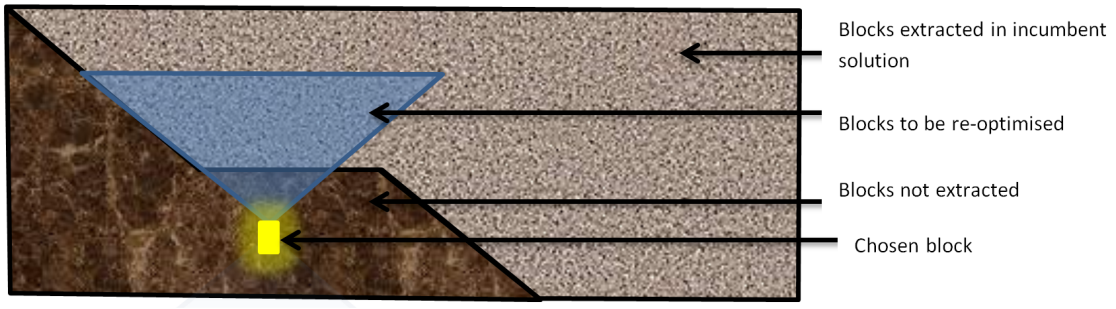

FIGURE 6.1: The Cone-above Strategy

[Amaya et al. 2009] have generated approximate solutions for the largest instances of the production scheduling problem thus far [Cullenbine et al. 2011]. They solve linear programming relaxations of the problem to acquire optimality measures for the solutions they obtain. The authors were not able to acquire an optimality bound for the largest problem in the case study set because solving the corresponding linear program proved to be difficult.

[Chicoisne et al. 2012] essentially solve the same formulation as [Amaya et al. 2009] except they propose an efficient decomposition method to solve intensive linear programming relaxations resulting from large problems that [Amaya et al. 2009] failed to acquire optimality bounds for. This decomposition method only works on a single capacity constraint per time period however, by using a rounding algorithm the authors claim it

can obtain solutions for multiple constraints per time period. The authors then use a set of methods, known as toposort heuristics, to acquire an initial solution. From here, a heuristic search, based on the algorithm found in [Amaya et al. 2009], can then be employed to acquire solutions to the formulation. Results produced on problem sets in [Chicoisne et al. 2012] are all within 3% optimality after 1 hour of computation. Their methodology consists of three steps: (*i*) the linear programming relaxation is solved, for one resource constraint, using the decomposition method known as the critical multiplier algorithm; (*ii*) The rounding heuristic is applied to the fractional solution to obtain a starting solution; (*iii*) Utilise the local-search heuristic to iteratively improve upon the starting solution from (*ii*).

6.4 Metaheuristic Methods

6.4.1 Evolutionary Algorithms

6.4.1.1 Genetic Algorithms

[Denby et al. 1994] define schedules as a combination of the ultimate pit and one extraction schedule. They seek to find optimal schedules by utilising a genetic algorithm (GA) that incorporates customary concepts for GAs, such as crossover and mutation. The method achieved good solutions on small problem sizes however, it struggles on larger instances that heralded untenable computation times.

It appears that genetic algorithms suffer from intractability issues with large block models. [Zhang 2006] combats this by aggregating blocks before employing a genetic algorithm. A comparison between the solutions obtained by [Zhang 2006] and those of IBM ILOG CPLEX (2009) on instances from BHP Billiton reveals that CPLEX is 2 - 4 times slower than [Zhang 2006] for the same NPV.

6.4.1.2 Particle Swarm Optimisation

[Ferland et al. 2007] make use of a particle swarm optimisation (PSO) metaheuristic to solve the OPMPSP. The authors suggest a formulation based on the *resource-constrained project scheduling problem* (RCPSP).

Definition 6.3 (Resource-constrained Project Scheduling Problem (RCPSP))**.**

The Resource-constrained Project Scheduling Problem (RCPSP) is defined as a set of activities that need to be completed, resources that are consumed by executing these activities and a set of constraints that must be satisfied. The idea is to maximise the objective function, a measure of a project schedule, by optimally assigning resources to activities at specific periods whilst not violating resource and other constraints.

An initial solution is obtained using a GRASP [Feo and Resende 1989] application. From here, the solution is evolved according to the particle swarm algorithm. Priority values, $Pr_i \in [0, 1]$, are given to each block i as they are given to tasks in a RCPSP. These values are not based purely on the economic value of each block but also its extraction would impact the extraction of other blocks - a *lookahead value* is thus calculated for blocks. This concept of *priority value encoding* was introduced by [Hartmann 1998] who used it in a RCPSP.

Definition 6.4 (Greedy Randomized Adaptive Search Procedure (GRASP))**.**

GRASP refers to a metaheuristic algorithm that is frequently applied to combinatorial optimisation problems. In general, a randomised solution will be constructed via several successive greedy iterations. This solution is then improved through further iterations within a local search.

[Ferland et al. 2007] perform various numerical tests ranging from identifying the sensitivity of key parameters in the algorithm to a comparison of the method to a more greedy approach. The authors concede that this method needs to be tested on larger, more realistically sized problems. Furthermore, there was little evidence of any significant sensitivity for the parameters in the algorithm.

[Khan et al. 2014] also look to solve the OPMPSP utilising particle swarm optimisation (PSO). They look at a host of previous PSOs and present their results.

6.4.2 Local Searches

6.4.2.1 Local Branching Heuristics

[Samavati et al. 2017] present a relatively new local branching heuristic to try and solve the OPMPSP. Their heuristic is a hybrid of sorts that involves dynamically adapting branching schemes with the original local branching proposal. Their version of the OPMPSP also takes into account lower bound resource constraints, which can potentially make the problem even more complex to solve.

6.4.2.2 Tabu Search

[Lamghari et al. 2012] present a Tabu search procedure to finding the optimal block extraction schedule for an open-pit mine. The model used in this approach is especially intricate because it attempts to capture metal-content geological uncertainty within its design as well as stochastic constraints and events. There are 2 conspicuous factors not often accounted for in most formulations for the OPMPSP namely, the aforementioned metal-content geological uncertainty and metal price volatility. An integral part of [Lamghari et al. 2012] that aims to capture this geological uncertainty is the usage of conditional simulations of the orebody. Instead of optimising over one block model, NPV is maximised on S equally likely block models.

Essential to any metaheuristic approach is an extensive exploration of the solution space. [Lamghari et al. 2012] employ 2 diversification strategies to achieve this. The first strategy utilises long-term memory strategy that keeps track of portions of the solution space that get explored. After a number of non-improving iterations of the Tabu search a new solution is generated in a region that have had little exploration and the Tabu search is restarted. The second strategy is based on the variable neighbourhood search by [Hansen and Mladenovic 2001].

[Lamghari et al. 2012] provide numerical results on realistically sized problems, revealing the efficiency of this particular approach to finding the optimal schedule. Numerical results identified Tabu search, using the long term memory strategy, as useful and robust. 92.5% of simulations produced solutions with optimality gaps of less than 4%. An additional advantage of this algorithm is its flexibility that offers the possibility of extensions and modifications. One enticing question is how one could incorporate parallel programming within this algorithmic structure. With the recent advent of *GPU programming* and the potential computational power that it may afford, this could greatly improve the performance of the solution approach. This is in fact a consideration that many in OPMPSP research have proposed, for example in [Amaya et al. 2009].

6.4.2.3 Simulated Annealing

[Kumral 2013] utilises a combination of *goal programming* and simulated annealing to simultaneously solve the OPMPSP and the ore-waste discrimination problem.

Definition 6.5 (Goal Programming)**.**

Goal programming is a mathematical programming paradigm that is concerned with optimising multiple objectives as opposed to a singular objective. For this reason, it can be thought of as an extension of linear programming.

The author employs a formulation, much like [Lamghari et al. 2012], that allows violations of mining and mill processing constraints to account for stochastic events. However, these violations act as penalty terms in the objective function and therefore there is an attempt to minimise these, as seen in equation (6.1):

$$\min \sum_{s=1}^{S}\sum_{t=1}^{T}(w_t^+\delta_{st}^+ + w_t^-\delta_{st}^-) + \sum_{s=1}^{S}\sum_{t=1}^{T}(\psi_t^+\zeta_{st}^+ + \psi_t^-\zeta_{st}^-) \tag{6.1}$$

w_t^+ and w_t^- are the penalty weightings in time period t for exceeding mining capacity by an amount δ_{st}^+ or falling short by an amount δ_{st}^- in block model simulation s. Likewise, ψ_t^+ and ψ_t^- are the penalty coefficients for exceeding ore processing capacity in period t by an amount ζ_{st}^+ or a shortfall amount of ζ_{st}^- in the block model simulation s.

An initial solution is first obtained by acquiring a block sequence using IBM ILOG CPLEX software, through its propriety exploitation of branch and bound and branch and cut methods. The simulated annealing metaheuristic is then applied to this initial solution.

[Kumral 2013] states that simulated annealing is pertinent for the OPMPSP. The algorithm fits in nicely with accessibility and precedence constraint checking, something that is more complicated with exact solution methods. Accessibility can be identified by using the upward and downward cone approach, highlighted in Figure 6.2. When a block is to be transferred to another time period the relative cone of blocks, upward or downward, is checked to identify if the shift is permissible. For example, imagine the simulated annealing algorithm wishes to schedule a block i for extraction in a later time period. Block i's downward cone of successor blocks, DC_i, is checked to ensure that the new extraction time of block i is not later than the extraction times of it's successors in

DC_i, which would violate precedence constraints. Mathematically, the check is expressed in equation (6.2):

$$x_{it} \leq x_{jt}, \quad j \in DC_i, \quad t = 1, ..., T. \tag{6.2}$$

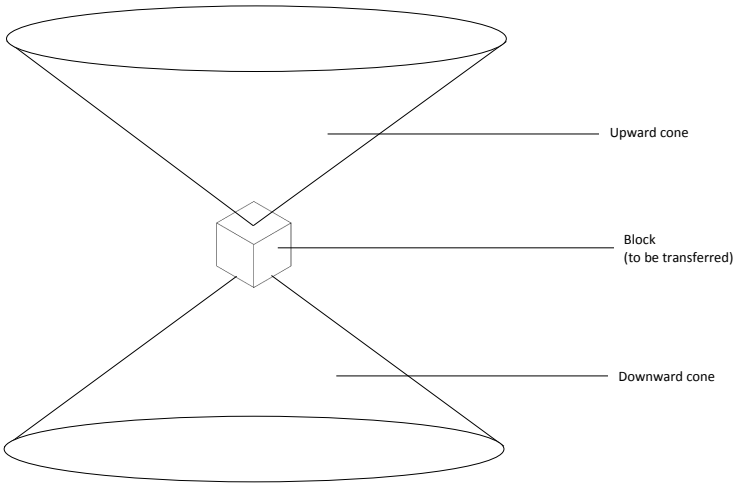

FIGURE 6.2: The Upward and Downward Cones

Simulated annealing also has an advantage over evolutionary algorithms, such as PSO and genetic algorithms. These approaches expend a lot of computation on updating generations. Additionally, they require a starting "generation" of initial solutions. Simulated Annealing needs to update and initialise one solution.

[Kumral 2013] lists problem size reduction and stochastic optimisation techniques as potential areas of improvement.

6.5 Model Anatomies and Analysis

The primary concern of the following research articles was not on developing a new enterprising algorithm to solve the open-pit mine production scheduling problem but rather bring to light modelling considerations that would make solving the subsequent formulation a more realistic depiction of open-pit mining.

As is customary with block model construction, a single orebody block model is created from a process such as kriging. Kriging involves drilling cores, obtaining information

about the content in these drill cores, interpolating between the data and then producing this "deterministic" block model. It is palpable to see that this model will not be entirely accurate given the often sporadic drillhole data. [Menabde et al. 2007] propose the use of conditional simulations, generations of equally probable realisation of the original block model, to cater for the uncertainty that is present in the construction of the block model. We have seen this strategy employed by [Lamghari et al. 2012] and [Kumral 2013]. [Menabde et al. 2007] make use of the statistical concepts of a variogram and probability distributions to obtain these equiprobable block models. The model also incorporates a collection of discrete cutoff grades.

[Kumral 2011] also makes mention of the various stochastic events that are involved in the open-pit mining production scheduling problem. The author proposes 2 methods to account for geo-metallurgical and financial uncertainty. The first method constructs a stochastic formulation with recourse actions. The second method expresses the scheduling as a maximin problem that primarily focusses on satisfying mill requirements rather than maximising the NPV.

[Kumral 2011] brings to light the faults that are traditionally made in models. Models usually consider recoveries, mining and processing costs as fixed however, there are a number of geo-metallurgical factors that can affect these parameters. Furthermore, mineralogical aspects have the potential to affect decision making as well. An example being the discovery of unwanted arsenic in a gold deposit that would invariably require additional costs. The author also motivates how cutoff grade should actually be independent for each block, a significant consideration. It should be noted that [Kumral 2011] specifically states that computational demand from these considerations is not of concern, the author rather wishes to account for stochasticity and factors that are so often overlooked in formulation construction.

[Kumral 2012] suggests that the open-pit mine production scheduling problem reduces to solving 3 integrated problems: (i) block sequencing; (ii) ore-waste discrimination with the aid of cutoff grades; (iii) the determination of production rates. What [Kumral 2012] attempts to do is amalgamate the solving of (i) and (ii) into a single model. Usually block sequencing and ore-waste discrimination (ie. determining cutoff grades) have been done independently with a priori cutoff grades. The model presented in [Kumral 2012] includes the added dimension in variables of destination, evident in the decision variable x_{ijd}. This of course is not foreign to the various formulations encountered in the production scheduling problem. Indeed, [Johnson 1968] was using this concept in 1968, see Section 5.3.2). [Kumral 2012] explains some of the erroneous assumptions and methods that are usually followed in determining cutoff grades. A case study is present in the paper which shows how the more detailed model accounts for 5% more NPV than the traditional

model. It is conceded that simplified models have the advantage of quicker solution times. Further work could include integrating production rates within the model or try and capture truck allocation and dispatching solving within the model.

6.6 Auxiliary Considerations

As mentioned previously, a block model is usually created as a result of kriging, an interpolation process. This process involves the drilling of holes to acquire samples of data on ore content, and information, for those drilled areas. It is therefore reasonable to assume that drilling more holes will produce greater certainty in the construction of the orebody block model thus, instilling greater confidence in interpolated block attribute values. [Froyland et al. 2007] try and evaluate the potential benefit in NPV of an open-pit mining project should additional drilling be considered. They investigate how conditional simulations are useful in allowing the rigorous valuation of a trade-off between the cost of extra drilling and more optimal schedules, since these schedules can be calculated on more accurate block models.

A big part of the life cycle of an open-pit mining venture is the reclamation phase (see Section 1.3). It is in this stage that an attempt is made to restore the mined area once operations have been concluded. [Zuckerberg et al. 2007] consider a model that attempts to cater for external waste dumping as well as in-pit dumping in the "voids" created by blasting. Not only does such in-pit dumping potentially aid in the reclamation phase of the venture but also can reduce costs associated with the transportation of this material after excavation. The model includes aggregation principles and the need to check that in-pit dumping does not forsake safeties. Invariably, the model will require more variables than usual therefore some sort of relaxation approaches will have to be employed for tractability reasons.

Chapter 7

Model Formulation

7.1 An Initial Model

Similar to the formulation detailed by equations (5.17)–(5.22), the model does not take into account multiple destinations for blocks. Furthermore, the model makes use of *at* decision variables as opposed to *by* decision variables:

$$\max \sum_{t=1}^{T} \sum_{i=1}^{N} v_{it} x_{it} \tag{7.1}$$

subject to:

$$\sum_{t=1}^{T} x_{it} \leq 1 \qquad i = 1, ..., N. \tag{7.2}$$

$$x_{it} - \sum_{\tau=1}^{t} x_{p\tau} \leq 0 \qquad i = 1, ..., N, \quad p \in P_i, \quad t = 1, ..., T \tag{7.3}$$

$$\sum_{i=1}^{N} m_i x_{it} \leq \bar{W}_t \qquad t = 1, ..., T. \tag{7.4}$$

$$\sum_{i=1}^{N} o_i m_i x_{it} \leq \bar{O}_t \qquad t = 1, ..., T. \tag{7.5}$$

$$x_{it} \in \{0, 1\} \qquad i = 1, ..., N, \quad t = 1, ..., T. \tag{7.6}$$

7.1.1 Formulation Overview

Equation (7.1) is the objective function. It seeks to maximise the NPV. Equation (7.2) represents the reserve constraint which makes sure that a block is mined at most once. The precedence constraint is established with equation (7.3). This form of the constraint is distinctly devoid of a graph network, as seen in previous formulations. There are two resource constraints, namely the mining constraint in equation (7.4) and the processing constraint in equation (7.5). Finally, there is the integrality constraint, equation (7.6).

7.1.1.1 Explanation of Significant Model Variables

In the objective function equation (7.1), the variable v_{it} represents the *present value* of a block i if extracted at a time period t.

Essentially, the present value of each block i at a period t provides us with a reflection of the current worth of the block should its economic value be realised at a time t. This valuation can be performed for each block with the following equation:

$$v_{it} = \frac{p_i}{(1+d_1)^t} \tag{7.7}$$

p_i represents the economic value of a block i and d_1 is the associated discount rate for the present-valuation.

Equation (7.6) is an integrality constraint and reveals x_{it} to be a binary decision variable with the following description:

$$x_{it} = \begin{cases} 1, & \text{if block } i \text{ is mined at time period t.} \\ 0, & \text{otherwise.} \end{cases} \tag{7.8}$$

The boolean variable o_i indicates whether a block i is an ore block or a waste block:

$$o_i = \begin{cases} 1, & \text{if block } i \text{ is an ore block - send for processing.} \\ 0, & \text{if block } i \text{ is a waste block - send to waste dump.} \end{cases} \tag{7.9}$$

$\bar{O}_t$ is the maximum mass that the ore mill can take in a time period t. Similarly, $\bar{W}_t$ is the maximum mass that can be mined in a time period t. m_i is the mass for a block i.

7.2 A Goal Programming Model with Penalty Terms

When dealing with a large, n-dimensional optimisation problem, such as the OPMPSP, traversing the solution space can often be a labourious process. This is especially true if local search methodologies form a significant component of the heuristic(s) being utilised. The algorithms encounter local optima and then slowly begin the process of escaping these "traps".

[Lamghari et al. 2012] and [Kumral 2013] combated this issue by the introduction of penalty terms into the problem formulation. The resource constraints of the model are now soft constraints. Schedules can now be produced that violate these constraints however, such solutions are penalised. The formulation below includes this functionality:

$$\max \sum_{t=1}^{T} \sum_{i=1}^{N} v_{it} x_{it} - \left(\sum_{t=1}^{T} (c_t^o \delta_t + c_t^m \zeta_t) \right) \tag{7.10}$$

subject to:

$$\sum_{t=1}^{T} x_{it} \leq 1 \qquad i = 1, ..., N. \tag{7.11}$$

$$x_{it} - \sum_{\tau=1}^{t} x_{p\tau} \leq 0 \qquad i = 1, ..., N, \quad p \in P_i, \quad t = 1, ..., T \tag{7.12}$$

$$\sum_{i=1}^{N} m_i x_{it} - \zeta_t^m \leq \bar{W}_t \qquad t = 1, ..., T. \tag{7.13}$$

$$\sum_{i=1}^{N} o_i m_i x_{it} - \delta_t^o \leq \bar{O}_t \qquad t = 1, ..., T. \tag{7.14}$$

$$x_{it} \in \{0, 1\} \qquad i = 1, ..., N, \quad t = 1, ..., T. \tag{7.15}$$

Equation (7.10) is the new objective function, now with contingent of penalty terms c_t^o, δ_t, c_t^m and ζ_t. c_t^o and c_t^m are the penalty weightings associated with exceeding ore processing and mining constraints in a time period t respectively. c_t^o is calculated as follows:

$$c_t^o = \frac{c^o}{(1 + d_2)^t} \tag{7.16}$$

c_t^m is calculated similarly to c_t^o in equation (7.16). c^o is the weighting without any discounting applied to the value according to the time period of the violation. d_2

represents the rate that determines the discount factor applied to violations. δ_t represents the amount of mass sent to the processing mill that exceeds $\bar{O}_t$ (the maximum ore processing capacity) in a time period t. Similarly, ζ_t is the total mass mined in a period t that exceeds the mining capacity $\bar{W}_t$. The goal programming formulation is completed with the transformation of the resource constraints with Equation (7.13) and Equation (7.14), effectively making them soft constraints.

The implementation component of this dissertation will largely be based off the aforementioned model formulation in this section, with some slight modifications for simplicity without comprising efficiency.

Chapter 8

The Precedence Constrained Knapsack Problem

Modern research literature in the OPMPSP sometimes exploits the structure of the Precedence Constrained Knapsack Problem (PCKP), another combinatorial optimisation problem. These problems are rather similar, with the exception that the PCKP is essentially the OPMPSP with only one time period. Otherwise, the objective function and constraints for both problems are almost synonymous.

[Bley *et al.* 2010] and [Fricke 2006] made several relations between the PCKP and the OPMPSP with the aim of strengthening formulations. This research seeks to exploit a set of methods used to solve a variant of the PCKP known as the *Multi-period Precedence Constrained Knapsack Problem (MPPCKP)*. These methods can be reverse-engineered for the OPMPSP since the MPPCKP is an equivalent problem.

Firstly, the general knapsack problem will be introduced. From here, the extension to the PCKP will be explained. This problem is then adapted to mimic the OPMPSP by ensuring a knapsack can be filled in multiple periods, the MPPCKP.

[Moreno et al. 2010] and [Chicoisne et al. 2012] showcased a set of methods that are commonly used to acquire solutions for the MPPCKP and how they can be used in the OPMPSP. These will be introduced and explained.

8.1 The Knapsack Problem (KP)

The Knapsack Problem is a combinatorial optimisation problem and can be defined as follows:

There are n items, each with weight and value attributes. The objective is to pack a subset of these items into a knapsack (or an entity representative of a knapsack) so that the resultant value of packed items is maximal and the capacity constraint is not violated. Furthermore, each item can only be packed once.

To be specific, the descriptions above are of the *0-1 Knapsack Problem*. Because an item is either packed or not packed into the knapsack the problem exhibits a boolean nature. Such problems can be aptly modelled as a binary integer program:

$$\max \sum_{i=1}^{n} v_i x_i \tag{8.1}$$

subject to:

$$\sum_{i=1}^{n} w_i x_i \leq W, \tag{8.2}$$

$$x_i \in \{0, 1\}, \quad i = 1, ..., n. \tag{8.3}$$

Equation (8.3) is the integrality constraint which expresses whether an item is either packed or not:

$$x_i = \begin{cases} 1 & \text{if item } i \text{ is packed into the knapsack,} \\ 0 & \text{otherwise.} \end{cases}$$

Equation (8.1) is the objective function of the 0-1 Knapsack Problem. It allows the realisation of the values of items, v_i, that have been decided to be packed and aims to maximise the sum of the realised values. Equation (8.2) is the capacity constraint and ensures that the sum of the realised weights, w_i, from packing does not exceed the weight capacity of the knapsack, W.

Research regarding knapsack problems started receiving a great deal of attention during the late 1950's. The huge strides that scientists such as George Dantzig made in the field of operations research cultivated an interest in optimisation problems such as the Knapsack Problem. Knapsack problems have found frequent application in finance and industry [Pisinger 1995].

Figure 8.1 is an optimisation problem that can be modelled as a knapsack problem. It is an example of a *capital budgeting problem* where there is $ 15 000 available to invest in

several investment opportunities. The aim is to maximise the total net present value realised from the chosen investment opportunities whilst not exceeding the investment limit.

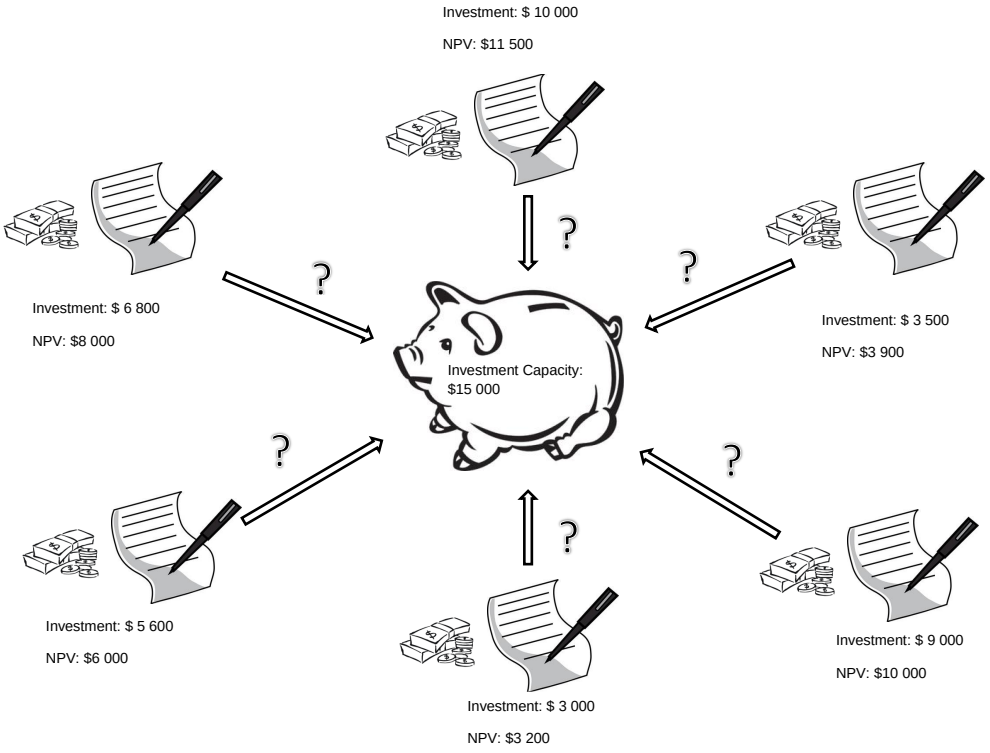

FIGURE 8.1: A Capital Budgeting Problem

Figure 8.2 reveals a graphical representation of the archetypal Knapsack Problem. The investment "piggy bank" and its investment limit can be thought of as the knapsack and its weight capacity. The investment opportunities, along with their investments and NPVs, can be thought of as the items, with their weights and values respectively.

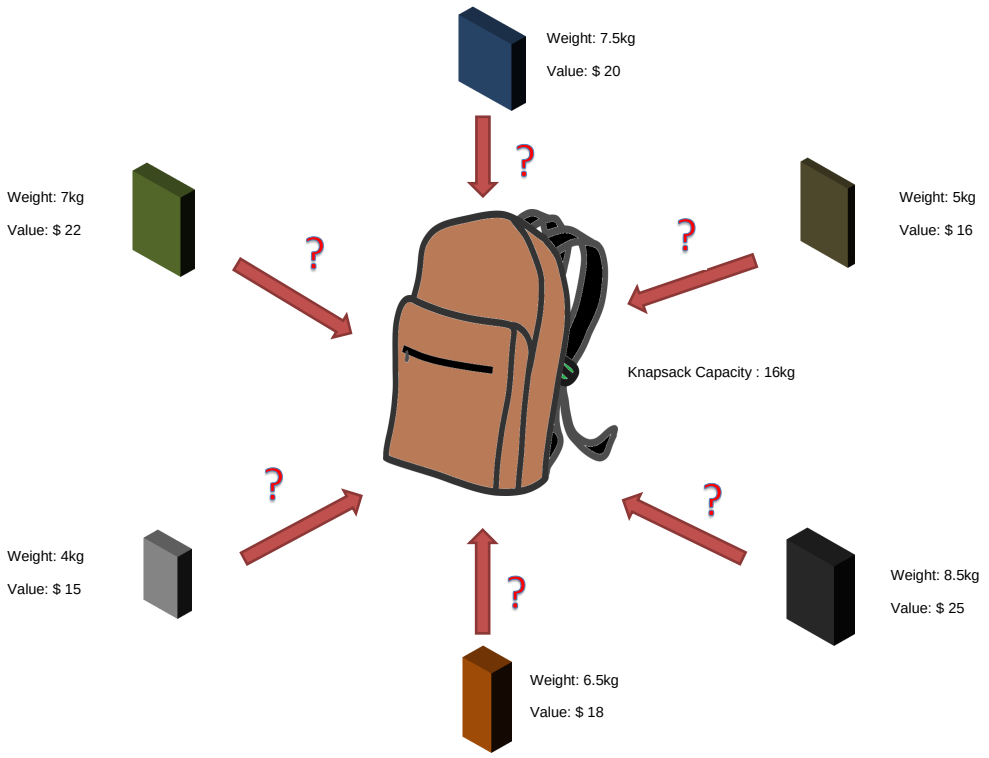

FIGURE 8.2: Graphical representation of the Knapsack Problem

[Pisinger 1995] states that all knapsack problems belong to a class of NP-Hard problems. This implies that it is highly unlikely that polynomial time algorithms will be developed to solve these problems. Knapsack problems are said to have an exponential worst case complexity. Despite this rather daunting fact, much research was directed toward the special structures of knapsack problems. The research enabled the effective solving of even large instances of knapsack problems with relative ease [Pisinger 1995].

Some popular techniques that usually accompany attempts to solve knapsack problems exactly are:

1. **Dynamic Programming** - solving subsets of the entire solution space.
2. **Branch-and-bound** - intelligent exploration of the solution space using bounds, pruning and node fathoming.
3. **Preprocessing** - some variables can be fixed a priori to allow for more efficient solving.
4. **State space relaxations** - "a dynamic programming relaxation" that scales coefficients by a fixed value. Optimality is sacrificed in favour of a faster approximate algorithm.

8.2 The Precedence Constrained Knapsack Problem

The Precedence Constrained Knapsack problem is an extension of the general Knapsack Problem. The aim is still to maximise realised values of packing items into a knapsack without exceeding its capacity however, there is an additional condition in the formulation in the form of a precedence constraint.

An item i can only be packed into the knapsack if, and only if, all the items in the set P_i are packed - all of item i's predecessors must be packed for item i to be packed.

$$\max \sum_{i=1}^{n} v_i x_i \tag{8.4}$$

subject to:

$$\sum_{i=1}^{n} w_i x_i \leq W, \tag{8.5}$$

$$x_i - x_p \leq 0, \quad i = 1, ..., n, \quad p \in P_i, \tag{8.6}$$

$$x_i \in \{0, 1\}, \quad i = 1, ..., n. \tag{8.7}$$

$$x_i = \begin{cases} 1 & \text{if item } i \text{ is packed into the knapsack,} \\ 0 & \text{otherwise} \end{cases}$$

Equation (8.6) is the introduced precedence constraint. It ensures that all items p in i's predecessor set, P_i, are packed before item i is packed. Of course, this can also be captured by using a graph network structure. Section 5.2.2 goes into detail how a graph structure captures the precedences between blocks in an open-pit mine block model.

Precedence relationships are inferred by logical and physical ties between items. We have seen the physical relationship between blocks in an open-pit mine block model. Overlying blocks need to **precede** their respective underlying blocks. The PCKP can also model other optimisation problems with precedences between items. For example, in project management, there is a logical ordering of activities in software development. A design phase has to logically come before the coding and testing phases.

Currently, the formulation for the Precedence Constrained Knapsack Problem is comparable to the formulation for the Ultimate Pit Problem, equations (5.5)–(5.7). In fact, from an open-pit mine planning perspective, this formulation is more detailed than

the UPL formulation in that there is a capacity constraint, equation (8.5). However, the aim is to identify its likeness to the Open-pit Mine Production Scheduling Problem (OPMPSP). There is still the notable absence of a temporal factor in the formulation. The addition of a time variable to the PCKP will result in another extension of the generalised knapsack problem known as the Multi-Period Precedence Constrained Knapsack Problem (MPPCKP).

8.3 The Analogous Multi-Period Precedence Constrained Knapsack Problem

The PCKP above, like the UPL, lacks a temporal factor in its formulation. With regards to open-pit mine planning and design, such models lack the necessary detail to effectively develop a realistic mining schedule. If we introduce a time factor to the PCKP problem it becomes the Multi-Period Precedence Constrained Knapsack Problem, which is equivalent to the OPMPSP.

A knapsack can now be filled in different time periods. The values of the items are dependent on the time period in which they are packed. Generally, items lose some value if they are packed in later periods rather than earlier ones [Moreno et al. 2010]. We saw this time-value-of-money concept in Section 5.3.1.1.

The formulation is presented below:

$$\max \sum_{i=1}^{n} v_{it} x_{it} \tag{8.8}$$

subject to:

$$\sum_{t=1}^{T} x_{it} \leq 1, \quad i = 1, ..., n, \tag{8.9}$$

$$\sum_{i=1}^{n} w_i x_{it} \leq W_t, \quad t = 1, ..., T, \tag{8.10}$$

$$x_i - \sum_{\tau=1}^{t} x_{p\tau} \leq 0, \quad i = 1, ..., n, \quad p \in P_i, \quad t = 1, ..., T, \tag{8.11}$$

$$x_i \in \{0, 1\}, \quad i = 1, ..., n, \quad t = 1, ..., T. \tag{8.12}$$

$$x_{it} = \begin{cases} 1 & \text{if item } i \text{ is packed into the knapsack in time period } t, \\ 0 & \text{otherwise.} \end{cases}$$

The introduction of the time factor has resulted in the addition of a new inequality, the reserve constraint in equation (8.9). This ensures that an item can be packed only once. The capacity and precedence constraints have changed to accommodate for the inclusion of time in the model. If we compare the formulation of equations (8.8)–(8.12) with the formulations of equations (7.1)–(7.6) we can see that they are essentially the same formulations except the OPMPSP has an extra resource, namely the processing mill resource, with an associated capacity constraint. We can therefore conclude that the OPMPSP is a knapsack problem, specifically the MPPCKP.

Figure 8.3 emphasises association between the OPMPSP and the MPPCKP. It presents a small 2D open-pit mine block model with only 3 time periods and 15 blocks. Mining capacity is neglected in this example however, we still consider the processing mill capacity which is 3 blocks per period. Blocks $\{1, 2, 3, 4, 5, 7, 8, 9, 13\}$ are considered ore blocks and blocks $\{6, 10, 11, 12, 14, 15\}$ are waste blocks. The manner in which the blocks have been "packed" into the time periods determines the order in which they have been mined. It is this ordering that determines the net profit. Figure 8.3 indicates a mining schedule of $\{2, 3, 4, 8, 1, 5, 7, 9, 13\}$. Recall that a block loses value the later it is mined. So if block 9 was ore-rich a better strategy may have been to mine blocks $\{3, 4, 5\}$ in period 1 so block 9 is accessible in the second period, realising its value as soon as possible so that it does not lose further value.

For the remainder of this section we will tend to reference "blocks" and the OPMPSP instead of "items" and the MPPCKP since the analogy between these two problems has been successfully drawn.

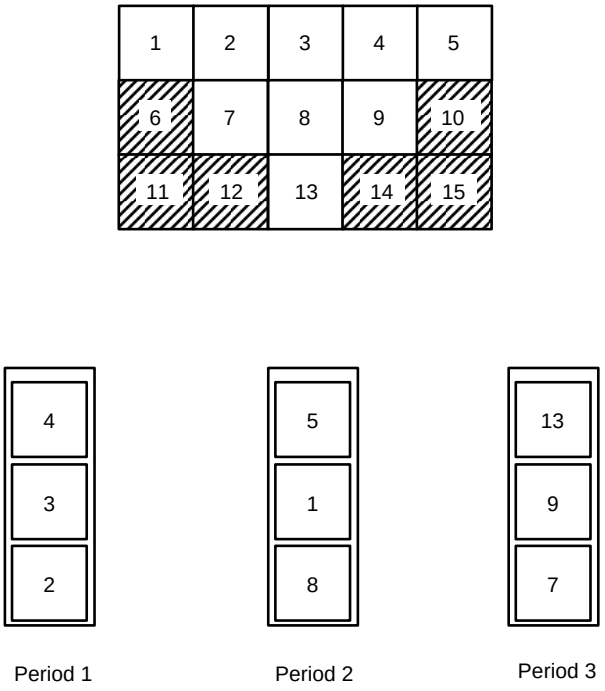

FIGURE 8.3: A Small MPPCKP Example - 2D Open-pit Mine Production Schedule

Toposort Heuristics are a set of methods that have proven very efficient at finding feasible solutions for the MPPCKP [Moreno et al. 2010]. Befitting for the OPMPSP, these methods are epitomised by the ordering of "items". The next section will introduce these methods.

8.4 Toposort Heuristics

In the previous section we saw that the ordering of blocks leads to the definition of a block extraction sequence. Each block has a position in the ordering such that all its predecessors come before it in the sequence, maintaining precedence in the context of the OPMPSP. That is if $\{p_1, p_2, ..., p_n\}$ is a permutation of blocks then if $p_i \in P_{p_j}$ it follows that $i < j$. If one keeps this rule in mind when generating orders it results in feasible solutions for the OPMPSP. A trivial method for finding a feasible solution is to sequentially schedule each block for extraction as early as possible whilst making sure it is not scheduled for extraction before its predecessors and does not violate any resource limitations. Of course, we also need to ensure that the resource constraints are not violated. This can be repeated until all blocks are scheduled. However, there are better

algorithms to generate orders. These methods will be described once a mathematical notation for these *topological orderings* has been defined [Chicoisne et al. 2012].

Consider a graph network of edges approach that models the precedence relationships between blocks in the block model, $G' = (V, E')$ and $|V| = n$. This graph differs from the traditional precedence graph in that $(i, j) \in E'$ implies that j is an immediate **successor** of i. A block j is a successor of a block i if there exists a direct path from i to j in the graph, $i \to j$. Let $V^-(i)$ be defined as the set of all blocks in the graph that are predecessors of i. A permutation of blocks $\{p_1, ..., p_n\}$ defines a topological ordering of V if $p_i \to p_j$ means $i < j$. It follows that $V^-(i) \subseteq \{p_1, ..., p_{i-1}\}$ $\forall i = 1, ..., n$ [Chicoisne et al. 2012].

In the OPMPSP the edges in the graph capture the physical precedences of the blocks. This results in G' being a directed acyclic graph.

The directed acyclic graph in Figure 8.4 is an example of what a 2D block model precedence graph may look like. Directed acyclic graphs have a significant property. They always admit **topological orderings** [Chicoisne et al. 2012]. We can conclude the following:

1. Devising a Toposort Heuristic is essentially finding a topological ordering from the directed acyclic precedence graph.
2. Because of the nature of these graphs we are always guaranteed that such an ordering exists.

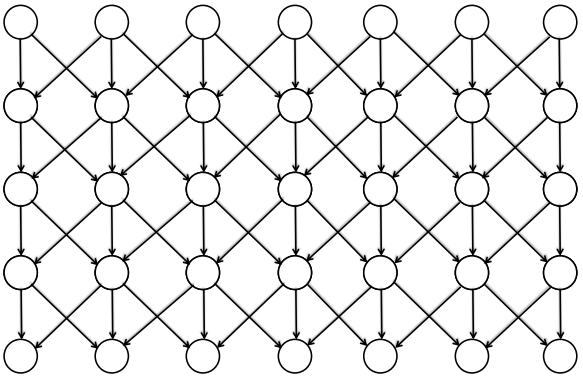

FIGURE 8.4: A Directed Acyclic Graph

Finding a topological ordering in precedence graph of a block model is equivalent to ordering blocks for the generation of the production schedule. There are a large amount

of possible topological orderings and so our objective should be to find an ordering that results in a schedule that maximises profit. We can introduce a weighting-mechanism that will allow us to select good orderings.

8.4.1 Topological Sorting

The process of generating a topological ordering involves the selection of blocks/vertices sequentially. Imagine one selects these according to the vertices seen in Figure 8.4, starting from the very top. The constraint is that a vertex can only be chosen if all the edges leading to it have an origin vertex that has already been selected. Certainly, there are fair amount of possible permutations, even in the small example that is Figure 8.4. However, if one assigns each block a weighting, and then sorts according to these weightings, we provide a rule for a heuristic where choice selection favours certain vertices, typically those with the greater weights. Thus, we have a *weighted topological ordering*.

Definition 8.1 (Weighted Topological Ordering)**.**

Assume a permutation $\{p_1, p_2, ..., p_n\}$ is a topological ordering. A topological ordering is weighted according to w if every pair of consecutive elements (blocks) p_i and p_{i+1} satisfies either:

1. p_i is a predecessor of p_{i+1}.
2. $w_{i+1} < w_i$.

Algorithm 9 reveals how this sorting can be achieved on a directed acyclic graph $G' = (V, E')$. $u \in V$ and $\epsilon(u)$ represents the incoming edges for vertex u.

Algorithm 9 Topological Sort of Vertices in a Directed Acyclic Graph, $G' = (V, E')$

Let $i \leftarrow 1$.
Let $n \leftarrow |V|$.
while $i < n$ **do**
    % select vertex with greatest weight and no incoming edges.
    $u \leftarrow \underset{u}{\text{argmax}}\{w_u : \epsilon(u) = \emptyset\}$.
    $V \leftarrow V \setminus \{u\}$. % remove u.
    $E \leftarrow E \setminus \{(u, v)\}, \forall\ v \in V$. % remove outgoing edges.
    $p_i \leftarrow u$.
    $i \leftarrow i + 1$.
end while

Algorithm 9 will return a topological ordering $\{p_1, p_2, ..., p_n\}$ of graph G' weighted according to w.

A vital constituent of this sorting procedure is the choice of weight function applied to the blocks/vertices. This function evaluates blocks and will determine their sequence. The sequence will determine the production schedule.

8.5 Weight Functions

8.5.1 The Greedy Weight Function

$$w_i = v_i, \quad i = 1, ..., n. \tag{8.13}$$

The blocks in the OPMPSP already have a "weight" associated with them. Recall that each block has an economic value, usually in USD or some other currency. It is possible to use this as the weighting function for all blocks, as seen in Equation (8.13).

Algorithm 9 will create a topological ordering that greedily favours blocks of greatest economic value at the beginning of the sequence. Such an approach is short-sighted since it does not take into account the blocks that are not yet accessible due to precedence constraints. The blocks are valued solely on their economic value. The accessibility they provide to potentially even more valuable blocks is negated. The method is completely "oblivious" to blocks that have not had all their predecessors mined yet. Figure 8.5 shows the failings of the greedy approach.

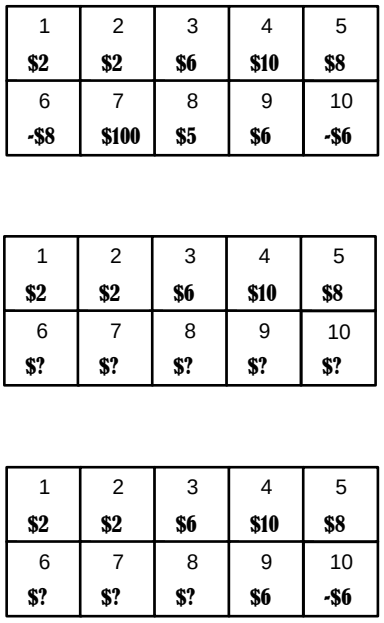

FIGURE 8.5: Short-sighted Greedy Weighting

Consider the small 2D block model with indicated economic values for extractions. Assume that 4 blocks can be mined in a single time period. Initially, the sorting would only consider the surface row, the underlying blocks are essentially invisible, as seen in the second instance of Figure 8.5. Because of the greedy nature of the weighting function the initial extraction sequence would be $\{4, 5, 3\}$ which would reveal blocks 9 and 10, as shown in the last instance of the figure. The period would conclude with the extraction of block 9. Block 7 is very valuable but because of the greedy procedure it will only be extracted in the second period and therefore lose a portion of its economic value. A better sequence would have been $\{3, 2, 1, 7\}$. The extraction of blocks $\{1, 2, 3\}$ would have allowed the extraction of the valuable block 7 in the first time period. The next weighting function, although not perfect, combats the short-sightedness of the greedy approach.

8.5.2 The Gershon Weight Function

Let $C(G') = (V, E'^{+})$ be the transitive closure of the graph G'. $A'_i = \{j \in V : (i, j) \in E'^{+}\} \cup \{i\}$ is then the set of all successors for block/vertex i.

$$w_i = \sum_{a \in A_i} v_a, \quad i = 1, ..., n. \tag{8.14}$$

Equation (8.14) is the weight function that was proposed by [Gershon 1987]. Each block, i, is weighted according to the sum of all the values of the blocks in the inverse-precedence set, A_i [Moreno et al. 2010]. Figure 8.6 shows how Gershon weighting would be implemented on a 15-block 2D model.

1 **$2**	2 **$2**	3 **$6**	4 **$10**	5 **$8**
6 **-$8**	7 **$100**	8 **$5**	9 **$6**	10 **-$6**
11 **-$2**	12 **$2**	13 **$6**	14 **$2**	15 **-$4**

1 **$2**	2 **$2**	3 **121/6**	4 **$10**	5 **$8**
6 **-$8**	7 **$100**	8 **$5**	9 **$6**	10 **-$6**
11 **-$2**	12 **$2**	13 **$6**	14 **$2**	15 **-$4**

1 **100**	2 **107**	3 **121**	4 **21**	5 **12**
6 **-8**	7 **106**	8 **15**	9 **10**	10 **-8**
11 **-2**	12 **2**	13 **6**	14 **2**	15 **-4**

FIGURE 8.6: Gershon Weighting

The first diagram shows the economic values of 15 blocks. The second diagram reveals how block 3's weighting becomes 121 through the sum of the economic values of itself and its successors ($121 = 6 + 100 + 5 + 6 - 2 + 2 + 6 + 2 - 4$). The final diagram shows the Gershon weights for all blocks in this example.

A Toposort algorithm that uses the Gershon weighting function shall henceforth be referred to as *GeTS*.

8.5.3 The Expected Extraction Time Weight Function

[Chicoisne et al. 2012] proposed a weight function that utilises the solution to the linear programming relaxation of the OPMPSP. It views the solution to this problem as the probability of extraction of a block in a certain time period:

$$P_{i,t} = (x^*_{i,t} - x^*_{i,t-1}), \quad i = 1, ..., n, \quad t = 1, ..., T. \tag{8.15}$$

Equation (8.15) represents the probability of a block i getting extracted in the time period t. [Chicoisne et al. 2012] developed this idea through identifying that:

- $0 \leq x_{i,0} \leq x_{i,1} \leq ... \leq x_{i,T} \leq 1, \quad \forall i.$
- Implies $0 \leq P_{i,t} \leq 1$, $t = 1, ..., T$.
- And $\sum_{t=1}^{T} P_{i,t} \leq 1$.

Should a block not be extracted by a time T it is then not extracted and given the pseudo extraction time of $T + 1$. We can define the expected extraction time for a block i by:

$$\mathbb{E}_i = \sum_{t=1}^{T} t(x^*_{i,t} - x^*_{i,t-1}) + (T + 1)(1 - x^*_{i,T}), \quad i = 1, ..., n. \tag{8.16}$$

For the precedence constraint to be maintained note that $\mathbb{E}_i \leq \mathbb{E}_j$ if $(i, j) \in E'$. Should x^* be all integer values, which implies it is optimal, then $\mathbb{E}_i$ is equal to the extraction time of block i or $T + 1$ if it is not to be extracted.

The expected-time weighting function can be defined as follows:

$$w_i = -\mathbb{E}_i, \quad i = 1, ..., n. \tag{8.17}$$

8.6 Creating a Production Schedule from a Weighted Topological Sort

Whatever weighting function we decide to use with a subsequent topological sort, we need to manifest it into a feasible production schedule for the OPMPSP. Algorithm 10 outlines how one would use the weighted sorted sequence to create a production schedule.

Algorithm 10 Toposort Heuristic

Input: $\{p_1, p_2, ..., p_n\}$, a permutation of blocks topologically sorted according to Algorithm 9.
Let $t_b \leftarrow 1, \forall\ b \in V$.
$i \leftarrow 1$.
while $i < n$ **do**
  Let $u \leftarrow p_i$.
  $i \leftarrow i + 1$.
  $t_u \leftarrow \max\{t_v : (v, u) \in E'\}$. % block u cannot be extracted before predecessors.
  $t_u \leftarrow \max\{t_u, \min\{t : c_{rt} \geq q_{ru} \quad \forall r = 1, ..., R\}\}$. % extract in earliest time period with sufficient resources.
  if $t_u \leq T$ **then**
    $c_{rt_u} \leftarrow c_{rt_u} - q_{ru}, \forall\ r = 1, ..., R$. % update resources.
  end if
end while

Algorithm 10 returns an extraction time for each block, thus creating a production schedule.

Note that there are only 2 resources in our model, namely the mining capacity per period and processing mill capacity per period.

From this point onwards, the application of the above mentioned algorithm shall be referred to as the *Expected Extraction Toposort* algorithm, or for convenience *ExTS*.

8.7 Conclusion

This chapter introduced the Knapsack Problem and showed how it can be extended to a variant known as the Multi-Period Precedence Constrained Knapsack Problem (MPPCKP). This allows for the filling of a "knapsack" in multiple time periods.

The MPPCKP was shown to be analogous to the Open-pit Mine Production Scheduling Problem (OPMPSP). Because these problems are equivalent there is the potential to exploit techniques and methods for the MPPCKP in the OPMPSP.

A set of methods utilised in finding solutions for the MPPCKP, known as Toposort Heuristics, were explained in the context of the OPMPSP. These methods can be used to find feasible production schedules. Specifically, the Expected-Time Toposort approach has shown much promise in literature. It can be used to acquire a good initial solution for the OPMPSP [Moreno et al. 2010].

Chapter 9

Tabu Search Metaheuristic for the OPMPSP

This chapter will explain the usage of tabu search metaheuristic in solving the OPMPSP, beginning with the vanilla form which is based predominantly on the methodology seen in [Lamghari et al. 2012]. It concludes with a piece on parameter optimisation, discovered from simulations on small problem instances, that are exploited in the results and implementations section later in this dissertation.

9.1 Vanilla Tabu Search

[Lamghari et al. 2012] present a tabu search approach for the OPMPSP with metal uncertainty. That is, they model for the stochastic nature of the orebody through incorporating a series of equally probable orebodies in their model. Although, the research of this dissertation does not account for metal uncertainty there are still facets of their algorithm which are easily migrated to the general OPMPSP.

The vanilla tabu search method, henceforth referred to as *Vanilla-TS*, operates on the problem formulation seen in Section 7.2. That is, a goal-programming model allowing for strategic oscillation (see Section 4.2.4.3), and an accomodation of reserve, slope, mining and processing constraints. [Lamghari et al. 2012] utilise a similar model but with lower bounded limits, the inclusion of metal constraints, and a slightly different objective function.

Definition 9.1 (Metal Constraints in OPMPSP)**.**

Metal constraints in the OPMPSP are formulation considerations that model upper and lower bound limitations on metal production as a result of ore processing. For example, there can be an upper limit on metal production because a maximum amount can only feasibly be sold during a time period or a lower limit due to the mining project's contractual obligations from investors.

9.1.1 An Abridgement of Vanilla-TS

The Vanilla-TS algorithm can be partitioned into 4 phases.

1. Acquiring an initial solution, a starting point for the algorithm.
2. A tabu search from a starting/reference schedule.
3. A diversification strategy generating a new starting/reference point.
4. A check of the ultimate termination criteria.

After the initial solution has been acquired a tabu search ensues. When the tabu search stopping criterion is satisfied phase 2 concludes and the best solution found within the tabu search is passed to the diversification strategy. The diversification strategy creates a new schedule from the post tabu search schedule and a number of other inputs. The algorithm's ultimate stopping criteria is then checked. Should the stopping criteria not be satisfied the algorithm returns to phase 2 with starting point the schedule generated at the conclusion of the diversification strategy. Otherwise, the entire algorithm terminates with output being the best schedule found during the whole procedure.

9.1.2 An Initial Solution for Vanilla-TS

Vanilla-TS uses the ExTS heuristic output as an starting point for its algorithm. The idea being that a good initial solution provides a sound foundation for the start of the algorithm and prevents unnecessary exploration of poor regions of the solution space. Starting point or reference schedules are denoted x_0. As will be described later, the algorithm's diversification strategy will generate new starting points for the tabu search during the course of the algorithm to prevent trappings in local optima.

9.1.3 Terminology and Key Concepts

9.1.3.1 Shifts and Schedules Notation

Consider a data structure x where x_i is the current time for which block i is scheduled for extraction. For example, $x_2 = 4$ would imply that block 2 is due for extraction in time period 4. x is essentially a data strucuture that stores a *feasible production schedule* for the OPMPSP. Some literature utilise a 2-dimensional, or sometimes 3-dimensional matrix (depending on the model), X to store extraction schedules. These matrices tend to be sparse and are therefore omitted from our coding implementation component of the research.

Definition 9.2 (A Feasible Production Schedule)**.**

A feasible production schedule, x, is a solution to the OPMPSP that does not violate the precedence constraint and every block in the block model has an extraction time.

A new solution is generated in the neighbourhood of x by setting the extraction time of a block i to time period t where $t \neq x_i$. In the context of the OPMPSP, this is known as a *shift* or *shifting*.

Definition 9.3 (Shift)**.**

A shift in a mining schedule x refers to the scheduling of a block i for extraction in time period t where $t \neq x_i$. The new solution generated by shifting is denoted $x \oplus (i, t)$.

There are 3 possible cases for shifts. A block can be "removed" from the schedule, never to be extracted. In the context of our data structures, this would be akin to setting $x_i = T + 1$. A block can be introduced to the schedule, where before it was not designated for any extraction time. Alternatively, a block can simply have its extraction time changed.

Further schedule notation includes the definitions of the current best schedule found so far in the algorithm and the current best solution found within the current nested tabu search. These are denoted as x_{best} and x^* with their respective objective function values $f(x^*)$ and $f(x_{best})$, calculated according to equation (7.10).

9.1.3.2 A Means to Monitor Individual Precedence Feasibilities

In line with Definition 9.2 and Definition 9.3, the new schedule generated from shifting $x_i = t$ is feasible if and only if $x_p \in [1, t]\ \forall\ p \in P_i$ and $x_s \in [t, T+1]\ \forall\ s \in S_i$. Recall that P_i represents the set of all predecessors for block i and S_i all the successors of block i. One could further simplify this necessary condition by stating that:

$$t \in [lp(x_i), es(x_i)] \tag{9.1}$$

where,

$$lp(x_i) = \max_{p \in P_i}\{x_p\} \tag{9.2}$$

$$es(x_i) = \min_{s \in S_i}\{x_s\} \tag{9.3}$$

Equation (9.2) is the latest extraction time of all of block i's predecessors and equation (9.3) the earliest extraction time of all the successors of block i. Therefore, in order for precedence constraints to be maintained, the shift to a time t for a block i must be in the range indicated by equation (9.1). One can employ a $2 \times N$ array, C, to store the lp and es values for each block, with $C_{1i} = lp(x_i)$ and $C_{2i} = es(x_i)$. This array will be most useful in determining feasible shifts during the course of the algorithm.

9.1.3.3 Memory Structures of Vanilla-TS

Similar to the description of the *tabu* memory structure described in Section 4.2.2, Vanilla-TS employs a tabu array, henceforth refered to as $Tabu$. When generating a new solution $x \oplus (i, t)$ by shifting block i to period t, the shift (i, x_i) becomes forbidden for the next θ tabu search iterations, $Tabu_{i,x_i} = \theta$. θ is a random positive integer in a predetermined range:

$$\theta \in [\theta_{min}, \theta_{max}], \quad \theta, \theta_{min}, \theta_{max} \in \mathbb{Z}^+.$$

A forbidden shift may be applied if the new solution generated is better than the current best solution found thus far by Vanilla-TS. This is forms familiar classical aspiration criterion that was described in Section 4.2.3.

Vanilla-TS utilises another memory structure known as a frequency matrix, denoted $\mathcal{F}$. $\mathcal{F}$ is a 2-dimensional array of size $N \times (T+1)$ that records the number of times a block i is shifted to a time period t. For example, $\mathcal{F}_{it} = 4$ would indicate that block i has been shifted 4 times to period t during the course of the algorithm. $\mathcal{F}$ is a vital constituent of the diversification strategy of Vanilla-TS. It is also used to break ties in deciding which shifts to select, as discussed later in this chapter.

9.1.3.4 Vanilla-TS Diversification Strategy

The diversification strategy is initiated after the tabu search portion of the algorithm. Vanilla-TS utilises the *long-term memory diversification strategy* outlined in [Lamghari et al. 2012]. The diversification strategy works from the x^* schedule.

The idea of a diversification strategy is to improve the efficiency of the search in the problem solution space. This can be achieved by shifting some blocks to periods where they have rarely been scheduled. To find these less frequent scheduling times one can utilise the frequency matrix $\mathcal{F}$. The rarest time period, that is not the current scheduled time or $T+1$, for each block i can be found:

$$t_i = \underset{t \neq x_i^*, T+1}{\operatorname{argmin}} \mathcal{F}_{it} \tag{9.4}$$

Let $\phi_i = \mathcal{F}_{it_i}$, the number of times block i was shifted to this rare time period t_i. The probability of selecting a block, i, for shifting to its infrequent period t_i is then set to be inversely proportional to its ϕ_i value.

$$\mathbb{P}b_i = \frac{1}{\phi_i} \tag{9.5}$$

Denote j the block randomly selected according to these probabilities and perform the shift, $x = x^* \oplus (j, t_j)$. Note that this shift neglected to check if the new schedule is feasible ie. that it satisfies precedence constraints. It may be that block j was shifted later and some successors of j may now be scheduled earlier than j, or that block j was scheduled earlier and some predecessors may now be designated for extraction after block j's extraction. These potential scenarios are written below:

$$\begin{cases} \text{if} \quad t_j > x_j^* \quad \text{possible that} \quad x_s < x_j, \quad s \in S_j, \\ \text{if} \quad t_j < x_j^* \quad \text{possible that} \quad x_p > x_j, \quad p \in P_j. \end{cases}$$

It is therefore necessary to retrieve the feasibility of the schedule should any of the above be the case.

Let ρ be the set of all predecessors or successors, depending on which case is being encountered, that violate the precedence constraints. For explanatory purposes, assume that the case is that of $t_j > x_j^*$. Then $\rho = \{s \in S_j : x_s < x_j\}$.

The feasibility retrieval process is iterative. A random element s of ρ is selected at each iteration. $\alpha_s = \max_{k \in P_s, k \notin \rho} \{x_k\}$ and $\beta_s = \min_{l \in S_s, l \notin \rho} \{x_l\}$ are then found where α_s is the latest predecessor of s not in ρ and β_s the earliest successor of s not in ρ. This troublesome block s is now scheduled a new time τ_s according to equation (9.6):

$$\tau_s = \underset{t \in [\alpha_s, \beta_s], t \neq x_s^*}{\operatorname{argmin}} \{\mathcal{F}_{st}\} \tag{9.6}$$

Therefore, τ_s is the most infrequent time period for block s in the range $[\alpha_s, \beta_s]$, but not its current extraction time. After performing the shift $x = x \oplus (s, \tau_s)$, both $Tabu$ and $\mathcal{F}$ are updated accordingly. s is then removed from ρ and the feasibility retrieval process continues until ρ is empty.

For the sake of brevity, the usage of the Long-term Memory structure mechanism in this research shall be referred to as LTM.

9.1.4 Vanilla-TS Algorithm

Now that the key concepts and terminology for Vanilla-TS have been laid out the algorithm composition can be described.

9.1.4.1 Nested Tabu Search

The first time the nested tabu search algorithm is run the starting/reference schedule, x_0 is from the schedule produced by the ExTS heuristic. This section explains the tabu search component which operates within the stopping criteria.

The tabu search terminates after *nitermax* consecutive non-improving iterations. *niter* is a counter that keeps track of the non-improving iterations in tabu search.

On the reference schedule, x_0, a large number of shifts are generated to conceive a large number of schedules. The objective function value of these schedules is then computed in-line with equation (7.10). The best non-tabu and tabu schedules are then stored. Recall that a tabu schedule is one that is generated with a tabu shift. That is, the proposed shift that generates the solution has already been selected less than θ iterations prior.

If the best tabu schedule is better than the best found in the entirety of the algorithm and is better than the non-tabu shift schedule then the classical aspiration criterion comes into effect. The schedule is accepted as the new reference schedule and x^* and x_{best} are updated accordingly, along with the memory structures. Otherwise, if the classical aspiration criterion condition fails, then the non-tabu shift schedule is set as the new reference schedule, along with the necessary relevant updates.

The new reference schedule x_0 may have illicited an update in x^*. If this is the case *niter* is reset to 0. If not, then *niter* is incremented. Furthermore, there is also a check to see if x^* is better than x_{best} ie. if the tabu search current best schedule is more profitable than the most profitable schedule found thus far in Vanilla-TS.

Algorithm 11 provides the pseudocode for the tabu search in Vanilla-TS:

Algorithm 11 Pseudo Code for Vanilla-TS

```
1  function TabuSearch
2  niter ← 0
3  while niter < nitermax do
4    Generate a large number of shifts on reference schedule x_0 to produce feasible
     schedule set, S.
5    for all x' ∈ S do
6      Compute f(x')
7    end for
8    Set x_Tabu as the best schedule in X that is tabu.
9    Set x_NT as the best schedule in X that is non-tabu.
10   if f(x_Tabu) > f(x_best) and f(x_Tabu) > f(x_NT) then
11     x_0 ← x_Tabu and f(x_0) ← f(x_Tabu)
12     Update x*, f(x*), x_best, f(x_best)
13     Update Tabu and F
14     niter ← 0
15   else
16     x_0 ← x_NT and f(x_0) ← f(x_NT)
17     Update Tabu and F
18     if f(x_0) > f(x*) then
19       Update x*, f(x*)
20       niter ← 0
21       if f(x*) > f(x_best) then
22         Update x_best, f(x_best)
23       end if
24     else
25       niter ← niter + 1
26     end if
27   end if
28 end while
```

9.1.4.2 Vanilla-TS Long Term Memory

The least frequent scheduled time period is computed for each block, t_i. The number of times that each block i has been shifted to this time period is then also calculated and stored as ϕ_i from which a probability distribution of selection is synthesised.

After randomly selecting a block j according to this distribution, a new schedule is created by performing the shift (j, t_j). Note that the relevant memory and data structures are updated every time there is a shift.

This shift may have generated a schedule that is infeasible due to violation of precedence constraints. The respective feasibility retrieval process then commences depending on if there is an infeasibility caused by block j shifting later or earlier.

The LTM portion of Vanilla-TS terminates when feasibility is assured and then duly returns a new reference schedule x_0 for the tabu search.

Algorithm 12 provides an outline of the pseudocode for this diversification strategy in Vanilla-TS.

Algorithm 12 Diversification Strategy for Vanilla-TS

function LTM
for each block i **do**
  $t_i \leftarrow \underset{t \neq x_i^*, T+1}{\operatorname{argmin}} \mathcal{F}_{it}$
  $\phi_i \leftarrow F_{it_i}$
  $\mathbb{P}[b_i] \leftarrow \frac{1}{\phi_i}$
end for
Randomly select block j according to probabilities $\mathbb{P}[b]$
$x \leftarrow x^* \oplus (j, t_j)$
Update $Tabu$, $\mathcal{F}$ and C.
if $t_j < x_j$ **then**
  $\rho \leftarrow \{p \in P_j : x_p > x_j\}$
  while ρ **not** empty **do**
    Randomly select element p from ρ
    $\alpha_p \leftarrow \max_{k \in P_p, k \notin \rho} \{x_k\}$
    $\beta_p \leftarrow \min_{l \in S_p, l \notin \rho} \{x_l\}$
    $\tau_p \leftarrow \underset{t \in [\alpha_p, \beta_p]}{\operatorname{argmin}} \{\mathcal{F}_{pt}\}$
    $x \leftarrow x \oplus (p, \tau_p)$
    Update $Tabu$, $\mathcal{F}$ and C.
  end while
else
  $\rho \leftarrow \{s \in S_j : x_s < x_j\}$
  while ρ **not** empty **do**
    Randomly select element s from ρ
    $\alpha_s \leftarrow \max_{k \in P_s, k \notin \rho} \{x_k\}$
    $\beta_s \leftarrow \min_{l \in S_s, l \notin \rho} \{x_l\}$
    $\tau_s \leftarrow \underset{t \in [\alpha_s, \beta_s]}{\operatorname{argmin}} \{\mathcal{F}_{st}\}$
    $x \leftarrow x \oplus (s, \tau_s)$
    Update $Tabu$, $\mathcal{F}$ and C.
  end while
end if
$x_0 \leftarrow x$

9.1.4.3 General Overview of Vanilla-TS

Given the explanations of the tabu search and diversification strategy portions, the Vanilla-TS algorithm can be outlined in Algorithm 13 below:

Algorithm 13 Vanilla-TS

Acquire reference schedule x_0 from ExTS heuristic
while Stopping criterion not satisfied **do**
    Call $TabuSearch$ with reference schedule x_0
    Output: x^*, the best schedule found within the $TabuSearch$ iteration.
    Call LTM with x^*
    Output: x_0, a new reference schedule for $TabuSearch$
end while
Output : x_{best} and $f(x_{best})$, the best schedule obtained during entire algorithm.

Chapter 10

Simulated Annealing Metaheuristic for the OPMPSP

This chapter will describe the various forms of simulated annealing that were implemented in this dissertation. An initial model is first described that is mostly derived from the simulated annealing algorithm seen in [Kumral 2013]. Modifications to this first algorithm are then introduced and explained.

10.1 Vanilla Simulated Annealing

[Kumral 2013] seeks to optimise two problems, a block sequencing and *ore-waste discrimination*. Ore-waste discrimination accommodates for varying cut-off grades, essentially also factoring in metal uncertainty, as seen in [Lamghari et al. 2012]. [Kumral 2013] introduces another dimension to his formulation that includes a number of simulated block models. These models have differing cut-off grades. For example, block i may be designated a waste block in simulation s but seen as an ore block in simulation s'. The vanilla simulated annealing algorithm of this research, henceforth referred to as *Vanilla-SA*, does not look to answer the ore-waste discrimation question but only creating an optimal mining schedule.

10.1.1 A Summary of Vanilla-SA

The progression of Vanilla-SA can be summarised as follows:

1. Calculate an initial solution for Vanilla-SA.

2. Randomly select a portion of blocks to shift.
3. Shift these blocks.
4. Accept this transition based on simulated annealing acceptance criteria and updating temperature parameter.
5. Repeat steps 2 - 4 for a given number of iterations.
6. Terminate if convergence is reached, otherwise return to step 2.

10.1.2 An Initial Solution for Vanilla-SA

Like Vanilla-TS, Vanilla-SA uses the schedule produced by the ExTS heuristic as an initial solution for its algorithm. This starting point is essentially a local optimum. However, Vanilla-SA has the mechanisms to escape local optima and explore the solution space for in hope of finding the global optimum.

10.1.3 Terminology and Key Concepts

10.1.3.1 Transition Mechanism

The transition mechanism works by randomly selecting the number of blocks, B, to shift during the current iteration. A random number $r \in [1, T]$ is then generated to determine which time period will experience this perturbation. Another random number is generated to determine the direction of the perturbation, backwards or forwards. The transition to a new solution is then implemented in such a way that precedence constraints (Equation (7.12)) are not violated. An example of how the transition mechanism would work on a small 2-dimensional set can be seen in Figure 10.1.

The left figure shows the original schedule. The time period to be perturbed is 4 and 2 blocks were randomly selected, indicated in this figure by the downward arrows and underlining. The right figure is the new schedule as a result of shifting the 2 blocks back a time period. These 2 blocks that were originally scheduled for extraction in time period 4 have now been moved to be extracted in time period 3.

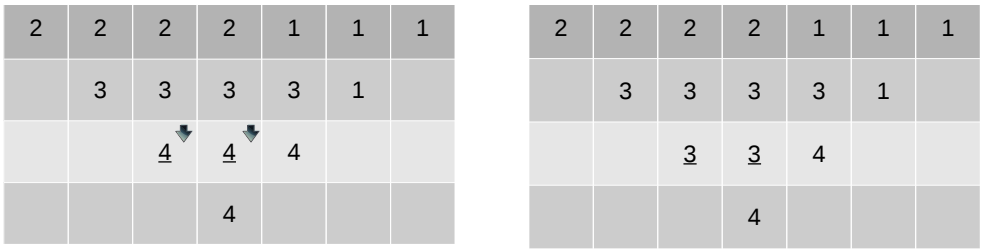

FIGURE 10.1: Example of Transition Mechanism in Vanilla-SA

10.1.3.2 Acceptance Criterion

The general acceptance criterion for simulated annealing is used in Vanilla-SA. That is, the acceptance criterion based around the Boltzmann distribution and explained in detail in section 4.1.4.

10.1.3.3 Temperature Adjustment through Dynamic Heating and Cooling

Unlike [Kumral 2013], whenever a transition passes the acceptance criterion the temperature is cooled by the following equation, with α and β been constants:

$$Temp = (1 - \beta) \times Temp \tag{10.1}$$

Whenever a solution is rejected then the temparture is heated according to the following equation:

$$Temp = (1 + \alpha) \times Temp \tag{10.2}$$

Cooling and heating in this fashion was found to offer the algorithm a more robust exploration of the solution space. It also enabled a faster progression than was given by using the temperature adjustments found in [Kumral 2013].

10.1.3.4 A Stopping Criterion Derived from Cooling and Heating Coefficients

Let k be the ratio of β to α, the cooling coefficient and heating coefficient respectively in equations. This implies that k heating steps balance with one cooling step. Theoretically, the temperature system should reach convergence when the ratio of cooling to heating is

k and thus, it is utilised as a stopping criterion. The determination of α and β therefore drives k, which will decide how Vanilla-TS terminates.

The empirical tests were done to decide upon good values for α and β that allowed for sufficient iterations, especially to escape local optima, thus discovering a good balance between heating and cooling.

10.1.4 Vanilla-SA Algorithm

The key concepts have been laid out for Vanilla-TS. This section will now elaborate on these concepts by explaining their use within the algorithm.

10.1.4.1 Simulated Annealing Operation

The simulated annealing loop of Vanilla-TS is the structural core of the algorithm and has L iterations, where L is the length of the Markov Chain. Therefore, the procedure described below will repeat L times.

Firstly, the number of blocks to be shifted must be randomly generated. This will be an integer between 1 and $Trans$, the maximum number of transitions allowed in a specific iteration. $Trans$ is a parameter of the algorithm and is defined prior to execution. Each block is then shifted either forward by one time period or backward by one time period, making sure that this transition does not violate the precedence constraints.

If this transition results in a schedule that is better than the previous schedule it is accepted. The old schedule is overwritten with the new schedule, a counter variable accrues another accepted move (for purpose later in the algorithm) and the temperature is dynamically cooled according to equation (10.1).

If the transitioning results in a schedule that is worse than the current schedule then it can still be accepted with probability around the acceptance criterion seen in equation (4.3). However, in the realm of a maximisation problem, in the setting of Vanilla-TS, that acceptance criterion has the following form:

$$\mathbb{P}_a = \exp\left\{\frac{f(y) - f(x)}{Temp}\right\}, \tag{10.3}$$

where $f(y)$ is the new proposed schedule as a result of the transition and $f(x)$ the current schedule.

Should this acceptance criterion pass then a counter variable accrues an acceptance iteration and the temperature is cooled. Otherwise, a counter variable will store that this iteration in the Markov Chain had a rejected transtion and the temperature is dynamically heated according to equation (10.2).

At the conclusion of the Markov Chain loop component of Vanilla-SA a new schedule is returned along with the number of accepted and rejected transitions. A ratio is calculated between the accepted and rejected moves. Should the absolute value of difference between k and this ratio be less than 3 Vanilla-SA terminates. If the ratio is greater than k then the temperature is heated according to equation (10.2). Otherwise, it is cooled by equation (10.1).

Algorithm 14 provides a pseudo code outline form of Vanilla-SA. Much of the intricacies of simulated annealing in general have been ommited since they have been discussed in section 4.1.4.

Algorithm 14 Vanilla-SA

Initialise parameters for Vanilla-SA : α, β, $Temp$, L and $Trans$
$k \leftarrow \frac{\beta}{\alpha}$
convergence $\leftarrow False$
$accepted \leftarrow 0$
$rejected \leftarrow 0$
while convergence *is* $False$ **do**
  for L iterations **do**
    Randomly select integer in range [1 $Trans$], blocks to move
    Move these blocks to a later time period or earlier time period
    Accept or reject the new schedule based on Acceptance Criterion
    if New schedule is accepted **then**
      Cool temperature: $Temp \leftarrow \frac{Temp}{1+\beta\ Temp}$
      $accepted \leftarrow accepted + 1$
    else
      Heat temperature: $Temp \leftarrow \frac{Temp}{1-\alpha\ Temp}$
      $rejected \leftarrow rejected + 1$
    end if
  end for
  $ratio \leftarrow \frac{rejected}{accepted}$
  if $|k - rejected| < 3$ **then**
    convergence $\leftarrow True$
  else if $ratio > k$ **then**
    $Temp = \frac{Temp}{1-\alpha\ Temp}$
  else
    $Temp = \frac{Temp}{1-\beta\ Temp}$
  end if
end while
Output : Returns final schedule with corresponding objective function value.

Chapter 11

Implementation and Results

11.1 Problem Instances and Computation Specs

The following algorithms, and subsequent results, are simulations run on the *kd model* instance obtainable from [Minelib], a copper deposit in Arizona USA. Some information on this mining instance can be found in Table 11.1 below:

TABLE 11.1: Parameters for *KD* mining instance

Parameter	Description	Value
n	number of blocks	$14,153$
$preds$	number of precedences	$219,778$
d_1	discount rate	0.15
$\bar{W}$	Upperbound mining capacity tonnage	∞
$\bar{O}$	Upperbound processing capacity tonnage	$10,000,000$
T	number of periods	12

Notice that there is no mining capacity constraints on this particular block model case.

We also include the *Newman* instance from [Minelib] to showcase the strength that TS has with smaller problem instances. Its parameters can be seen in the Table 11.2 below:

TABLE 11.2: Parameters for *Newman* mining instance

Parameter	Description	Value
n	number of blocks	$1,060$
$preds$	number of precedences	$3,922$
d_1	discount rate	0.08
$\bar{W}$	Upperbound mining capacity tonnage	$2,000,000$
$\bar{O}$	Upperbound processing capacity tonnage	$1,100,000$
T	number of periods	6

It should be noted that we do not agree with the interpretation of the *time value of money* or discounting reflected in the results in [Minelib]. There it is assumed that blocks mined in the first period are devoid of discounting ie. a piece of ground designated for mining in the first period will realise its full profit. This is akin to assuming that extraction and processing takes place instantaneously.

For the purposes of comparative feedback from our results we will also assume this in the subsequent sections.

All computations were run in the technical computing language, MATLAB. Our machine specifications are given below in Table 11.3:

TABLE 11.3: Computer Specifications

Feature	Hardware
Processor	Intel(R) 8 Core(TM) i7-3770K CPU @ 3.50 GHz
RAM	16 GB
System Type	64-bit OS
Graphics	NVIDIA GeForce GTX670

11.2 Vanilla Initial Solutions

The weighting functions of [Gershon 1987] (see Section 8.5.2) and the Expected extraction time (Section 8.5.3) were applied. On these block weighting foundations, the Toposort heuristic described in Algorithm 9 was utilised. Recall that these two initial solutions are referred to as Gershon Toposort (GeTS) and Expected extraction time Toposort (ExTS) [Chicoisne et al. 2012]. The analysis and results that follow are related to computations on the KD mining instance.

11.2.1 NPV Analysis

Table 11.4 shows the numerical results for NPV per period for the ExTS and GeTS algorithms as well as the cumulative NPV for both of these approaches. What we notice is that there is actually no need for periods 11 and 12 as the extraction is completed, evidenced by the 0 NPV values for those periods. Figure 11.1, Figure 11.2 and Figure 11.3 represent these numerical results graphically. The bars represent the NPV realised during that particular time period whereas the line component of the plots represents the cumulative NPV attained as we progress through the mining periods.

The first thing that one notices is that GeTS outperforms ExTS. This is best seen by inspection of Figure 11.3. This is rather surprising since ExTS is a more modern algorithm in this problem. However, the random nature of both algorithms can explain how GeTS could have gotten a better solution. Both NPV profiles reveal a poor start to the extraction scheme followed-up by marked improvement, only to plateau somewhat in the latter periods. It should be noted that no extraction takes place in the last 2 periods.

According to [Espinoza et al. 2012], [Muñoz 2012] has the best NPV solution for the KD block model instance to date, with an NPV of $\$396,858,193$ and corresponding optimality gap of 3.1%. The initial solutions of GeTS and ExTS have optimality gaps of 19.63% and 21.55% respectively. These are poor but it should be mentioned that they are obtained in execution times of ≈ 10 seconds and this is measured against an LP relaxation that is not attainable.

TABLE 11.4: NPV Results for GeTS and ExTS

Time Period	GeTS NPV/Period	GeTS Cumulative NPV	ExTS NPV/Period	ExTS Cumulative NPV
1	-3.31×10^7	-3.31×10^7	-2.90×10^7	-2.90×10^7
2	6.26×10^7	2.95×10^7	5.99×10^7	3.09×10^7
3	5.62×10^7	8.57×10^7	5.91×10^7	9.00×10^7
4	5.44×10^7	1.40×10^8	5.85×10^7	1.49×10^8
5	4.93×10^7	1.89×10^8	3.03×10^7	1.79×10^8
6	4.19×10^7	2.31×10^8	3.40×10^7	2.13×10^8
7	3.61×10^7	2.67×10^8	3.46×10^7	2.47×10^8
8	3.02×10^7	2.98×10^8	2.89×10^7	2.76×10^8
9	2.33×10^7	3.21×10^8	2.84×10^7	3.05×10^8
10	8.12×10^6	3.29×10^8	1.65×10^7	3.21×10^8
11	0	3.29×10^8	0	3.21×10^8
12	0	3.29×10^8	0	3.21×10^8

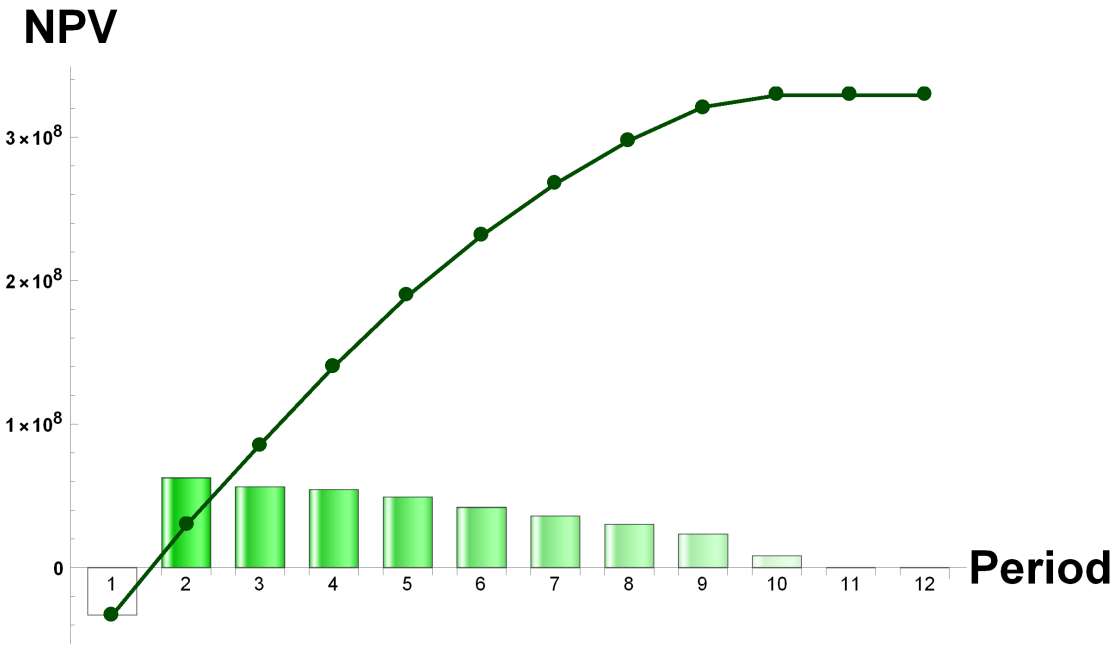

FIGURE 11.1: NPV Results - GeTS

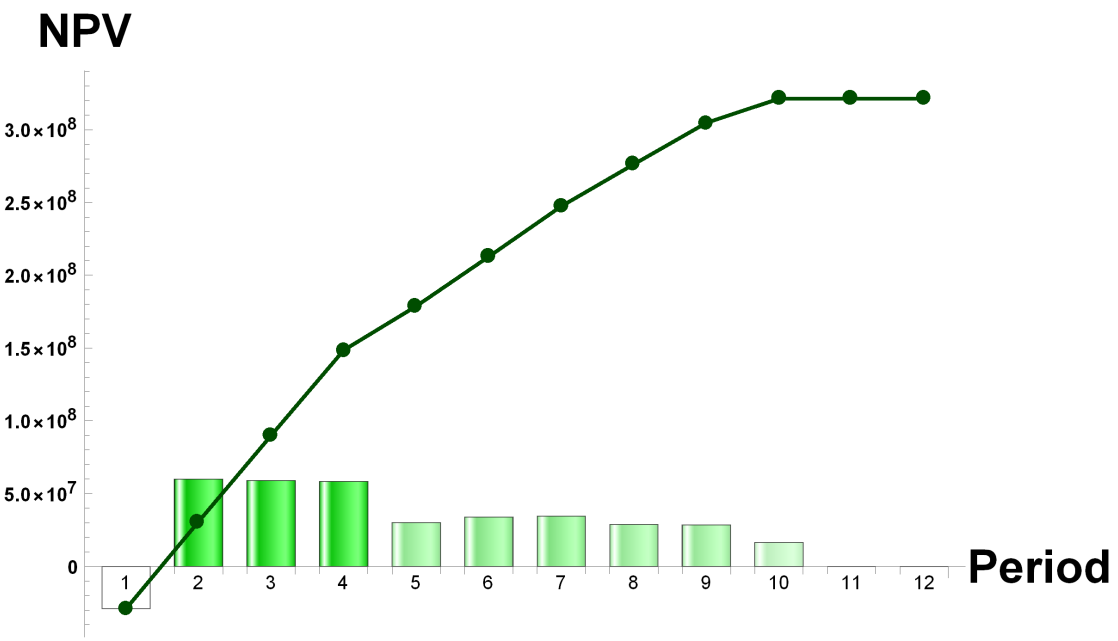

FIGURE 11.2: NPV Results - ExTS

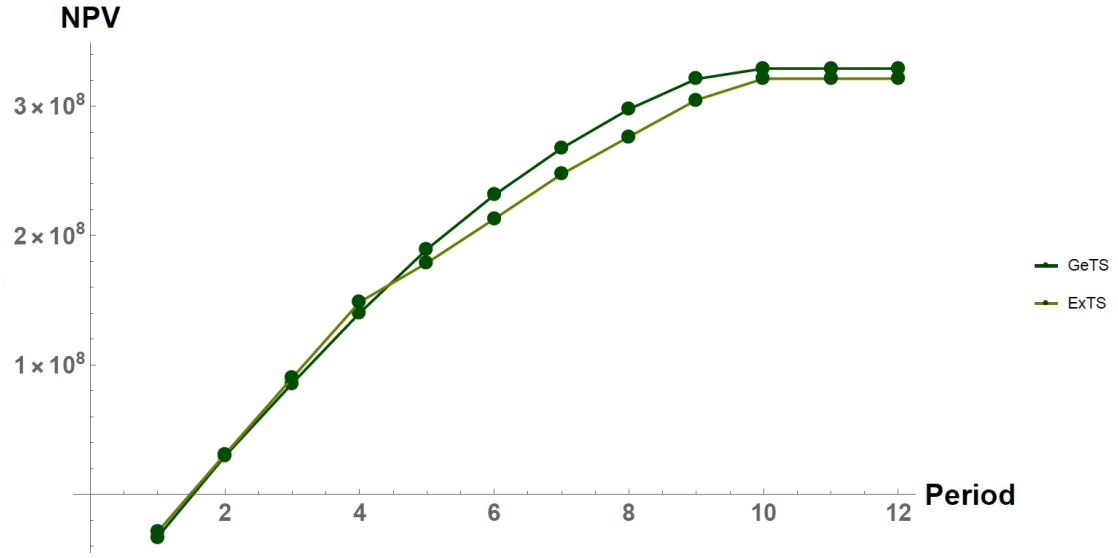

FIGURE 11.3: NPV Comparison - GeTS and ExTS

Ideally, we would like to try and acquire better initial solutions. This will increase the chances eventually bettering the NPV solution obtained by [Muñoz 2012] once further heuristics are applied.

A concerning observation of Figure 11.3 is the "tailing-off" of the cumulative NPV line graphs of both GeTS and ExTS. Of course, this would not be such an issue if the slope of these lines exhibited increase however, we identify a decreasing trend that starts in the latter-periods. Further graphical evidence of this can be seen in Figure 11.3 and Figure 11.2. The higher, darker bars represent the more profitable time periods during

the mining venture. One should expect early periods to be profitable. The earlier ore blocks are excavated the less their economic realisations are discounted. In general, as we progress to later periods, we will see a decline in profitability as discounting grows. However, we are confronted with a poor start to the mining venture, something which would not instil confidence in investors.

Ensuring that ore blocks are mined as soon as possible and waste blocks are mined as late as possible, if at all, will increase the NPV of our initial solutions. This will be discussed in a later section.

11.2.2 Processing Production Activity Analysis

Table 11.5 shows the numerical results for the mass (in tons) sent to the processing plant per period for the ExTS and GeTS algorithms as well as the accumulation of this rock mass at these processing plants for both of these approaches. We again notice 0 values in the 11^{th} and 12^{th} periods, indicating that mining has concluded since no processing is taking place.

Figures 11.4 and 11.5 reveal that both the GeTS and ExTS schedules send the maximum mass to the processing plant during periods 1 - 10, with minor differences between the two initial solutions. This efficiency is brought about by the workings of the Toposort algorithm. Processing plants at maximum capacity are generally indicative of a good schedule. It can mean that valuable ore blocks have not been left to devalue by extracting them in later periods.

From this however, one can infer that GeTS was better at scheduling the extraction of the ore blocks.

TABLE 11.5: Processing Production Statistics for GeTS and ExTS

Time Period	GeTS Processing/Period	GeTS Cumulative Processing	ExTS Processing/Period	ExTS Cumulative Processing
1	9.9882×10^6	9.9882×10^6	9.9947×10^6	9.9947×10^6
2	9.9849×10^6	1.9973×10^7	9.9944×10^6	1.9989×10^7
3	9.9940×10^6	2.9967×10^7	9.9995×10^6	2.9989×10^7
4	9.9860×10^6	3.9953×10^7	9.9862×10^6	3.9975×10^7
5	9.9890×10^6	4.9942×10^7	9.9936×10^6	4.9968×10^7
6	9.9844×10^6	5.9927×10^7	9.9881×10^6	5.9957×10^7
7	9.9977×10^6	6.9924×10^7	9.9859×10^6	6.9942×10^7
8	9.9910×10^6	7.9915×10^7	9.9845×10^6	7.9927×10^7
9	9.9952×10^6	8.9911×10^7	9.9839×10^6	8.9911×10^7
10	6.2669×10^6	9.6177×10^7	6.2666×10^6	9.6177×10^7
11	0	9.6177×10^7	0	9.6177×10^7
12	0	9.6177×10^7	0	9.6177×10^7

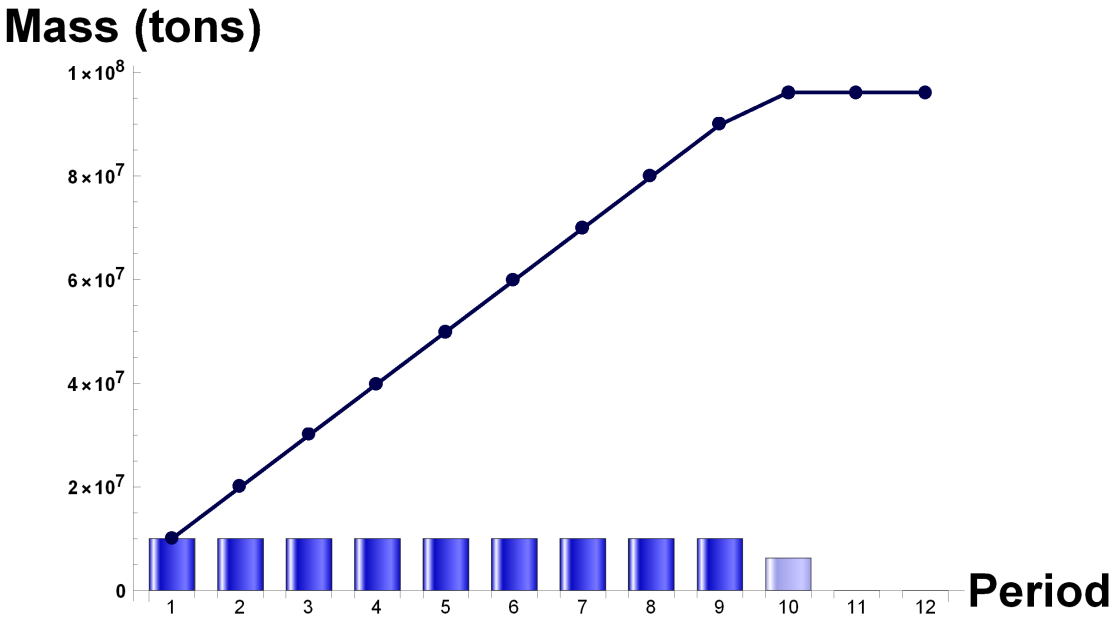

FIGURE 11.4: Processing Production - GeTS

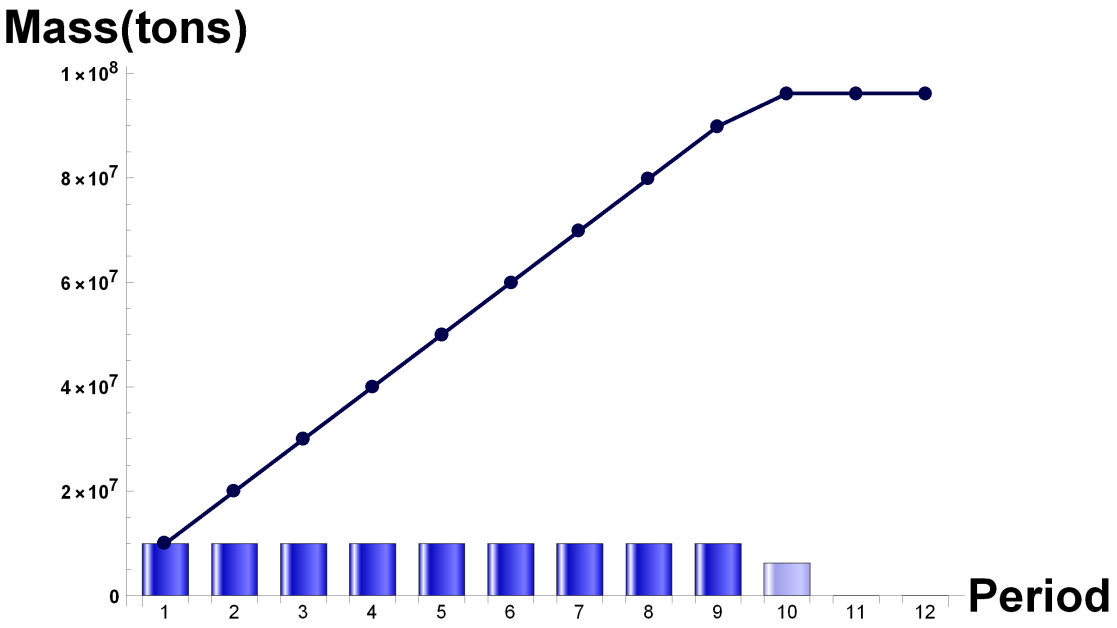

FIGURE 11.5: Processing Production - ExTS

11.2.3 Conclusion

Analysis of the NPV and processing production revealed both algorithms are devoid of glaring inefficiencies. However, blocks should only be extracted if it is economically viable. Waste blocks should not be extracted at all unless it is to free up access to rich ore blocks. Furthermore, ore blocks should be extracted as early as possible.

Additionally, a constraint that is not considered in our model, but is a realistic proposal, is an NPV minimum constraint per period. One would find the confidence of investors wavering if a period showed a negative NPV. Even if the schedule ensured positive economic gain for future periods, it would prove difficult explaining to those who have vested interests in the mining venture.

The next section will describe the development of a greedy-like heuristic that is applied to the schedules of GeTS and ExTS in order to allocate waste blocks for mining as late as possible, or never to be extracted, as well as designating ore blocks for extraction as early as possible. This will be done within the confines of the precedence and resource constraints and should greatly benefit the performance metrics of the schedules. The section will then conclude with an analysis of the new schedule(s).

11.3 A Pseudo-Greedy Search (PGS)

As we saw in the previous section, GeTS and ExTS gave relatively good solutions however, after some analysis it was identified that there is room for improvement.

We describe here a means to improve upon these initial solutions. This is applicable to any stage of a heuristic algorithm to solve the OPMPSP. It is one of the contributions of this research dissertation.

Section 11.2.1 revealed a decline in NPV for both GeTS and ExTS in the latter periods of the schedules. This is indicative of the needless mining of waste blocks. An immediate, if rather specious, solution to this issue is to schedule waste blocks for extraction as late as possible, if at all. For this we need to make use of the data structure C described in Section 9.1.3.2.

Let $\mathbb{W}$ denote the set of all waste blocks that have the potential for later extraction for each block k in our mining instance,

$$\mathbb{W} = \{j : v_j < 0\} \cap \{k : C_{2k} > x_k\}$$

Recall that C is a data structure that keeps track of the earliest possible extraction time for a particular block, k, as well as the latest possible extraction time. The latest possible extraction time is stored in the second dimension, as seen in the above equation.

We then denote l_i the latest possible extraction time for a block i currently scheduled for extraction at a time x_i. This is essentially the earliest time, x_s, seen in the set of block i's successors, S_i.

$$l_i = \min_{s \in S_i} x_s$$

The logic behind determining this value l_i is that a block i cannot be extracted later than the earliest time amongst its successors.

The objective is to set the extraction time of these blocks found in $\mathbb{W}$ to as late as possible, with aid of l_i values, whilst making sure that the resource constraints are still upheld. Essentially, we change the extraction time of the block i, a member of our candidate waste set $\mathbb{W}$, to be this latest possible extraction time l_i. Therefore, we amend the production schedule as follows:

$$x_i = l_i, \quad i \in \mathbb{W} \tag{11.1}$$

Due to the time value of money component in OPMPSP, this will lessen the NPV impact for extracting negative value blocks. Some waste blocks may have their extraction times set to $T + 1$, meaning that their negative value will never be realised because they are never extracted. Scheduling waste block extractions as late as possible for a given schedule creates a new schedule with improved NPV however, the next step in PGS has the potential to further increase the NPV.

Referring to Figure 11.4 and 11.5, the ExTS and GeTS schedules have similar ore block processing, yet they have differing NPV values. This points towards a potential schedule differing in terms of ore blocks getting mined earlier. Mining rich blocks earlier lessens the impact of discounting revenue flow. Therefore, it would be a natural improvement on any schedule to ensure that ore blocks are mined as soon as possible. Once again we make use of the C structure from the Vanilla-TS algorithm.

Let $\mathbb{O}$ denote the set of all ore blocks that hold some promise of being scheduled for extraction earlier,

$$\mathbb{O} = \{j : v_j > 0\} \cap \{k : C_{1k} < x_k\}$$

and e_i the earliest possible extraction time for a block i currently scheduled for extraction at a time x_i. This will be the latest extraction time of all the blocks p in the predecessor set P_i.

$$e_i = \max_{p \in P_i} x_p$$

The rationale behind this is that a block i cannot be extracted before the latest extraction time amongst its predecessors.

The extraction time for the block i can thus be set to the earliest possible extraction time without violating the precedence constraint and if resource constraints are not violated.

$$x_i = e_i, \quad i \in \mathbb{O} \tag{11.2}$$

The approach operates by first scheduling waste blocks for later time periods, seen in equation (11.1), and then ore blocks as early as possible, equation (11.2). The algorithm terminates if there is a iteration fails to bring about the movement of a single waste block followed consecutively by a failure to move a single ore block.

Should $\mathbb{W}$ not be empty, a random block from $\mathbb{W}$ is chosen to be moved to the latest possible time period. If this fails the algorithms makes another set of attempts until a valid candidate from $\mathbb{W}$ is randomly selected. The rescheduling of the waste blocks stops when the counter reaches a predetermined maximum or if $\mathbb{W}$ is empty. The ore block rescheduling component of the algorithm performs in the same manner with random blocks in $\mathbb{O}$ being scheduled as soon as possible rather than as late as possible.

It is fully presented below in Algorithm 15:

Algorithm 15 Outline of the Pseudo-Greedy Search

```
Inputs: {C, f, x}
wastesMoved ← True
oresMoved ← True
while wastesMoved = True OR oresMoved = True do
  % Move waste blocks.
  wastesMoved ← False
  Find set 𝕎
  count ← 0
  while 𝕎 is not empty and count < countmax do
    Randomly select element, i, from 𝕎
    Find l_i, the latest possible extraction time for i
    that does not violate any resource constraints.
    if l_i = ∅ then
      count ← count + 1
    else
      wastesMoved ← True
      x_i ← l_i
      Update f(x), C and resource constraints
      Find set 𝕎
      count ← 0
    end if
  end while
  % Move ore blocks.
  oresMoved ← False
  Find set 𝕆
  count ← 0
  while 𝕆 is not empty and count < countmax do
    Randomly select element, i, from 𝕆
    Find e_i, the earliest possible extraction time for i
    that does not violate resource constraints.
    if e_i = ∅ then
      count ← count + 1
    else
      oresMoved ← True
      x_i ← e_i
      Update f(x), C and resource constraints
      Find set 𝕆
      count ← 0
    end if
  end while
end while
```

Due to the superior performance of GeTS over ExTs in the *KD* instance, PGS was only applied to GeTs. The following section investigates the extent of the improvement that PGS achieved from the initial production schedule produced by GeTs. The statistics of processing production is also analysed.

11.3.1 NPV Analysis of PGS

Referring to Table 11.6, one can identify the improvement that PGS has made on GeTS. The majority of early periods see more realised NPV. The cumulative, and thus final NPV of the schedule, is improved by the application of PGS. NPV increases by 15.78%, in monetary terms, $\$51,930,320$. One also sees that the production schedule is completed within 10 periods, as evidenced by the 0 values in Table 11.6.

The functionality of the PGS algorithm is aptly highlighted in Figure 11.6. The histogram portion has a skewness to the left. This is what PGS is trying to extenuate by attempting to realise the profits as early as possible. Compared to the GeTS and ExTS graphs, the cumulative line graph for PGS flattens at around the same periods. This is evident upon inspection of Figure 11.7, a comparison of the cumulative NPVs on a period to period basis.

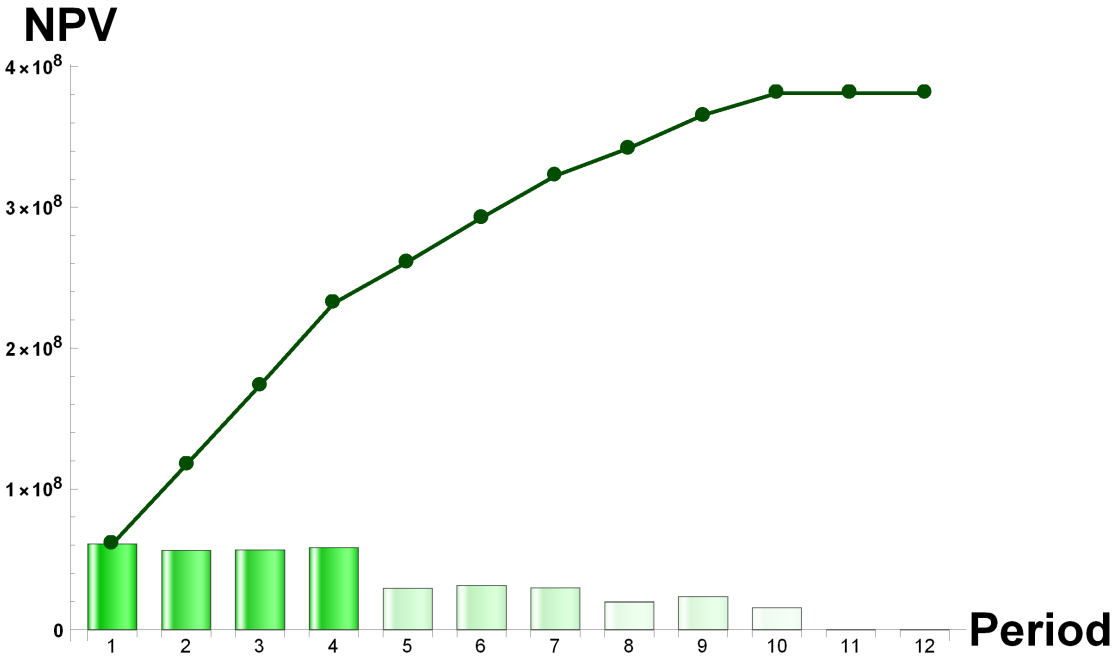

FIGURE 11.6: NPV Performance of PGS on GeTS

TABLE 11.6: NPV Results for ExTS and PGS

Time Period	ExTS NPV/Period	ExTS Cumulative NPV	PGS NPV/Period	PGS Cumulative NPV
1	-3.3099×10^7	-3.3099×10^7	6.0860×10^7	6.0860×10^7
2	6.2562×10^7	2.9463×10^7	5.6189×10^7	1.1705×10^8
3	5.6225×10^7	8.5688×10^7	5.6570×10^7	1.7362×10^8
4	5.4411×10^7	1.4010×10^8	5.8296×10^7	2.3191×10^8
5	4.9283×10^7	1.8938×10^8	2.9449×10^7	2.6136×10^8
6	4.1936×10^7	2.3132×10^8	3.1238×10^7	2.9260×10^8
7	3.6101×10^7	2.6742×10^8	2.9822×10^7	3.2242×10^8
8	3.0216×10^7	2.9763×10^8	1.9709×10^7	3.4213×10^8
9	2.3341×10^7	3.2098×10^8	2.3390×10^7	3.6552×10^8
10	8.1225×10^6	3.2910×10^8	1.5506×10^7	3.8103×10^8
11	0	3.2910×10^8	0	3.8103×10^8
12	0	3.2910×10^8	0	3.8103×10^8

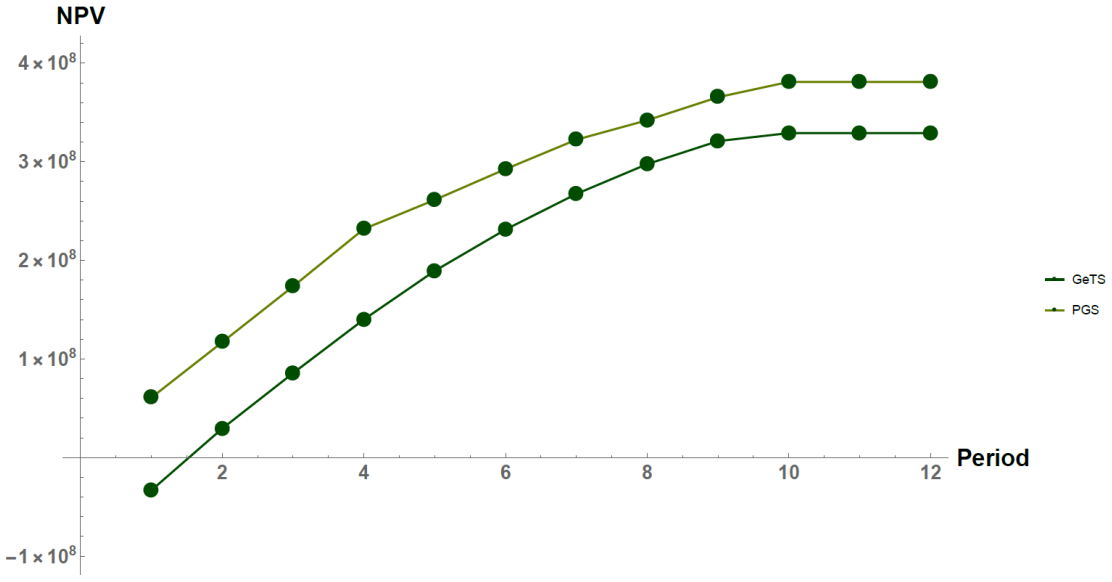

FIGURE 11.7: NPV Comparison - PGS and GeTS

11.3.2 Processing Production Analysis

Table 11.7 reveals the tonnage of rock processed per period for GeTS and PGS. The small difference between PGS and GeTS with regards to processing production suggests that the primary source of improvement on GeTS with PGS is the rescheduling of waste blocks for as late as possible.

TABLE 11.7: Processing Production Statistics for ExTS and PGS

Time Period	GeTs Processing/Period	GeTs Cumulative Processing	PGS Processing/Period	PGS Cumulative Processing
1	9.9882×10^6	9.9882×10^6	9.9947×10^6	9.9947×10^6
2	9.9849×10^6	1.9973×10^7	9.9944×10^6	1.9989×10^7
3	9.9940×10^6	2.9967×10^7	9.9995×10^6	2.9989×10^7
4	9.9860×10^6	3.9953×10^7	9.9862×10^6	3.9975×10^7
5	9.9890×10^6	4.9942×10^7	9.9936×10^6	4.9968×10^7
6	9.9844×10^6	5.9927×10^7	9.9881×10^6	5.9957×10^7
7	9.9977×10^6	6.9924×10^7	9.9859×10^6	6.9942×10^7
8	9.9910×10^6	7.9915×10^7	9.9845×10^6	7.9927×10^7
9	9.9952×10^6	8.9911×10^7	9.9839×10^6	8.9911×10^7
10	6.2669×10^6	9.6177×10^7	6.2666×10^6	9.6177×10^7
11	0	9.6177×10^7	0	9.6177×10^7
12	0	9.6177×10^7	0	9.6177×10^7

11.3.3 Conclusion

The NPV of PGS exceeds that of ExTS and GeTS, significantly improving upon these initial algorithms. The reasoning in the previous section with the analysis of ExTS and GeTS proved to be correct - there are a number of waste blocks that are needlessly mined in the GeTS and ExTS schedules or could have been scheduled for extraction later. The tonnage of ore blocks processed remains relatively unchanged between GeTS and PGS.

This points further towards a more optimal schedule with waste blocks being moved later and ore blocks getting mined earlier.

11.4 Tabu Search

This section focuses on the vanilla implementation of tabu search, detailed in section 9.1. The algorithm was run on the *KD* mine instance freely obtainable from [Minelib].

A smaller problem is first examined with an attempt at parameter optimisation before moving on to the larger *KD* model, our fundamental instance for algorithm simulations and resultant performance.

We then look at the NPV performance and analyse the processing scheduling produced by this algorithm and contrast it to PGS.

11.4.1 Tabu Search Specific Parameters

The tabu search algorithm (which shall be periodically be referred to as *TS*) by [Lamghari et al. 2012] has a number of fundamental parameters that the authors provide no guidance on how they are selected. Naturally, discerning the optimal parameters on often very large problems is not easy. Running thousands of simulations requires copious amounts of computing time and power.

However, we are of the belief that it would be beneficial, if not necessary, to look at these parameter choices to some degree to provide some guidance on differing problem types and sizes.

The key parameters in question are:

1. **Tabu Limits Range**, θ: a random number in a predetermined range is chosen when designating a shift as tabu.
2. **Number of Shifts**, $shiftsNum$: the number of potential shifts generated per iteration of the tabu search.
3. **Maximum non-improving solutions**, $nitermax$: the number of shifts allowed before exiting the current iteration of the tabu search and employing a diversification strategy.
4. **Diversifications**, $divs$: The number of iterations of the tabu search, determined by the number of diversifications employed - the original stopping criterion for tabu search.

Each of these parameters were tested with values determined as a function of the number of blocks and / or the number of time periods. For example:

$$nitermax = kn, \quad k = 0.1, 0.2, .., 1 \tag{11.3}$$

Table 11.8 details a generalisation of the parameters specific to the tabu search algorithm:

TABLE 11.8: Important Parameters for TS Algorithm

Parameter	Generalised Formula	*KD Value*
$\theta \in [\theta_{min}, \theta_{max}]$	$[\lfloor \frac{n}{4} \rfloor ; \lfloor \frac{3n}{4} \rfloor]$	$[3,538; 10,614]$
c^o	Inspection	5
c^m	Inspection	8
$nitermax$	n	$14,153$
$shiftsNum$	$shiftsNum > n$	$shiftsNum >> 14,153$
$divs$	$5n$	$70,765$

The *generalised formula* column denotes the formulae one should consider using when determining the respective value for the parameters. The KD Value column then indicates the exact parameter value that was used in the computations, based on statistics from the KD mining instance.

Some consideration needs to be made in determining the tabu limits. Tabu limits that are too small will lessen the exploration capabilities that tabu search possesses whilst tabu limits that are too large hamper traversals of the solution space too much, since too many shifts will be forbidden.

The number of non-successive iterations ($nitermax$) cannot be too small because the local search will "give-up" too quickly and hastily move to diversification, effectively jumping to another neighbourhood of the solution space and leaving the old neighbourhood relatively unexplored. If $nitermax$ is too large then one runs the risk of languishing too long in a local optima or a futile neighbourhood.

It was discovered that the feasible possible shifts, $shiftsNum$, is not a vital parameter when it comes to solution quality. However, one cannot simply assume generating a significantly small amount of shifts will lead to decent solution quality. Furthermore, we

make use of the vectorisation (see Definition 11.1) benefits of the *MATLAB* numerical computing environment. MATLAB is optimised to handle operations pertaining to vectors and matrices. We therefore utilise these data structures, as opposed to *object orientated* programming (see Definition 11.2) models, to increase the efficiency of our code. These principles were extensively employed within our coding frameworks. Operations like generating possible shifts were greatly improved using this approach. Therefore, one will notice from Table 11.8 that a much greater number of shifts can be utilised than 14,153 due to strong coding practices.

A prime example of the benefit for good coding practice that we made use of can be seen in the pronounced differences in computation efficiency between function calls on discovering feasible shifts in the TS algorithm. Figure 11.8 reveals this below.

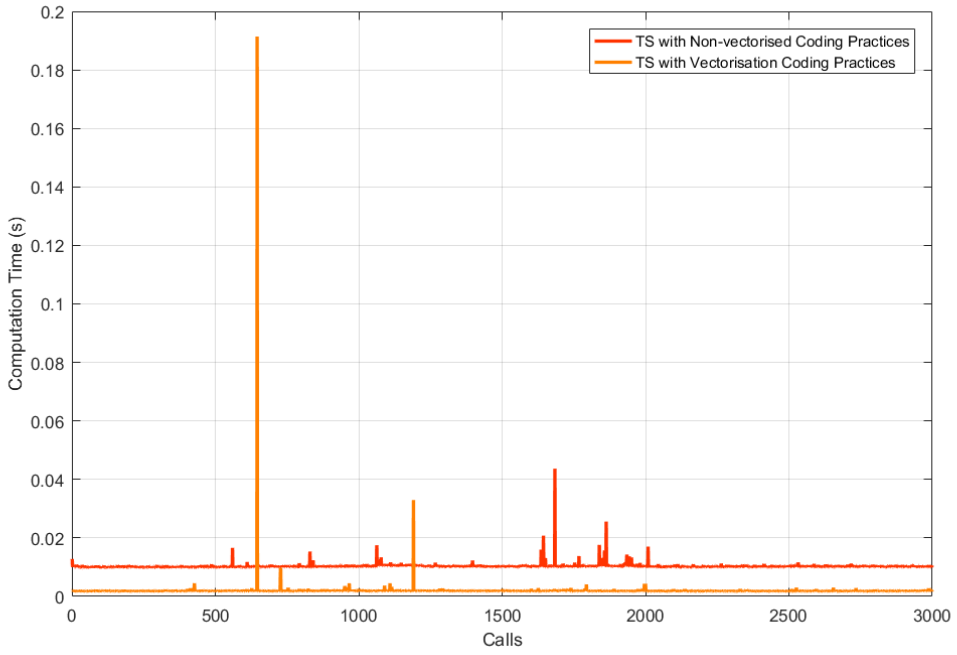

FIGURE 11.8: Feasible Shift Computation Times

Over 3000 calls the average computation time for our vectorised code was 0.002 seconds versus an average of 0.01 seconds on the non-vectorised code. Therefore, the good coding practices we employed for this component of the code decreased computation time by a factor of 5 on average.

Definition 11.1 (Vectorisation)**.**

Some CPUs have an instruction which allows them to apply simultaneous computations on several slices of data. Vectorisation is the operation that processes several pieces of data in parallel as opposed to looping through every element. Good coding practices in MATLAB allow for efficient computations through vectorisation.

Definition 11.2 (Object Oriented Programming (OOP))**.**

OOP is a programming methodology based on data structures been modelled as objects. These objects are categorised into classes and come with functioning and operations that one would relate to these object instances.

11.4.2 NPV Analysis of Tabu Search

The Tabu Search heuristic generally peforms quite well on the mining instance problems seen in [Minelib].

Figure 11.9 below displays the NPV performance of Tabu Search on the *Newman* mining instance, as a proof of concept with the generalised Tabu Search formulae. Given the structure of the orebody, it is not surprising a lot of value is realised at the start of the mining venture, an important consideration for attracting mining investors as the mine quickly becomes profitable. Furthermore, the production schedule did not need to make use of more than 3 time periods.

Using *IBM ILOG CPLEX Optimization Studio* within the MATLAB programming environment it required 5.35 hours of computing time to find the optimal solution of $\$2.4177 \times 10^7$. Tabu Search was run for 5 hours and achieved this optimal solution in 5 hours, see Table 11.9 below:

TABLE 11.9: Tabu Search Performance vs Optimal Solution using CPLEX vs [Muñoz 2012]

Algorithm / Optimisation Software	Monetary Value ($)	Execution Time (hours)
Vanilla Tabu Search	2.4177×10^7	5
IBM CPLEX	2.4177×10^7	5.35
[Muñoz 2012]	2.348×10^7	N/A

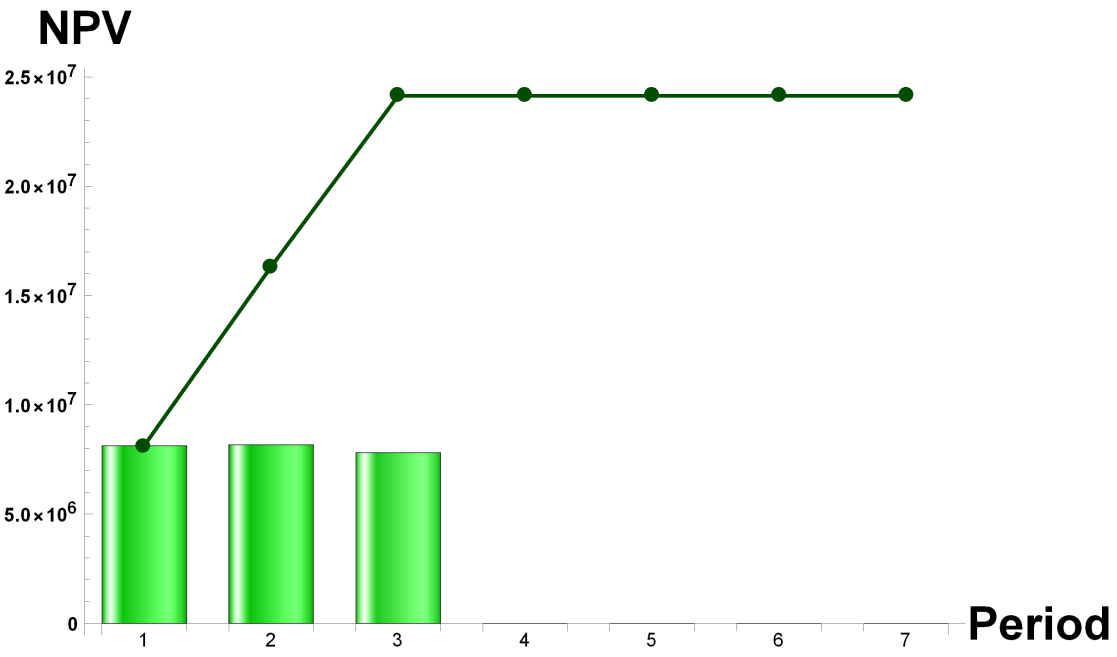

FIGURE 11.9: NPV Performance of Tabu Search on Newman Mining Example

Of course, the results are on an appreciably smaller example than the *KD* copper deposit. It should be noted that these NPV values are higher than that in [Muñoz 2012]. This proof of concept demonstrated the prowess of the algorithm with the added benefit of vectorisation and efficient parameter selection.

The *KD* problem instance poses further challenges than the *Newman* model above. Not only are we exposed to a more granular block model, with over 14,000 blocks, and 12 time periods, but also more precedences than in the proof of concept model.

Unlike the *Newman* model, the *KD* model is too large to be solved optimally by the IBM CPLEX Software. The best solutions to date can be found in [Minelib] and were achieved by the methods found in [Muñoz 2012], with a linear programming relaxation

solution of $\$409,498,555$ and a solution of $\$396,858,193$, corresponding to a LP gap of 3.1%.

The Vanilla Tabu Search, with its optimised code shift generation and vectorisation in MATLAB, acquires a solution of $\$395,849,841$ over a 24 hour computation time. The optimality gap for this is 3.33% and is slightly worse than the solution found in [Muñoz 2012].

Referring to Figure 11.10 below, we see a steady progression of the best solution found. However, there are lengthy patches of little improvement followed by steep increases.

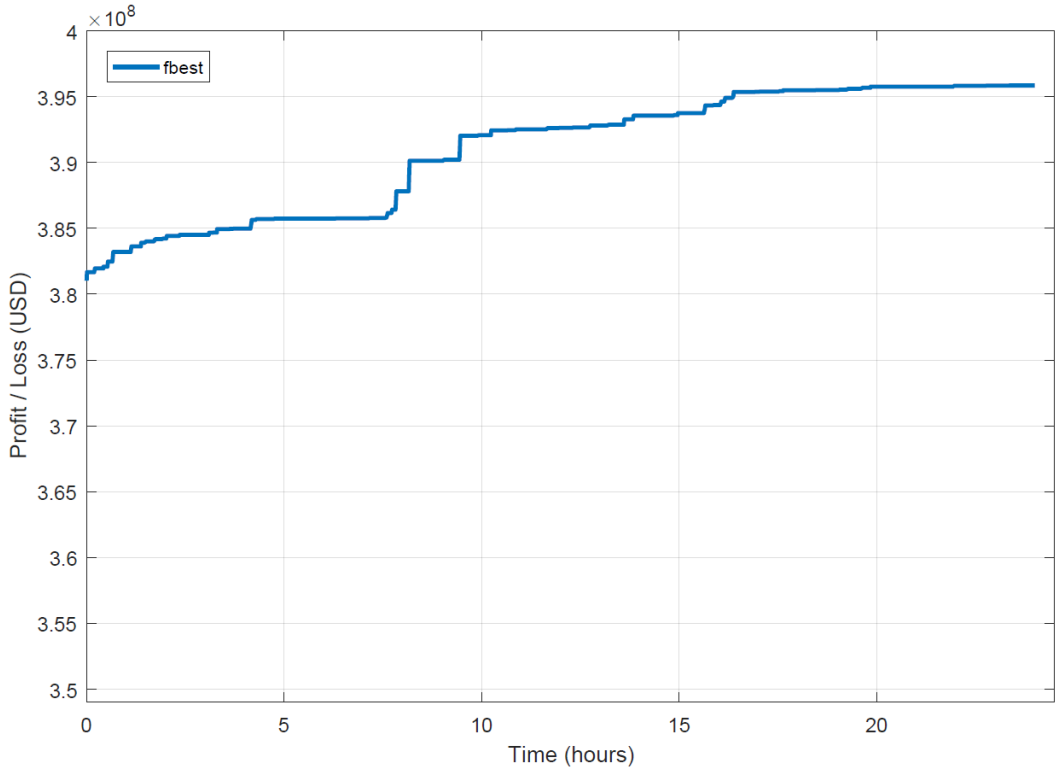

FIGURE 11.10: Progession of best found solution using TS

We note that there is strong performance from the onset of the algorithm. This suggests that the initial solution is far from a local optimum. There is also significant improvement before the 10-hour mark. Progress from here onwards struggles to make any inroads with only modest improvements.

One might question how exploratory is the tabu search with its memory structures and diversifications. Indeed, solving the OPMPSP requires an extensive traversal of the solution space, given that it is such a complex dimensioned problem. Figure 11.11 displays the increasing best objective function values (fbest) but also the many production schedules that were traversed in an effort to improve upon the solution. Notice that the

algorithm even went so far as to experience a 35 million USD drawdown just post the 5^{th} hour of computation time in an effort to effectively explore paths to better solutions.

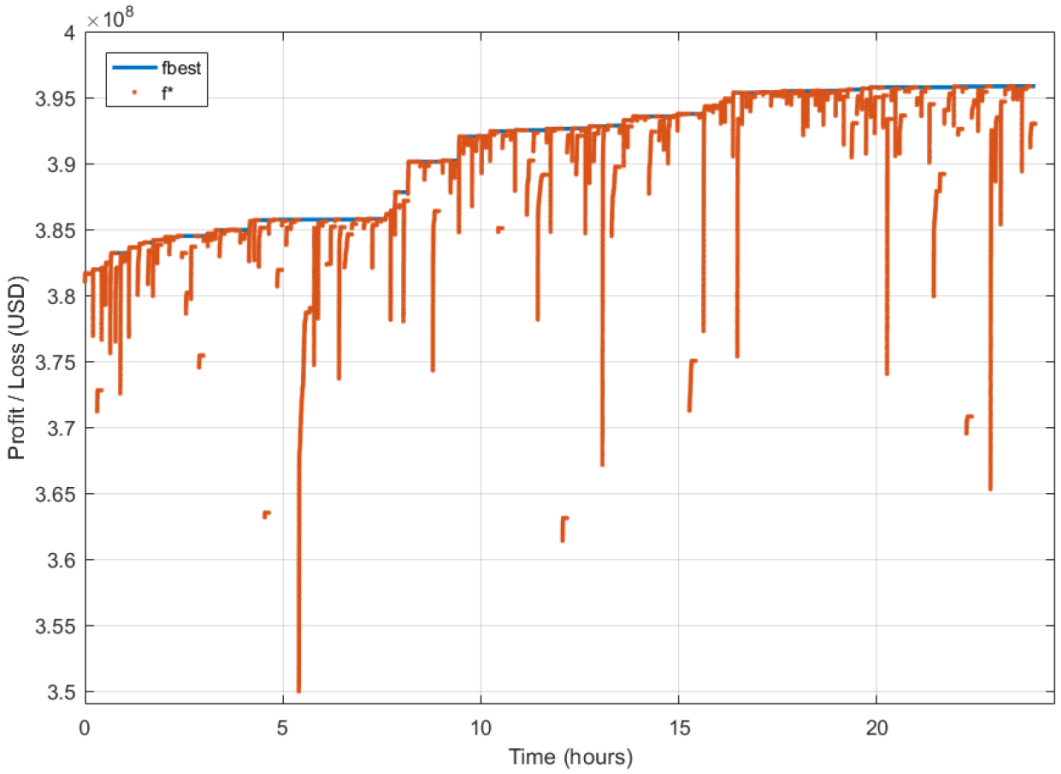

FIGURE 11.11: Vanilla TS exploration for solution improvements

Referring to Table 11.10 below we see the consistent outperformance of TS over PGS. On most occasions TS generates more profits than PGS per period, particularly evident in the early periods. This suggests a schedule that acquires more rich ore blocks earlier before the diminishing returns of discounting come to bear.

Figure 11.12 shows the NPV profile of the TS algorithm graphically. We see that the algorithm produces a production schedule that attempts to realise as much NPV as possible in early periods, fading slightly as we progress to later periods.

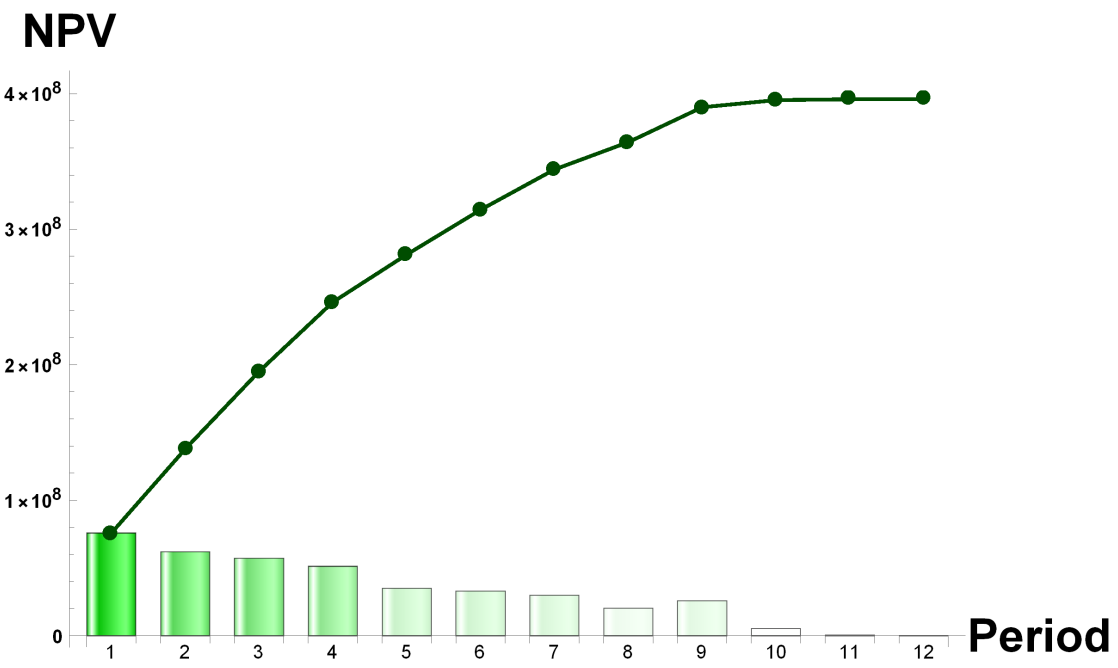

FIGURE 11.12: NPV performance of TS on KD Model

Below in Figure 11.13 we see the cumulative NPV performance of TS vs PGS. The slowing improvement of NPV is prevalent in both mining schedules however, we see the glaring improvement that TS brings.

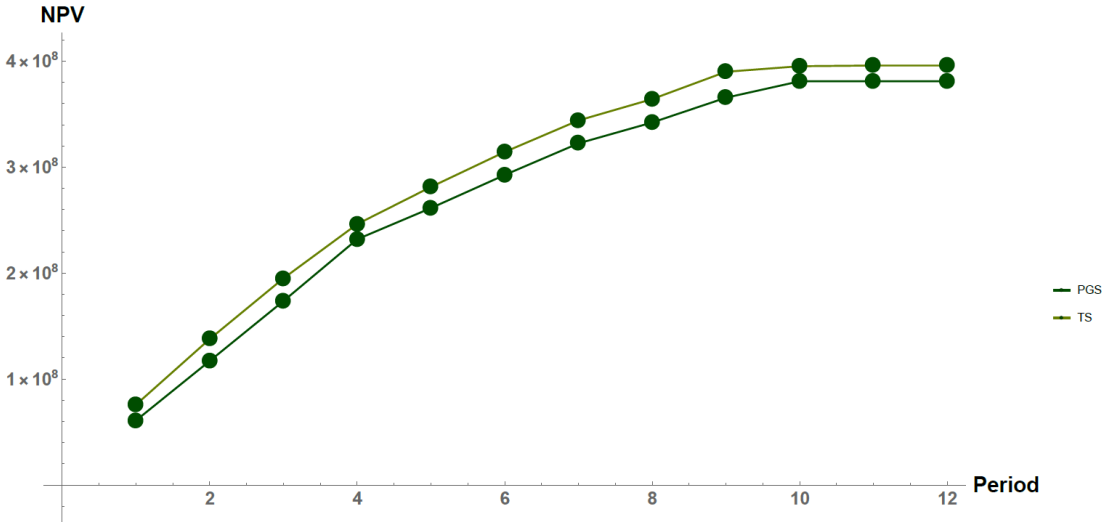

FIGURE 11.13: NPV performance of TS vs PGS

11.4.3 Processing Production Analysis

The NPV performance of Tabu Search is superior to that of PGS. Further investigation of the processing statistics of the schedule produced by TS reveals further information

TABLE 11.10: NPV Statistics for PGS and TS

Time Period	PGS NPV/Period	PGS Cumulative NPV	TS NPV/Period	TS Cumulative NPV
1	6.0860×10^7	6.0860×10^7	7.5761×10^7	7.5761×10^7
2	5.6189×10^7	1.1705×10^8	6.1994×10^7	1.3775×10^8
3	5.6570×10^7	1.7362×10^8	5.7275×10^7	1.9503×10^8
4	5.8296×10^7	2.3191×10^8	5.1196×10^7	2.4623×10^8
5	2.9449×10^7	2.6136×10^8	3.5073×10^7	2.8130×10^8
6	3.1238×10^7	2.9260×10^8	3.3021×10^7	3.1432×10^8
7	2.9822×10^7	3.2242×10^8	2.9741×10^7	3.4406×10^8
8	1.9709×10^7	3.4213×10^8	2.0161×10^7	3.6422×10^8
9	2.3390×10^7	3.6552×10^8	2.5764×10^7	3.8999×10^8
10	1.5506×10^7	3.8103×10^8	5.2731×10^6	3.9526×10^8
11	0	3.8103×10^8	6.3865×10^5	3.9590×10^8
12	0	3.8103×10^8	0	3.9590×10^8

and confirms our suspicions about the reasons of the improved performance. That is, the mining of rich ore blocks in earlier periods. The numbers in Table 11.11 expose this advantage of TS. Notice that TS processing mass is greater than that of PGS right up until the 10^{th} period. More tonnage of ore blocks in the TS schedule are being sent to the processing mill on a relative basis when compared to the schedule produced by PGS.

Figure 11.14 divulges the processing activity of the production schedule produced by the TS algorithm. Recall that the maximum that can be processed in a given period is $10,000,000$ tons of ore. The early periods of the TS production schedule effectively utilise the full amount possible per period. Notice also that processing takes place in all periods apart from the last.

We see from Figure 11.15 that it is very difficult to discern the difference between the processing activity between the 2 algorithm production schedules, although evident in Table 11.11. This suggests that a portion of the NPV improvement also originates from

the more intelligent mining of waste blocks, not only to secure rich ore blocks earlier, but also as late as possible, when viable.

TABLE 11.11: Processing Statistics for PGS and TS

Time Period	PGS Processing/Period	PGS Cumulative Processing	TS Processing/Period	TS Cumulative Processing
1	9.9947×10^6	9.9947×10^6	1.0000×10^7	1.0000×10^7
2	9.9944×10^6	1.9989×10^7	1.0000×10^7	2.0000×10^7
3	9.9995×10^6	2.9989×10^7	9.9930×10^6	3.0000×10^7
4	9.9862×10^6	3.9975×10^7	9.9986×10^6	4.0000×10^7
5	9.9936×10^6	4.9968×10^7	9.9923×10^6	4.9993×10^7
6	9.9881×10^6	5.9957×10^7	1.0000×10^7	5.9994×10^7
7	9.9859×10^6	6.9942×10^7	9.9980×10^6	6.9992×10^7
8	9.9845×10^6	7.9927×10^7	9.9864×10^6	7.9978×10^7
9	9.9839×10^6	8.9911×10^7	9.9853×10^6	8.9964×10^7
10	6.2666×10^6	9.6177×10^7	5.3074×10^6	9.5271×10^7
11	0	9.6177×10^7	9.0648×10^5	9.6177×10^7
12	0	9.6177×10^7	$0.0000E + 00$	9.6177×10^7

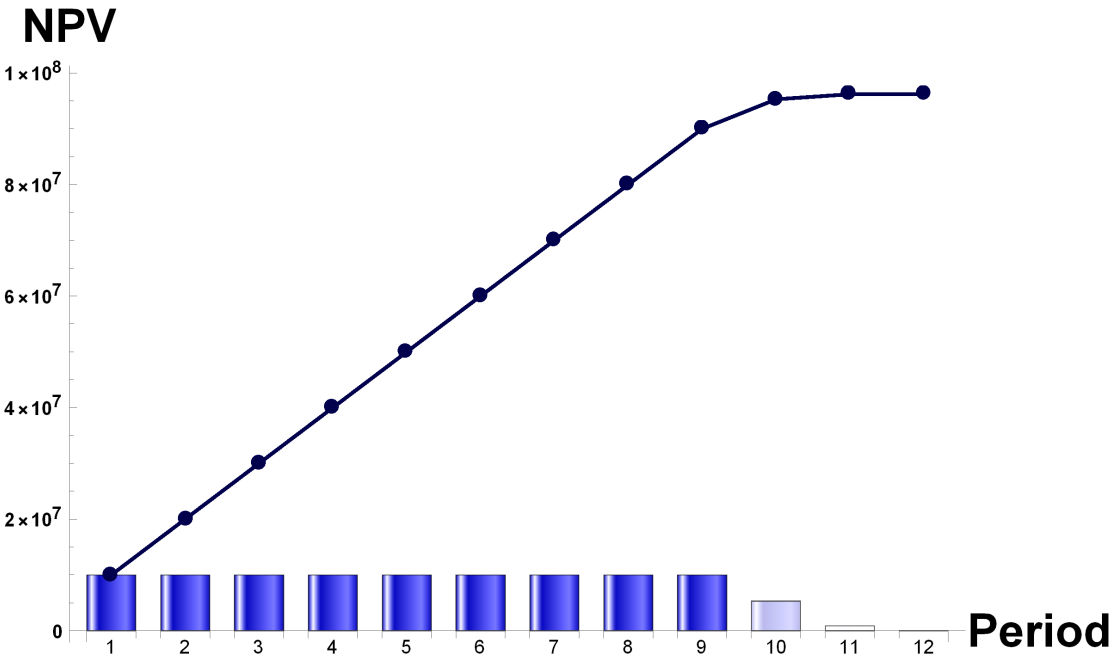

FIGURE 11.14: Processing activity of TS on KD Model

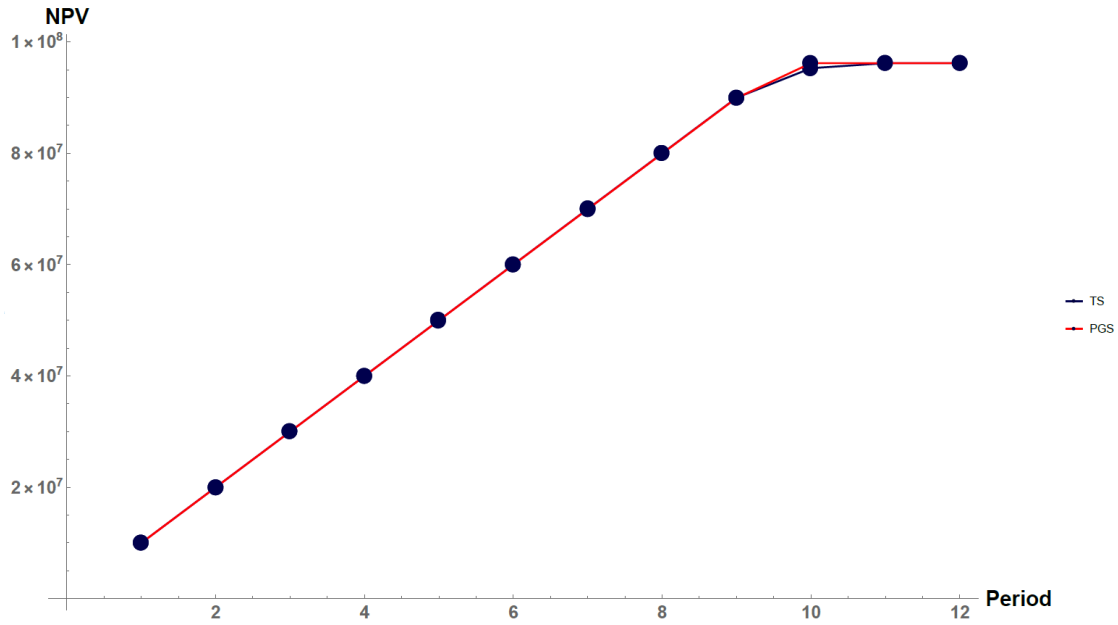

FIGURE 11.15: Processing activity of TS vs PGS

11.4.4 Conclusion

Tabu Search shows some promise in generating a strong production schedule. It makes a notable improvement on the initial solution discovered by PGS. However, it comes short of bettering the solution found by [Muñoz 2012]. The progression of TS also appears to stagnate rapidly during the latter stages of the exploration of the solution space,

as exposed in Figure 11.10. A potential cause for this is that the algorithm operates one shift at a time. In a problem instance with a plethora of blocks and precedence relationships, this single shift movement each computation step is sometimes not sufficient. The objective is to escape local optima as quickly as possible and thus single shifts may be too sluggish.

11.5 Vanilla-SA

This section will detail the algorithmic performances of Vanilla-SA(which will periodically be referred to as *SA*), outlined in Chapter 10. Vanilla-SA, much like Tabu Search, will also utilise the solution produced by PGS.

Concepts of the simulated annealing algorithm, such as temperature, and how it behaves in unison with other concepts, will also be displayed.

It should be mentioned that the Vanilla-SA approach, as it is given in this dissertation, achieves a higher NPV than the best solution found for this problem instance.

11.5.1 Simulated Annealing Specific Parameters

As with TS, SA has a collection of parameters specific to its functioning, which are vital to its effective performance. For a revision of these parameters and their interplay within the SA algorithm the reader can refer to 10.1.4.

The crucial parameters in question are:

1. **Length of Markov Chain**, L: the length of the Markov chain loop component of the simulated annealing algorithm.
2. **Temperature**, $Temp$: the "temperature" component of the simulated annealing algorithm that determines the likelihood of acceptance or rejection inherent in the Boltzmann condition.
3. **Heating coefficient**, α: the coefficient that determines how much the temperature is heating upon a solution's rejection.
4. **Cooling coefficient**, β: the coefficient that determines how much the temperature is cooled upon a solution's acceptance.

The length of the Markov Chain does not hold significant relevance in our implementation of the SA algorithm since we run the algorithm over a predetermined time frame,

specifically 24 hours. Where it does have a marked purpose is when testing for convergence of the algorithm. Nonetheless, we recommend the generalised value seen in Table 11.12 to allow for effective exploration within the algorithm before checking for convergence.

The next three parameters, namely the initial temperature, the heating and cooling coefficients, are imperative to the effective functioning of SA. The initial temperature forms a pivotal component in the determination of the Boltzmann condition, see Equation (10.3). If the initial temperature is too large the algorithm will be too lenient on its acceptance criteria and one will sacrifice too much searching for a means to escape local optima. If the initial temperature is too low the acceptance criteria is too strict and we run the risk of not exploring the solution space enough. A reliable solution to these issues is to set the initial temperature value close to the initial NPV value less the next NPV value. In the case of the KD mine instance, with our PGS solution of 3.82×10^8 the next step is roughly 1×10^5 USD from the initial solution.

α and β are equally important to the exploration component of the SA algorithm. Ideally, to preserve the principles of the simulated annealing metaheuristic, the cooling rate should be greater than the heating rate. The degree and value of these coefficients need to be tested over several iterations of the algorithm. One needs to identify progression in terms of solution quality but also a varying temperature to display the attempts to explore other parts of the solution space. During our testing we found that the cooling and heating numbers should be appreciably small and the cooling rate should be roughly 40 times more than the heating rate.

TABLE 11.12: Imperative Parameters for the SA Algorithm

Parameter	Generalised Formula	*KD Value*
L	$\approx \frac{1}{3}n$	$5,000$
$Temp$	$\approx \lvert f(x_0) - f(x_1) \rvert$	1×10^5
α	Empirically tested	0.0005
β	Empirically tested	0.02
$Trans$	$[\approx 0.001n; \approx 0.003n]$	$[15; 50]$

11.5.2 NPV Analysis

The simulated annealing algorithm is a strong performer when solving the KD instance mining problem.

Figure 11.16 below reveals progression of the best solutions found by SA over a 24 hour period. We notice the algorithm makes good progress in the early stages, only to level out from about the 5 hour mark until the end of the 24 hour computation period we impose.

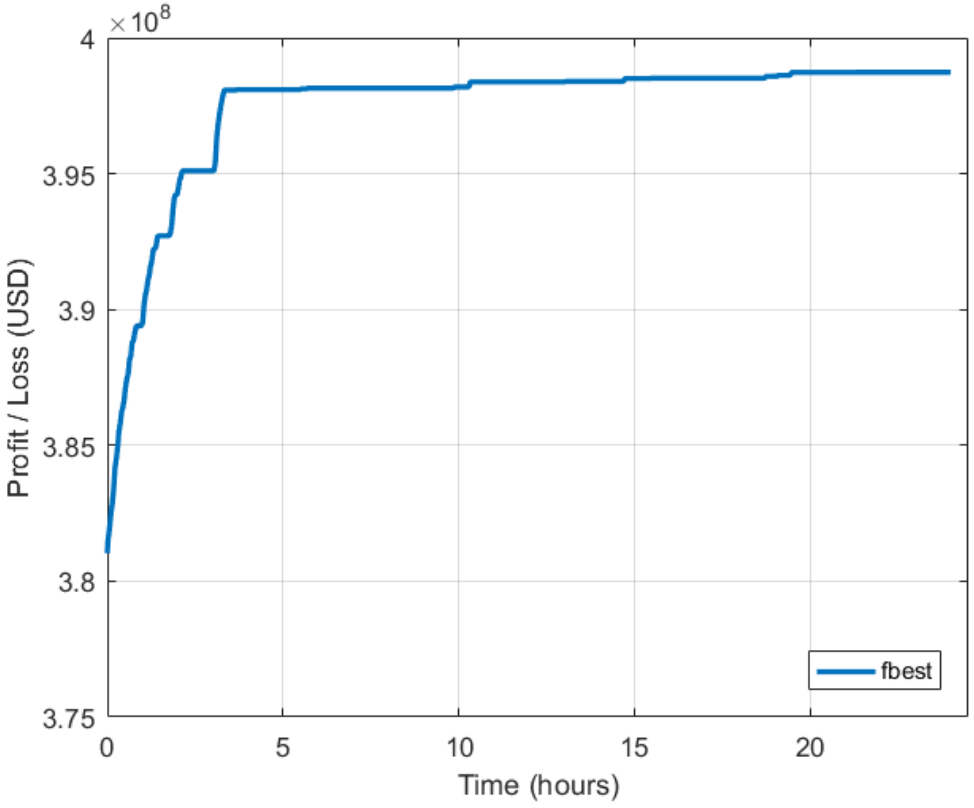

FIGURE 11.16: Best solutions progression of SA algorithm

Figure 11.17 portrays the fluctuation of the temperature through the SA algorithm's progression. The temperature values appear to be largely random, with no noticeable pattern emerging. By pure observation, one might conclude that the initial value of $10,000$ acts as long term mean that the temperature reverts to once it moves too far away from it. We see this in the period between 5 - 10 hours as well as the 10 - 15 hours. The 24 hour mark may have been in the midst of a period where the temperature had drifted away from this mean and was in the process of making its way back closer to this $10,000$ level.

There does not appear to be any discernible relationship between the objective function value movements and the temperature fluctuations.

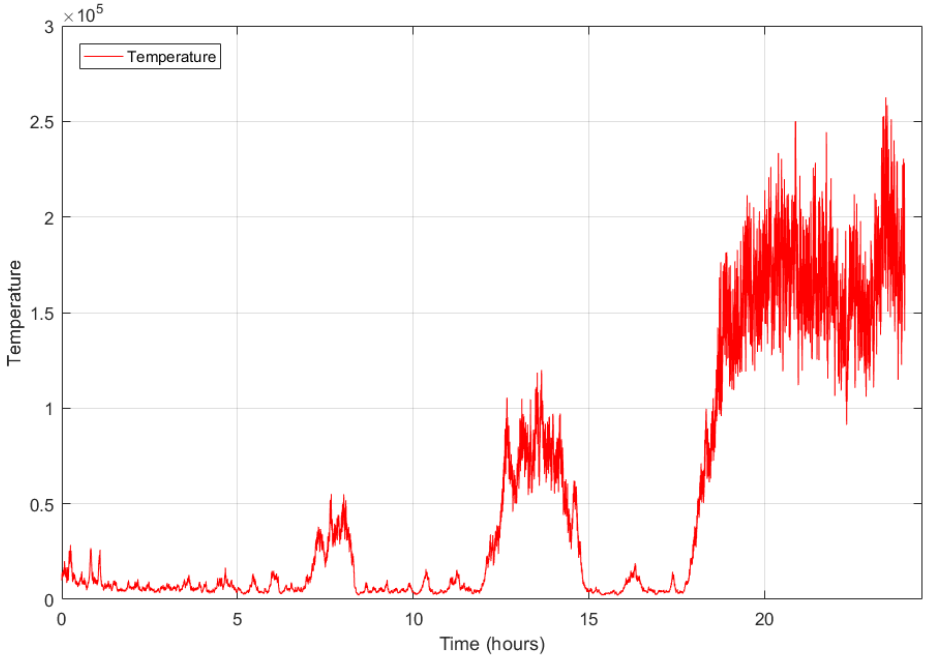

FIGURE 11.17: Temperature Fluctuations through SA Algorithm

Figure 11.18 displays the NPV profile of the SA algorithm. We identify that the algorithm produces a production schedule that attempts to realise as much NPV as possible in early periods, fading slightly as we progress to later periods.

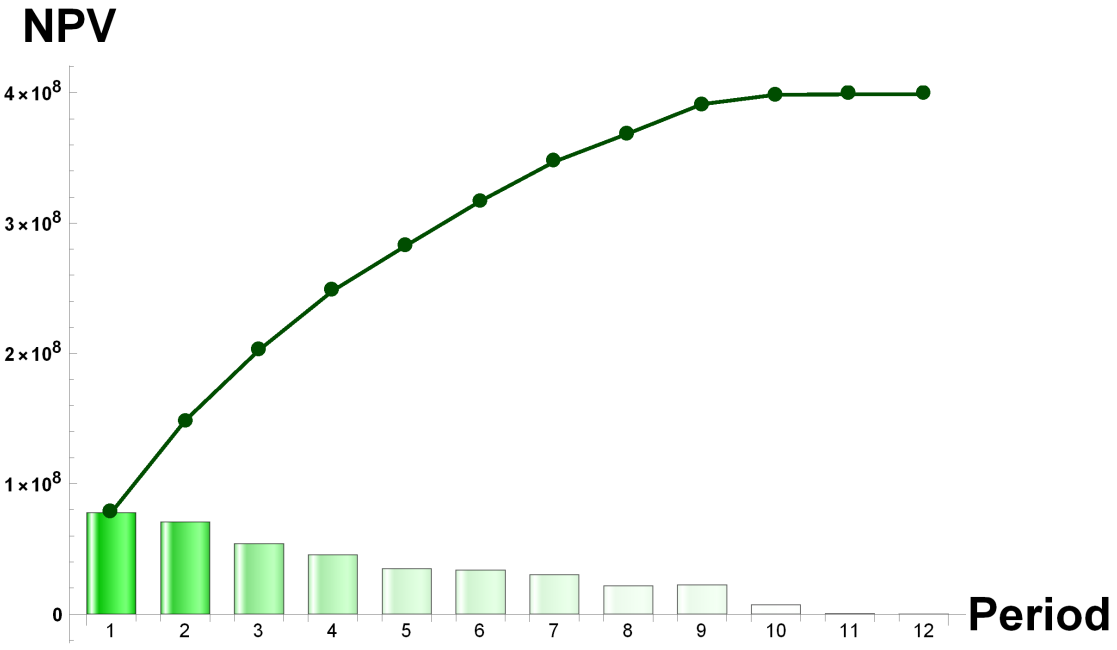

FIGURE 11.18: NPV profile of SA Algorithm

The SA Algorithm performs better than [Muñoz 2012] and the TS algorithm, establishing a new best found objective function value of 398, 731, 386 USD. Table 11.13 below provides a review of the objective function values and their associated LP gaps.

TABLE 11.13: Recap of Algorithm Objective Function Value Performances on KD Model

Algorithm	Objective Function ($)	LP Gap
SA	398, 731, 386	2.6%
TS	395, 849, 841	3.3%
[Muñoz 2012]	396, 858, 193	3.1%

A comparative view of NPV with TS and SA can be seen in Figure 11.19 below. Due to the margins of improvement being relatively small it is not immediately clear but upon closer inspection we do see SA performing better than TS in almost all the extraction periods.

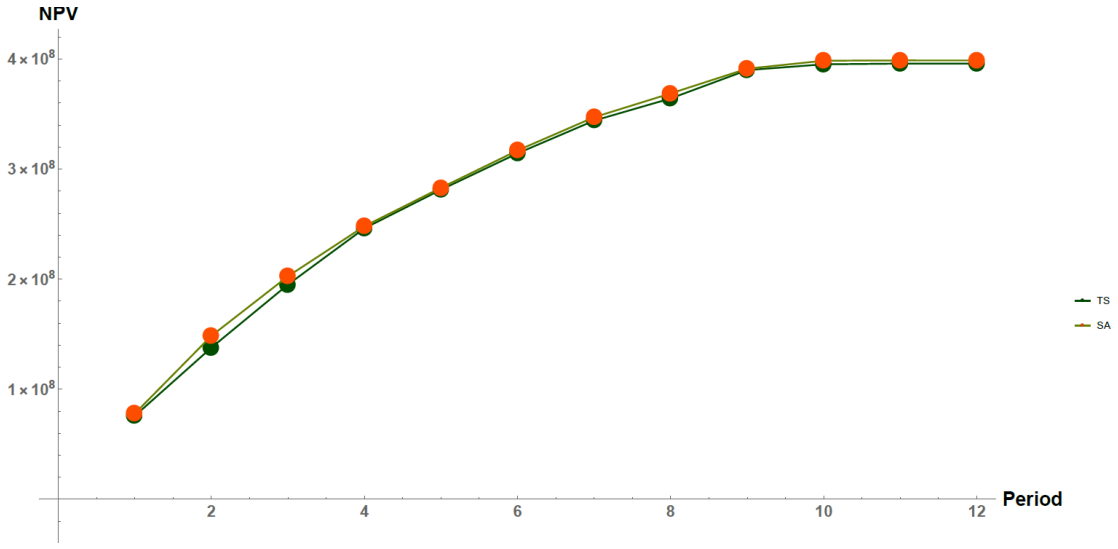

FIGURE 11.19: NPV Comparison of TS vs SA

Table 11.14 reveals the disparity in NPV performance between the 2 algorithms. The numbers reveal that SA's strength lies mainly in the later periods where TS suffers. The 10^{th} period is a prime example of this were SA achieves an NPV of 7.241×10^6 whereas TS can only muster 5.273×10^6.

Another point of interest in the SA algorithm is that we see negative NPV in period 12. This means that we could potentially have an even stronger performing production schedule had no mining occurred in the last period.

TABLE 11.14: NPV Statistics of SA and TS

Time Period	TS NPV/Period	TS Cumulative NPV	SA NPV/Period	SA Cumulative NPV
1	7.5761×10^7	7.5761×10^7	7.7903×10^7	7.7903×10^7
2	6.1994×10^7	1.3775×10^8	7.0565×10^7	1.4847×10^8
3	5.7275×10^7	1.9503×10^8	5.4128×10^7	2.0260×10^8
4	5.1196×10^7	2.4623×10^8	4.5557×10^7	2.4815×10^8
5	3.5073×10^7	2.8130×10^8	3.4895×10^7	2.8305×10^8
6	3.3021×10^7	3.1432×10^8	3.3907×10^7	3.1695×10^8
7	2.9741×10^7	3.4406×10^8	3.0192×10^7	3.4715×10^8
8	2.0161×10^7	3.6422×10^8	2.1627×10^7	3.6877×10^8
9	2.5764×10^7	3.8999×10^8	2.2506×10^7	3.9128×10^8
10	5.2731×10^6	3.9526×10^8	7.2410×10^6	3.9852×10^8
11	6.3865×10^5	3.9590×10^8	2.7425×10^5	3.9879×10^8
12	0	3.9590×10^8	-3.1457×10^4	3.9876×10^8

11.5.3 Processing Production Analysis

We now look at the processing productivity of the SA algorithm, which may provide some insights into its impressive performance.

In the below Figure 11.20 we see the cumulative tonnage, as well as the per period, processed in the production schedule of SA. Table 11.15 highlights how the production schedule uses its full allocation of processing capacity for every period until period 10.

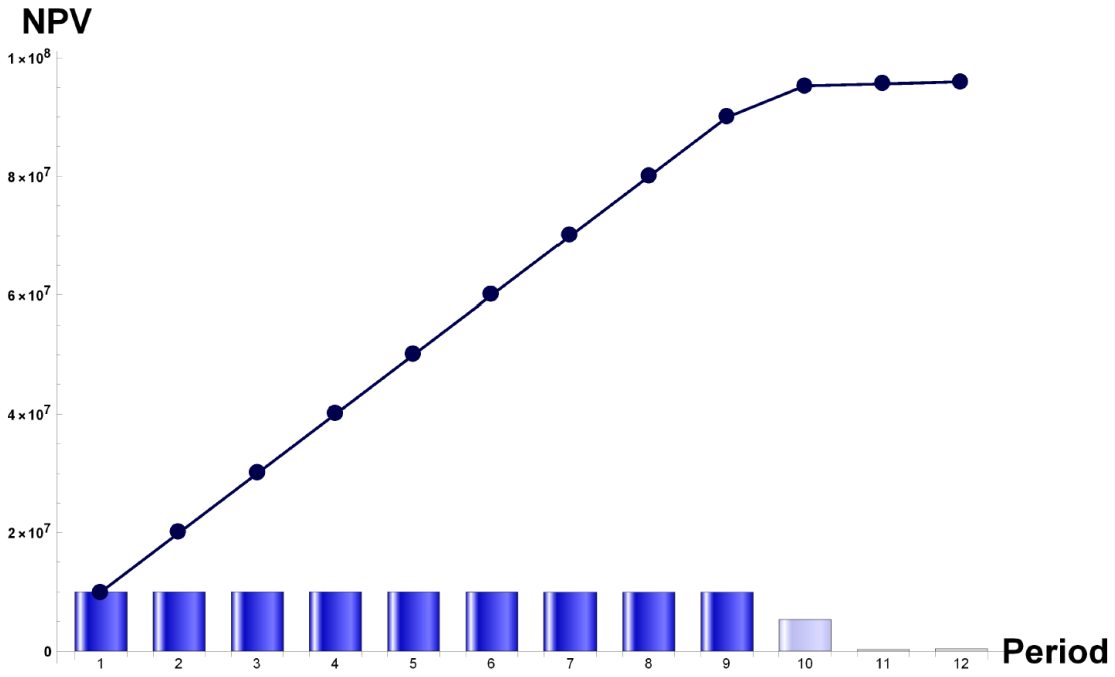

FIGURE 11.20: Processing Profile of the SA Algorithm

11.5.4 Conclusion

The SA algorithm performs rather well on the KD mining model instance. It surpasses the NPV values found by both [Muñoz 2012] as well as the TS algorithm, see Table 11.13. There are a number of explanations for this.

Firstly, given the nature and the size of the mining model instance, TS appears quite sluggish in its operations of making one shift at each iteration. In contrast, SA makes several shifts during each iteration. Furthermore, this is a parameter that can be altered with the $Trans$ variable.

Secondly, the long-term memory diversification strategy of TS, which forms the fundamental exploration component of the solution space, is perhaps too drastic in its shift to a new neighbourhood when comparing it to the Boltzmann condition in SA. The new resultant solution is usually in a new neighbourhood that holds little prospect of finding even a good solution as initially it often violates many resources constraints from the start. Echoing the issue stated in the first point, TS must then slowly iteratively improve the production schedules from a poor starting point. SA on the other hand, with its Boltzmann condition, heating and cooling procedures, effectively explores other neighbourhoods without meandering too far into highly infeasible production schedules.

Table 11.15: Processing Statistics of SA and TS

Time Period	TS Processing/Period	TS Cumulative Processing	SA Processing/Period	SA Cumulative Processing
1	1.0000×10^7	1.0000×10^7	1.0000×10^7	1.0000×10^7
2	1.0000×10^7	2.0000×10^7	1.0000×10^7	2.0000×10^7
3	9.9930×10^6	3.0000×10^7	1.0000×10^7	3.0000×10^7
4	9.9986×10^6	4.0000×10^7	9.9989×10^6	4.0000×10^7
5	9.9923×10^6	4.9993×10^7	9.9961×10^6	5.0000×10^7
6	1.0000×10^7	5.9994×10^7	9.9890×10^6	5.9990×10^7
7	9.9980×10^6	6.9992×10^7	9.9872×10^6	6.9978×10^7
8	9.9864×10^6	7.9978×10^7	9.9859×10^6	7.9964×10^7
9	9.9853×10^6	8.9964×10^7	9.9857×10^6	8.9949×10^7
10	5.3074×10^6	9.5271×10^7	5.3390×10^6	9.5288×10^7
11	9.0648×10^5	9.6177×10^7	2.9136×10^5	9.5580×10^7
12	0	9.6177×10^7	4.0398×10^5	9.5984×10^7

11.6 A Dual Interchange Algorithm (DIA)

We propose a probabilistic selection mechanism for finding a good production schedule that ensures the presence of the features of both the TS and SA algorithms. This is another contribution of this research towards the OPMPSP but can find application in other optimisation problems.

11.6.1 Motivation

Through our analysis emerged a clear superior algorithm for the KD mining instance. SA made clearer strides to producing a superior production schedule, whilst consuming less computation time.

However, each algorithm does exhibit certain potential strength characteristics that are not immediately noticeable. Visually, by inspection of Figures 11.10 and 11.16, we might propose that although TS makes slow progress in production schedule improvement, there is improvement nonetheless. Whereas, SA makes a clear-cut improvement right from the beginning but struggles to make sustained progress afterwards.

One can also look to the strong performance of TS in its solution for the smaller Marvin mining instance, achieving a near optimal solution in less computation time than an Optimisation Studio Software (IBM ILOG CPLEX). Conversely, SA's performance on the Marvin instance was not as impressive. This implies that TS performs better on smaller problems than SA, and in the scope of larger instances, may perform better within subsections of the solution space.

11.6.2 Probabilistic Selection Mechanism

We propose a dual utilisation of both the TS and SA algorithms through probabilistic selection dynamically determined by perceived performance. To expand on this, if one algorithm is performing better than the other it increases the chances of it getting randomly selected for the next iteration, or selection phase, in the hope that improvement will continue within the current stage of the exploration. If it should not perform well the parent algorithm will diminish its chances of selection for the next phase. Through this it may emerge that one algorithm holds an advantage over the other within certain situations or stages during the solution space traversal. For example, there may be periods where ever laborious progression may be preferred over lengthy periods of no improvement, followed by drastic improvement.

We envision an algorithm that selects either the SA or TS algorithms during each time period or iteration. We designate this to be the *parent algorithm*.

Let α denote the probability of selecting the SA algorithm for the current iteration, or time period, for the parent algorithm. Similarly, let β denote the probability of selecting the TS algorithm. The static value at any point in time for these probability variables are represented in Equations (11.4) and (11.5) below.

$$\alpha = \begin{cases} 0.9 & \text{if } \alpha \geq 0.9, \\ 0.1 & \text{if } \alpha \leq 0.1, \\ 1 - \beta & \text{otherwise.} \end{cases} \tag{11.4}$$

$$\beta = \begin{cases} 0.9 & \text{if } \beta \geq 0.9, \\ 0.1 & \text{if } \beta \leq 0.1, \\ 1 - \alpha & \text{otherwise.} \end{cases} \tag{11.5}$$

Where α, $\beta \in [0, 1]$.

We run the DIA algorithm in 2 settings. Both settings run for a total time of 48 hours. The first setting will make an inquiry into the selected algorithms performance every 5 minutes, whereas the second setting will only query every 30 minutes. In essence, the first setting is more greedy and the second setting affords the selected algorithm more time to show improvement.

The extended computation time, from 24 hours to 48 hours, will provide for clear insight into any perceived benefit from either algorithm selection.

The variables of α and β need to be dynamically altered according to the recognised performance of either SA or TS respectively. For this purpose, we introduce the probability altering variable, γ.

γ should dynamically reward an algorithm for adding value to developing an efficient production schedule, or "punish" an algorithm for not making progress. Equation (11.6) below details how γ is determined at each time interlude.

$$\gamma = \begin{cases} 0.2 & \text{if } \frac{f(x_{best}^{t+1})}{f(x_{best}^{t})} > 1, \\ 0.05 & \text{if } \frac{f(x^{t+1})}{f(x^{t})} > 1, \\ -0.05 & \text{otherwise.} \end{cases} \tag{11.6}$$

Each time period interlude then sees either α or β updated according to Equations (11.7) or (11.8) respectively:

$$\alpha = \alpha + \gamma \qquad \text{if SA was selected,} \tag{11.7}$$

$$\beta = \beta + \gamma \qquad \text{if TS was selected.} \tag{11.8}$$

The full pseudo code for the DAI algorithm can be seen below in Algorithm 16.

11.6.3 DIA Behaviour

Upon investigation of Figure 11.21, we see that the progression of the best solution found is not dissimilar from that of Vanilla-TS or Vanilla-SA. The line plot indicates that the best solution found makes strong progress early on but experiences rather lethargic periods leading up to and after the 10^{th} hour. We also note that there is a clear developing bias for the TS algorithm, identified by looking at the bars. SA's probability of selection, α never reaches the heights of 50% during the entirety of the algorithm over 48 hours. Only coming close at 0.45 during the later stages of the runtime.

We do notice that α sees spikes during times of slow progression, centred around the 10 hour and 35 to 45 hour marks. However, the brevity of these "spikes" implies that SA does not make significant progress in its apportioned 5 minute window.

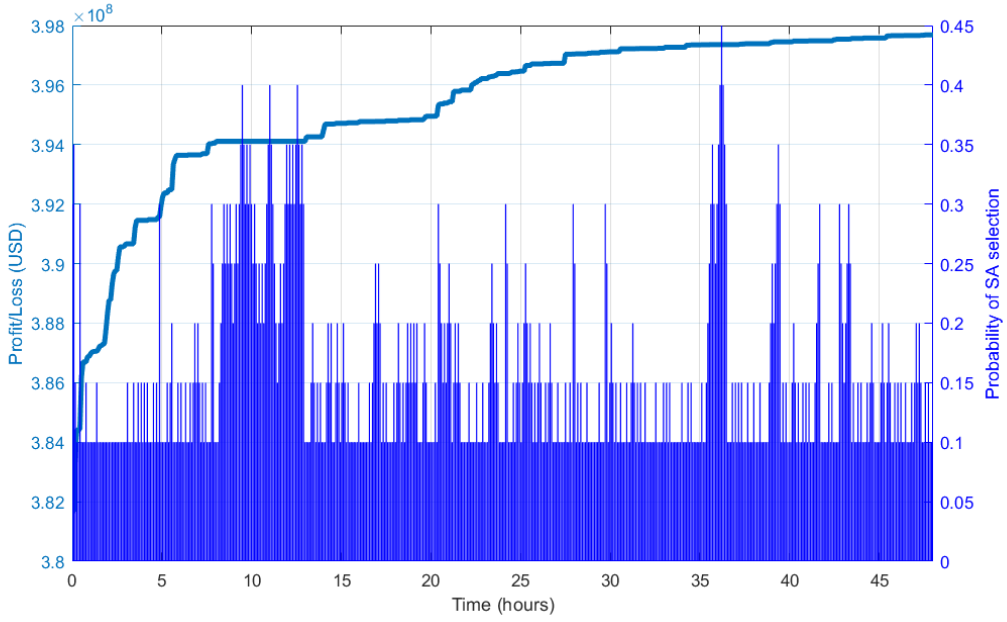

FIGURE 11.21: Best solutions progression of DIA algorithm with 5 minute interludes

Table 11.16 displays the 24 hour performances of the algorithms. Notice that after 24 hours, DIA with the 5 minute interludes, was unable to surpass the solutions found by [Muñoz 2012] and of course SA. Indeed, the DIA algorithm in its current form favours TS, which did reveal its weakness in sluggish and labourious improvement. Certainly, it does perform better with the addition of integration within DIA with SA, which was impressive in the 24 hour runtime.

We consider that 5 minute interludes may not be enough to allow SA to have a chance to flourish. We allowed the DIA algorithm to run with 30 minute interludes. The progression of the best solution found is presented in the below Figure 11.22.

TABLE 11.16: Recap of Algorithm Objective Function Value Performances on KD Model, including DIA

Algorithm	Objective Function 24hours ($)	LP Gap
SA	$398,731,386$	2.6%
TS	$395,849,841$	3.3%
$DIA(5min)$	$396,386,814$	3.2%
$DIA(30min)$	$396,400,732$	3.2%
[Muñoz 2012]	$396,858,193$	3.1%

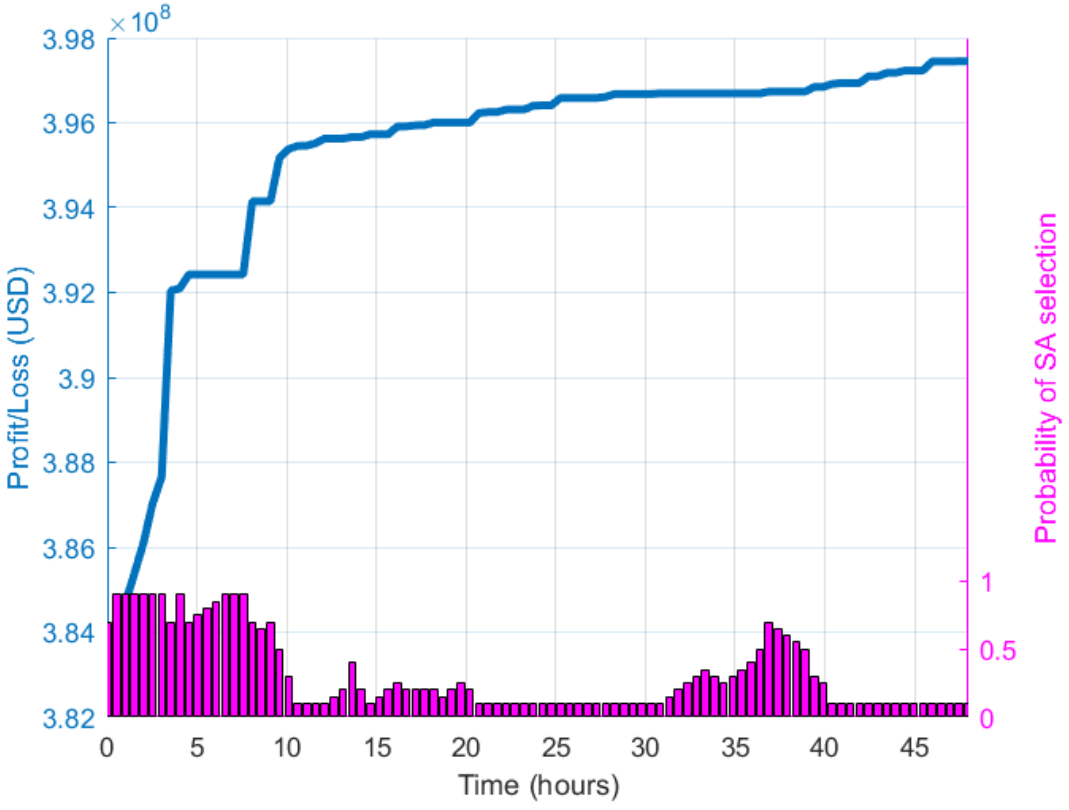

FIGURE 11.22: Best solutions progression of DIA algorithm with 30 minute interludes

We notice that the 30 minute version has a steeper increase in the initial stages of the algorithm. We also notice more strips of probability for SA, which confirms our thoughts that SA requires longer interlude periods to actually realise its strengths. This being said, the latter periods still see a prevalence for the TS algorithm. Over a 24 hour period TS still struggles to surpass the solution achieved by [Muñoz 2012]. Over a 48 hour

period, both the 5 minute and 30 minute versions of DIA manage to surpass the best NPV acquired.

Algorithm 16 Overview of DIA

Initialise parameters for SA and TS.
while Current computation time $\leq$ Alloted time **do**
  Set x,$f(x)$ values as x^0 and $f(x^0)$ respectively
  Set x_{best} and $f(x_{best})$ values as x^0_{best} and $f(x^0_{best})$ respectively
  Generate uniformly distributed random number, $r \in (0, 1)$
  if $r < \alpha$ **then**
    Run SA algorithm, Output: $x, f(x), x_{best}, f(x_{best})$
    if $\frac{f(x_{best})}{f(x^0_{best})} > 1$ **then**
      $\gamma = 0.2$
    else if $\frac{f(x)}{f(x^0)} > 1$ **then**
      $\gamma = 0.05$
    else
      $\gamma = -0.05$
    end if
    $\alpha = \alpha + \gamma, \quad \alpha \in [0.1, 0.9]$
    $\beta = 1 - \alpha, \quad \beta \in [0.1, 0.9]$
  else
    Run TS algorithm, Output: $x, f(x), x_{best}, f(x_{best})$
    if $\frac{f(x_{best})}{f(x^0_{best})} > 1$ **then**
      $\gamma = 0.2$
    else if $\frac{f(x)}{f(x^0)} > 1$ **then**
      $\gamma = 0.05$
    else
      $\gamma = -0.05$
    end if
    $\beta = \beta + \gamma, \quad \beta \in [0.1, 0.9]$
    $\alpha = 1 - \beta, \quad \alpha \in [0.1, 0.9]$
  end if
end while

11.6.4 NPV Analysis

The 30 minute interlude version of DIA performs better than the 5 minute interlude version of DIA over a 24 hour period however, over the full 48 hour computation period the 5 minute variant achieves a higher NPV value. It is for this reason that we will

focus our NPV analysis on the 5 minute interlude form of DIA, since a greater objective function is valued more highly than efficient computation time in this case.

As we can see from the below in Figure 11.23, DIA shows the same profile traits of the algorithms that came before it. There is vast NPV improvement in the early stages of the production schedule only to abate as we get to the latter periods. This should not be too surprising since in itself it is a blend of both TS and SA through the probabilistic selection mechanism.

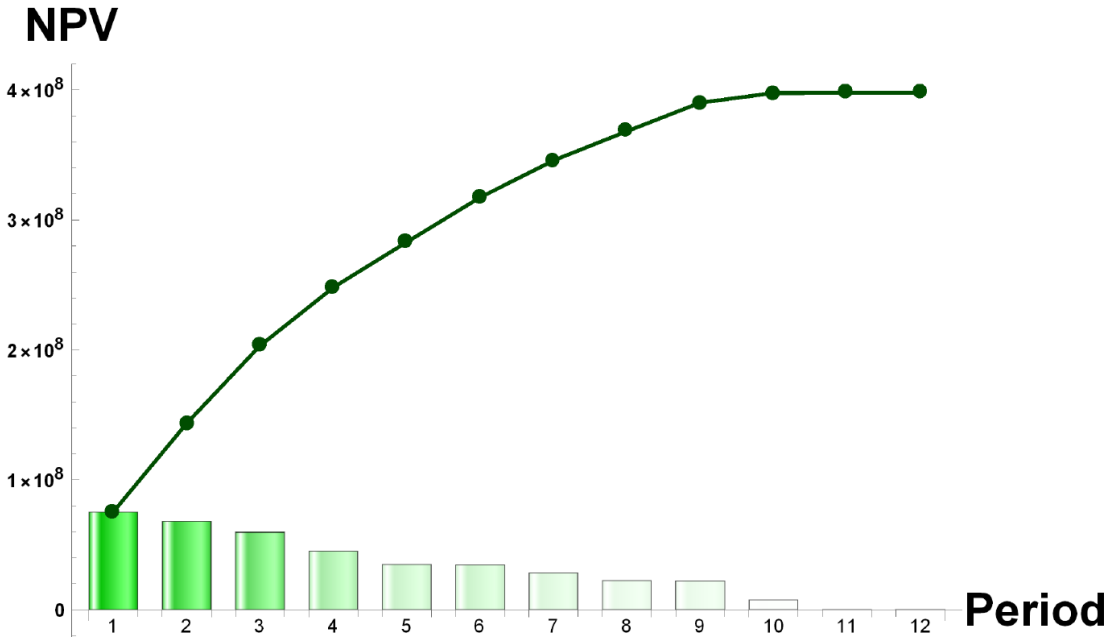

FIGURE 11.23: NPV profile of DIA Algorithm

It has already been established that SA is the superior algorithm. It achieves the highest NPV value amongst its compatriots in a computation time of 24 hours. For the purposes of comparison we include Figure 11.24 below, which is a comparison of DIA vs SA.

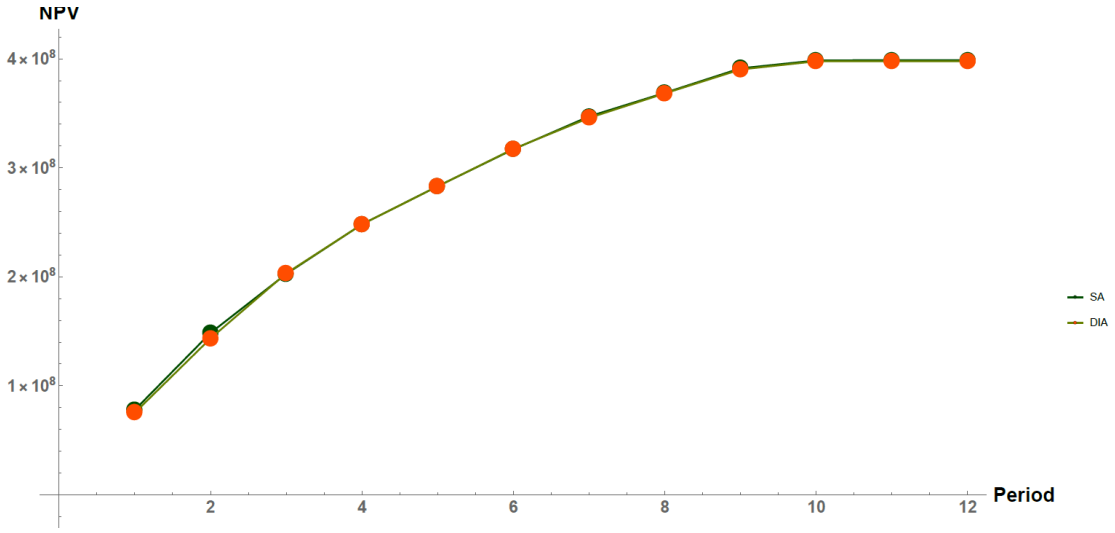

FIGURE 11.24: NPV Comparison : SA vs DIA

As can be seen in the figure, it is hard to discern the periods where SA outperforms DIA. Close inspection reveals that there are early periods where SA extracts slightly more value, followed by later periods. This is confirmed in table 11.17 below.

TABLE 11.17: NPV Statistics of DIA and SA

Time Period	DIA NPV/Period	DIA Cumulative NPV	SA NPV/Period	SA Cumulative NPV
1	7.5274×10^7	7.5274×10^7	7.7903×10^7	7.7903×10^7
2	6.8178×10^7	1.4345×10^8	7.0565×10^7	1.4847×10^8
3	5.9720×10^7	2.0317×10^8	5.4128×10^7	2.0260×10^8
4	4.4921×10^7	2.4809×10^8	4.5557×10^7	2.4815×10^8
5	3.4886×10^7	2.8298×10^8	3.4895×10^7	2.8305×10^8
6	3.4350×10^7	3.1733×10^8	3.3907×10^7	3.1695×10^8
7	2.8440×10^7	3.4577×10^8	3.0192×10^7	3.4715×10^8
8	2.2474×10^7	3.6824×10^8	2.1627×10^7	3.6877×10^8
9	2.1976×10^7	3.9022×10^8	2.2506×10^7	3.9128×10^8
10	7.4935×10^6	3.9771×10^8	7.2410×10^6	3.9852×10^8
11	7.2211×10^4	3.9778×10^8	2.7425×10^5	3.9879×10^8
12	-2.3414×10^4	3.9776×10^8	-3.1457×10^4	3.9876×10^8

11.6.5 Processing Production Analysis

We investigate the processing profile generated by the DIA algorithm over the 48 hours of computing time, with 5 minute interludes.

Figure 11.25 displays the profile of DIA. Initial inspection shows that it has much the same profile as our other metaheuristic solutions. Processing is maximised right up until period 10, which is confirmed in table 11.18.

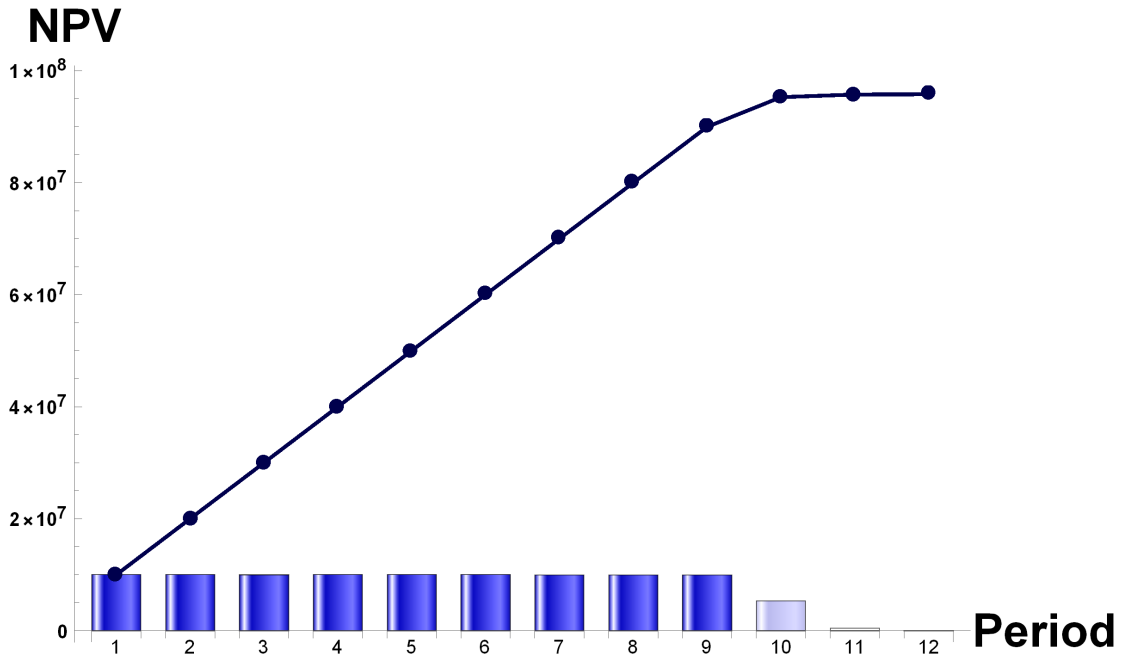

FIGURE 11.25: Processing Profile of the DIA Algorithm

Table 11.18: Processing Statistics of DIA and SA

Time Period	DIA Processing/Period	DIA Cumulative Processing	SA Processing/Period	SA Cumulative Processing
1	1.0000×10^7	1.0000×10^7	1.0000×10^7	1.0000×10^7
2	1.0000×10^7	2.0000×10^7	1.0000×10^7	2.0000×10^7
3	9.9919×10^6	3.0000×10^7	1.0000×10^7	3.0000×10^7
4	1.0000×10^7	4.0000×10^7	9.9989×10^6	4.0000×10^7
5	1.0000×10^7	5.0000×10^7	9.9961×10^6	5.0000×10^7
6	1.0000×10^7	6.0000×10^7	9.9890×10^6	5.9990×10^7
7	9.9877×10^6	6.9994×10^7	9.9872×10^6	6.9978×10^7
8	9.9881×10^6	7.9983×10^7	9.9859×10^6	7.9964×10^7
9	9.9846×10^6	8.9967×10^7	9.9857×10^6	8.9949×10^7
10	5.3044×10^6	9.5272×10^7	5.3390×10^6	9.5288×10^7
11	4.6914×10^5	9.5741×10^7	2.9136×10^5	9.5580×10^7
12	4.8540×10^4	9.5789×10^7	4.0398×10^5	9.5984×10^7

11.6.6 Conclusion

The DIA algorithm showed that we have contributed in developing a good production schedule. That being said, it failed to surpass SA in terms of solution quality, both over the 24 hour period as well as the 48 hour period, with both the 30 minute interlude version and the 5 minute version.

The algorithm did display some strong profile characteristics in both NPV as well as processing productivity. Its NPV values were relatively high in the earlier periods, suggesting that the production schedule seeks to quickly extract rich ore blocks before the time value of money diminishes the profit to be derived from their extraction. Its full exploitation of processing capacity in the earlier periods points towards a strong production schedule that does not waste its resource capacities in an effort to extract as much value as possible from the orebody.

A potential weakness of DIA is in the evaluation of algorithm probabilities. SA was found to be the superior algorithm but falls out of favour quickly with the overseeing parent algorithm that is the decision mechanism within DIA. Even when affording SA more time to make progress within DIA it still was on the back foot relative to TS selections. This implies that DIA is quite "greedy" in its focus, and perhaps short-sighted.

However, DIA still produced an able production schedule that was not far off the NPV feats achieved by SA. It also demonstrated clear improvement upon the basal form of TS. Examination of objective function values and probabilities display the perceived benefit of the probabilistic selection mechanism where periods lacking in progress saw a move to higher SA probabilities to escape these weak phases - only to see moves back to TS selections.

This produces some questions around the γ variable, the key distributor of probabilities per algorithm for each interlude. The interplay between the probabilities and NPV values suggests that potentially SA helps the parent algorithm escape weak periods, which the TS algorithm then capitalises on. TS is left with the majority of the rewards in probability from the γ variable.

Chapter 12

Conclusion

The Open Pit Mine Production Scheduling Problem (OPMPSP) is a complex optimisation problem within the Operations Research discipline.

The existing methods in literature are capable of producing strong solutions to this problem, producing near optimal production schedules to extract profit. Their benefit to mining endeavours, and certainly the South African mining industry, is indisputable.

However, these methods can be improved upon through analysis and adaptation. This was achieved through reproducing the algorithms, running them on real-world mining instances, optimising parameters and utilising the modern coding architectures, frameworks and computation power now available.

The methods we explored in this research form only part of a wide array of metaheuristics. The OPMPSP likeness to the Precedence Constrained Knapsack Problem makes it a candidate for these types of methods to find optimal production schedules.

The Tabu Search (TS) algorithm displayed the characteristic of prolonged improvement, if rather slow. It was the weakest performer amongst the explored methods in terms of NPV however, it was shown that it does perform well on smaller problem instances, such as the Marvin block model instance. Using SIMD principles within the MATLAB programming language, TS became more robust. We omitted the need to constrain the number of shifts since the computation time needed to produce these shifts was broadly negligible. It was highlighted that parameter value selection is critical to the effective functioning of TS. A number of suggested generalised formulae were proposed depending on problem size.

The Simulated Annealing (SA) algorithm had a distinguished performance on achieving an objective function value higher than the best found before this research in [Muñoz 2012].

Although there are long periods of little improvement on NPV, SA showed better exploration of the solution space than TS and has more chance of finding better solutions than TS because of this. Its shifting mechanism, or how it proposes new production schedules, involves more shifts within iteration steps and is seen as an advantage over TS when it comes to larger problems. Analysis was also done on the SA parameters and they are equally as important as TS to the efficient implementation of the algorithm.

The Dual Interchange Algorithm (DIA) method revealed the potential benefits of combining 2 algorithms under the supervision of a parent algorithm, where this parent algorithm assigns probability for selection to these algorithms based on their seen contributions to improving the objective function. This approach was shown to be better than TS but underperform when compared to SA in its vanilla form.

DIA revealed a bias towards TS as the reward variable, γ, which controls the probabilities of selection, favours immediate improvement even if its is not a large improvement. Recall that this was one of the traits of the TS algorithm. To investigate this further we explored 2 variants of the DIA algorithm. The first having an interlude every 5 minutes and the second having an iterlude of every 30 minutes. The 30 minute version did afford SA more opportunities but the bias was still clear for TS under these settings.

All algorithms produced a production schedule that favoured extraction of rich ore blocks as soon as possible before they fall victim to diminish profits through discounting. This was displayed in their NPV and processing profiles.

At its core, a strong production schedule is one that mines unavoidable waste blocks as late as possible as well as mining economically feasible ore blocks as early as possible. Through this realisation we introduced the Pseudo Greedy Search (PGS), which seeks to improve a production schedule within its local neighbourhood of the solution space by actively searching for waste blocks that can be moved to later periods and ore blocks that can be moved to earlier periods, without violating any constraints of the algorithm. This continues until no waste blocks or ore blocks can be moved in such a fashion that benefits the NPV of the production schedule.

Through our research we came across certain ideas that we envisage would be beneficial for future work on this interesting problem, as well as mining ventures. These are laid out in the proceeding section.

12.1 Further Work

12.1.1 Further Algorithm Parameter Optimisations

An area where we foresee further work is in the space of parameter optimisations. There was some work done here within this dissertation but we believe that there is scope for more.

A number of the parameters were ascertained through observation by simulation. Ideally, parameters should be determined through quantitative means where possible. There is an inherent amount of randomness, despite any number of simulations one implements. Therefore, it provides more comfort as scientists to verify certain parameter values through means other than empirical.

Parameters remain static throughout the life of the algorithms. With the rise of machine learning and artificial intelligence it seems untenable to not consider these methods in today's scientific climate. Machine learning would offer the opportunity to have parameters that adapt and change dynamically depending on the progress of the development of the production schedule. For example, in the case of DIA, the γ constant could be determined agent learning based on objective function value improvement along with state space exploration. SA showed more promise in exploring other regions of the state space than TS and thus achieving better production schedules. Machine learning approaches could grant SA more leeway in exploration than often short-sighted probability increase that was often afforded to TS.

Machine learning could also find application within the TS or SA algorithms themselves. For example, the cooling and heating procedures are rather linear in the case of SA. In its implementation within the OPMPSP there may be more optimal heating and cooling procedures that could be algorithmically deciphered by machine learning approaches, rather than a simple constant factor of cooling coupled with constant factors of heating.

12.1.2 GPU (Graphics Processing Unit) Multithreading

In the current computation landscape, it is not uncommon to find most personal computers, and even the majority of our cellular phones (smartphones), equipped with parallel processing capabilities. The splitting of processes into partitions for simultaneous execution started with CPUs. Initially, a single CPU core would have multiple threads running in tandem. This was then extended to include multiple cores, each with their own simultaneous computations.

Graphics cards are similar to computer motherboards in that it is essentially a circuit board that houses a CPU (central processing unit) and RAM (random access memory). The CPU component for a graphics card is known as a GPU (graphics processing unit). GPUs are designed predominantly for graphics rendering processing, which involves a host of complex and sometimes intensive calculations.

These calculations and operations have chains that are separate from one another and thus execute independently. Many sought to exploit the potential of harnessing the computational prowess of GPUs for matters outside of graphics rendering. GPUs were seen as promising tools to alleviate computation bottlenecks, thus making simulations, calculations and algorithms more efficient and powerful. Their use can be found in a variety of industries. For example, GPUs are being utilised in solving complex problems in computational finance.

GPU cores do not have the raw processing power of CPU cores however, they are havens for a paradigm of parallelisation known as SIMD (Single instruction multiple data). Our programming language of choice in this research has been MATLAB, which enjoys the benefits of SIMD as well. Not to mention, MATLAB comes with support for GPU integration. The end of this section displays a simplistic view of SIMD, displayed in Figure 12.1.

It is our belief that many of the algorithms and metaheuristics for solving problems such as the OPMPSP can be made more powerful through the exploitation of the computational power available in these modern times of Operations Research. Many of our coding practices within TS, SA and DIA employed some form of SIMD but never in the GPU domain.

Exploring this further could greatly improve the performance of these algorithms and open up further opportunities for solving more complex versions of the problem as well as create more feasible options to solving this problem that were seen as infeasible before.

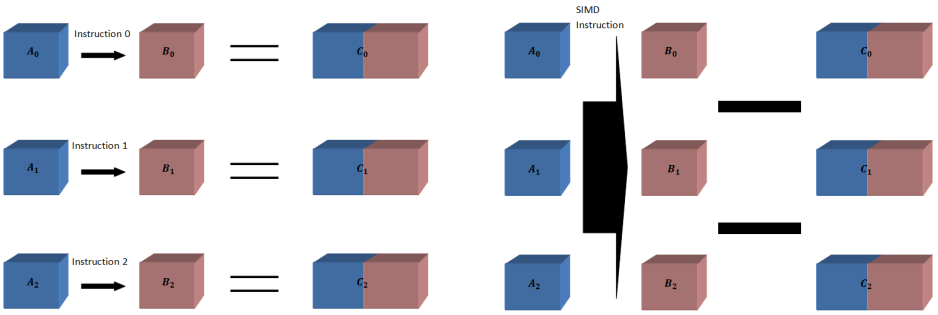

FIGURE 12.1: A contrast of a scalar operation and a SIMD operation

12.1.3 Modelling Ore Stochastics

The process of Kriging, which was described at the beginning of this dissertation, is not a definitive classification of the orebody. There is a range of mathematical procedures, including interpolation, that gives the mining venture an idea of how the orebody appears and the location of ore deposits.

[Lamghari et al. 2012] acknowledge this and adapt their model to include several equally probable instances of the orebody. Of course this makes the quest to find a strong production schedule even more arduous as the model becomes more complex. However, one cannot argue with the logic behind their model, given the nature of the creation of the simulated orebody.

Such complex models become more manageable as scientists and mathematicians harness the increasingly powerful developments in computation capacities. A compelling example has been mentioned in the case of GPU programming, seen in section 12.1.2.

Below is the model formulation we utilised in this dissertation with the inclusion of S equally probable block models.

$$\max \frac{1}{S}\left\{\sum_{s=1}^{S}\sum_{t=1}^{T}\sum_{i=1}^{N} v_{its}x_{it} - \left(\sum_{s=1}^{S}\sum_{t=1}^{T}(c_t^o\delta_{ts} + c_t^m\zeta_{ts})\right)\right\} \tag{12.1}$$

subject to:

$$\sum_{t=1}^{T} x_{it} \leq 1 \qquad i = 1, ..., N. \tag{12.2}$$

$$x_{it} - \sum_{\tau=1}^{t} x_{p\tau} \leq 0 \qquad i = 1, ..., N, \quad p \in P_i, \quad t = 1, ..., T. \tag{12.3}$$

$$\sum_{i=1}^{N} m_i x_{it} - \zeta_{ts}^m \leq \bar{W}_t \qquad t = 1, ..., T, \quad s = 1, ..., S. \tag{12.4}$$

$$\sum_{i=1}^{N} o_i m_i x_{it} - \delta_{ts}^o \leq \bar{O}_t \qquad t = 1, ..., T, \quad s = 1, ..., S. \tag{12.5}$$

$$x_{it} \in \{0, 1\} \qquad i = 1, ..., N, \quad t = 1, ..., T. \tag{12.6}$$

12.1.4 Modelling of Underlying Commodity

Inspired from the work on numerous simulations of the orebody by [Lamghari et al. 2012], we suggest further research into the OPMPSP to include simulations or cases of the underlying commodity price behaviour. After all, it would not be illogical to assume that the values of each block in the block model are heavily dependent on the price of the commodity in question.

We foresee such an inclusion into modelling to be challenging. Indeed, forecasting of future securities prices is not a trivial task and such questions have created an established industry in the financial research discipline. Doing so would inherently involve some risk factor in the formulation of the model. For example, when solving the OPMPSP for a open pit gold mine, with the view that the gold price will increase over the next 10 years will produce a different schedule, as well as different NPVs, if the gold price had in fact declined.

12.1.5 Tabu Search - Simulated Annealing Hybrid Algorithm

TS and SA both come with their own set of algorithmic features and behaviours in solving the OPMPSP. We propose further research be done on attempting to combine certain aspects of these 2 algorithms within a single algorithm that utilises some of the stronger components of both of the originals.

For instance, the long-term memory structure found in TS is in some regards superior to the Boltzmann transitioning in SA, since SA can at times venture too far from optima. In generating a proposed shift for SA one could rather generate a number of the best shifts rather than a collection of random shifts. One could also determine the likelihood of these shifts occurring by utilising the long-term memory structure array, F, that was seen in TS.

Bibliography

[Akaike et al. 1999] Akaike, A. and Dagdelen, *A strategic production scheduling method for an open pit mine*, In Proceedings of the 28th Applications of Computers and Operations in the Mineral Industries Conference (APCOM), C. Dardano, M. Francisco and J. Proud, eds., Golden, CO, 729-738, 1999.

[Albert 2006] Albert, S., *Solving Mixed Integer Linear Programs Using Branch and Cut Algorithm*, Master of Mathematics, NCSU, 2006.

[Amaya et al. 2009] Amaya, J., Espinoza, D., Goycoolea, M., Moreno, E., Prevost, T., Rubio, E., *A Scalable Approach to Optimal Block Scheduling*, Proc. APCOM 2009 (The Canadian Institute of Mining, Metallurgy and Petroleum, Vancouver, British Columbia, Canada), 567575, 2009.

[Bley *et al.* 2010] Bley, A., Boland, N., Fricke, C., Froyland, G. , *A strengthened formulation and cutting planes for the open pit mine production scheduling problem*, Computers and Operations Research 37 (9) , pp. 1641-1647, 2010.

[Bloomberg] Bloomberg L.P. (2016) Bloomberg database/mobile app. Retrieved Dec. 30, 2016.

[Burne-Jones 1861] Burne-Jones, Sir Edward,. Theseus and the Minotaur in the Labyrinth, 1861, Tile Design. Birmingham Museum and Art Gallery, Birmingham.

[Caccetta and Hill 2003] Caccetta, L., Hill, S.P., *An application of branch and cut to open pit mine scheduling*, Journal of Global Optimization 27 (2-3) , pp. 349-365, 2003.

[Chamber of Mines F&F 2016] Chamber of Mines of South Africa, *Facts & Figures 2016*, Available from: Chamber of Mines South Africa, http://www.chamberofmines.org.za/industry-news/publications/facts-and-figures, [June,2017]

[Chamber of Mines F&F 2012] Chamber of Mines of South Africa, *Facts & Figures 2012*, Available from: Chamber of Mines South Africa, http://www.chamberofmines.org.za/industry-news/publications/facts-and-figures, [June,2012]

[Chamber of Mines AR 2012] Chamber of Mines of South Africa, *Annual Mining Sector Report 2012*, Available from: Chamber of Mines South Africa, http://www.bullion.org.za, [12 Dec 2012]

[Chamber of Mines AR 2015] Chamber of Mines of South Africa, *Annual Mining Sector Report 2015*, Available from: Chamber of Mines South Africa, http://www.chamberofmines.org.za/industry-news/publications/annual-reports/send/15-archived/226-annual-review-2015, [17 June 2017]

[Chicoisne et al. 2012] Chicoisne, R. , Espinoza, D. , Goycoolea, M. , Moreno, E. , Rubio, E. , *A New Algorithm for the Open-pit Mine Production Scheduling Problem*, Operations Research, 60(3) pp. 517-528, 2012.

[Clausen 2003] Clausen, J., *Branch and Bound Algorithms - Principles and Examples*, Department of Computer Science, University of Copenhagen, 2003.

[Cullenbine et al. 2011] Cullenbine, C., Wood, R.K., Newman, A., *A sliding time window heuristic for open pit mine block sequencing*, Optimization Letters 5 (3) , pp. 365-377, 2011.

[Dahl and Mannino 2012] Dahl, G., Mannino, C.,*Notes on Combinatorial Optimisation*, Department of Mathematics and Department of Informatics, CMA, University of Oslo, Norway. Accessed 10 March 2014. http://heim.ifi.uio.no/~geird/comb_notes.pdf

[Dantzig and Thapa 2003] Dantzig, G.B., Thapa, M.N., *Linear Programming 2: Theory and Extensions*, Springer Series in Operations Research and Financial Engineering, 2003.

[Denby et al. 1994] Denby, B., Schofield, D., *Open-pit design and scheduling by use of genetic algorithms*, Transactions - Institution of Mining and Metallurgy, Section A 103 (Jan-April) , pp. A21-A26, 1994.

[Dagdelen et al. 1986] Dagdelen, K., Johnson, T., *Optimum Open Pit Mine Production Scheduling by Lagrangian Parametrization*, Proc. 19th Internat. Appl. Comput. Oper. Res. Mineral Indust. (APCOM) Sympos., SME, Littleton, CO, 127141, 1986.

[Espinoza et al. 2012] Espinoza, D., Goycoolea, M., Moreno, E., Newman, A. , *MineLib: A Library of Open Pit Mining Problems*, Annals of Operations Research , pp. 1-22, 2012.

[Feo and Resende 1989] Feo, F.A, Resende M.G.C, *A probabilistic heuristic for a computationally difficult set covering problem*, Operations Research Letters, 8:6771, 1989.

[Ferland et al. 2007] Ferland, J.A., Amaya, J., Djuimo, M.S., *Application of a particle swarm algorithm to the capacitated open pit mining problem*, Studies in Computational Intelligence 76 , pp. 127-133, 2007.

[Fricke 2006] Fricke, C., *Applications of integer programming in open pit mining*, PhD thesis, Department of Mathematics and Statistics, The University of Melbourne, 2006.

[Froyland et al. 2007] Froyland, G., Menabde, M., Stone, P., Hodson, D., *The value of additional drilling to open pit mining projects*, Australasian Institute of Mining and Metallurgy Publication Series , pp. 245-252, 2007.

[Gendreau et al. 2005] Gendreau, M., Potvin, J., *Tabu Search*, Search Methodologies : Introductory Tutorials in Optimization and Decision Support Techniques, pp. 165 - 186, 2005.

[Gershon 1987] Gershon, M.E., *Heuristic Approaches for Mine Planning and Production Scheduling*, International Journal of Mining and Geological Engineering, Volume 5, Issue 1, pp. 1-13.

[Glover 1986] Glover, F., *Future paths for integer programming and links to artificial intelligence*, Computers and Operations Research, 5, 533-549, 1986.

[Hartmann 1998] Hartmann, S., *A Competitive Genetic Algorithm for the Resource-constrained Project Scheduling*, Naval Research Logistics, 456, pp 733 - 750, 1998.

[Hansen and Mladenovic 2001] Hansen, P., Mladenovic, N., *Variable Neighbourhood Search: Principles and Applications*, European Journal of Operations Research 130(3), pp. 449 - 467.

[Jackson 1999] Jackson, T., *The Boer War*, London: Channel 4 Books, 1999. Print.

[Johnson 1968] Johnson T.B, *Optimum Open-pit Mine Production Scheduling*, Ph.D. Thesis, Operations Research Department, University of California, Berkley, Berkley, 1968.

[Johnson et al. 1989] Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C., *Optimization by simulated annealing. An experimental evaluation. Part I. Graph partitioning*, Operations Research, Volume 37, Issue 6, pp. 865 - 892, 1989.

[Kawahata 2007] Kawahata, K., *A new algorithm to solve large scale mine production scheduling problems by using the Lagrangian relaxation method*, Ph.D. thesis, Colorado School of Mines, 2007.

[Khan et al. 2014] Khan, A., Niemann-Delius, C., *Production Scheduling of Open Pit Mines Using Particle Swarm Optimization Algorithm*, Advances in Operations Research, Volume 2014, Article 208502.

[Kim 1978] Kim Y.C, *Ultimate Pit Limit Design Methodologies using Computer Models - state of the art*, Mining Engineering, 28: 1454 - 1459, 1978.

[Kirkpatrick et al. 1983] Kirkpatrick, S., Gelatt, Jr., C.D., Vecchi, M.P., *Optimization by Simulated Annealing*, Science, Volume 220, Issue 4598, pp. 671 - 680, 1983.

[Kumral 2011] Kumral, M., *Incorporating Geo-metallurgical Information into Mine Production Scheduling*, Journal of the Operational Research Society 62 (1) , pp. 60-68, 2011.

[Kumral 2012] Kumral, M., *Production planning of mines: Optimisation of block sequencing and destination*, International Journal of Mining, Reclamation and Environment 26 (2) , pp. 93-103, 2012.

[Kumral 2013] Kumral, M., *Optimizing Orewaste Discrimination and Block Sequencing through Simulated Annealing*, Applied Soft Computing Journal, Volume 13, Issue 8, pp. 3737 - 3744, 2013.

[Lamghari et al. 2012] Lamghari, A., Dimitrakopoulos, R. , *A Diversified Tabu Search Approach for the Open-pit Mine Production Scheduling Problem with Metal Uncertainty*, European Journal of Operational Research 222 (3) , pp. 642-652, 2012

[Lane 1997] Lane, K.F., *The Economic Definition of Ore, Cut-off Grades in Theory and Practice*, Mining Journal Books Limited, Second Edition, London, 1988.

[Lane 1988] Lane, K.F., *The Economic Definition of Ore, Cut-off Grades in Theory and Practice*, Mining Journal Books Limited, First Edition, London, 1988.

[Lerchs and Grossmann 1965] Lerchs, H., Grossmann, I.F., *Optimum Design of Open-pit Mines*, Transactions CIM, Vol. LXVIII: 17-24, 1965.

[Financial Times] Financial Times (2018) FT Markets Summary. Retrieved Jan. 02, 2018.

[Mathematical Optimization Society] Mathematical Optimization Society, *About the Mathematical Optimization Society*, Accessed 23 August 2012. http://www.mathopt.org/

[Menabde et al. 2007] Menabde, M., Froyland, G., Stone, P., Yeates, G.A., *Mining schedule optimisation for conditionally simulated orebodies*, Australasian Institute of Mining and Metallurgy Publication Series , pp. 379-383, 2007.

[Metropolis et al. 1953] Metropolis, W., Rosenbluth, A., Rosenbluth, M., Teller, A., Teller, E., *Equation of State Calculations by Fast Computing Machines*, J. Chem. Phys. 21, 1087 - 1092, 1953.

[Minelib] Retrieved Dec. 5 2014, http://mansci-web.uai.cl/minelib/Help.xhtml

[Minnit 2004] Minnit, R.C.A., *Cut-off Grade Determination for the Maximum Value of a Small Wits-type Gold Mining Operation*, Journal of the South African Institute of Mining and Metallurgy, 2004.

[Moreno et al. 2010] Moreno, E., Espinoza, D., Goycoolea, M., *Large-scale multi-period precedence constrained knapsack problem: A mining application*, Electronic Notes in Discrete Mathematics, 2010.

[Muñoz 2012] Muñoz, G., *Modelos de optimizacion lineal entera y aplicaciones a la mineria*, MSc Thesis, Dept. Mathematical Engineering, Universidad de Chile, 2012.

[Nemhauser and Wolsey 1988] Nemhauser, G.L., Wolsey, L.A., *Integer and Combinatorial Optimization*, Wiley-Interscience, New York, 1988.

[Newman *et al.* 2010] Newman, A.M., Rubio, E., Caro, R., Weintraub, A., Eurek, K. , *A review of operations research in mine planning*, Interfaces 40 (3) , pp. 222-245, 2010.

[onlygold.com] http://onlygold.com/info/Historical-Gold-Prices.asp

[Picard 1976] Picard, J.C, *Maximum Closure of a Graph and Applications to Combinatorial Problems*, Management Science, 22: 1268-1272, 1976.

[Pisinger 1995] Pisinger, D., *Algorithms for Knapsack Problems*, Phd thesis, Department of Computer Science, University of Copenhagen, 1995.

[Samavati et al. 2017] Samavati, M., Essam, D., Nehring, M., Sarker R., *A local branching heuristic for the open pit mine production scheduling problem*, European Journal of Operations Research, Volume 257, Issue 1, 2017.

[Schrijver 2005] Schrijver, A.,*On the History of Combinatorial Optimization (till 1960)*, In Handbook of Discrete Optimization, pp. 1-68, edited by Aardal, K., Nemhauser, G.L., Weismantel, R., 2005.

[Struik 1987] Struik, D.J., *A Concise History of Mathematics*, 4^{th} edition, New York: Dover Publications, 1987.

[Sutton and Barto 1998] Sutton, R.S., Barto, A.G., *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 1998.

[Wolsey 1998] Wolsey, L.A., *Integer Programming*, John Wiley & Sons, New York, 1998.

[Zhang 2006] Zhang, M., *Combining genetic algorithms and topological sort to optimize open-pit mine plans*, Proceedings of the 15th Mine Planning and Equipment Selection, pp. 1234 1239, 2006.

[Zuckerberg et al. 2007] Zuckerberg, M., Stone, P., Pasyar, R., Mader, E., *Joint ore extraction and in-pit dumping optimisation*, Australasian Institute of Mining and Metallurgy Publication Series , pp. 137-140, 2007.