OPEN-PIT MINE PLANNING WITH OPERATIONAL CONSTRAINTS

by

Matthew Vernon Deutsch

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Doctor of Philosophy (Mining and Earth Systems Engineering).

Golden, Colorado

Date _____

Signed: _____

Matthew Vernon Deutsch

Signed: _____

Dr. Kadri Dağdelen
Thesis Advisor

Signed: _____

Dr. Thys B. Johnson
Thesis Co-Advisor

Golden, Colorado

Date _____

Signed: _____

Dr. Merritt Steve Enders
Professor and Department Head
Department of Mining Engineering

ABSTRACT

Open-pit mines must be designed to develop the Earth's natural resources in the most responsible, sustainable, and economic way. Traditional mine planning optimization methods do not consider operational constraints; such as minimum mining width or minimum pushback width constraints, and often do not generate realistic, actionable designs. This dissertation develops techniques to incorporate operational constraints into open-pit mine planning which allows for engineers to more accurately convert mineral resources into mineral reserves and better evaluate the economic viability of open-pit mining projects. A major practical challenge is that the resulting mathematical models are very large, with potentially hundreds of millions of variables and constraints. Addressing this challenge and delivering tools which are usable on real-world 3D datasets requires a theoretically motivated and computationally grounded approach.

The first contribution of this dissertation is an efficient implementation of the pseudoflow algorithm for the well known ultimate pit problem. Modest theoretical improvements and practical computational improvements combine to create a fast and efficient open source ultimate pit optimizer, called "MineFlow," which is more performant than all evaluated commercial alternatives. A model with sixteen million blocks which takes over three minutes to solve with a commercial ultimate pit optimizer is solved in nine seconds with this implementation.

The second contribution is a formulation and methodology for the ultimate pit problem with minimum mining width constraints. These operational constraints restrict the shape of the ultimate pit in order to provide suitably large operating areas which can accommodate the large machinery used in open-pit mining. This problem is shown to be $\mathcal{NP}$-complete and several optimization approaches are developed in order to compute high quality results for large block models in a reasonable amount of time. The two most effective approaches use Lagrangian relaxation and the Bienstock-Zuckerberg algorithm which are modified for this problem.

Moreover, the formulation is extended to open pit direct block scheduling problems with operational constraints and solved using a newly developed method based on the Bienstock-Zuckerberg algorithm. This approach is applicable to large, realistic, open-pit planning problems that span multiple time periods and multiple possible destinations for each block.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Bienstock Zuckerberg . . . . . BZ

Economic Block Value . . . . . EBV

Integer Linear Programming . . . . . ILP

Mixed Integer Programming . . . . . MIP

Net Present Value . . . . . NPV

Random Access Memory . . . . . RAM

Selective Mining Unit . . . . . SMU

ACKNOWLEDGMENTS

I have led a very privileged and fortunate life which has allowed me to pursue this undertaking. This good fortune has extended to a great many wonderful people in my life which I would like to acknowledge and thank here.

Special thanks are dedicated to my advisors: Dr Kadri Dağdelen and Dr Thys Johnson who both provided guidance and assistance throughout the process of preparing this dissertation. I also greatly appreciate the contributions and considerations from Dr Alexandra Newman, Dr Dinesh Mehta, and Dr Rennie Kaunda. All of these individuals contributed not only ideas and advice relating to the work itself, but also important life lessons.

I would also like to thank Dr Marcelo Godoy, Larry Allen, Dr Conor Meagher, and Dr Ryan Goodfellow at the Newmont Mining Corporation who provided many useful discussions and the majority of the funding for the work contained herein. Bill Blattner, Bruce Ramsay, Dr Keith Ramsay, and Rob Hardman at Maptek also deserve special thanks for their advice and for generously allowing me to split my time between developing software and pursuing this effort.

My good fortune has also extended to my family and friends. Thank you to Pauline, Clayton, Jared, and Kelsey for your encouragement, advice, and support. Thank you to my friends for the much needed distractions and emotional support.

However my greatest thanks are owed to my wife Amanda and my children Philip, Isaac, and Lilah. They had to put up with the most stress and trouble to make this whole thing a reality and despite that they maintained a positive attitude and provided never-ending love and support. Especially Amanda whose tireless efforts and self sacrifice throughout this endeavor can not be overstated. Thank you.

For Amanda

CHAPTER 1
INTRODUCTION

1.1 Problem Setting and Background

Our society is built on natural resources, many of which are extracted from the Earth via large scale mining operations. It is of the utmost importance that these limited resources are developed in a responsible and sustainable way such that future generations will be able to enjoy a quality of life that is the same, or better, than our own. The mining companies which explore for, plan, and ultimately develop these natural mineral deposits require suitable mathematical tools and techniques in order to make the best decisions possible. Better, and more informed, decisions in the planning process will lead to better plans that yield an improved return on investment and also provide more benefit for local communities and accommodate our growing global needs.

In recent decades the global trends in mining have been towards mining larger and lower grade deposits. The demand for metal is increasing and improvements in all aspects of the mining and mineral processing chain have made these previously uneconomic deposits viable. Underground mining, especially through large scale bulk mining methods such as block caving, are increasingly being chosen for green and brownfield projects alike. However, in many situations surface mining is still preferred due to lower capital and operating costs, among other considerations.

The field of operations research is a vital part of mine planning. The techniques provided by operations research allow practitioners to form mathematical models of the complex engineering problems that they face and analyze them in order to gain necessary insight. The solutions provided by operations research guides mining engineers in navigating the complex operational, geological, financial, environmental, and social challenges in mining. In this dissertation these tools are developed further in order to provide additional insight, tackle previously under-represented operational constraints, and provide necessary improvements in the field of long range open-pit mine planning which are applicable to operating and future surface mines.

A mathematical model is only useful in so far as it represents the real underlying decision problem, and how well the resulting analysis can be used. A poor model can often do more harm than good, and unrealistic or inappropriate assumptions in the modeling process can mislead

engineers and practitioners, sometimes drastically. For this work, the emphasis is on more accurately representing underlying decision problems and more closely matching the mathematical models with the realities of open-pit mining. Specifically operational constraints, or constraints that must be imposed on the plan because large equipment is used, are considered herein.

The unfortunate reality, however, is that increased accuracy generally leads to increased complexity and can limit the applicability to smaller and less useful models because the computational process of solving for the correct answer takes an inordinate amount of time. Therefore, special emphasis has been placed on developing techniques that are computationally efficient and are directly usable with full sized, realistic, 3D datasets from real mining operations.

1.2 Goals and Outline

The goal of this dissertation is to develop high quality and computationally efficient models which addresses the most fundamental operational constraints in open-pit long range mine planning. The models must be correct, flexible, and applicable to a wide range of real world deposits. To achieve this goal the dissertation is structured as follows.

Chapter 2 reviews relevant literature and establishes the necessary prerequisites in the fields of mine planning and operations research. The focus is on long range mine planning problems, specifically the ultimate pit limit problem and the block scheduling problem. Current approaches to incorporating operational constraints are discussed.

Chapter 3 documents a series of improvements to the foundational ultimate pit problem. The general network flow based pseudoflow algorithm is dissected and reimagined solely for the ultimate pit problem. Small optimizations are presented with a dramatic impact on the computational performance of the algorithm. This provides the ability to solve hundreds or even thousands of ultimate pit problems extremely rapidly. A novel notation is developed for the pseudoflow algorithm with moderate pedagogical value, and a novel means by which the accuracy of a given precedence pattern can be computed is presented. Finally, Chapter 3, concludes with a computational comparison between this new, and permissive open-source, implementation of the pseudoflow algorithm and long standing commercial alternatives.

Chapter 4 extends the ultimate pit problem to consider minimum mining width constraints which are conventionally addressed late in the mine design process by hand. Incorporating these

operational constraints directly into the ultimate pit problem and using proper optimization methods ensures that the pit designs accurately reflect how they will be developed and eventually produced. This allows for more accurate reserve estimates and better decision making opportunities. Several different solvers are developed varying in complexity from simple geometric heuristics to iterative optimization procedures based on the tenets of duality theory. Another computational comparison is completed to evaluate the proposed solvers by their solution quality and overall speed. Run time is of the utmost importance in real world applications especially when evaluating uncertainty and performing sensitivity analysis which require solving multiple problems in sequence.

Chapter 5 describes various methods for modeling operational constraints in the more complicated area of direct block scheduling. Where the ultimate pit problem is concerned with *what* material should be mined, the block scheduling problem additionally considers *when* to mine different areas of the deposit as well as to what destination the mined material should be sent. This problem's mathematical model is a large scale mixed integer linear program that is very difficult to solve directly. This chapter describes a new algorithm to solve this problem iteratively using a modified Bienstock Zuckerberg decomposition algorithm.

Finally, Chapter 6 contains the final conclusions and documents avenues for future research. Many of the limitations associated with this work are also discussed.

The appendices contain two more specialized developments which are better suited to appearing after the main text. Appendix A presents a computational complexity proof associated with the main problem considered in Chapter 4. The ultimate pit problem with minimum mining width constraints is shown to belong to the class of $\mathcal{NP}$-complete problems for which no known polynomial time algorithm currently exists. This is shown by reducing the 3-SAT problem to a simplified variant of the ultimate pit problem with minimum mining width constraints.

Appendix B contains a novel dynamic programming algorithm for selecting a subset of pits to serve as pushbacks following parametric analysis in long range mine planning. This algorithm is motivated by the fast ultimate pit optimization approach developed in Chapter 3 which allows for practitioners to compute many more nested pits than typical. Selecting an evenly spaced set of pushbacks from these hundreds of nested pits automatically is possible with this new algorithm which removes a long standing inconvenience faced by mine planners.

CHAPTER 2

OPEN-PIT MINE PLANNING: REVIEW AND ALGORITHMS

Mining is responsible for providing many of the raw materials necessary to build and maintain our civilization. The design, planning, and scheduling of mining operations is critically important to maintaining our way of life and providing the building blocks of our society. The mine design process helps determine the economic viability of mining projects around the globe and provides plans which guide the operation through development, production, and reclamation. This is an extremely difficult and complex problem because it is both very large with many different concerns including geological, financial, environmental, and social; and contains many different interdependent systems and interactions.

To contend with this complexity mining engineers and technologists in the mining industry use a wide range of techniques from operations research and other fields in order to make the best decisions possible. The concepts and prior work described in this chapter provide the necessary background and the basis for the new methods introduced in the subsequent chapters.

Section 2.1 reviews the broader field of open-pit mine planning including its main objectives, modeling, and the modern approach to open-pit mine planning. The circular nature of mine planning is introduced and several of the subproblems are discussed. The subproblems most relevant to this dissertation are then described in the following sections.

Section 2.2 discusses the foundational ultimate pit problem which helped launch the field of computational open-pit mine planning. The ultimate pit problem has a well defined mathematical structure which has afforded multiple different exact solution techniques and heuristics which are described herein. The max-flow / min-cut approach to identifying the ultimate pit limit is described in detail as it is used extensively when considering operational constraints and is further developed in Chapter 3.

Where the ultimate pit problem is concerned only with *what* is mined throughout the life of the mine, the block scheduling problem concerns itself with *when* to mine it and how to process the mined material. Section 2.3 discusses the block scheduling problem where the ultimate pit is divided into a set of smaller nested pits which are extracted in sequence. These nested pits are

also referred to as pushbacks, phases, or cutbacks and are chosen to maximize the net present value (NPV) of the mine while adhering to additional constraints such as minimum and maximum mill grades, blending, stockpiling and more.

Section 2.4 presents prior work relating to extending the ultimate pit problem and the block scheduling problem to consider operational constraints. Special attention is given to extensions which consider minimum mining width constraints as many are implemented and considered in Chapter 4.

Section 2.5 presents the necessary background around mathematical programming and optimization. The Lagrangian relaxation approach to dualizing certain constraints is given special attention as it is used later in the dissertation to accommodate operational constraints.

2.1 Open-Pit Mine Planning

Open-pit mines are characterized by mining deposits which are near-surface using horizontal benches and dumping waste material outside of the pit limits. Deeper deposits are generally more well suited to underground methods which can leave much of the waste undisturbed by using a variety of methods to support overlying rock and extract material through shafts or declines. Underground mines generally have higher development costs, longer start-up times, and are less preferable to open-pit mines all else being equal [1]. Quarries are distinct from open-pit mining in that a quarry extracts aggregate material or dimensioned stone instead of selectively extracting ore, and strip-mines depart from open-pit mining by operating with larger, shallower footprints and depositing the waste in previously mined panels [2].

Production rates in excess of 100 MT/year are achievable with open-pit mines, where even the largest block caving underground mines rarely exceed 30 MT/year [3]. This high production is possible in part by using larger equipment and many concurrent working areas which are more difficult in confined underground workings which also have to contend with complex ventilation requirements. Open-pit mines are well suited for extracting industrial metals including iron ore and copper, along with precious metals such as gold so long as the ore bodies are near enough to the surface to maintain an operable stripping ratio. The high production rates and lower mining costs allows even low-grade porphyry copper deposits to be economical with open-pit techniques.

An operating open-pit mine has several main features which are in place to accommodate the large equipment and mining operations. The pits themselves are large holes in the ground which are divided into many horizontal benches. These benches typically range in height from 10 to 20 meters which depends on geotechnical considerations, the equipment being used, and the deposit itself. Some benches, which are currently being excavated, are called working benches and must be of a sufficient size to host the necessary equipment. The pit also maintains at least one large haul road and several smaller access roads. These roads provide necessary access to the working benches and a means by which material is transported to the relevant processing facilities or waste dumps using haul trucks.

Open-pit mines generally extract much more overburden and waste than ore which has to be put somewhere. It is important to handle this waste material as little as possible and great care is generally taken to avoid having to re-handle it. The waste, by definition, is not processed for revenue but may require processing to remove deleterious elements or avoid issues associated with acid rock drainage or other concerns. Waste is typically stored in large waste dumps which, at least in the early stages of an open-pit mine, are located outside of the pit limits.

Different ores have different processing requirements that may involve many unit operations including comminution, classification, and concentration. The facilities for these operations may be on the mine site, or accessed by transporting the raw bulk material or intermediate products by rail, large conveyor systems, cargo ships, or other bulk transportation methods. If the facilities are on site their by-products may also have to be stored on site or processed to avoid untenable environmental impacts. Many mineral processing plants generate tailings; small particles which have been separated from the elements of interest and are mixed with water and other processing reagents. These tailings are stored within large tailings storage facilities on site which have to be carefully engineered and located, as moving them later is essentially impossible.

Historically open-pit mine planning was completed by hand, with hand drawn sections and maps. This approach relied on trial and error, tedious hand calculations, and manual smoothing between cross sections [4]. Many decisions during this era of mine planning were based on visual criteria that emphasized operational concerns or rules of thumb. Alternative mine designs were expensive and time consuming to consider so only a very small number of trials or alternatives could be generated.

Following the development of computers and the acceptance of computerized methods for mine planning many of the traditional limitations were lifted. Computers are capable of evaluating many different designs and solving different subproblems that are infeasible to solve by hand. In general, operations research has proven to be one of the most important tools for the modern open-pit mine planning engineer and a great deal of research has been completed in this area [5]. The following sections describe much of the modern, computerized, approach to open-pit mine planning including its objectives, the mine planning process, mathematical modeling, and the most relevant subproblems.

2.1.1 Objectives of Open-Pit Mining Operations

The objective of an open-pit mining operation is to extract and process relevant material in the most cost effective and economic way possible. Generally the most commonly accepted objective is to maximize the net present value (NPV) of the project which accounts for the time value of money and rewards generating profit as early as possible typically by reducing early capital costs and mining high value material quickly. However the NPV of a project is not all-encompassing and modern mining projects are increasingly designed to be developed in a sustainable manner that responsibly considers the social and environmental impact of the operation both locally and globally. Some mines may be developed or operated for reasons other than purely economic; such as for social reasons or as a matter of national security. Even in these circumstances decisions are generally still made based on economic criteria [6].

The net present value of a project is calculated as follows:

$$\text{NPV} = \sum_{t=1}^{n} \frac{R_t}{(1+i)^t} \tag{2.1}$$

R_t is the net cash flow at time t, i is the discount rate, and n is the number of time periods. The discount rate is generally a difficult parameter to define and is often dictated by corporate policy and other concerns. There are valid criticisms of the NPV approach to mine design - especially when the project may have long-term benefits which can be hard to quantify [7]. Additionally it is often common in the mining industry to use a relatively high discount rate such that the denominator in each term of Equation 2.1 grows rapidly and the cash flows in later years

become much less meaningful, which may not be desired.

It is imperative for the open-pit mine plan to operate in line with all relevant legal, environmental, and social requirements. These objectives generally take the form of constraints which necessarily reduce the net present value. Indeed, all secondary objectives can only lead to a decrease in NPV which should always be quantified and understood [8].

Mine planning is a field which is rife with uncertainty. The typical open-pit mine plan samples a minuscule fraction of the deposit prior to mining which can lead to large unknowns with respect to the underlying deposit geometry and grades. The mine planner does not know with a high degree of precision how much ore there is, or its exact makeup. Additionally, the fundamental economic quantities such as commodity price and external costs are uncertain. The value of these parameters are dictated by external markets and may change drastically during the decades that the mine is operating. Some mines are located in countries where political upheaval represents a substantial risk. Addressing uncertainty explicitly, instead of simply being conservative, is preferred [8]. This may involve incorporating risk into the objective, such as using a Risk-adjusted NPV or incorporating a maximum allowable level of risk as a constraint.

2.1.2 Mine Planning Process

The design of open-pit mines is characterized by a cyclical process of assuming, planning, evaluating, refining, and then revisiting the original assumptions. It is not feasible to design an entire open-pit mine plan straight through because there is no way to make a single starting decision that does not depend on later ones.

Consider, for example, trying to define the ultimate extents of the open-pit. These extents are very useful for quantifying the contained resource, and thereby the total potential revenue from the mining operation. Additionally, the extents help inform where to place any supporting infrastructure such as dumps, processing facilities, and tailings ponds. The ultimate extents also give a planner a means to assess the total mined tonnage and thereby the mine life, provided that the mining rate is known. However, it is not possible to determine the optimal ultimate open-pit extents without knowing the mining cost which directly depends on things such as the infrastructure locations, chosen equipment, and so on. Therefore the overall open-pit mine planning process is a circular process which relies on slowly converging on a good mine plan

through repeatedly revisiting earlier decisions and assumptions [9–11].

In general an open-pit mine is planned and designed at three main levels of detail:

- *Long Range Plans* are typically updated yearly or at longer intervals and provide a basis for reporting contained reserves and contain the actionable life-of-mine plan. A long range plan gives a good handle on the mineral inventory and decisions which have a lasting impact, such as where infrastructure is placed and any major operational changes that may require large capital expenses. Long-term price forecasts are considered and a higher degree of uncertainty is accepted. In a long range plan the optimization problems are concerned with much larger volumes at a lower level of precision.

- *Medium Range Plans* are updated at a higher frequency than long range plans and generally dictate the operation on a monthly or quarterly basis. A medium term plan would only be concerned with short-term price forecasts and decisions that impact the more immediate operation such as blending or stockpiling concerns. Medium range plans may look at extraction sequences on a monthly or quarterly level with consideration for equipment allocation or shorter term haul roads.

- *Short Range Plans* consider smaller volumes of material with higher precision and greater frequency. Short range plans may be developed weekly or to dictate a single day's operation. They consider the current state of the mine, the plant, and immediately available blast hole data to decide where material should be routed to maximize profit and maintain appropriate blending. The decisions made at this timescale are irreversible and have immediate impacts on the operation.

The information effect is an important aspect of open-pit mine planning. The long range plans are developed with different data availability than short range plans, as new data is constantly being collected and incorporated throughout the mine life. This new data allows short range plans to operate with less uncertainty, but flexibility is greatly reduced as it is not possible to revisit earlier decisions at this stage.

2.1.3 Open-Pit Mine Modeling

An open-pit mine model is a mathematical representation of a real-world mine developed to aid in decision making and provide insight. These models depart from the real world in many ways to make them more manageable and more useful. Models are designed in conjunction with geologists, geostatisticians, geotechnical engineers, and others. These models are then used for a wide range of tasks described in Section 2.1.4.

In this section we describe the general character of open-pit mine models, how they are constructed, the typical variables present in a workable model and the relevant global parameters. The most fundamental set of geometric constraints to open-pit mine modeling, the precedence constraints, are also described.

2.1.3.1 Model Framework

The primary type of model used in modern open-pit mine planning is the block model. A block model divides the area of interest into a set of non-overlapping volumes, called blocks, which are then populated with a wide range of different attributes. Regular block models are constructed from blocks all of the same shape and size arranged in a regular rectangular grid, however irregular block models are also used which may have blocks of different shapes or sizes.

The regular block model is the most common and consists of a few main parameters.

- The *origin*, which is a point in space defined relative to some datum or at some site specific location. Commonly set to the lowest, leftmost, frontmost point of the model but may be at any of the other 8 corners of the model. The origin is either at the centroid of the block or its outside corner. In Figure 2.1 the origin is denoted with the dot and notated o_x, o_y, o_z.

- The *block size* or, equivalently, the *block spacing*. This parameter defines the shape of each individual block and is chosen to balance a few concerns. The vertical size of the blocks are typically linked to the bench height, on the range of 10 to 20 meters [12], and the horizontal dimensions are generally similar to maintain a roughly cubical aspect ratio. In Figure 2.1 the block sizes are notated s_x, s_y, s_z.

- The *number of blocks* along the 3 coordinate axes. This, along with the block size and origin, controls the extent of the block model. In Figure 2.1 the number of blocks is notated

n_x, n_y, n_z.

- The *block model orientation*, often a single azimuth angle, allows for the model to rotate around the z axis and not be aligned with the underlying coordinate system in order to more closely follow the local geology. Block models are very rarely rotated again around the x or y axes.

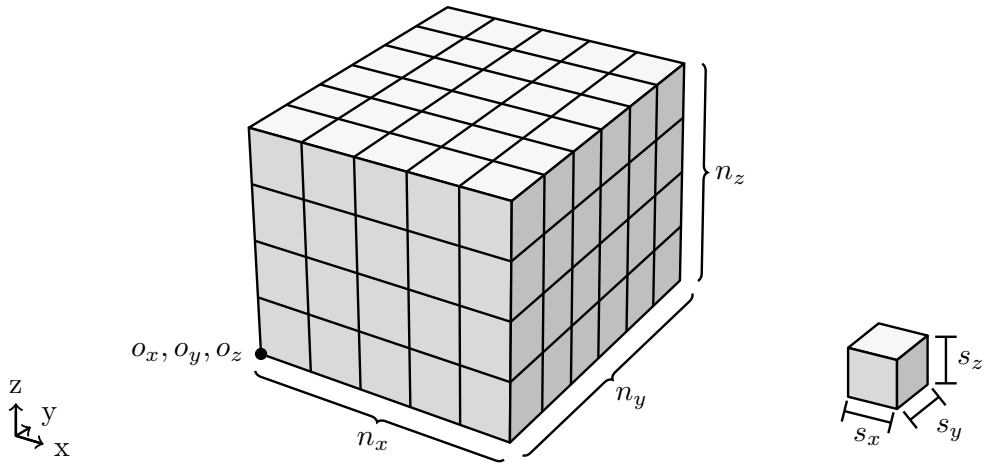


Figure 2.1 The structure and parameters of a regular 3D block model

Each block in a regular block model is often considered a *selective mining unit* or SMU, which corresponds to "The smallest volume of material that can be selectively extracted as ore or waste" [13]. However, in Section 2.4 we see evidence that open-pit mine planners do not typically allow for mine plans to treat single blocks, or even a handful, as extractable. Instead most mine plans require a small contiguous group of blocks to be extracted all as one.

The primary reason that most mines do not scale the blocks themselves larger to avoid this issue is that you lose out on modeling precision, you may have difficulty recreating the precedence constraints discussed in Section 2.1.3.4, and there may be costly aliasing effects. Larger blocks include additional dilution which may significantly impact the valuation of a deposit and mine plan. If that dilution is in excess of the bare minimum dilution required due to a mine's operating parameters, it could lead to under-estimating the value of a deposit and making sub-optimal decisions or incurring substantial opportunity costs.

On the other hand, if a modeler elects to reduce the block size in order to more precisely represent the underlying geology they face two major challenges. The first is that it is much harder to accurately estimate smaller block sizes and the uncertainty inherent in these estimates may not be properly understood or accounted for in downstream applications. The second is more practical; as you reduce the size of each block you must also increase the number of blocks in order to retain coverage over the area of interest which can quickly overwhelm downstream algorithms and procedures. Reducing the size of blocks in half along each axis leads to an eight times increase in the number of blocks required.

Regular block models generally lead to computationally efficient algorithms and are well suited to the procedures in the following sections. For example, regular block models do not need to store every block's individual coordinates as they can be calculated rapidly from the block's indices. If the model is organized such that 0^{th} block is the lowest, frontmost, leftmost block and the blocks are numbered sequentially first along the x direction, then y, and finally z the block's one dimensional index i is given as:

$$i = i_z \cdot n_y \cdot n_x + i_y \cdot n_x + i_x \tag{2.2}$$

Where the number of blocks in the x, y, and z directions are denoted n_x, n_y, n_z respectively. The indices along the x, y, and z directions are similarly denoted i_x, i_y, i_z. The individual indices can be recovered from the one dimensional index i using the following three equations using integer division (truncating the fractional part) and where % is the modulo operator which yields the remainder following division.

$$i_x = i \ \% \ n_x \tag{2.3}$$

$$i_y = \frac{i}{n_x} \ \% \ n_y \tag{2.4}$$

$$i_z = \frac{i}{n_x \cdot n_y} \tag{2.5}$$

Regular block models, with their efficient and consistent structure, are well suited for accurately representing precedence constraints (Section 2.1.3.4), and minimum mining widths (Section 2.4). However, there are downsides to regular block models. They generally include a lot of superfluous information especially around the edges and the lower benches which wastes space

and slows processing. Additionally there may be certain deposit types for which a one size fits all approach is not warranted.

Irregular block models are much more varied and flexible than regular block models and therefore more complex. Sub-blocked models retain rectangular prism shaped blocks but vary the sizes of individual blocks to align with the underlying geometry. For example, large waste regions may be modeled with just a few blocks that take up a lot of space. Then along a geologic contact the blocks are made much smaller to capture the boundary. A vertical cross section through a sub blocked model is included on the left in Figure 2.2. This form of block model is more typically used with underground mines, but still has its place in open-pit mine modeling.

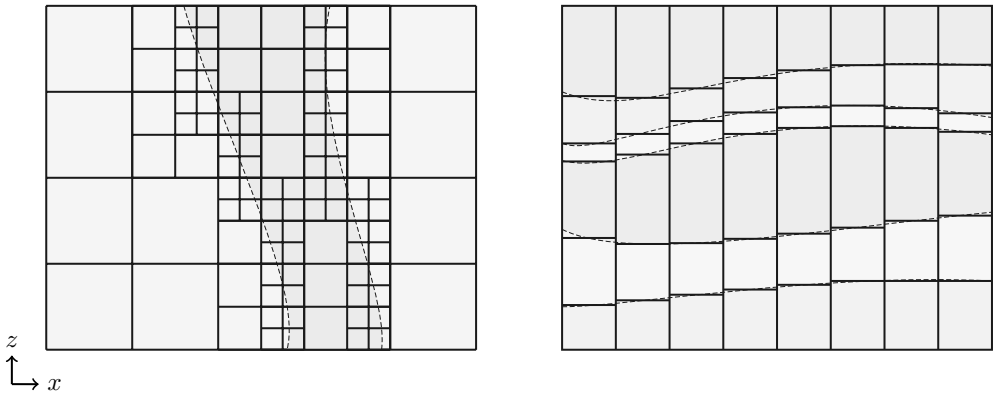


Figure 2.2 Vertical sections through two irregular block model types (left) a sub-blocked model (right) a stratigraphic block model

A second form of irregular block model, the stratigraphic block model, is shown on the right in Figure 2.2. This type of block model is used for stratigraphic ore bodies where there are generally laterally expansive layers where the thickness of each layer is important to capture accurately. There are many other different types of irregular block models including ones which deviate from rectangular prisms and those which cover irregular non rectilinear areas. However, by far the most common model framework is the regular 3D block model.

2.1.3.2 Model Estimation and Simulation

Once the framework, typically a regular 3D block model, is defined geologists and geostatisticians use a wide range of data and techniques to fill that model with the relevant information required for mine planning. Rossi and Deutsch [13], describe the four main

components of the mineral resource estimation process as:

- *Data collection and management*, which includes concerns involving drilling and sampling theory. The truism 'Garbage in Garbage out' is especially relevant during model estimation and simulation as any errors at this stage propagate through all the following stages and lead to erroneous information which can lead to costly mistakes and opportunity costs.

- *Geologic interpretation and modeling*, is the process by which geologic data and mathematical techniques are combined to create realistic geologic domains for downstream steps. The domain model assigns rock types, or similar classifications, to areas within the model and can be used to evaluate contained tonnage and provides the groupings within which the downstream grade estimation and simulation techniques are applied.

- *Grades assignment*, involves identifying the variables of interest and filling them in around and between the drill holes. These are generally the concentrations of different elements, called 'grades', or other continuous properties. However, geometallurgical variables, which may be nonlinear, might also be considered; which necessitates more complicated procedures. The grade assignment process can use many different approaches that have several practical and theoretical challenges which need to be overcome to create a accurate and precise model free of bias or other types of error.

- *Assessing and managing geologic and grade uncertainty.* As previously discussed, there are many sources of uncertainty in the mining process and some of these uncertainties can be evaluated during model estimation. There is uncertainty in the data, the geologic modeling, and within the grade assignment process which must be quantified and considered carefully.

Note that there may be several different models used for different purposes in mine planning all following the above four steps. This is in part due to the information effect discussed in Section 2.1.2. For example, the new data obtained from blast holes may permit a higher resolution model over a smaller area, such as a single blast or bench, that can be used in short term planning.

The data used for model estimation and simulation must be of good quality and free of systematic bias. It must be representative; such that it is spatially spread out and not overly concentrated in one area or biased to specific types of material. This can be complicated by both

the expense of collecting data and the desire of project managers to drill high value holes that provide better sounding press than statistical merit. Mining operations sample a minute fraction of the mineral deposit through different means including; trenches, drill holes; including both reverse circulation and diamond drill holes, and test pits. The sampled material is then processed through a variety of tests and assay procedures to quantify the relevant mineral concentrations.

Site specific data may also be inferred through geophysical methods such as magnetic surveys, electromagnetic methods, or reflection seismology [14]. Geophysical data has different considerations than directly sampled data, but the principles of ensuring quality control, understanding sampling variance, and avoiding errors are the same. In all situations the data must be correctly managed, safely stored, and verified through regular audits as all of the subsequent steps use that data to develop realistic models which are then used for mine planning and design.

The geologic model is responsible for defining estimation domains and dictating how to pool data together into relevant stationary zones. A mining operation is primarily concerned with making decisions to most economically develop the deposit so the geologic model is designed to provide the necessary information for this task. Domains identified in a geologic model may be based on both alteration and structural geologic variables depending on the mineral deposit. Stationarity refers to only grouping variables of like statistical and geologic properties together. Combining data from different statistical populations leads to poorer quality estimates.

There are several techniques used to develop geologic models including; manual interpretation on sections, indicator Kriging methods, implicit modeling, machine learning approaches and others [15]. Each technique has different advantages, disadvantages, and areas of applicability. In all cases the mixing of different mineralization types should be strictly minimized as this leads to poor quality estimates that may skew resulting tonnage and minerals concentration estimates. It is important to consider uncertainty within the geologic model as the boundaries are never sufficiently sampled to give an absolutely precise delineation between zones. Some methods can consider uncertainty directly; as is the case with indicator simulation or using the so-called 'uncertainty parameter' within implicit modeling, however others may require additional effort. This information has to be accurately communicated to the downstream mine planning engineer and other model users.

Once a domain model has been developed, audited, and verified it is necessary to estimate mineral concentrations and other variables of interest within the identified domains. The variables in mineral deposits exhibit spatial dependence because they are formed through natural processes following well defined, if not completely understood, rules [16]. It is valuable to describe this spatial dependence, typically through a variogram or similar measure, in order to inform the estimation process. Earlier, less sophisticated, estimation techniques, such as nearest neighbor or inverse distance, do not incorporate a variogram directly.

Kriging is a method for estimating continuous variables using a direct model of spatial variability that minimizes the expected error variance [13]. There are several types of Kriging with varied levels of applicability to different models including simple Kriging, ordinary Kriging, Kriging with a trend or external drift model. Each approach is selected based on the geologic and statistical nature of the domain, data availability and the variable of interest [17]. Regardless of the method selected, modelers must justify their choice and perform various checks including cross validation, and visual and statistical verification. Additionally models should be calibrated to other data sources such as production data if possible.

Of extreme importance to the entire modeling process is the understanding that the models are subsequently used to make specific mine planning decisions. The technical nature of the model and the different considerations must be accurately transmitted to the downstream consumers of the models, including considerations such as how dilution has been handled and how to use each variable in the model.

As introduced in Section 2.1.1, uncertainty is inherent in the mine planning and geologic modeling process. A typical mining operation only samples a minuscule fraction of the mineral deposit before development. The estimation techniques introduced above provide a single, hopefully high quality, estimate for the unsampled locations but this averaging process does not explicitly consider the geologic uncertainty. Simulation is a process by which multiple geologic realizations can be generated from the input data and, when taken as a whole, can aid in understanding. One major advantage of simulation is demonstrated in Figure 2.3, where we see how multiple realizations can be used to create multiple plans which, when taken as an ensemble, provide information on multiple possible outcomes.

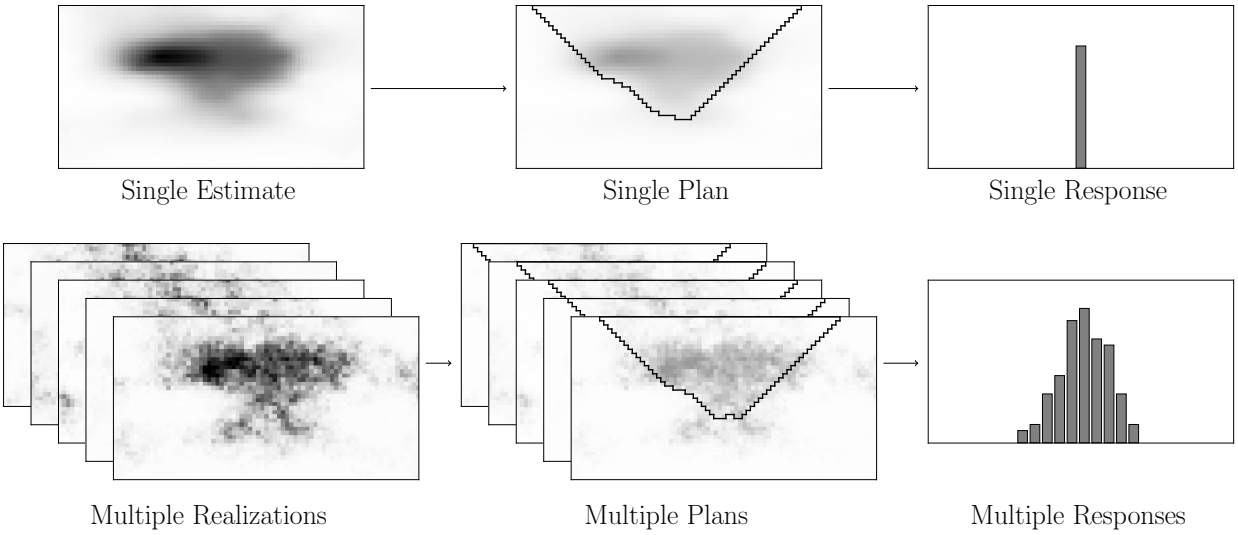


Figure 2.3 Applying mine planning to estimated and simulated models. A single response provides no information on uncertainty. Multiple realizations can provide more information

A growing challenge in mine planning is to properly account for the geologic uncertainty. However, the workflow implied by Figure 2.3 is not the ultimate ideal. Instead of determining the best mine plan for a specific realization or even the best plans for multiple realizations it is preferred to create the best mine plan in the presence of uncertainty.

2.1.3.3 Typical Model Variables and Global Parameters

An open-pit mining operation is planned and developed in response to a wide range of model variables and global parameters. A model variable is a metric that varies by location throughout the mining area and is typically stored within the block model described earlier. Parameters do not vary by location however they may vary by time. All inputs including model variables, global parameters, and other information are uncertain and considered as either estimates or as a part of some potentially unknown distribution.

At minimum a geologic model for mine planning consists of a usable rock type model, density, relevant mineral concentrations, and a resource classification. More sophisticated models may include additional variables such as geometallurgical indices, or additional information that can help guide the mine planning process.

The most impactful global parameter on the open-pit mine planning process is the commodity price [8]. This parameter can singlehandedly make or break a mining operation and

fundamentally alters the way that the mine operates including dictating the optimal production rate and the final pit limits. Unfortunately the commodity price is also one of the most uncertain parameters and depends on many factors outside of the mining operation's control. To combat this uncertainty mines may negotiate longer term contracts or use a variety of other economic and financial means to try and avoid disruptive price drops and still be able to take advantage of price increases.

The planning engineer typically designs multiple plans and schedules based around at least three different price forecasts: a middling price forecast which aligns with the general expectation, a low price forecast which is pessimistic, and a high price optimistic forecast. In certain circumstances many more price forecasts can be considered which are generated using a simulation process or a method akin to random walks as shown in Figure 2.4.

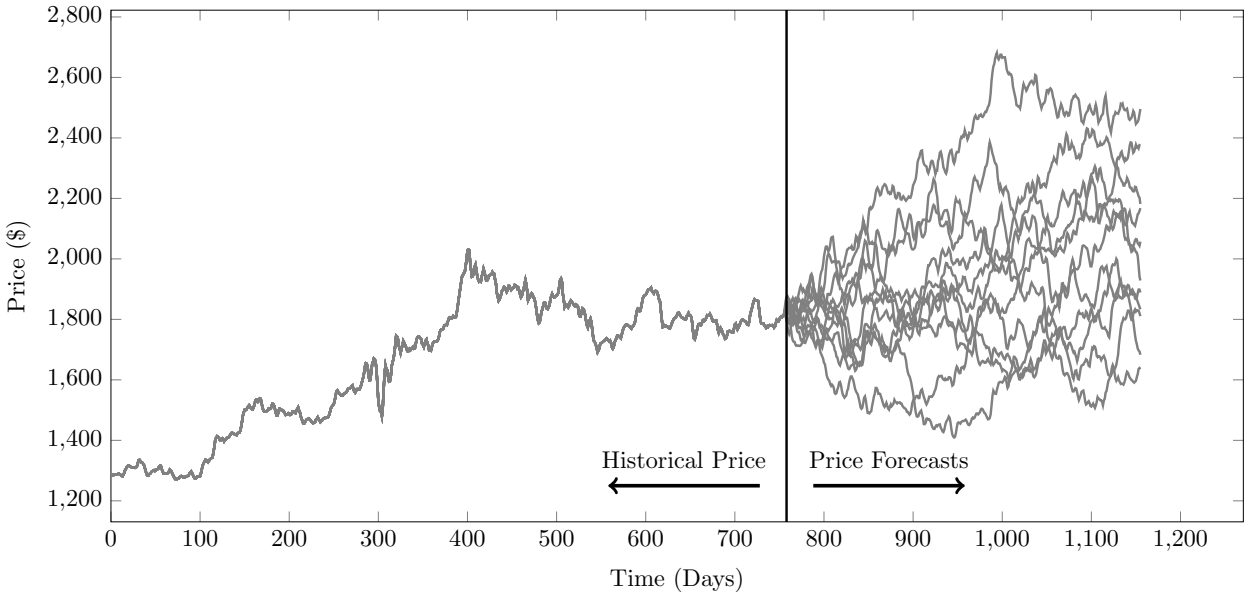


Figure 2.4 Example price forecasts for mine planning generated by using random walks

These variables and global parameters are drivers for some the most important design parameters of an open-pit mine planning operation [8] including:

- The total *product produced* which is an upper bound on any potential revenue from processing a block and directly informs material routing and valuation. For example in a gold mining operation the gold grade (typically measured in grams per tonne) along with

density and the volume of the block gives an upper bound on the potential product available for recovery. No current processing methodology can achieve 100% recovery of the minerals of interest, and different element interactions can be quite complicated.

- The *mining costs* include costs associated with drilling, blasting, loading, and hauling the material. Mining costs are applied on a dollar per ton basis and varies by depth, and distance to the relevant processing facility or waste dump.

- The *processing costs* are applied to material which is processed in, for example, a processing plant or a leach pad. Processing costs have to include the cost of any reagents used, energy consumed, and tailings rehabilitation.

- The *material destination*, or material classification, is the selected process stream for the mined material. A given mine may have many different possible material classifications including different streams for waste and ore of different grades and properties.

- The *economic block value*, or EBV, is a combination of the above design parameters which indicates the, generally undiscounted, value of a block if it were mined and sent to the appropriate material destination. This parameter may be provided on a per destination basis, or assuming that the material can be processed via the most economic process.

- The *pit slopes* dictate the precedence constraints described in the following section and vary based on rock type, and direction. Large scale structural geology features, such as faults and joint sets, have a strong impact on the allowable pit slopes.

The economic block value is of utmost importance in the following chapters. At its core economic block value is defined as revenues less costs. A simple economic block value formula may have the following form:

$$v_b = T_o \cdot g \cdot r \cdot P - T_o \cdot PC - T \cdot MC \tag{2.6}$$

Where v_b is the economic block value for block b. T_o is the ore tonnage, T is the total tonnage, g is the grade, r is the recovery as a proportion of recovered material between 0 and 1. P is the commodity price, PC is the processing cost in dollars per ton of ore, and MC is the mining cost.

In practice a much more complicated procedure is often used to calculate economic block value based on a wide range of additional local and global variables and parameters [18, 19].

In Chapters 3 and 4 the economic block value variable and pit slopes are used. Chapter 5 requires additional variables including the discount rate, material classification, and others.

2.1.3.4 Precedence Constraints

Unlike underground mines, which use a variety of techniques to support the material above the deposit, open-pit mines progress downwards, removing successive benches of material in a cone-like shape. The slope of the cone is based on the strength of the material being mined, its geotechnical properties, and other structural geologic factors. This slope generally varies by both location and direction throughout the mining area as the composition of the material changes and certain geologic features are more prone to different failure modes in different directions. Geotechnical engineers are responsible for using field data and sophisticated modeling techniques to determine safe pit slopes that protect individuals, equipment, and prevent slides that could lead to injury, expensive re-handling, and other difficulties.

The geotechnical understanding of the area is used to create precedence constraints, which are used in combination with the variables described in Section 2.1.3.3 as inputs for the techniques in Section 2.1.4. Precedence constraints encode the physical relationships between blocks and indicate that 'this block cannot be mined, until all of these other blocks are mined.' They are generally modeled as pairwise relationships between blocks and denoted with a directed arc from the lower, base block to the higher block. Precedence relationships are common across any optimization or scheduling technique that operates on an open-pit mine and must be constructed correctly and efficiently to avoid unsafe deviations from the geotechnical model.

Lerchs and Grossmann proposed two precedence schemes in their seminal 1965 paper on the ultimate pit problem [20]. The first scheme uses an irregular model where the odd layers are offset from the even layers by half of the block size; which is an uncommon block model configuration. The second precedence scheme, commonly now called the 1:5 pattern, connects each block to the five blocks immediately above it in a cross pattern which is locally precise but globally inaccurate. When the 1:5 pattern is applied with larger block models and over several benches it creates diamond shaped pits that deviate substantially from 45° in the off axis directions.

To account for the deviations from 45° in the off axis directions, Gilbert in 1966 developed the 1:5:9 pattern which varies the precedence scheme by level [21]. The 1:5:9 pattern alternates between connecting every block to the five above it in a cross pattern and connecting every block to the nine above it in a square. Lipkewich and Borgman introduced the "Knight's move" pattern in 1969 which connects each base block to the five immediately above it in a cross and eight more blocks two benches above that are offset by a knight's move as in chess [22]. Note that both of these patterns only approximates 45° when the block model has isometric block sizes. These small precedence patterns are shown in Figure 2.5.

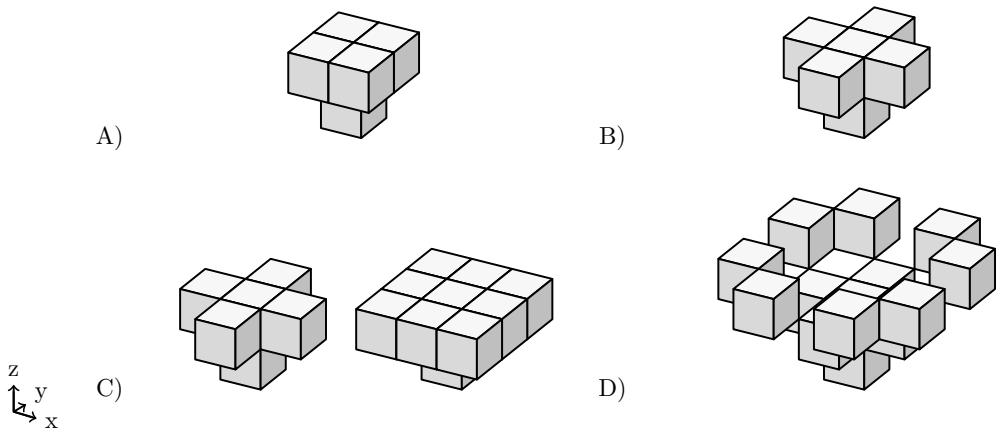


Figure 2.5 Small precedence schemes. (A, B) from Lerchs and Grossmann, (C) the "1:5:9" scheme from Gilbert, (D) The "Knight's Move" scheme from Lipkewich and Borgman [20–22]

Chen, in 1976, apply variable slope angles and varying cone templates across the deposit to more faithfully recreate the geometrical and operational constraints [23]. Pit slopes that vary by direction require some form of interpolation between the specified directions. Researchers have used splines, ellipsoids, and inverse distance interpolation for this purpose [24–26].

Precedence schemes must be generated so that they realistically reflect the geometrical constraints and such that they are computationally tractable and efficient. The techniques in subsequent chapters are sensitive to the problem size, so generating unnecessary precedence constraints should be avoided. The "Minimum search patterns" from Caccetta and Giannini are designed to accurately recreate arbitrary slope requirements and consist of the fewest constraints possible [24]. Their approach is included in Algorithm 1, and is discussed further in Section 3.2.2 alongside an efficient and open-source implementation.

Algorithm 1: Algorithm to generate minimum search patterns, adapted from Caccetta and Giannini 1988[24]

Input : A list of azimuth slope pairs A: as a list of pairs defining the corresponding azimuth and pit slope.

Input : The maximum number of benches in the output pattern N_z. Note: this is generally less than the number of benches in the input model.

Input : The block dimensions: s_x, s_y, s_z

Output: The minimum search pattern S: as a list of 3-tuple offsets in the x, y, and z directions.

```
// Maintain a list of tagged blocks, as offsets
```
$T \leftarrow \emptyset$;

for $l \leftarrow 1$ **to** N_z **do**
    for *all untagged blocks on this level which violate the slope constraints* **do**
        $S \leftarrow S \cup \{\text{this block}\}$;
        $T \leftarrow T \cup \{\text{this block}\}$;
    for *all tagged blocks* **do**
        Apply current search pattern to this block;

2.1.4 Open-Pit Planning Subproblems

An open-pit mine plan consists of several different studies at different levels of detail all with different individual subproblems. The scope of a particular study may allow for large strategic changes to the mine's development or instead operate within a pre-defined strategy which was chosen based on an earlier study. The subproblems present in the open-pit planning process are applied as necessary to answer specific questions and inform subsequent decision making [27, 28] Some of the most important subproblems in the open-pit mine planning process are listed below.

- The *ultimate pit problem* was originally described in 1965 by Lerchs and Grossmann [20] and is concerned with determining the final limits of an open-pit mining operation such that mining any more material would require removing so much waste that it would be uneconomic. The majority of this dissertation is devoted to this subproblem and it is discussed in Section 2.2 in detail and further in Chapters 3 and 4.

- *Pit parameterization* is the process by which many ultimate pits are constructed for further analysis originally found in Lerchs and Grossmann's seminal paper and further described by Matheron in 1975 [29]. There are several techniques for pit parameterization including using a revenue factor, cost factor, or a constant decrement on block values among others

[11, 30, 31]. Although pit parameterization can give some indication of an overall sequence, it does not typically divide the material into yearly volumes or handle other relevant concerns. Pit parameterization is discussed in 2.3.1, and a small extension is presented in Appendix B.

- *Block scheduling* is an extension to pit parameterization in that the output is still a set of nested production volumes, however, the optimization process and relevant parameters are substantially more complex. A production schedule directly considers the time value of money along with blending, stockpiling, and other concerns. This subproblem is discussed in more detail in Section 2.3 and extended to consider operational constraints in Chapter 5.

- *Cut off grade optimization* is used to determine the best cutoffs to discriminate between ore and waste during the scheduling process. Using conventional break-even cutoff grades does not directly optimize NPV and instead maximizes undiscounted cash flows [32]. For this reason a higher cutoff grade is commonly used early in the process which is lowered to the break-even cutoff grade over time [33, 34].

- *Grade control polygon optimization* is applied in short term planning on a daily or weekly basis to define operable mining polygons to assign material to the different processes [35–38]. This optimization problem is directly related to the problems discussed in this dissertation, in that it requires directly handling minimum mining widths, although it falls outside the scope of this work.

Grade control polygon optimization is a subproblem considered in the short term planning space with smaller volumes of material, typically a small portion of a bench that corresponds to a single blast. A block model containing economic block value variables for each possible destination is constructed, as shown on the left in Figure 2.6. The reclassified model, on the right, is constructed to reclassify the blocks to satisfy minimum mining width constraints and maximize economic value.

This section provided an overview of just a few of the problems typically faced by the open-pit mine planning engineer. There are many additional subproblems in open-pit mine planning which can vary from deposit to deposit and mine to mine such as optimizing the haulage network and

infrastructure placement. The following sections discuss two of the fundamental subproblems in more detail which are the focus of this dissertation: The ultimate pit problem, and the block scheduling problem.

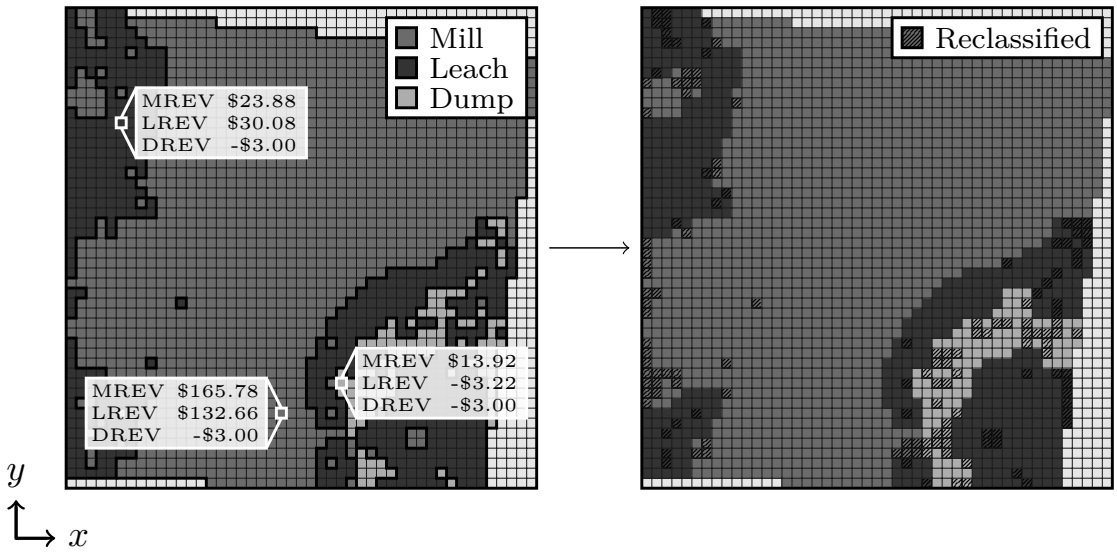


Figure 2.6 The grade control polygon optimization problem takes a bench classification (left) and creates an operable set of polygons (right). The operable polygons satisfy minimum mining width constraints and maximize undiscounted cash flow

2.2 The Ultimate Pit Problem

The ultimate pit problem, [20], is an important subproblem in the open-pit mine planning process. Its narrow focus allows it to be applied to large models and give initial results that can be used to inform infrastructure placement, assist in equipment selection, perform parametric analysis, and guide many other aspects of designing and scheduling a surface mining operation [39–42]. The ultimate pit problem possesses a unique mathematical structure which makes it amenable to several different approaches including very efficient methods from network optimization. Specifically the problem can be cast as a network flow problem which is the current best approach for large models with dozens or hundreds of millions of blocks.

Graphically the ultimate pit problem is shown in Figure 2.7 for a single vertical section through a synthetic ore body. The economic block values and the precedence constraints are used to identify the blocks which together maximize the undiscounted value and satisfy the precedence constraints. This subset of blocks is called the ultimate pit and is not only used in the context of

determining when a mining operation should cease production but also in informing many engineering decisions in the mine planning process, generating outer bounds for scheduling, and several other contexts.

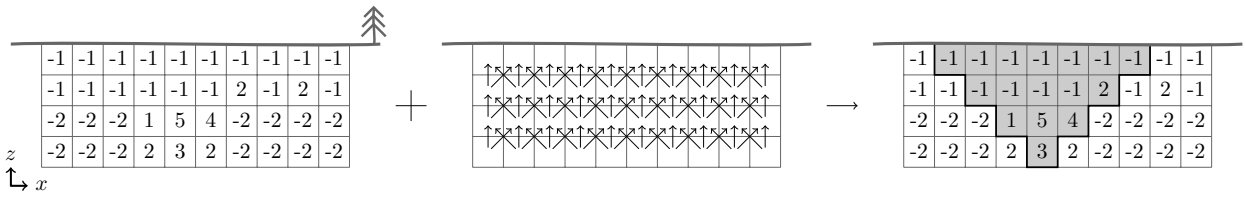


Figure 2.7 The ultimate pit problem requires an economic block value model, (left) and precedence constraints, as directed arcs between blocks, (middle) in order to identify the ultimate pit limits (right)

In practice the ultimate pit can be calculated rapidly for very large 3D models. The example 3D ultimate pit shown in Figure 2.8 is for a relatively small model with 374,400 input blocks and 7,116,016 precedence constraints however even models with hundreds of millions of blocks and billions of precedence constraints are attainable. This is not unrealistic; as many modern open-pit mining complexes use very large models with smaller blocks for a variety of reasons [43].

The remainder of this section describes three formulations for the ultimate pit problem including the original network based formulation, the linear programming formulation, and the max-flow / min-cut formulation. Then four methods for solving the ultimate pit problem are described: The Lerchs & Grossmann algorithm, the heuristic floating / moving cone algorithm, the pseudoflow algorithm, and the special case 2D dynamic programming algorithm. Finally we present a brief survey of alternative methods which are less commonly used.

2.2.1 Original Network Formulation

In 1965, Lerchs and Grossmann originally described the ultimate pit problem in the context of a custom network algorithm for determining the smallest maximum valued closure of a weighted directed network [20]. A network, also called a graph, is a structure for modeling pairwise relationships between objects. Networks are extremely useful structures for all kinds of real world problems and are commonly found in many different areas in open-pit mine optimization and planning.

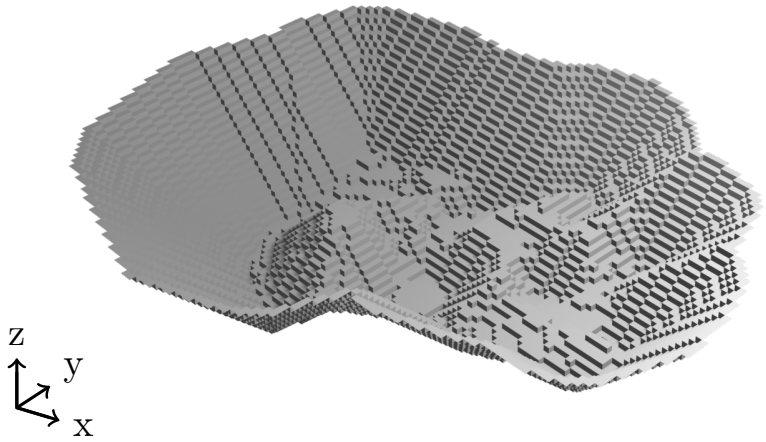


Figure 2.8 An example 3D ultimate pit calculated from a 374,400 block model

The *nodes* of the network represent some object and are connected in pairwise relationships via *directed* or *undirected arcs*. A directed arc is used when there is some order to the nodes, or a specific orientation inherent in the relationship. The beginning node is called the *tail*, and the ending node is called the *head*. Undirected arcs do not have any order but are merely used to indicate that the relationship exists. A *weighted* network may have weights associated with nodes, arcs, or both. The weights are some scalar value that could have many different meanings depending on the context.

In Lerchs and Grossmann's description of the ultimate pit problem blocks are represented by weighted nodes with the weight being the economic block value. Blocks are then connected by directed arcs from lower blocks to upper blocks following the precedence constraints. The resulting network has no cycles by construction and therefore is classified as a *directed acyclic network*. A *closure* of a directed acyclic network is a set of nodes such that there are no arcs with their tail inside the closure and their head outside. All closures of this network are valid pits and do not violate any precedence constraints. The ultimate pit is the smallest maximum valued closure and Lerchs and Grossmann developed an algorithm for identifying this closure, described in Section 2.2.4.

Figure 2.9 shows an example network that corresponds to a very small example ultimate pit problem. This model consists of only 12 blocks which are not arranged in the typical rectilinear grid, instead they each only have two precedence constraints indicated by the directed arcs. The

marked closure is the maximum valued closure and corresponds to the ultimate pit with a total contained value of 2 units.

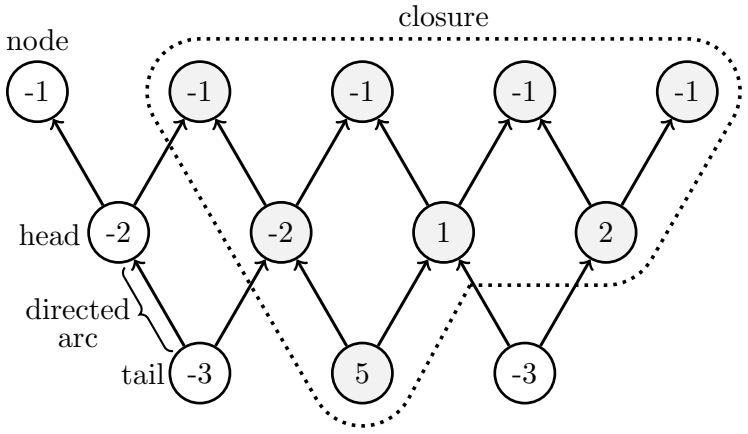


Figure 2.9 Example weighted directed acyclic network with labeled maximum valued closure

A few more definitions are required to properly describe the Lerchs and Grossmann algorithm and network flow approaches. *Paths* are sequences of arcs such that each subsequent arc shares a common node. In many situations paths are constrained such that they can only go through nodes and arcs at most once and they may be restricted to only travel along directed arcs from tail to head. Paths in networks with undirected arcs do not have this restriction. In directed networks where the sequence of arcs does not have any orientation restrictions this is sometimes called a *chain*. A network such that any two nodes are connected by exactly one path is called a *tree*. The tree may contain a special node designated a *root* from which there could be many *subtrees* or *branches*, which are themselves trees when 'detached' from the root.

2.2.2 Linear Programming Formulation

The linear programming formulation for the ultimate pit problem is useful theoretically and as a means of communicating the problem clearly. Ultimate pit solvers do not typically use this formulation directly with the simplex algorithm or another general purpose linear programming solver; but it is still useful and is strongly related to the subsequent max-flow / min-cut formulation in Section 2.2.3. Linear programming is discussed in more detail in Section 2.5.2.

The ultimate pit problem as a linear program is defined as follows [44–46]:

Sets:

- $b \in \mathcal{B}$, the set of all blocks. Commonly based on a regular block model. See Section 2.1.3.1.
- $\hat{b} \in \hat{\mathcal{B}}_b$, the set of *antecedent* blocks that must be mined if block b is to be mined in order to honor pit slope requirements. These sets are derived from the precedence constraints described in Section 2.1.3.4.

Parameters:

- v_b, the economic block value of block b. See Section 2.1.3.3.

Variables:

- X_b, the proportion of block b which is mined in the ultimate pit

The Ultimate Pit Problem:

$$\text{maximize} \quad \sum_{b \in \mathcal{B}} v_b X_b \tag{2.7}$$

$$\text{s.t.} \quad X_b - X_{\hat{b}} \leq 0 \qquad \forall b \in \mathcal{B}, \hat{b} \in \hat{\mathcal{B}}_b \tag{2.8}$$

$$0 \leq X_b \leq 1 \qquad \forall b \in \mathcal{B} \tag{2.9}$$

Equation 2.7 maximizes the total contained value of ore and waste blocks mined within the ultimate pit. Equation 2.8 enforces precedence constraints. Note that for any block b to attain a value of 1 all of its antecedent blocks $\hat{b}$ must already have a value of 1 or else this constraint would be violated. Equation 2.9 enforces bounds on the main variable to disallow mining a negative proportion or more material than is present.

Importantly it is not necessary to restrict the variable X_b to only integral values. This is inherently satisfied because the precedence constraints form a totally unimodular system [47].

2.2.3 Min Cut Formulation

In 1968 Johnson showed the ultimate pit problem can be formulated as a max-flow / min-cut network problem [44]. This reformulation was also demonstrated by Picard in 1976 alongside further mathematical justifications [48]. The construction is based on taking the dual of the linear

programming formulation in the previous section and performing some appropriate manipulations. Johnson recommended Ford and Fulkerson's labeling algorithm to solve the max-flow / min-cut formulation of the ultimate pit problem. This formulation gives rise to the currently fastest known methods of solving the ultimate pit problem; specifically Hochbaum's pseudoflow algorithm [46].

A flow network differs from other networks in a few respects. First two nodes are identified as the *source*, denoted S, and the *sink*, denoted T. Each directed arc is then capable of carrying some non negative integer amount of *flow* from its tail to its head up to some specified capacity. Each node, other than the source and the sink, are required to satisfy a *flow balance* constraint such that the inflow is equal to the outflow. Flow networks are used for many purposes such as modeling traffic on roads, fluids flowing through pipes, power flowing along electrical grids, and others [49]. Ford and Fulkerson developed much of the initial theory on flow networks including the important max-flow / min-cut theorem [50].

Each well-formed flow network generally has many possible valid flows, which are assignments of flow values to each arc in the network. As previously mentioned these flows are constrained such that the flow on an arc cannot exceed the arc's capacity, and also such that for every node (except the source and sink) the total inflow into the node is equal to the total outflow from the node. Each flow network has a *max flow* which is an assignment of flow such that the outflow from the sink (and thereby the inflow into the source) is maximized. There are many procedures for determining the max flow including augmenting paths [50, 51], the push-relabel algorithm [52, 53], and the pseudoflow algorithm [46, 54, 55].

Also in any given flow network there are many ways to *cut* the network into two pieces. This cut is accomplished by removing a set of arcs, called the *cut-set*. If, once the cut-set arcs are removed, the flow network is divided neatly into two pieces with the source on one side, the sink on the other, and such that there are no paths along directed arcs from the source to the sink this cut is called an *s-t cut*. In an *s-t* cut it is only required to remove arcs which are directed from the source side to the sink side and any arcs that are directed from the sink side to the source side are not a part of the *s-t* cut. The capacity of the *s-t* cut is the sum of the arc's capacity in its cut-set. The *s-t* cut which corresponds to the minimum capacity is called the *minimum cut* of the flow network.

Ford and Fulkerson's max-flow / min-cut theorem states that the maximum flow of a flow network is equal to the minimum cut of the same flow network. Intuitively this is because the arcs which are a part of the minimum cut's cut-set form the 'bottleneck' of the flow network. There is no way to sneak more flow through the cut-set without increasing its capacity. The formal proof of this theorem proves that if the maximum flow did not equal the minimum cut there would be a contradiction [50].

For our purposes, in the mining industry, this is a very useful result because using Johnson's construction our desired ultimate pit directly corresponds to the source side of the minimum cut. If one is able to find the maximum flow through some means then the minimum cut can be extracted, and thereby the ultimate pit. Or, if an algorithm gives the minimum cut directly this also immediately reveals the ultimate pit.

Practically a given ultimate pit problem is transformed by representing each block as a node in the network, however unlike Lerchs and Grossmann's network formulation the nodes do not have any associated weight. The source and sink nodes are added, and each node corresponding to a positively valued block is connected by a directed arc *from the source* with a capacity equal to the economic value. Nodes with negative values are connected with a directed arc *towards the sink* with a capacity equal to the absolute value of their economic value. Nodes that correspond to blocks with zero economic value do not need to be connected to the source or the sink. Finally, each precedence constraint is incorporated by including a directed arc between the lower and higher block with infinite capacity. An example transformation of an ultimate pit problem is shown in Figure 2.10.

In Figure 2.10 the capacity of each arc in the flow network is the denominator of the fraction shown on each arc. The flow, which in this case corresponds to the maximum flow, is the numerator. The minimum cut is represented by the dashed line which goes through the four bolded arcs, the sum of the capacities of the arcs in this set is 9 which is equal to the maximum flow. The difference between this value and the sum of the capacities of the source adjacent arcs is the ultimate pit value. Note that the nodes on the source side of the minimum cut, which are colored darker, are exactly the ultimate pit.

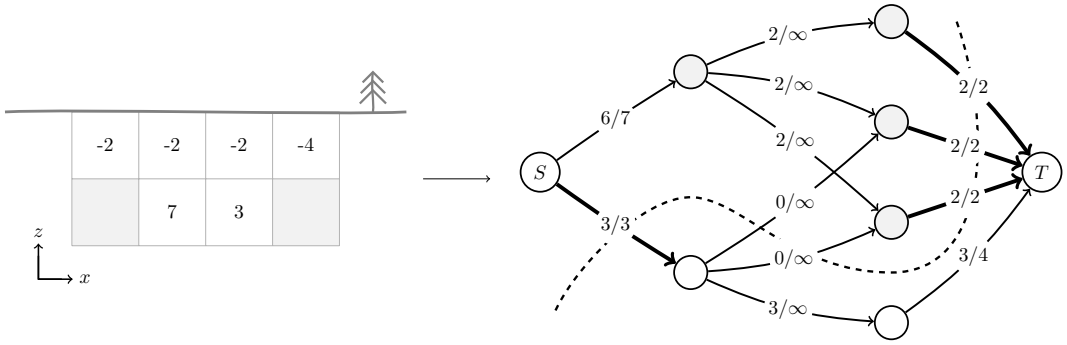


Figure 2.10 Example transformation of an ultimate pit problem (left) into source-sink form for solving with a max-flow or min-cut algorithm. Figure adapted with permission from Deutsch, Dağdelen, and Johnson 2022 [56]

2.2.4 The Lerchs & Grossmann Algorithm

In their seminal 1965 paper, Lerchs and Grossmann proposed two approaches to solving the ultimate pit problem [20]. The simpler approach based on dynamic programming which is only applicable to two dimensional cross sections is described in Section 2.2.7. The more general approach, based on network theory, is reviewed here.

The Lerchs and Grossmann algorithm is an iterative algorithm which identifies the maximum closure of a directed network. It begins with a infeasible solution and iterates through primal infeasible solutions until it identifies the first primal feasible solution which is the optimal solution. The solution at each stage is the current collection of *strong* nodes, which begins as the set of all positive blocks and ends as the set corresponding to the ultimate pit.

Instead of operating with the original directed network which contains all of the precedence constraints, the Lerchs and Grossmann algorithm uses an augmented network which contains a special artificial root node which is originally connected to every node with a directed arc beginning at the artificial root. The augmented network is a tree at initialization and remains a tree throughout the entire process. Nodes within the augmented network are denoted as strong or weak depending on the sum of all the associated block values within their branch. Initially positive nodes are classified as strong, and non-positive nodes are classified as weak. This initialization step is shown in Figure 2.11, the example ultimate pit problem (left) is transformed into the augmented network (right) with each node connected to the artificial root. The strong branches (colored darker) form the initial solution.

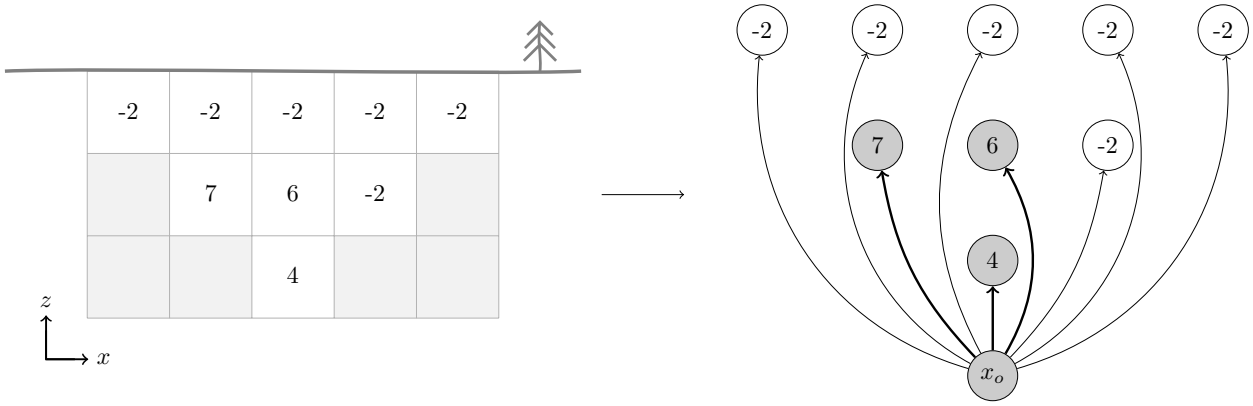


Figure 2.11 The initialization step for the Lerchs and Grossmann algorithm.

In addition to classifying nodes based on strength, arcs within the augmented network are classified based on their direction and strength. The direction classification of an arc depends on its direction relative to the chain originating at the artificial root and passing through the arc. If the arc is pointing toward the root it is considered a *m-arc* or *minus-arc* and if it is pointing away from the root it is a *p-arc* or a *plus-arc*. The strength of an arc depends on the net value of the nodes which are supported by the arc, this is called the *mass* of the branch. If the mass supported by an arc is positive it is a strong arc, and if non-positive it is a weak arc. The augmented network is said to be normalized if and only if all strong arcs are adjacent to the artificial root.

The two main steps of the Lerchs and Grossmann algorithm are to first 'move toward feasibility' and then 'normalize' the resulting tree. These two steps are repeated until the move toward feasibility is no longer possible at which point the ultimate pit is identified. In the linear programming context the move toward feasibility corresponds to a change of basis on the path toward primal feasibility. The normalization step is to fix the current iteration when it strays from dual feasibility [57].

In practice the Lerchs and Grossmann algorithm continuously scans over all of the precedence arcs and once an arc is identified that has a 'strong' tail and a 'weak' head it is selected for the move toward feasibility. In the move toward feasibility the offending precedence arc is added to the augmented network, which creates a cycle which has to be broken so that the augmented network can remain a tree. The Lerchs and Grossmann algorithm breaks the cycle by removing the arc which is adjacent to the artificial root. If, following the introduction of the offending

precedence arc, there are any arcs which are strong p-arcs that are not adjacent to the artificial root the normalize procedure must be used to prepare the tree for the next move toward feasibility.

A strong p-arc in the non-normalized tree is replaced with the artificial arc between the artificial root and the head of the p-arc. A strong m-arc is replaced with the arc between the artificial root and the tail of the m-arc. In effect this is to guard against inappropriate allocations of values between the branches of the tree. For example, a strong p-arc implies that a node farther along the branch is using its value to pay for blocks that are not within its cone of extraction. This is avoided by the normalization step as shown in Figure 2.12.

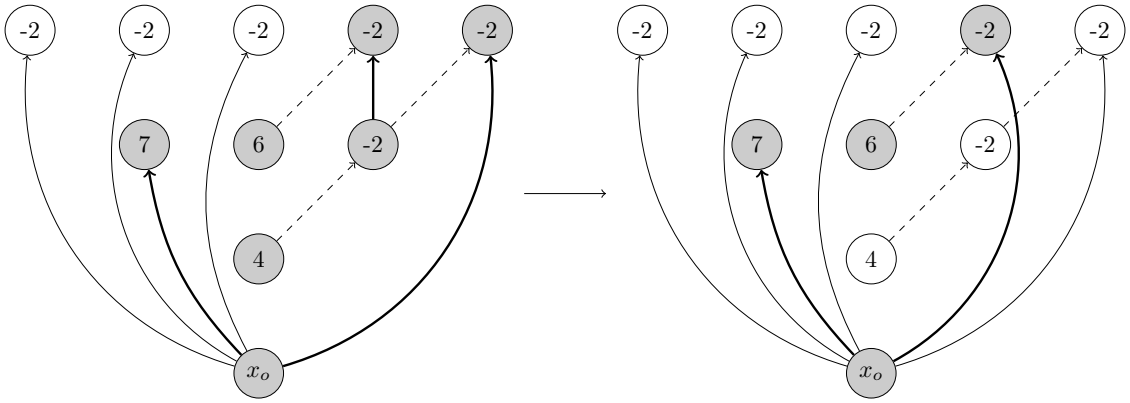


Figure 2.12 The normalization step in the Lerchs and Grossmann algorithm transforms the non normalized tree (left) to the normalized tree (right) by replacing any non root adjacent strong arcs.

The Lerchs and Grossmann algorithm terminates when there are no possible moves toward feasibility at which point the remaining strong arcs are the smallest maximum valued closure of the directed network: the ultimate pit. Practically this may require many complete scans over the entire precedence network, along with many moves toward feasibility and normalization steps. The performance of the Lerchs and Grossmann algorithm is not very good when the number of nodes and precedence constraints is large. Heuristic methods, primarily the floating cone method in the following section, were preferred for many years following 1965 until a strong commercial implementation of the Lerchs and Grossmann algorithm was developed by Whittle [34]. In recent years the Pseudoflow algorithm is the fastest approach to the ultimate pit problem.

2.2.5 Floating Cone Methods

The floating cone method was originally described by Pana in 1965 in the context of simulating the mining process in order to determine both the ultimate pit limits and the optimal mining sequence [58, 59]. Pana, and members of the Systems and Data Processing Division at Kennecott Copper Corporation, developed a system whereby estimated geologic attributes were transformed into economic block values, coded on punch cards, and the mining sequence was determined by evaluating many hundreds or thousands of 'frustums,' or cones, of material. An open-pit mine can be thought of as a set of intersecting cones which together define the ultimate pit limits. Pana described an approach which later became known as the 'floating cone' because of the way one can visualize an inverted cone floating from block to block and either mining that entire cone or not.

Floating cone algorithms are iterative in nature and work to define the ultimate pit limits by successively visiting different possible cone bottom locations and deciding to extract that cone based on the net contained value. That is, if a cone contains blocks that together have a positive economic value then it is extracted and if non-positive it is left in place. The mining engineer defines the cones in the same manner as precedence constraints, with the added flexibility that they can very easily define minimum bottom widths by simply requiring the cone to consist of multiple blocks at the bottom. Floating cone algorithms terminate once there are no remaining cones containing a positive value.

The fundamental issue with floating cone methods is that they do not correctly consider the contribution of multiple cones at one time. This can lead to both overmining: wherein a larger cone than necessary is mined that includes a subset of material with net non-positive economic value that does not need to be mined, and undermining: wherein the floating cone method is unable to identify a situation whereby two or more cones could 'share' the cost of extracting negative valued material and end up net positive [27, 60–62].

The example in Figure 2.13 demonstrates both overmining and undermining, on the left and right respectively. The true ultimate pit in both models (hatched section) has a value of 2 and relies on sharing the top middle waste block between the two ore blocks. However when the floating cone algorithm is applied to the model on the left it creates the too large pit that

incorrectly mines two extra blocks, and when the floating cone algorithm is applied to the model on the right it is unable to find any economic cone to extract and terminates with the 'mine-nothing' solution.

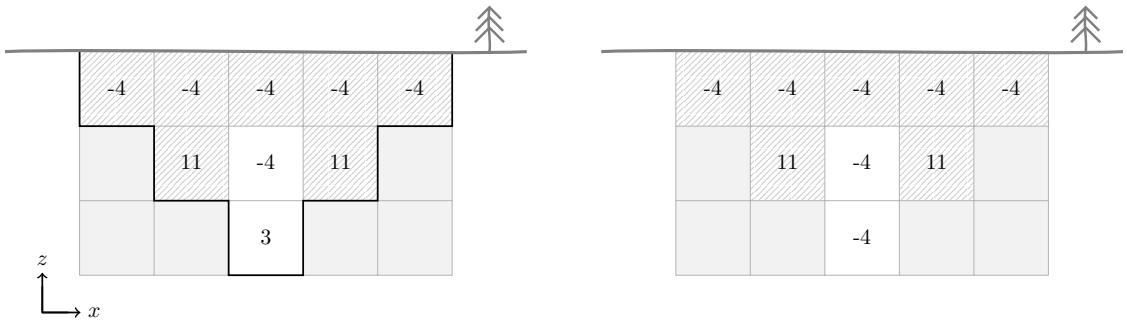


Figure 2.13 Two issues with the conventional floating cone algorithm. The pit on the left is too large, and no pit is identified on the right which is too small compared to the optimum ultimate pit (hatched blocks).

Several authors have worked to address these issues to varying degrees of success. Wright, in 1999, introduced the floating cone II algorithm which was able to outperform the original approach by varying the float sequence and adjusting the extraction criteria in certain situations [63]. This was later modified and further developed by Khalokakaie in 2006, Kakaie in 2012, among many others as recently as 2022 [64–68]. Floating cone methods are inherently heuristic methods and do not guarantee the optimal ultimate pit limits in all scenarios, however their strength comes from the ease with which minimum mining width constraints can be incorporated and their relative ease of implementation and use. Interestingly with the advent of flow based techniques, and efficient implementations of exact ultimate pit optimization algorithms, floating cone methods are often slower than their optimal counterparts when used without any operational extensions.

2.2.6 The Pseudoflow Algorithm

The pseudoflow algorithm is very similar to the Lerchs and Grossmann algorithm, in that both algorithms start with a primal infeasible solution and move towards feasibility by incorporating violated precedence constraints at each iteration [46, 54, 55]. In Chapter 3 a variation of the pseudoflow algorithm that is customized specifically for the ultimate pit problem along with relevant implementation details is described. Therefore, this section focuses on some of

the practical differences between the pseudoflow algorithm and the Lerchs and Grossmann algorithm instead of the details of the pseudoflow algorithm which are discussed later.

The biggest practical difference between the pseudoflow algorithm and the Lerchs and Grossmann algorithm is that the pseudoflow algorithm is much faster. A computational comparison conducted by Hochbaum and Chen in 2000 showed the same model being solved by the push-relabel algorithm in forty minutes whereas the Lerchs and Grossmann algorithm took around five and a half hours [69]. Hochbaum indicated in the same paper that the pseudoflow algorithm, which was in part inspired by this comparison, ran faster than all known implementations of the push-relabel algorithm. Muir, in 2007, showed how the pseudoflow algorithm was superior to the Lerchs and Grossmann algorithm in practice [70]. Chandran and Hochbaum also showed that the pseudoflow algorithm was superior to known implementations of the push-relabel algorithm [71].

In 2015, a comparison between all three algorithms; the Lerchs and Grossmann algorithm, the push-relabel algorithm, and the pseudoflow algorithm was conducted [40]. Deutsch et al. showed for a model with 16 million blocks that pseudoflow could compute identical results in four seconds, to those by push-relabel in nine seconds, and Lerchs and Grossmann in forty five minutes. With one particularly egregious dataset, which consisted of a steeply dipping vertical ore body, the Lerchs and Grossmann algorithm took fifteen hours compared to pseudoflow requiring a mere nine seconds.

Another practical difference is that the pseudoflow algorithm, which is based on routing units of flow around a network, is limited to operating with integral economic block values, unlike the Lerchs and Grossmann algorithm which can use floating point numbers [72]. This is not a detriment for mining engineers, as fractional economic block values can be multiplied by a large constant and then rounded to the nearest integer while retaining the same solution. In certain circumstances there may be a concern for integer overflow within the solution process, but this implementation detail should be handled by any serious ultimate pit solver.

There are several different variants of the pseudoflow algorithm which control the order in which nodes are processed and the means by which nodes are labeled. Labeling, which is another implementation detail responsible for much of the performance improvements relative to the conventional Lerchs and Grossmann algorithm, is discussed further in the subsequent chapter.

However the general notion is to assign a monotonically increasing label to nodes that prevents the algorithm from reprocessing certain nodes until other nodes have been processed. This has the effect of avoiding certain sequences of merging operations that would require additional iterations and is one of the key differentiators between the pseudoflow algorithm and the Lerchs and Grossmann algorithm. Of the lowest-label and highest-label strategies described by Hochbaum in 2001, the highest-label variant generally performs the best on ultimate pit problems [46, 70].

2.2.7 2D Dynamic Programming Algorithm

The two-dimensional ultimate pit algorithm from Lerchs and Grossmann is a dynamic programming based technique to solve a special case of the ultimate pit problem [20]. The precedence constraints are restricted to the simple case of requiring each mined block to mine the block immediately above it and the two blocks above and to the left and right. A dynamic programming method is characterized with a straightforward data preparation step, a iteration through the section, and a traceback step. In Section 4.3, a novel extension of this algorithm which accounts for minimum mining width constraints is developed as a part of this dissertation.

The data preparation step is to construct an initial tableau which converts the economic block values into a cumulative value model which corresponds to extracting the entire column of material above each block. This avoids having to recompute these cumulative values over and over again throughout the process. An example of this data preparation step is shown in Figure 4.8.

The iteration step proceeds down each column individually from left to right keeping the top row zero, and filling in each cell of a new tableau as the maximum of three previously computed values corresponding to mining an additional bench downwards, mining straight across and reducing the depth of the pit by one bench. The entries of this new tableau represent the maximum possible contributions of the columns to the left and thereby construct the ultimate pit one column at a time from the left of the section. Traceback information is recorded on a per block basis indicating which of the three options were the best.

Lerchs and Grossmann describe keeping the top row as zero throughout the entire process and following the traceback information from the maximum valued block in the first row [20]. This can present difficulties when there are multiple disparate pits within the cross section, because there is no means by which value can be transferred between the pits without mining the

uppermost bench, which generally consists of overburden, and reduces the value. It is therefore better to only set the leftmost value of the upper air row to zero and use the iteration step mostly unchanged, allowing the values of the 'air blocks' to increase if necessary. Finally, the traceback step should be initiated from the rightmost block in the air row instead of the maximum valued block in the first bench. This minor issue in Lerchs and Grossmann's orignal description was addressed early on [22, 73], and can also be seen corrected in Section 4.3.

2.2.8 Alternate methods

Zhao and Kim proposed another network theoretic algorithm in 1991 that was purported to have better performance than the Lerchs and Grossmann algorithm [74]. Their algorithm is of a similar character and operates with a similar direct network representation of the problem, however Zhao indicates that the normalization step is avoided by enforcing a different set of invariants between operations [75].

The original dynamic programming approach from Lerchs and Grossmann was restricted to two dimensional cross sections of an open-pit mine. Johnson and Sharp, in 1971 showed a possible means to apply this approach to a three dimensional mine which involved running the algorithm repeatedly on sections and then smoothing between those sections [73]. This approach found use as it is substantially less computationally intense than exact approaches. As computers have become more powerful and faster algorithms are more available a heuristic is no longer applicable when the optimal answer is known.

Koenigsberg, in 1982, present a three dimensional application of dynamic programming to the ultimate pit problem [76]. This approach has faced valid criticism for not identifying the optimal ultimate pit contours due to how it handles precedence constraints and how in certain cases it creates additional constraints which preclude the true optimal result [77–79]. A proper three dimensional dynamic programming approach to the ultimate pit problem which flexibly handles precedence constraints has not been developed.

As previously mentioned in Section 2.2.3 any max-flow/min-cut algorithm would be usable with the ultimate pit problem. Such as the push-relabel algorithm [52, 53], the Ford-Fulkerson algorithm [50], the Edmonds-Karp algorithm [80], Dinic's algorithm [81], and several others [82–85].

It is not expected that *all* max flow techniques will always be faster than the Lerchs and Grossmann algorithm. In Lerchs and Grossmann's original paper they specifically mention that their direct approach is preferred for "obvious reasons," although they neglect to say what those reasons are. It is possible that the max flow algorithms available at the time, which have quite steep memory requirements, were not well suited to the existing computer hardware. Yegulalp and Aries applied the excess-scaling max flow algorithm from Ahuja and Orlin to the ultimate pit problem but were unable to solve problems as quickly as the Lerchs and Grossmann implementation from Whittle [86, 87].

Recently, in 2022, Chen et al. have described an algorithm for determining the maximum flow in near-linear time which could be the fastest approach yet described [88]. This approach appears to be quite involved, relying on identifying certain minimum cycles and using custom data structures. An implementation of Chen's algorithm does not yet exist however this could be a very valuable result for large models if the algorithm translates well to existing computer hardware.

2.3 The Block Scheduling Problem

Where the ultimate pit problem is concerned with determining which blocks should be mined at all, the block scheduling problem is concerned with when those blocks should be mined, and where they should be routed. With these additional concerns, the dimensionality of the problem vastly increases which allows practitioners to define additional constraints which more closely approximate the true open-pit mining process. However, this increased flexibility and accuracy also increases the problem's complexity and the required solution time. Block scheduling problems are more complicated, more specialized, and less tractable than ultimate pit problems.

Overcoming these challenges has received a great deal of academic and commercial effort over the years. Researchers have evaluated a variety of approaches. In Section 2.3.1 the pushback approach to block scheduling is discussed. This approach begins with the ultimate pit, and then calculates a sequence of nested pits through some means, which approximate an extraction sequence wherein parts of the smallest nested pit are extracted first, followed by the next largest pit next, and so on until the ultimate pit. Nested pits only consider the time component of the block scheduling problem, and only indirectly. Integer programming is often employed to properly handle routing, blending, stockpiling, and more relevant concerns which is discussed in Section

2.3.2. Finally Section 2.3.3 discussed several other approaches to the block scheduling problem that do not rely on Integer programming, and instead use heuristics to inform long range open-pit mine planning.

2.3.1 The Pushback Approach to Block Scheduling

Pushbacks, or phases, are nested pits calculated via the ultimate pit problem and some pit parameterization method introduced in Section 2.1.4. The central idea is that these pushbacks approximate an optimal extraction sequence where benches from the smallest pit are generally extracted before those in subsequent pushbacks.

Using pushbacks to approximate the block scheduling problem first appeared in Lerchs and Grossmann's paper which introduced the ultimate pit problem [20]. Lerchs and Grossmann proposed reducing the block values by a constant, and resolving for the ultimate pit with the modified block values. This pit will be the same or smaller than the original ultimate pit, and if smaller it will be smaller such that it tends to prefer the higher valued blocks. By repeating this process, reducing by even larger constants, an entire series of nested pits can be determined which Lerchs and Grossmann claimed to maximize the integral of the cash flow curve. One useful way of thinking about this process is by dualizing a constraint of the form $\sum_x X_x \leq T$ into the objective where T is some tonnage constraint whose value isn't important. The dual on this constraint is exactly the constant that Lerchs and Grossmann use to decrease all of the block values.

Matheron later expanded on the idea of pit parameterization alongside Vallet in the late 70s [29, 30]. Dağdelen and Francois-Bongarcon used more finely grained variations on the commodity prices and mining / processing costs [31]. Whittle also developed several approaches to pit parameterization based on price and cost factors [11]. Meagher, Dimitrakopoulos and Avis review several of these approaches, and others in a recent review paper [89].

A key feature of pit parameterization is that the pits are nested within one another and do not overlap. This makes each pit potentially a good candidate to use for downstream pushback design and can be used to determine rough production schedules or used in more sophisticated block scheduling algorithms. A disadvantage of pit parameterization is that in some circumstances the ore body does not lend itself to creating operable pushbacks. For example, a steeply dipping vertical ore body often generates nested pits that are concentric cones which do not share a

common wall and do not form an efficient sequence.

Additionally the volumes of the nested pits do not vary linearly with the parameterization factor which can complicate the pushback selection process[1]. This typically happens when a large volume of ore suddenly becomes economic when a certain profit threshold is exceeded.

However the greatest disadvantage of using a pushback generating, or pit parameterization, approach to block scheduling is that it simply does not adequately address the important constraints. The nested pits are not guaranteed to form yearly or quarterly production volumes and do not consider material routing, capacity constraints, blending, or any other relevant constraint. To address these concerns more sophisticated methods are required.

2.3.2 Block Scheduling with Integer Programming

Johnson developed a full featured formulation for the mine production scheduling problem that, if solved to optimality, would report an open-pit schedule which maximizes NPV and addresses the most important scheduling constraints [44]. The model incorporates a cutoff grade strategy which varies throughout the life of the mine which allows blocks to be routed appropriately. Much like the ultimate pit problem precedence constraints are considered, but so too are capacity limitations on the relevant mining processes (for example a mill and leach pad), and average grade requirements. Johnson's formulation has provided the basis for many future efforts.

Two review papers which address production scheduling with integer programming have been developed by Newman et al., and Fathollahzadeh et al. [5, 42]. The general thrust of research in this area has been to incorporate additional constraints or to find more tractable means of obtaining solutions.

All block schedules incorporate time as a dimension into the model, and are used to indicate when blocks should be mined. Under the NPV objective function model typical in open-pit mine planning it would be vastly preferable to mine all ore blocks as soon as possible. The constraints in place to prevent this are capacity constraints on either the total volume of material mined, or on a per-destination basis. For example, a given mill may only be able to process a few million tons in any year which is incorporated as a knapsack style constraint into the optimization model.

[1] A novel dynamic programming based approach to this gap problem is developed in Appendix B

Risk constraints are a common addition to mine schedules following Johnson's seminal work. Dimitrakopoulos et al. incorporate grade uncertainty and risk into open-pit design by incorporating additional constraints and considering geostatistical simulation [90–92]. Godoy presented a multi-stage approach for profitable risk management [93]. Van Dunem incorporated a form of risk constraints based on limiting the number of blocks mined that are classified as measured, indicated, or inferred in a given year [94].

Another example of an additional constraint is in mining complexes with both a surface and underground component. King et al. developed a model which incorporated the surface-to-underground transition, leaving space for a suitable crown pillar [95]. However, the model was still very difficult to solve and a linear relaxation approach was used alongside a rounding heuristic to achieve an integer solution.

Several researchers have worked to relax certain constraints or change the variable types in order to solve larger models. Gershon allowed for continuous variables in certain parts of the formulation to allow certain blocks to be mined partially [9].

One method of improving the tractability of the block scheduling problem is to reduce the number of variables. Several approaches rely on aggregated blocks together into larger groups of blocks that are considered as one unit [96–98]. These aggregation approaches yield smaller problems with fewer variables that are easier to solve however the larger aggregated units may not provide enough granularity to identify the optimal mine design for the original problem.

A useful approach to solving large block scheduling problems is via Lagrangian relaxation, which is discussed further in Section 2.5.3 and in Chapter 4. This approach was originally introduced by Dağdelen in the mid 1980s [27]. The value of this approach is that the sub problem remains a network model which has a useful mathematical structure which can be exploited for very fast solutions. Several researchers have expanded on Dağdelen's approach included Tachefine and Sumois who use an alternative method to obtain the necessary Lagrange multipliers and Akaike who worked to incorporate stockpiles and a more involved cutoff grade strategy [99, 100].

The Bienstock Zuckerberg algorithm is a very useful column generation approach used for the block scheduling problem which takes advantage of the large precedence constraint structure and the handful of knapsack constraints [101, 102]. This algorithm is discussed further in Section 2.5.4. Aras, in 2018, showed in detail how the BZ algorithm could be used for direct block

scheduling for complex datasets with three destinations: a mill, a leach pad, and a waste dump [103]. Additionally, Aras incorporated uncertainty and risk into the scheduling process by limiting the number of blocks that are measured, indicated, or inferred on a per period basis. A novel integerization method was also developed which achieves a small gap between the integer solution and the linear relaxation in practice.

2.3.3 Heuristic methods

The number of variables and constraints in the open-pit mine planning process is often too large for exact methods. The techniques described in the previous section to aggregate variables and solve relaxed models help to alleviate some of the difficulty, although there has also been efforts towards using heuristics and other inexact methods that sacrifice an optimality guarantee to be faster. Several approaches have been developed by Chicoisne et al., Lamghari et al., Lambert et al., and others [104–106]. Many of these heuristic methods are discussed further in Fathollahzadeh et al [42]. Several of these methods use meta-heuristics, of which several are described in Section 2.5.5.

One distinct advantage of a heuristic approach to open-pit mine design is that it is much easier to incorporate nonlinear or complicated constraints that cannot be modeled directly with integer programming, or other exact methods that are more prescriptive. Additionally, heuristics can be used not only to generate feasible solutions but also to improve existing ones. For example, the local search heuristic developed by Amaya et al. can take an initial integer feasible solution and evaluate nearby solutions in order to find a local optimum [107].

2.4 Open-Pit Mine Planning with Operational Constraints

Both the ultimate pit problem and the block scheduling problem do not, in their classical descriptions, consider operational constraints such as minimum mining width constraints or minimum pushback width constraints. The ultimate pit problem considers only the barest minimum of constraints. Straightforward bounds on the variable preclude impossible values and precedence constraints govern the shape of the ultimate pit to ensure geotechnical stability. Outside of these considerations there is no prerogative other than to simply maximize the undiscounted economic value. The block scheduling problem has many constraints but the most

common formulations are generally concerned with managing blending, plant capacities, sequencing, and certain economical considerations instead of operational ones.

Many of the researchers that have developed extensions to the ultimate pit problem and block scheduling problem to handle operational constraints, including minimum mining width constraints, are discussed in this section.

The very first floating cone methods from Pana and Carlson were able to handle minimum mining width constraints by restricting the volumes of extraction to consist of several blocks [58, 59]. Whittle, in 1990, had this to say about floating cone methods:

> Apart from being easy to understand and program, the one advantage that the floating cone method has over other methods is that, if instead of using just one block the program uses a disk of blocks as its starting point, then this can ensure a particular minimum mining width at the bottom of the pit [108].

Of course extending the floating cone algorithm to address minimum mining width constraints does not address the fundamental concerns with floating cone methods described in Section 2.2.5. Overmining and undermining errors will still occur and may even be exacerbated.

Wharton, in 1997, describe a series of geometric operators on nested pits used to create more operable designs and assess the impact of minimum mining width on the NPV of long-term schedules [109]. The original nested pits used as input are calculated with conventional parametric analysis and the Lerchs Grossmann algorithm using, for example, a mining cost adjustment factor. The user then specifies a rectangular mining width template and a iterative procedure is carried out to remove small contiguous blocks, remove protrusions along the outer pit wall, handle inaccessible blocks, remove small holes, and overall perform some geometric cleaning. This cleaning is performed based only on the shape of the nested pits and does not consider block values. Run times and typical block model sizes are not reported, but the description implies a constant number of linear passes over the model and is likely to be very quick if programmed efficiently. However, the impact on NPV is substantial. In their case study NPV decreases between 3 and 22% depending on how aggressive the cleaning is.

An early optimization based approach that was not solely geometric was described by Dimitrakopoulos in 2004. Dimitrakopoulos et al. develop a "risk-based production-scheduling

formulation for complex, multielement deposits", and incorporate specific equipment constraints and mining sequence feasibility [91] . Their model provides a benefit for mining blocks in the same period as blocks within their immediate neighborhood, or equivalently a penalty when adjacent blocks are mined in different periods. This benefit is realized by applying cost coefficients, which are determined through trial and error, to relevant terms in the objective function and not as hard constraints. Using cost coefficients allows the user to specify how important mining width constraints are to them in actual dollar terms, however this is typically not straightforward to determine a priori and requires iterating on those parameters considering the final results. This formulation is applied to a model with 2030 blocks and the elapsed time is not reported.

Stone et al. in 2007, describe an in-house optimization tool developed at BHP Billiton called 'Blasor' which generates optimized mine schedules [110]. Blasor internally uses CPLEX, a well known MILP solver developed by IBM, in order to solve specially constructed formulations accounting for mining, transport, comminution, and market constraints. Minimum mining width constraints and pushback width constraints are specifically handled by aggregating blocks together using a 'proprietary fuzzy clustering algorithm' and scheduling based on those operational units. Additionally Blasor provides a graphical tool to allow practitioners to make manual modifications to incorporate other operational constraints which are difficult to encode algorithmically. The precise details are unavailable.

In 2008 Zhang introduced a heuristic approach to incorporate minimum mining width constraints in nested pits similar to Wharton in 1997 [109, 111]. The initial heuristic operator developed in 2008 applies three geometric operators in an unspecified sequence:

- *Removing drop cuts:* small contiguous groups of blocks which are smaller than some constant and are surrounded by later phases or blocks outside of the pit are removed.
- *Small wall removal:* similar to the drop cut removal step, this operator removes small contiguous groups that contact blocks from earlier phases.
- *Interior minimum mining width enforcement:* reassign blocks which do not satisfy a rectangular mining width template to nearby common phases.

These operators do not consider economic value and instead aim to make the pit operational by changing relatively few blocks. It is generally true that changing fewer blocks leads to a smaller reduction in economic value with most datasets. However, it is possible to construct counterexamples and a true optimizing approach must not rely on this. The following year Zhang extended their heuristic cleaning technique with a meta-heuristic that would use the cleaning as a sub procedure [112]. The working solution is repeatedly modified in a stochastic manner then made feasible by the cleaning procedure. This new solution would be accepted as the working solution always if the NPV improves, but also if the NPV reduces with some decreasing probability. Accepting a 'worse' design with some non-zero probability is one of the defining features of simulated annealing and works to prevent the algorithm from getting stuck in local optima. Zhang applied their method to two synthetic models of 16,000 and 120,000 blocks respectively but do not indicate runtime.

Another two optimization formulations for the block scheduling problem with minimum mining width constraints are given by Pourrahimian in 2009 [113]. Their first formulation restricts blocks such that they can only be extracted in a given period if a certain number of nearby blocks are also extracted in that period. This does not explicitly disallow inoperable configurations of blocks, but it does preclude many common issues (such as single block pit bottoms). Crucially, because this extra constraint is appended to a full optimization method it does consider economic block values. The second works by aggregating blocks prior to optimization which had the added benefit of making the model much smaller and easier to optimize while handling mining width considerations by design. The authors apply both of their methods to a single bench of 415 blocks and do not report the total run time.

A sliding time window heuristic method to a variant of the general open-pit block sequencing problem handling multiple periods and typical resource constraints is given by Cullenbine in 2011 [114]. They additionally incorporated a small operational consideration. Each block in Cullenbine et al's model is required to extract the five blocks above in a conventional cross sign configuration but also required to extract at least one of the neighboring blocks on the same level. This additional constraint precludes single block pit bottoms, and other locations where a single block is mined in a period by itself with no nearby support. The largest example considered by Cullenbine et al. contains 25,620 blocks and achieves an optimality gap of 4.3% in just under

three hours. However the impact of the operational constraint itself is not the object of this work and is not isolated.

Pourrahimian and Cullenbine both incorporate the idea of restricting blocks to be mined if and only if a minimum number of additional blocks within the original block's neighborhood are also mined. As the neighborhood grows the minimum number of blocks must also grow, but you cannot, in general, rule out inoperable configurations with only these kind of constraints. Additionally, if the minimum number of blocks grows too large relative to the size of the neighborhood then the constraint is too restrictive, and may lead to poor results.

Instead of incorporating constraints directly into the optimization model Tabesh, in 2014, suggests clustering blocks together into operable shapes and scheduling on those results [115]. This form of clustering can be guided based on both the perceived performance of the clusters in downstream scheduling steps and such that they are big enough for operation. Additionally this form of clustering precludes minimum pushback width violations.

Version 10 of Maptek Vulcan included a tool called the "Automated Pit Designer" which took pit numbers, from conventional nested pit analysis, and created polygonal designs - without ramps - satisfying some operational parameters [116]. The help documentation indicates three operational preprocesses which may be applied to modify the input pit numbers to satisfy mining width constraints. Two are based on mathematical morphology operators [117], and the third is a custom geometric operator to 'snap walls' together between nested pits. Mathematical morphology is a useful tool for operational constraints relating to minimum mining width and is discussed in more detail later, however it is a geometric method and does not consider block values. These routines operate on a bench by bench basis and do not consider precedence constraints, as the pit slopes are handled later explicitly in the polygonization process, this allows them to operate very quickly and have been applied to models with tens of millions of blocks in seconds. In full disclosure, the author of this dissertation developed this version of the automated pit designer in version 10 of Maptek Vulcan.

In Figure 2.14 there are three planar sections through an example ultimate pit model to demonstrate some of the mathematical morphology procedures used in Maptek Vulcan's Automated Pit Designer. The first section on the left exhibits several examples of an inoperable ultimate pit model. There are both missing blocks, which would realistically be mined, and

isolated blocks which would realistically be either not mined or mined along with their neighbors. The middle section shows the result following a cleaning operation consisting of a closing of four blocks and an opening of three blocks, and the right section shows a more extensive cleaning operation consisting of a closing of seven blocks and opening of seven blocks. In both cases the actual economic values of the blocks are not considered and although the resulting sections are now operational, the sacrificed value may be much greater than necessary.

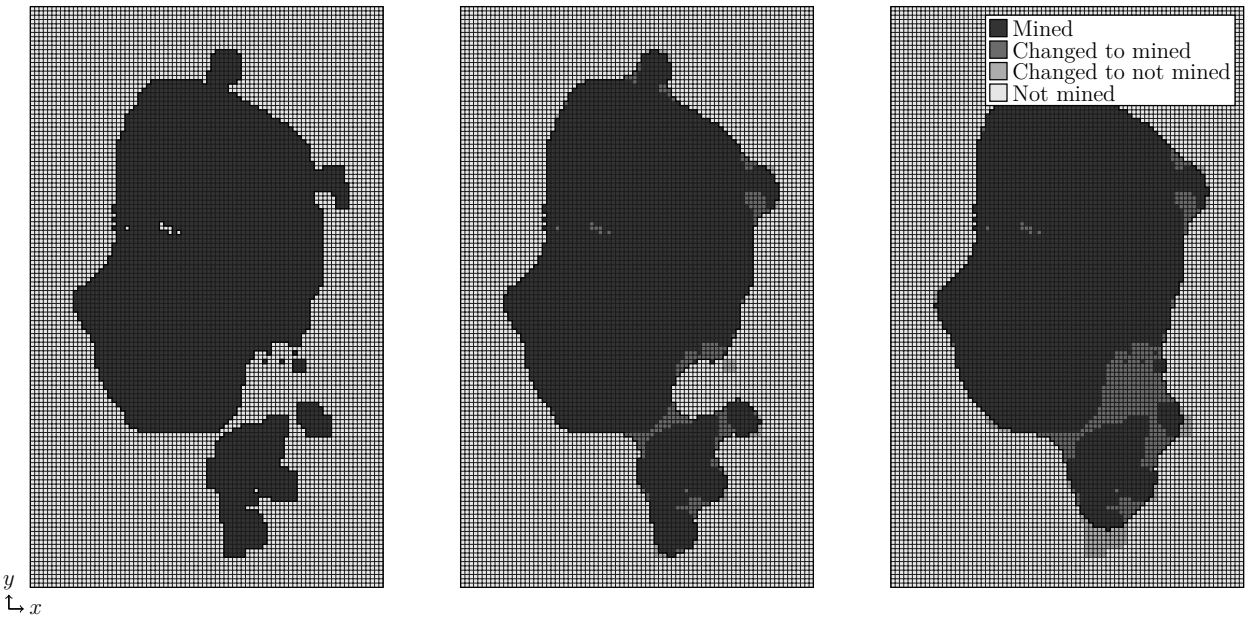


Figure 2.14 Cleaning an ultimate pit with mathematical morphology as with the Maptek Vulcan automated pit designer. The initial planar section on the left is moderately cleaned (middle), and aggressively cleaned (right).

Juarez et al. in 2014, introduce a technique whereby operational constraints are considered within a broader tree search style heuristic approach to open-pit mine scheduling and phase design [118]. Their algorithm, and associated implementation, consider minimum mining widths and minimum pushback widths by only generating designs which satisfy operational constraints in the tree search. Typical block model sizes and times are not reported.

Bai et al. in 2018, describe a custom mathematical morphology approach to handling minimum mining width and minimum push-back width constraints that does not explicitly consider the value of the design changes [119]. Their method goes beyond the basic application of the standard mathematical morphology operators by considering connected components, carefully

considering cycling, and accounting for precedence constraints. They report that their methods are suitable on reasonably small block models, an ultimate pit with 550,000 blocks took 2 hours to incorporate mining width constraints.

An additional clustering approach is presented by Farmer et al. in 2018 [120]. This approach is similar to Pourrahimian's, described earlier in that the blocks are clustered together into larger operable groups prior to schedule optimization. Specifically Farmer et al. amalgamate the scheduled blocks using breadth first search, and divide the aggregations into mineable and non-mineable groups before proceeding. Their specification for minimum mining width is based on the number of connected blocks in each spatial dimension which corresponds to a rectangular mining width. They also describe a heuristic post process for smoothing phase designs and avoiding both minimum mining width and minimum pushback width violations. The method is applied to two orebodies however the model sizes and achieved runtimes are not reported.

Deutsch, in 2019, introduced a formulation for the ultimate pit problem with a minimum mining width based on auxiliary variables [121]. These auxiliary variables follow arbitrary mining width sets such that before any block is mined at least one of its corresponding operable mining width sets must be completely mined. This formulation is presented as a maximum satisfiability problem and was only applied to very small 2D examples, on the order of a few thousand blocks. The two sets of mining width specific constraints in this formulation are appended to a full optimization approach and guarantee optimal economic results satisfying operational constraints if ran to completition. The general maximum satisfiability solvers tested did not scale to full size models however the formulation has merit and is developed further in Chapter 4.

Muir, in 2020, presents a practical method to incorporating minimum mining width constraints into ultimate pit models and successfully apply it to large models with over 21 million blocks [122]. Their method involves solving for the ultimate pit, modifying it, and resolving multiple times - the aforementioned 21 million block model took 10 iterations and achieved a usable 2x2 mining width satisfying pit in 2 hours and twelve minutes. Muir's method encodes known mining width violating patterns along with an 'appropriate' action which are then used to correct pit designs. The process is applicable to large, practical models, and is quite involved. The main drawbacks of this method are that only 2x2 mining width templates are currently supported, and there is no guarantee of optimal results.

A formulation for the geometrically constrained ultimate pit problem also using arbitrary mining width sets and auxiliary variables was introduced by Nancel in 2021, however they also consider additional operational constraints [123]. They incorporate specific constraints to disallow thin connections between adjoining operable zones and to avoid 'cavities' or small collections of unmined blocks contained within the ultimate pit. Nancel et al. 2021 describe approaches to preprocess the input and show that on a moderately sized block model, on the order of half a million blocks, preprocessing can reduce the total solution time for the geometrically constrained ultimate pit problem from just under 20 minutes to just over a minute. Nancel et al. 2021 use off-the-shelf optimization software, Gurobi in this instance, to solve their ultimate pit and block scheduling problems.

Yarmuch et al. 2021 consider operational constraints including mining width, block connectivity, and ramp access by introducing a so called 'compactness factor' to the objective [124]. The compactness factor preferentially guides the optimization model to select blocks which are close to the designed ramp when optimizing a single pushback. This method is applied to either very small models, or models which have been made small by aggregating blocks together. Manual intervention is required to decide on an appropriate compactness factor. They also consider a 'closeness factor' whereby the output of the optimization model is biased to align with a predefined manual, operable, schedule.

Another paper from Yarmuch et al. solve an open-pit pushback design problem considering mining width and connectivity [125]. They use rectangular mining width elements and restrict blocks from being assigned to specific pushbacks similar to [121] and [123]. If a block is assigned to a pushback, then at least one of the rectangular mining width templates must be assigned to that pushback and all of the blocks within that pushback must be extracted. Extensive preprocessing and sliding window approximations are required to reduce the size of the problem. Their method is applied to small models, on the order of a few tens of thousands of blocks, and take many hours to achieve results with an optimality gap of around 6%.

2.5 Optimization

The three main components of an optimization problem are the decision variables which encapsulate the different choices available, the constraints which limit those choices, and the

objective function which ranks the different possible outcomes. In open-pit mine planning there are many subproblems which use these components to define optimization models, or mathematical programs, which can be analyzed to guide decision making in the planning and operating phases of an open-pit mining project. Fundamentally optimization is applicable to many different problem domains within engineering, management, and more [126, 127].

This section discusses the necessary background regarding optimization within the field of operations research which is used in this dissertation to develop tools for open-pit mine design with operational constraints. Section 2.5.1 describes the processes for developing and applying optimization models to real world problems. The main limitations of this approach are discussed. Section 2.5.2 introduces the linear programming paradigm for mathematical optimization which is an extremely useful approach for problems that are inherently linear, or can be approximated as such. A relevant approach to solving large linear programming problems, Lagrangian relaxation, is described in Section 2.5.3 because it is used in the following chapters alongside necessary modifications and extensions. Finally, Section 2.5.5 describes relevant heuristic approaches which are useful when exact approaches are too inflexible or too slow.

2.5.1 Solving Problems with Optimization

Within this chapter several decision problems have already been presented such as; *which* blocks should be mined and *how* should those blocks be routed. Or *when* should different areas of the deposit be developed in order to maximize net present value while satisfying relevant environmental and operational constraints. Within these problems there is an element of choice. There must be some flexibility in what can be done, or how a desired outcome can be achieved, for optimization to be relevant. The flexibility within the system is always bounded by relevant constraints which capture the real-world nature of the problem. Additionally, there must be some means of evaluating different outcomes or decisions.

The process of applying optimization begins with taking the problem and *modeling* it in some capacity. The variables, constraints, and objective function are all defined in such a manner to capture the essence of the problem while considering the tractability and validity of the model. Tractability is the degree to which the model can practically be solved and the extent to which it admits necessary analysis. An extremely large model with billions of variables and interrelated

constraints might be desired to accurately represent the original problem, but if there is no current technology that can generate a solution then that model is not very tractable, and therefore not very useful. On the other hand, the validity of a model describes the extent to which the resulting inferences and conclusions are applicable to the original real world problem. Simplifications in the modeling process which improve the tractability of a model generally have a negative effect on that model's validity, and the trade-off between these two concerns is a omnipresent dilemma within the modeling process.

The model must then be analyzed in order to draw necessary conclusions. Various technologies, including some discussed in the following sections, and mathematical analyses are used in order to extract relevant information from the model, such as the optimal decision policy. The conclusions are derived from the model and not from the original problem, so they may need to be modified and certainly considered within the context of any concessions taken during the modeling process before making any final decisions. Additionally, the problem and modeling process may need to be revisited once the conclusions are reviewed.

It is extremely important to understand the disconnect between the real-world problem and the mathematical model, which is a fundamental limitation of optimization. For a wide variety of reasons models can never fully capture every possible outcome and consideration within the real world. But that does not preclude optimization as a valuable technique in real world scenarios, because without optimization one would find themselves adrift in an endless sea of possibilities and concerns with only 'rules of thumb' and their 'best judgment' to guide them. Even with the numerous concessions and approximations required, open-pit mine planning benefits from the judicious application of optimization techniques to guide decision making.

There are many approaches that may be considered to take a mathematical model through to its conclusions. These approaches can generally be divided into two groups, *exact* methods that provide not only an optimal solution but also a certificate which guarantees that it is as good as possible, and *in-exact* or *heuristic* methods which generally provide a good solution but cannot guarantee optimality. Of the exact methods there are many approaches such as naïve enumeration, dynamic programming, network methods, linear programming, maximum satisfiability, and others. Heuristic approaches include a wide range of ad-hoc methods, simulated annealing, genetic algorithms, tabu search, and others. In the following section several of these

approaches are considered in detail as they are used in the following chapters to develop high quality solutions to relevant problems in open-pit mine planning.

2.5.2 Linear Programming

Linear programming is a technique whereby a linear system is analyzed to find a vector which maximizes (or minimizes) some linear objective function subject to linear equality and inequality constraints [128–130].

A straightforward linear program expressed in standard form with vector notation is given in statements 2.10 to 2.12.

$$\text{maximize} \quad cX \tag{2.10}$$

$$\text{s.t.} \quad AX = b \tag{2.11}$$

$$X \geq 0 \tag{2.12}$$

In this notation c are the objective function coefficients, X are the decision variables, A is the $i \times j$ matrix, where i is the number of rows (or constraints), and j is the number of columns (or variables), and b are the righthand side of the constraints. It is possible to switch the sense of the objective from maximize to minimize, or to change the equalities of constraints from $\leq$ to $\geq$ or $=$ and still remain a linear program. But this format is preferred because it is straightforward to transform any other formats to this one. For mine planning many problems can be expressed as linear programs, which is very useful because linear programs are generally quite easy to solve with the simplex algorithm or interior point methods [126, 129].

Every linear program has an associated dual, which is a closely related linear programming problem which uses the same parameters. For a linear program in standard form the dual is given in Equations 2.13 to 2.14.

$$\text{minimize} \quad bV \tag{2.13}$$

$$\text{s.t.} \quad A^T v \geq c \tag{2.14}$$

A^T is the transpose of the original A matrix and v are the new dual variables, which are unrestricted when the original constraints are equality constraints. We have already seen the

application of duality to optimization problems in open-pit mine planning. In Section 2.2.3 the dual of the ultimate pit problem was used to create an equivalent network flow model.

In many real world applications of linear programming it is desirable to restrict the decision variables to integer values. This greatly increases the difficulty of the problem because duality is no longer applicable, and many of the most useful theoretical developments are disrupted. Integer linear programs (ILPs) are typically solved through a combination of analyzing their linear relaxations and branch and bound, which is a exponential technique which enumerates many integer solutions in order to find the best [126].

2.5.3 Lagrangian Relaxation

Lagrangian relaxation is a strategy that can be applied to integer programming problems to help compute the linear-programming relaxation of very large models more quickly. In some cases the Lagrangian relaxation can even be used to give a tighter bound than the linear relaxation, although this is not guaranteed. The results from a Lagrangian relaxation model can also be rounded to provide an heuristic ILP solution, or used for other purposes.

Lagrangian relaxation relaxes specific constraints from the input model but does not remove them entirely. Instead, these relaxed constraints are dualized into the objective function and their violation is penalized by using an appropriate multiplier - called a Lagrange multiplier denoted with λ. For a particular constraint the new objective will have the new term in Equation 2.15.

$$... + \lambda_i \left(b_i - \sum_j a_{i,j} X_j \right) + ... \tag{2.15}$$

Where λ_i is the Lagrange multiplier for this constraint, b_i is the right hand side of the constraint, X_j are the variables, and $a_{i,j}$ are the constraint coefficients for each row i and each column j. The sign of λ_i is carefully controlled based on the sense of the objective and the direction of the inequality in the constraint. A constraint of the form $\sigma_i a_{i,j} X_j \leq b_i$ requires a non-negative ($\lambda_i \geq 0$) multiplier when maximizing, and non-positive when minimizing. A constraint of the form $\sigma_i a_{i,j} X_j \geq b_i$ requires a non-positive ($\lambda_i \leq 0$) multiplier when maximizing, and non-negative when minimizing. Equality constraints have unrestricted multipliers regardless of the objective sense.

There are two important aspects of a Lagrangian relaxation that make it a valid operation.

- Every feasible solution of the original model is feasible within the relaxed model. This is straightforward to see, because *removing* constraints will never exclude additional solutions.
- And, the objective value in the relaxed model for every feasible solution must be equal to or better than the objective function in the full model. This follows because of the sign rules on the λ_i multipliers, and how the new terms in the objective are constructed. A solution which satisfies a constraint will have a term in the objective of the necessary sign. For example a constraint of the form $\sum_j a_{i,j}X_j \leq b_i$ when maximizing, will have a term that is of the form $\lambda_i \left(b_i - \sum_j a_{i,j}X_j \right)$ where $lambda_i$ is non negative and, because the constraint is satisfied, $\left(b_i - \sum_j a_{i,j}X_j \right)$ will be non-negative.

The primary goal when using Lagrangian relaxation is to determine the best possible bound on the solution to the original ILP. However, if an optimal solution to the Lagrangian relaxation is found such that it is feasible for the full model and either all multipliers are zero or the constraint is satisfied at equality - the solution satisfies complementary slackness - it is optimal in the full model.

In some cases, the solution to the Lagrangian relaxation will be a tighter bound on the integer solution than the straightforward linear relaxation. However this is not guaranteed. If the constraints which were chosen to dualize in the Lagrangian relaxation admit too easy of a model, that is, one that can be solved by linear programming alone, then the bound will not be improved [126]. The Lagrangian relaxation guided solver in Section 4.4.6 must contend with this fact.

Additionally there is a practical challenge which arises when using Lagrangian relaxation. The values of the multipliers must be determined through some means, which can be difficult. It is often possible to ascertain how to improve the multipliers after a solution is determined, and potentially when improvement is unlikely which can inform when to stop. A popular method of determining Lagrangian multipliers is by subgradient search. In this method the full collection of Lagrange multipliers is updated at each iteration by using a step size and the subgradient which is computed by looking at the previous solution's relaxed constraints. That is, the new multipliers are given in Equation 2.16.

$$\lambda_i^{t+1} \leftarrow \lambda_i^t + s_t \delta\lambda \tag{2.16}$$

Where λ_i^t is the multiplier for constraint i for iteration t, s_t is the step size for iteration t, and $\delta\lambda$ follows in Equation 2.17. The step size, s_t, is a matter for some flexibility as well. But it is known that a step size which converges to zero, when the sum of all step sizes does not, will converge [126].

$$\delta\lambda \leftarrow \frac{(b - AX^t)}{||b - AX^t||} \tag{2.17}$$

Where b is the vector of right hand sides to the dualized constraints, A is the constraint coefficients, and X^t are the variable values as calculated for the current iterations solution. All λ values may need to be projected in order to satisfy the necessary sign conventions as discussed earlier. If an optimal solution is found that satisfies complementary slackness we can terminate with the optimal answer. However if this fortunate scenario does not occur, then the best solution and the current bound can be reported once the step size has reached a very small number or whenever further computation does not seem justified.

2.5.4 The Bienstock-Zuckerberg algorithm

The Bienstock-Zuckerberg algorithm (BZ) was first discussed in Section 2.3.2, where it has, in recent years, seen use for solving specific instances of the open-pit block scheduling problem. At its core BZ is an extension of the column generation approach to solving linear programming problems [101, 102]. Bienstock has colloquially referred to the approach as "Column generation on steroids".

Column generation is an approach where the large scale linear program is decomposed into a master problem and sub-problem. Column generation is primarily useful where the optimization needs to address combinatorially many decision options that can be re-expressed as columns (variables representing full solutions) in a partial master problem [126]. This partial master problem considers the few columns currently available in order to inform a column generating sub problem about which additional columns may be necessary in order to improve the objective. The duals from the solution to the partial master are used to inform the column generating procedure,

which in the presence of complicating side constraints may be a heuristic. If there is no way to construct attractive new columns then the model terminates with an optimal, or near-optimal, solution to the original problem.

The BZ algorithm deviates from the conventional column generation procedure in two ways. The first is to require the columns to always contain the optimal solution of the master problem in the previous iteration, which is used at times during the algorithm to prevent the number of columns from getting too large. The second is to construct the columns as orthogonal 0-1 vectors. Orthogonal means that for each variable in the full master problem it obtains the value of one in exactly one column and no others.

The BZ algorithm is most applicable to problems where there is a large submatrix consisting of $X_i \leq X_j$ constraints which can be solved with a network flow procedure, and fewer knapsack constraints which are of the form: $\sum_i X_i \leq y$. The precedence constraints are placed in the column generating subproblem, and both the precedence constraints and knapsacks are retained in the master.

2.5.5 Heuristic Optimization

Linear programming, and other similar techniques, can admit *exact* solutions to a given optimization model which are *provably* as good, or better, than any other possible solution in terms of objective function value. A heuristic is an approach that admits a feasible solution, in that it satisfies all necessary constraints, and generally tries to achieve as good a solution as possible but it is not guaranteed to obtain the exact optimum. In general exact approaches are much more satisfying and are preferred. If one is comfortable with the model they have developed and any assumptions therein, the exact optimal result is going to give the best feasible solution alongside a certificate that no other solution is going to be better. However this may not be possible for large models which cannot easily be made smaller without sacrificing model validity.

In these instances a heuristic approach to optimization, either ad-hoc or following an established meta-heuristic methodology may be appropriate. Additionally, often the losses from settling for a heuristic instead of the exact optimal result will not exceed the losses and variations from the concessions taken in the modeling processes or variations and uncertainties present in the input data or parameters [126]. An exact optimal solution to a shaky model with uncertain

data that takes several days to compute is not that much better than a 99% solution computed in a few minutes in most applications. Therefore, just as optimization in general must be applied judiciously with adequate understanding of any limitations so too must the solution methodology, exact or heuristic, be selected with understanding of the relevant trade-offs.

Ad-hoc heuristic methods are customized specifically for a particular real world problem and optimization model. They are developed and implemented independently for a particular problem with a narrow range of input differences. An ad-hoc approach can take advantage of specific problem attributes that may admit generating feasible solutions or improving solutions efficiently. They may incorporate more general approaches such as greedily selecting the current best known value for a particular variable or performing a basic local search that explores 'nearby' feasible solutions before selecting the best outcome. However, they will always only be usable for the problem for which they were designed, and although they admit high quality solutions for a specific problem they can be difficult and costly to implement.

Meta-heuristic approaches instead rely on some higher level strategy, which may have been inspired by some natural process or at least guided by some high level understanding of optimization. Most meta-heuristics, and most ad-hoc approaches, fall under the general paradigm of generating or searching through many feasible solutions and selecting the best solution as the final answer. Where meta-heuristics differ is that they provide a strategy and criteria for guiding that search to be as effective as possible.

Simulated annealing is a meta-heuristic inspired by the natural process of annealing whereby particles arrange themselves into high strength configurations during a slow cooling process [126, 131, 132]. The general idea is to take a current solution modify it through some appropriate means and compute the change in objective value. If the change is an improvement, for example the objective value increases when maximizing, the change is accepted and incorporated into the solution for the next iteration. However, if the change would not improve the objective it is only accepted with some decreasing probability which corresponds with the 'temperature' analogue in the overall process. Incorporating non-improving moves into the optimization process allows for more solutions to be explored, potentially incorporating solutions that 'break out' of local optima.

Genetic algorithms fall within the broader group of evolutionary meta-heuristics which maintain a larger *population* of possible solutions and provide operations for combining and

managing the population [126, 133]. One possible operation for combining two solutions together is the *cross-over* which takes parts of each solution to form a new solution. At regular intervals the population is culled by removing low performing solutions and incorporating other random solutions.

There are many other metaheuristics, including tabu search, particle swarm optimization, ant colony optimization, all of which employ different approaches to obtain high quality solutions.

2.6 Discussion

The necessary background on open-pit mine planning, its associated algorithms, and underlying theory have been introduced in this chapter. This information forms the basis for the improvements to the pit optimization process (Chapter 3), the developments in including minimum mining width constraints (Chapter 4), and extending those developments to the direct block scheduling problem with operational constraints (Chapter 5).

CHAPTER 3

AN IMPROVED ULTIMATE PIT SOLVER – MINEFLOW

A fundamental component of open-pit mine planning is the ultimate pit problem, where the undiscounted value of an open-pit is maximized subject to precedence constraints. It is used for many different purposes, as discussed in Section 2.2, including as a subproblem in the optimization procedures described in Chapters 4 and 5. In all circumstances it is preferable to compute the provably optimal results as quickly as possible, and in many circumstances it is advantageous to be able to modify block values and recompute the ultimate pit without having to start everything from the beginning. For these reasons, and to facilitate future research efforts in open-pit mine planning, a fast, extensible, open-source, ultimate pit solver named MineFlow is developed in this chapter. MineFlow, at its core, is a specialized and customized implementation of Hochbaum's pseudoflow algorithm specifically for use with mining problems.

Section 3.1 describes the pseudoflow algorithm in detail and customizes it specifically to the ultimate pit problem. Hochbaum's pseudoflow algorithm solves the more general max-flow min-cut problem and must contend with a few complexities that are not present in the ultimate pit problem. Removing that unnecessary complexity from the algorithm and taking advantage of the ultimate pit problem's special structure allows for a faster implementation. Additionally, this section describes a novel notation for the pseudoflow algorithm which helps to make it easier to understand and communicate to new researchers and practitioners.

Section 3.2 expands on several of the important implementation details which serve to make this implementation much more performant than available alternatives. Specifically the importance of lazily generating precedence constraints, using the minimum search patterns from Caccetta and Giannini, and other details are discussed.

Finally, Section 3.3 presents a computational comparison which highlights the tangible benefits of the theoretical and practical improvements developed in this chapter. This approach uses less memory and less computer time than currently available commercial implementations of the pseudoflow algorithm and computes identical results.

This chapter is adapted, in part, from 'An Open-Source Program for Efficiently Computing Ultimate Pit Limits: MineFlow' by Matthew Deutsch, Dr Kadri Dağdelen, and Dr Thys Johnson published in March of 2022 during the development of this dissertation [56]. This adaptation is developed with permission from the licensor: Springer Nature and Natural Resources Research. The source code for the implementation described in this chapter, approximately 6,000 lines of C++, is readily available from `https://github.com/mineflowcsm/mineflow` and is licensed under the permissive MIT license to facilitate further development and collaboration from both academic and commercial partners. During the development of this thesis MineFlow has already been adopted by research groups at commercial mining software companies. In a private communication to the author, one researcher disclosed that MineFlow was able to solve some of their problems five times faster than their previous implementation.

3.1 The Pseudoflow Algorithm

The pseudoflow algorithm is a highly performant max-flow min-cut algorithm inspired by the venerable Lerchs and Grossmann algorithm [46, 54]. The pseudoflow algorithm is of great practical importance due to its ability to compute ultimate pits very rapidly and is the current preferred approach for ultimate pit optimization in long range mine design as discussed in section 2.2.6. This increased speed allows companies to decrease turnaround time, avoid expensive downtime associated with waiting for results to be computed, and opens the door to valuable analyses which incorporate sensitivity analysis and uncertainty management. Additionally, several of the approaches described in the following chapters rely on solving multiple ultimate pit problems to incorporate minimum mining widths and develop high level mine schedules.

Section 3.1.1 reviews the necessary notation and nomenclature regarding the network models which forms the framework for the pseudoflow algorithm. Sections 3.1.2 to 3.1.7 describe the algorithm in detail and introduce and illustrate the notation for the pseudoflow algorithm developed herein. Within these sections any departures from the conventional pseudoflow algorithm are highlighted, as it is these departures (along with the implementation details described in Section 3.2) that make this implementation a valuable addition to the mine planning engineer's toolkit. Finally relevant literature on the computational complexity of the pseudoflow algorithm is briefly discussed in Section 3.1.8.

3.1.1 Network Preliminaries

A network, or graph, is a mathematical structure which models pairwise relationships between objects. Networks are used to represent things which are both abstract and concrete. Many problems become simpler when thought of through the lens of networks because turning a real world problem into a network requires one to think cogently about which of the components of the problem can be combined and represented as *nodes* and then how best to model the relationships between those components with *directed* or *undirected arcs* [49].

Arcs are used to represent pairwise relationships between nodes. A directed arc between two nodes indicates that the arc has a special orientation. The beginning and ending nodes of a directed arc are called the *tail* and *head* respectively. An undirected arc between two nodes does not have an order and only indicates that a relationship exists between the nodes.

A sequence of arcs traversed in any direction between two different nodes is a *path*. Paths are defined such that they only go through nodes and arcs at most once. If a network is constructed such that any two nodes are connected by exactly one path it is called a *tree*, an example tree is shown on the left in Figure 3.1. Often trees have a special node designated the *root* node from which there could be many *sub trees* or *branches*.

In the ultimate pit problem, blocks are represented as nodes and precedence constraints as directed arcs. This is a special kind of network called a *directed acyclic graph*. Acyclic means that the network does not contain any directed cycles. Acyclicity is inherent in precedence graphs because each block only depends on blocks that are above them in elevation.

A *closure* of a directed network is a set of nodes such that there are no arcs with their tails inside the closure and their heads outside. In the ultimate pit problem all closures of the network are valid pits because the restriction on directed arcs ensures there are no precedence violations. An example closure is shown on the right in Figure 3.1. The ultimate pit, therefore, is the smallest maximum valued closure of the precedence graph.

Networks are used to model many different real-world problems, one of special interest here are network flow problems. In a network flow each arc has a maximum allowed capacity and an associated flow. Two special nodes are identified as the *source*, denoted with a S, and the *sink* with a T. Flow originates at the source node and terminates at the sink node. Usually, every

other node must satisfy a *flow-balance* constraint which requires that the amount of flow into the node be equal to the amount of flow leaving. Network models can be used to model fluids in pipes, power in an electrical grid, traffic on roads, and other similar things [49]. In the context of ultimate pit analysis the flows on arcs can actually be thought of as flowing money. The flow corresponds to money moving around and paying for the extraction of necessary blocks. This analogy is expanded upon and justified in future chapters. A small example network flow model is shown in Figure 3.2.

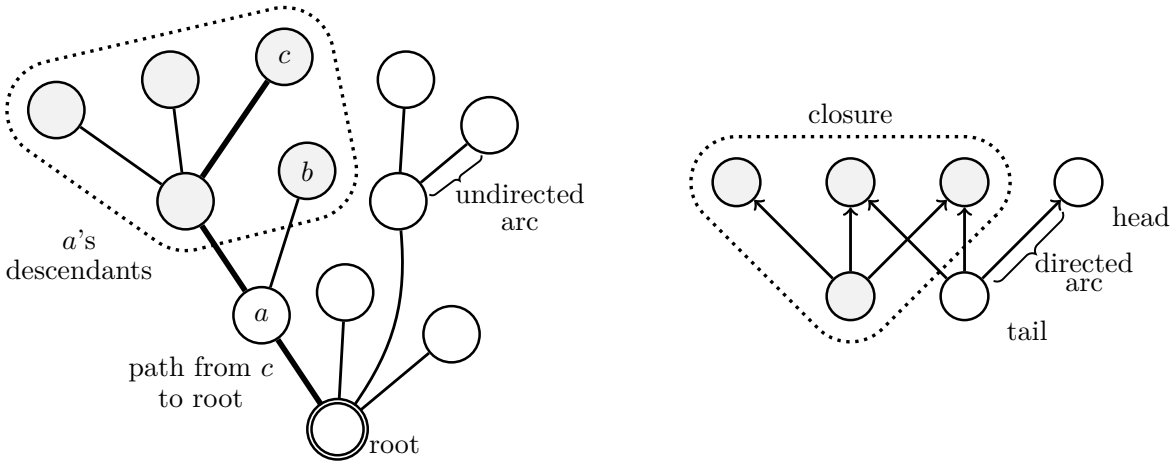


Figure 3.1 Left: a network which is a tree with associated terminology. Right: a network which is a directed acyclic graph. Figure adapted with permission from Deutsch, Dağdelen, and Johnson 2022 [56]

In Figure 3.2 the current flow from the source to the sink is four units, however it is possible to route additional flow through this network. The bolded arcs through the middle of the network can carry one additional unit of flow, and the path along the top could take an additional two units. If both paths were saturated the flow for this network would be seven, which is the *maximum flow*.

In a network there are many ways to cut the network into two pieces. If the partitions are organized such that one side contains the source and the other side contains the sink then these two sets are called an *s-t* cut. In an arbitrary cut the arcs that cross from one partition to the other are said to be a part of the cut-set. However, in *s-t* cuts only arcs going from the source side to the sink side are included.

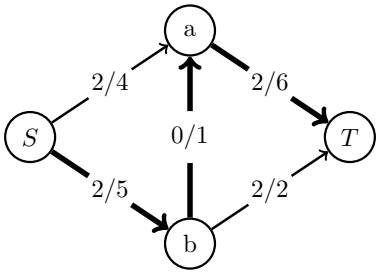


Figure 3.2 An example network flow model. Source S and sink T nodes are labeled. Numbers on arcs indicate 'flow' / 'capacity'. The bolded arcs show a possible augmenting path. Figure adapted with permission from Deutsch, Dağdelen, and Johnson 2022 [56]

For the network in Figure 3.2 there are four possible s-t cuts as shown in Figure 3.3. Note that the cut-set corresponding to cut 4 only consists of two arcs despite appearing to go through three. This is because the middle arc, from b to a, goes from the sink side to the source side.

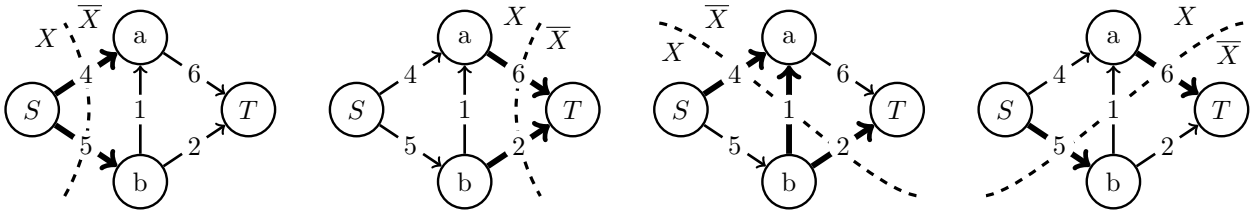


Figure 3.3 The four different possible s-t cuts for the network in Figure 3.2. Numbers on arcs are the arc's capacity. The *cut-set* arcs are bolded and the total cut capacity is shown below each cut. Figure adapted with permission from Deutsch, Dağdelen, and Johnson 2022 [56]

The *capacity* of an s-t cut is the sum of the arc's capacities in its cut-set. The capacity for each possible cut in Figure 3.3 is given in Table 3.1.

Table 3.1 The source set, sink set, cut-set, and capacity of the four possible cuts for the graph in Figure 3.2.

S-T cut	X	$\bar{X}$	Cut-set	Capacity
1	$\{S\}$	$\{a, b, T\}$	$\{(S, a), (S, b)\}$	9
2	$\{S, a, b\}$	$\{T\}$	$\{(a, T), (b, T)\}$	8
3	$\{S, b\}$	$\{a, T\}$	$\{(S, a), (b, T)\}$	7
4	$\{S, a\}$	$\{b, T\}$	$\{(S, b), (a, T)\}$	11

It so happens that the maximum possible flow through a network is equal to the capacity of the minimum cut. This is formalized in the max-flow min-cut theorem. Intuitively, the arcs in the cut-set of the minimum cut correspond to the 'bottle-neck' of the network flow model. There is no way to fit more flow through the cut-set without increasing its capacity, and if there was some other path around the bottle-neck then it would not be a valid s-t cut.

Most formal proofs of this theorem confirm the above intuition by showing that if the max flow did not equal the minimum cut there would be a contradiction. A very early discussion of this theorem appears in Ford and Fulkerson 1962 [50].

3.1.2 Pseudoflow Preliminaries

Hochbaum devised the pseudoflow algorithm to solve the general max-flow min-cut problem. In order to use the pseudoflow algorithm for the ultimate pit problem, the ultimate pit problem must first be transformed into the source-sink form described in Section 2.2.3. In brief; a directed arc with capacity equal to the economic block value is connected from the source to each positive-valued block, and a directed arc with capacity equal to the absolute value of the block's economic block value is connected from each negative block to the sink. Finally, arcs with infinite capacity are connected for every precedence constraint. Applying a suitable max-flow min-cut algorithm to this network identifies the ultimate pit as the source set.

The pseudoflow algorithm is very similar to the Lerchs and Grossmann algorithm and was, in part, inspired by it [46]. Both algorithms operate iteratively by selecting a violated precedence constraint, enforcing it, and then adjusting necessary information for the next iteration. Where the pseudoflow algorithm deviates from the Lerchs and Grossmann algorithm is that it leverages the idea of flow instead of mass, it provides some machinery for selecting which precedence constraint to introduce, and maintains different structures which are easier to update efficiently.

At each iteration the pseudoflow algorithm maintains a *pseudoflow* on the network and a special structure called a *normalized tree*. A pseudoflow is a relaxed flow where nodes are not required to satisfy the flow balance constraints described in Section 3.1.1. Nodes which have more inflow than outflow are said to have an *excess* and nodes with more outflow than inflow have a *deficit*.

The normalized tree is a subset of arcs from the network such that there is exactly one unique path from each node to either the source or the sink node. The tree remains a tree for the entire algorithm so any changes to the tree require adding and dropping arcs simultaneously. 'Normalized,' in this context, requires that only nodes which are immediately adjacent to the source or sink nodes are permitted to carry excesses or deficits.

Here a departure from the conventional description of the pseudoflow algorithm is taken regarding the so called main root of the normalized tree and the source and sink nodes. Hochbaum often combines the source and sink nodes into a single node called the main root, whereas here they are left separate as two distinct S and T nodes. This is done for several reasons. Firstly, it reinforces the max flow nature of the problem where one can imagine flow, in this case 'money' or 'value', traveling from the source to the sink. Secondly, it is much easier to draw and keep the arcs associated with positive and negative blocks from crossing and getting in the way of one another. In most real applications of the ultimate pit problem the negative valued blocks (waste) are at higher elevations than positive valued blocks (ore), so it is beneficial to imagine the source at the bottom and the sink at the top. This does have the unfortunate effect of making the normalized tree appear to be disconnected and not much like a tree, however when reasoning about the tree either consider both the source or the sink nodes as valid *main roots* or imagine an extra tree arc between the source and sink nodes making it a true tree.

The nodes which are immediately adjacent to either the source or the sink are the only nodes that can have some excess or deficit and are called roots (not to be confused with the main root terminology used by Hochbaum). Roots with excesses are said to be strong, and all of the nodes within their respective subtrees are also strong. Roots which satisfies the flow balance constraint, or ones that have a deficit, are said to be weak, and all of the nodes within their subtrees are also weak.

Finally each node in the pseudoflow algorithm has an associated label which is a non negative integer used to preclude certain sequences of merging operations that can negatively impact performance. Labeling is not strictly necessary for the operation or correctness of the pseudoflow algorithm but was included as a performance optimization. Therefore labels are not included in the notation nor the algorithm description in Sections 3.1.3 to 3.1.5, but are discussed separately in Section 3.1.7.

3.1.3 Pseudoflow Notation

Figure 3.4 introduces the new notation developed for the pseudoflow algorithm. This notation makes it far easier to understand how the pseudoflow algorithm operates and keep track of progress when completing iterations by hand. It is true that experienced practitioners will generally not perform pseudoflow steps manually and this notation will never be applied to problems that are even approaching a realistic size. However, similar to the way that the tableau method is useful to novice researchers learning about the simplex algorithm, this notation is useful to novice researchers learning about the pseudoflow algorithm.

On the left in Figure 3.4 the numbers on arcs are the current flow of the arc. If the arc does not have a number, then the flow is zero. The numbers inside nodes are the current excesses (when positive) and deficits (when negatives). If a number is omitted inside a node then it is zero and this means that the node satisfies the flow balance constraint.

In this notation the capacity is not indicated on any arc. This is because the capacities of all precedence arcs, arcs which do not connect to either the source or the sink, are infinity. And the flow along the value arcs, arcs which are connected to either the source or the sink, are always kept at their maximum capacity.

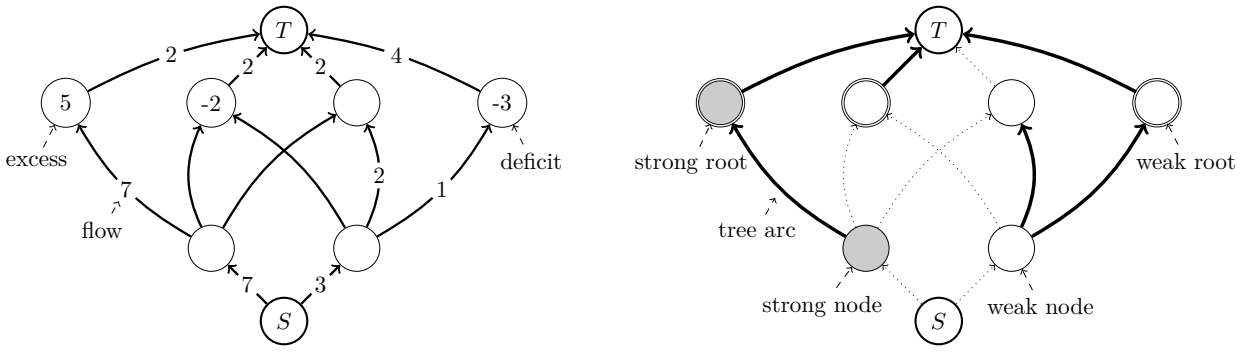


Figure 3.4 Left: The maintained pseudoflow on the flow network is notated with numbers on arcs for the flow, and numbers within nodes for excesses or deficits. Right: Thick arcs are a part of the normalized tree, and dotted arcs are not. Gray nodes are strong, white nodes are weak. Figure adapted with permission from Deutsch, Dağdelen, and Johnson 2022 [56]

This is one of the main departures from the conventional pseudoflow algorithm incorporated into this description. The conventional approach must keep track of, and continuously check, capacity information on all arcs because they are generally not the same value nor are they all

infinite. Additionally, in the conventional pseudoflow algorithm the flow on source and sink adjacent arcs is modified during the second stage of the algorithm where the maximum flow is recovered from the minimum cut - however this is not required. Once the minimum cut is identified so too is the ultimate pit and no additional work is necessary as the flow values are not the main goal in this application.

The normalized tree is notated by making its member arcs bold, and the arcs which are not a part of the normalized tree dotted or dashed. This is shown on the right in Figure 3.4. Nodes which are roots are notated with a double circle and contain either an excess or a deficit. Strong nodes are shaded gray and weak nodes are left unshaded. These two pieces of notation are then superimposed on top of one another during the execution of the algorithm.

3.1.4 Initialization

The first step in the pseudoflow algorithm is to construct an initial normalized tree and an initial pseudoflow. It is possible to start the pseudoflow algorithm from any normalized tree with a valid pseudoflow, but the simplest starting point is to fully saturate all source and sink adjacent arcs, and include all source and sink adjacent arcs in the normalized tree as in Figure 3.5. This creates an excess on each positive valued block, and a deficit on each negative valued block.

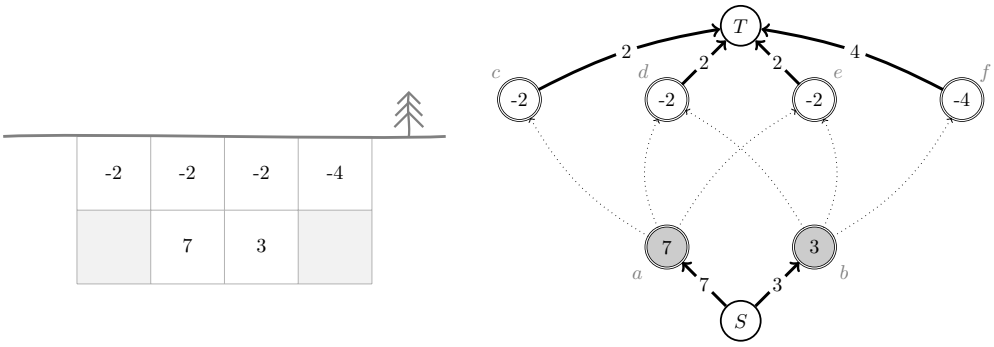


Figure 3.5 Left: The input ultimate pit problem. Right: The initial normalized tree, letters near nodes are the node names and not a part of the notation.

The slight modification to how capacity is handled in this implementation requires the pseudoflow to be initialized with maximum flow on the source and sink adjacent arcs. The nodes must also be initialized with a valid label, it is sufficient to set the label of the strong nodes to one and the label of the weak nodes to zero.

3.1.5 Algorithm Steps

At each iteration of the pseudoflow algorithm an arc corresponding to a precedence constraint between a strong node and a weak node is selected to be the *merger* arc. This merger arc is then introduced into the normalized tree and the arc between the strong root and the source or sink is removed. The pseudoflow is then adjusted so that the tree remains normalized. This requires adjusting flows along the path from the strong root to the week root, and is called a *merge*. During this merging process there may be tree arcs which must be removed from the normalized tree resulting in new branches with new associated roots which is called a *split*. Finally, when there are no more precedence arcs between strong and weak nodes the algorithm terminates and all the remaining strong nodes constitutes the ultimate pit.

There is no need to continue with the flow recovery step present in the original pseudoflow algorithm, because the ultimate pit is the sole goal of this approach. Recovering the max flow also does not have any side effects which could aid in re-initialization or solving similar problems.

In practice choosing which merger arc to use has a large impact on the performance of the algorithm. The pseudoflow algorithm uses a labeling scheme, discussed further in Section 3.1.7. These labels help guarantee reasonable performance and avoid several of the problems which plague the Lerchs and Grossmann algorithm. In this section, however, the labels are not used so that the small example demonstrates all of the necessary components of the pseudoflow algorithm.

Once the appropriate merger arc is chosen the algorithm identifies the strong root associated with the underlying strong node, and the weak root with the overlying weak node. The strong root will, by definition, have some positive excess. The normalized tree must then be updates as in Algorithm 2.

One of the departures from the conventional pseudoflow algorithm occurs during the walk from the strong root to the weak root. If the arc is directed in line with this path the flow along the arc must be increased by the current δ which intuitively corresponds to using currently available funds to pay for overlying negative valued blocks. These funds will then be 'spent' by the negative valued blocks by directing flow to the sink. In the more general case it would be necessary to ensure that this increase in flow does not lead to a capacity violation, however in the context of solving solely for the ultimate pit this check is not necessary because all of these arcs

Algorithm 2: The merge procedure in the modified pseudoflow algorithm, adapted from Hochbaum 2001 [46]

```
// Update the normalized tree
Remove the tree arc connected to the strong root;
Add the merger arc;
δ ← the excess of the strong root
for all of the arcs along the path from the strong root to the weak root do
    if the arc is directed in line with the path then
        Increase the flow on the arc by δ;
    else
        // Try to decrease the flow on the arc by δ
        if the flow is greater than δ then
            Set the flow the current flow less δ;
        else
            Split flow on this arc;
            // See following Algorithm for details on splitting
```

are precedence constraints with infinite capacity.

During the merge operation if an arc is oriented opposite to the direction of the path from the strong root to the weak root and the current flow on that arc is less than the currently available excess, δ, a split is required. If this happens it means that at some stage earlier in the algorithm a strong root became weak after supporting some of the nodes within the current strong root's cone of influence. Therefore the weak nodes which are currently being merged with the current strong root had, at one point, an underlying positive block supporting them and they may be connected inappropriately for this new step. The tree must be split into two subtrees at this location so that we avoid configurations where unnecessary negative valued blocks are included in a strong subtree. This allows for the two valid outcomes: the subtree remains weak and is correctly excluded from the ultimate pit, or the positive values blocks within the subtree are sufficient to support the negative valued blocks above them with the new 'help' following the merge.

At later stages of the algorithm, after many merge operations, there may be several splitting operations during the course of a single merge operation. The splitting operation follows in Algorithm 3.

When there are no longer any precedence arcs between strong nodes and weak nodes the algorithm terminates. Any nodes which are still classified as strong are then the ultimate pit, and the sum of their excesses is the ultimate pit value.

Algorithm 3: The split procedure in the modified pseudoflow algorithm, adapted from Hochbaum 2001 [46]

$\delta \leftarrow$ the flow along the splitting arc ;
The flow along the splitting arc $\leftarrow 0$;
`// Note that this leaves a positive excess at the head node`
`// Update the normalized tree`
Remove the split arc;
Add the arc from the head node to the root;
Continue with the merging operation using the new δ;

3.1.6 Example

This example serves to illustrate both the algorithm and the notation developed herein. The dataset for the example is very small, consisting of only six blocks. In order to force a splitting operation a specific sequence of merge steps are required as shown on the left in Figure 3.6.

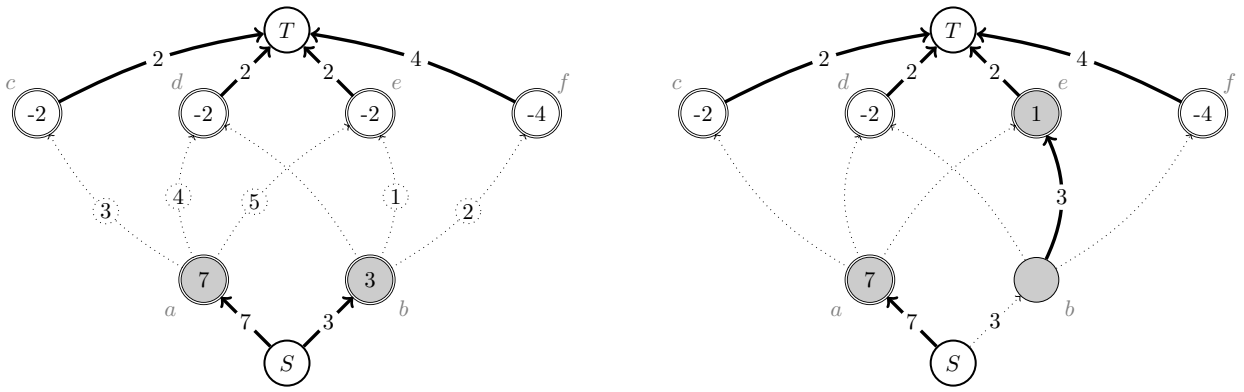


Figure 3.6 Left: The starting network, circled numbers on nodes indicate the order of merging arcs in this example. Right: The result of the first merge between nodes b and e. Figure adapted with permission from Deutsch, Dağdelen, and Johnson 2022 [56]

The result of the first merge, between b and e, is shown on the right in Figure 3.6. The strong root, b, remains strong following the first merge and the weak root, e, becomes strong.

The result of the second merge, between b and f, is shown on the left in Figure 3.7. This merge operation has the consequence of leaving the new root, f, with a deficit which reclassifies the entire subtree as weak. The third merge, between a and c, and the fourth merge, between a and d, are similar to the first merge and proceed with no complications. The result is included on the left in Figure 3.8.

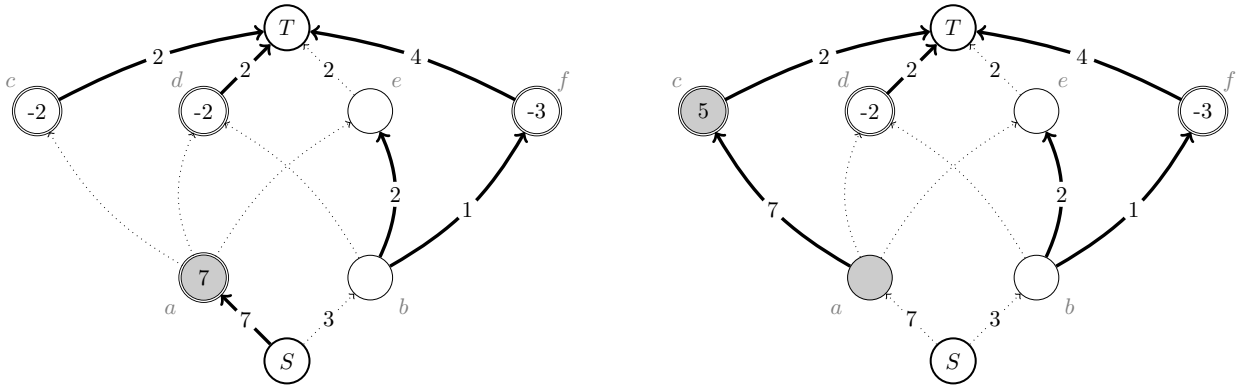


Figure 3.7 Left: The result of the second merge between nodes b and f. Right: The result of the third merge between nodes a and c. Figure adapted with permission from Deutsch, Dağdelen, and Johnson 2022 [56]

The final merge operation between nodes a and e begins by pushing d's excess along the path towards the week root. The flow between a and d is reduced by three, the flow between a and e is increased by three but then there is a problem. The flow between b and e should be reduced by three (which is the current value of δ), but this would lead to a negative flow which is not allowed. The flow, therefore, is reduced to zero leaving one unit of excess flow on e and splitting the tree.

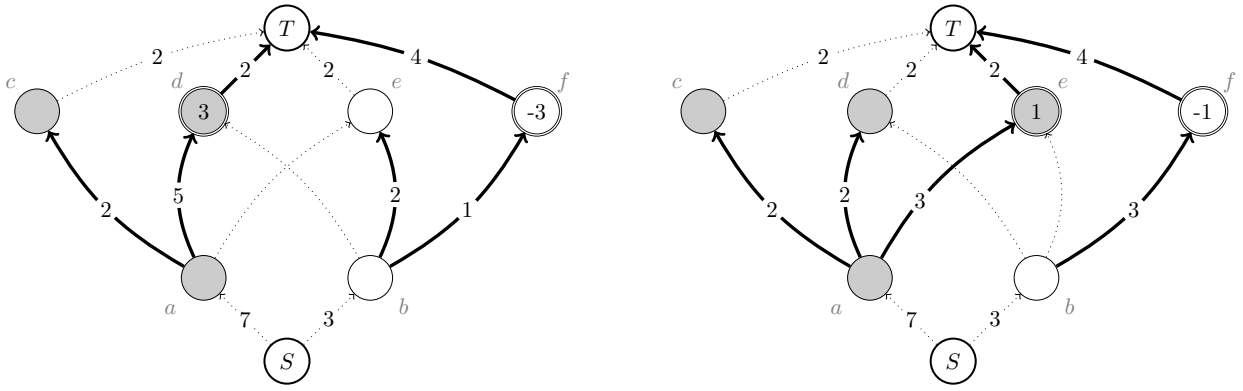


Figure 3.8 Left: The result of the fourth merge between nodes a and d. Right: The result of the fifth merge between nodes a and e which requires splitting on the arc between b and e. Figure adapted with permission from Deutsch, Dağdelen, and Johnson 2022 [56]

At this point there are no longer any strong nodes with overlying weak nodes and the algorithm terminates with the ultimate pit indicated by all remaining strong nodes. The sum of the excess across all of the strong roots is the value of the ultimate pit.

3.1.7 Labeling

Hochbaum describes a labeling scheme to improve the performance of the pseudoflow algorithm. Intuitively, the labeling scheme forces the algorithm to carefully choose the merger arc at each iteration; avoiding certain sequences of merger arcs which would require more iterations that necessary. Specifically, the labeling scheme makes it such that the merger arc between nodes s and w cannot be used again until the labels on both s and w have increased by at least one.

The original labeling scheme in [54] is as follows: All nodes are initially assigned a label of one. When selecting a merger arc between a strong node s and a weak node w the algorithm *must* select an arc such that the label of w is as low as possible. Once the merger arc is selected, the label of all strong nodes is set to the maximum of its current label and $l_w + 1$. These straightforward steps are enough to improve both the theoretical complexity of the pseudoflow algorithm and the practical performance.

One aspect of this labeling scheme which is not ideal is that once a merger arc is selected this may trigger a relabeling operation across all strong nodes, of which there could be many hundreds of thousands. This is not strictly necessary. Chandran and Hochbaum [71] provide an alternate labeling scheme that delays relabeling and improves the computational efficiency. MineFlow uses the more performant delayed labeling scheme which is also present in Hochbaum's implementation which allows for strong nodes to have different labels throughout the execution of the algorithm.

The labeling scheme primarily becomes important in large problems because it can vastly limit the number of possible merger arcs available at any iteration while simultaneously ensuring that those merger arcs will have a substantial effect. This is because labeling encourages connecting strong nodes to weak nodes that haven't been considered yet. In the conventional Lerchs and Grossmann algorithm there is no guidance on which precedence constraint should be considered, so there is no protection from undesirable sequences. Many Lerchs and Grossmann implementations continuously loop over *all* precedence constraints testing if they are between a strong and weak node until a full loop is completed with no changes. Only then is the algorithm terminated. However, with labeling incorporated this is not necessary and more efficient stopping criteria are available.

MineFlow, and Hochbaum's implementation, divide strong roots into a set of buckets which are differentiated by their label number. At each step a strong node is selected from the bucket with the *highest* label number, and considered for possible merger arcs. It is possible to select a strong root with the lowest label number instead but experimentally this yields poorer performance. Once no more strong roots are available the algorithm terminates.

If at any stage during the algorithm a strong root is evaluated and no overlying weak nodes exist then this strong root and its entire subtree are removed from future consideration. Because the tree is normalized, and there can be no 'hanging' weak nodes either, this is a valid and important practical optimization.

An example showing how labeling works to preclude undesirable merges is included in Figure 3.9. The value of block c is reduced from -2 to -4 compared to the example in section 3.1.6 and the sequence of merger arcs is modified as shown on the left. Following the first merge operation between nodes a and e the labels of a and b are increased to two. The state of the network after the first three merges is shown on the right in Figure 3.9. At this point nodes a, b, d and e all have a label of 2 and the only allowed merge is between b and f - because f has the lowest available label. This merge will lead to the immediate termination of the algorithm as all blocks will become weak and no strong nodes will remain. If the labeling scheme were *not* used then the next merger arc could be between b and e or b and f. Either of these choices would require several additional iterations before the correct answer is identified.

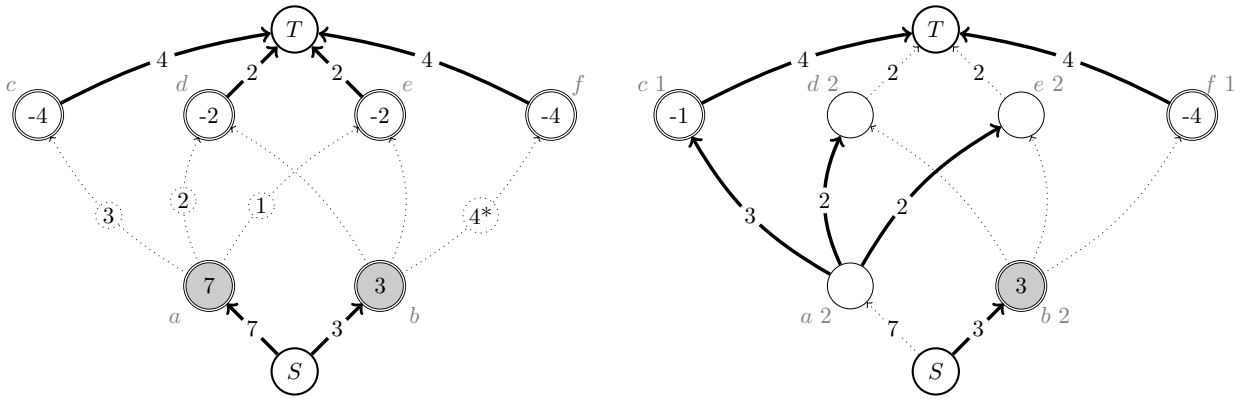


Figure 3.9 Left: The modified example with a different sequence of merges as numbers on arcs. Right: The network after three merges. Labels are given as numbers next to the node names. Figure adapted with permission from Deutsch, Dağdelen, and Johnson 2022 [56]

3.1.8 Pseudoflow Complexity

The modifications developed in the preceding sections do not alter the computational complexity of the pseudoflow algorithm as originally developed by Hochbaum. They do, however, have an impact on the practical performance (Section 3.3). The computational complexity is important, as it dictates how the algorithm is expected to behave as the problem size grows. Mine planning engineers are often using larger and larger block models with more sophisticated geometrical requirements in order to represent as much of reality as possible in their mine models and facilitate improved decision making. Therefore, it is useful to understand how the algorithm might respond to larger models.

Hochbaum, in 2008, showed that the labeling pseudoflow algorithm, with integer block values, has a complexity of $\mathcal{O}(mn \log n)$ where m is the number of arcs and n is the number of nodes [54]. The complexity can be improved to $\mathcal{O}(n^3)$ and even $\mathcal{O}(nm \log \frac{n^2}{m})$ by incorporating specific data structures [55]. Some of these data structures can be tricky to implement in a manner that their higher constant time requirements are outweighed by their lower computational complexity. The applicability of these more sophisticated data structures was not evaluated.

3.2 The MineFlow Implementation

MineFlow is implemented as a C++ library which exposes a few straightforward classes for defining precedence graphs and computing ultimate pit limits with the modified pseudoflow algorithm. Additionally a very simple command line executable is provided for smaller explicit datasets, or more simple datasets with regular block models.

There are three main components to an ultimate pit optimizer: The block values, the precedence constraints, and the solver itself. Each of these components was carefully designed to obtain the highest degree of performance and lowest memory requirements.

3.2.1 Block Values

The block values used for the ultimate pit problem are calculated from a wide range of input variables and parameters as in Section 2.1.3.3. In MineFlow this calculation is assumed to have been done prior to invoking the library or executable and is considered outside of its scope.

The arithmetic required while solving the ultimate pit problem is very simple; block values are only added, subtracted, and compared with one another or zero. No multiplication, division, or other more sophisticated operations are required but a careless implementation can still lead to issues. The core arithmetic must be precise.

Floating point arithmetic, as typically implemented on modern computers, is inadvisable for the max flow problem. When using floating point numbers there is the potential for loss of precision and even entering an infinite loop. A very simple network which exhibits this behavior is given in Althaus and Mehlhorn 1998 [134]. Additionally, Hochbaum's original developments regarding the computational complexity of the generic pseudoflow algorithm rely on integer capacities to ensure either that the total excess of the strong nodes is strictly reduced or at least one weak node becomes strong [54]. This is used to show that the algorithm always terminates in a finite number of steps.

Therefore block values in MineFlow must be provided as integer values. This is not typically a concern for mining engineers, practitioners, and other end users of the library because input values can be multiplied by a positive constant and rounded to an integer value. One practical consideration is that the integer values should not be so large as to potentially lead to overflow. The default data type for block values in MineFlow is a signed 64 bit integer however the GNU Multiple Precision Arithmetic Library can also be used [135]. This library provides a datatype which is an arbitrarily large integer limited only by the computer's available memory. This precludes overflow but has a negative impact on performance as each individual operation is slower.

A fundamental tenet of MineFlow, which is further developed in Section 3.2.2, is to minimize expending effort on steps which are not necessary. A more conventional ultimate pit optimizer would generally collect *all* of the block values, define *all* of the precedence constraints, and then begin solving for the ultimate pit. However in most real world datasets only a small fraction of the block values and precedence constraints are ever used. So it is inefficient to be so pedantic in the implementation. MineFlow must only know all of the positive valued blocks at the onset of the optimization procedure as it is only the positive valued blocks, and their antecedents, that could be included in the final answer.

The block values are defined as a list of positive block identifiers, their values, and a *function* which can be queried for other block values as necessary. This even allows the caller to use MineFlow in cases where the entire block model is not fully defined and avoid the potential for inappropriate edge effects when the pit extends beyond the original block model limits. If the total number of possible blocks is known then it can be provided to MineFlow to avoid having to use a hash map between block identifiers and nodes within the precedence graph. This can lead to a decrease in runtime.

3.2.2 Precedence Constraints

Precedence constraints are defined on a per block basis as a list of other block identifiers. It is very important not to generate all precedence constraints because they are not all necessary and it wastes a substantial amount of time. In MineFlow precedence constraints are defined via a user provided callback function that when given a base block identifier returns a, possibly empty, list of antecedent block identifiers. This allows the solver to only request precedence constraints as necessary and is responsible for much of the speed improvements in this library over the commercial implementations which often generate all of the precedence constraints upfront.

Because the solver only considers the user provided block identifiers to define precedence constraints it naturally supports subblocks or other irregular block models. The only restriction is that those user provided precedence constraints do not form a cycle. This restriction is not enforced by the library because it would take extra time to check for cycles and is unlikely to occur in real applications. The higher level routines which are used to define precedence constraints will generally prevent cycles.

Users do not expect to provide precedence constraints as lists of other blocks, but instead prefer to use their geometrical information directly. In practice precedence constraints vary by both location and direction following geometrical constraints and are often specified by a list of azimuth slope pairs on a per block basis, Figure 3.10. Many blocks often share the same azimuth slope pair list because they are considered to be members of the same geotechnical zone. The pit slope between the given azimuths must be interpolated. MineFlow provides linear and cubic interpolation for this purpose.

Azimuth	Slope
0°	30°
45°	40°
90°	45°
135°	45°
180°	45°
270°	40°

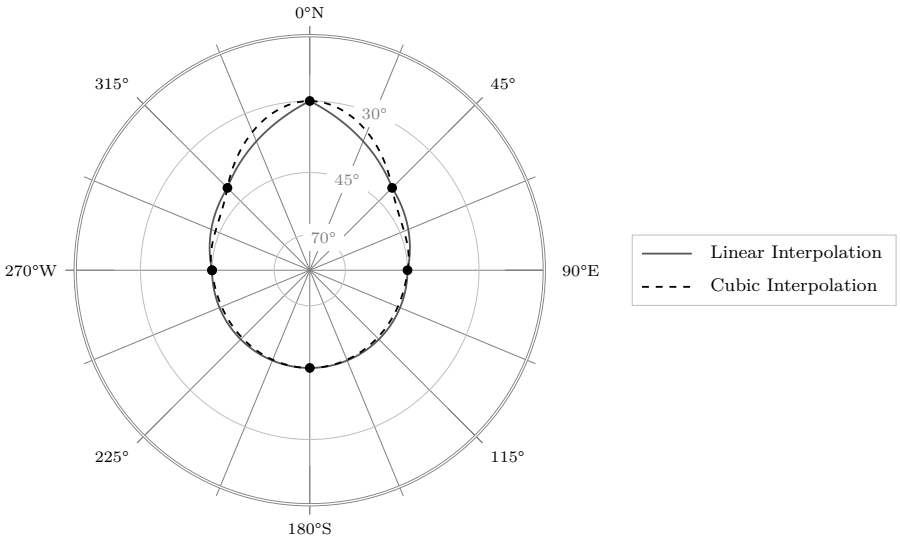


Figure 3.10 An example slope definition with six azimuth slope pairs. Linear and cubic interpolation for unspecified directions is shown.

If a slope definition, as a list of azimuth slope pairs, is used with an irregular block model then the list of antecedent blocks for any given base block is not necessarily easy to determine. Sorting the block coordinates or using an appropriate acceleration structure, such as an R-tree, may be appropriate in these circumstances. However, most of the time ultimate pits are calculated using regular block models and pre-computing the set of antecedent blocks for a given slope definition is warranted. MineFlow implements the minimum search pattern paradigm from Caccetta and Giannini, introduced in Section 2.1.3.4.

To define a minimum search pattern the slope definition is required along with the block dimensions and a maximum offset in the z direction. The routine will then determine the smallest set of offsets that accurately recreates that slope definition for that block model relying on the transitive nature of precedence constraints in order to include all necessary blocks.

An example minimum search pattern is included in Figure 3.11. This is a plan view of a regular block model with cubical blocks where the numbers inside blocks indicate the z offset. The central square is connected to the five blocks immediately above in a cross pattern, but to recreate 45° pit slopes it is necessary to include additional blocks at higher elevations. The important feature of this pattern is that it has the fewest number of blocks possible which corresponds to far fewer precedence constraints.

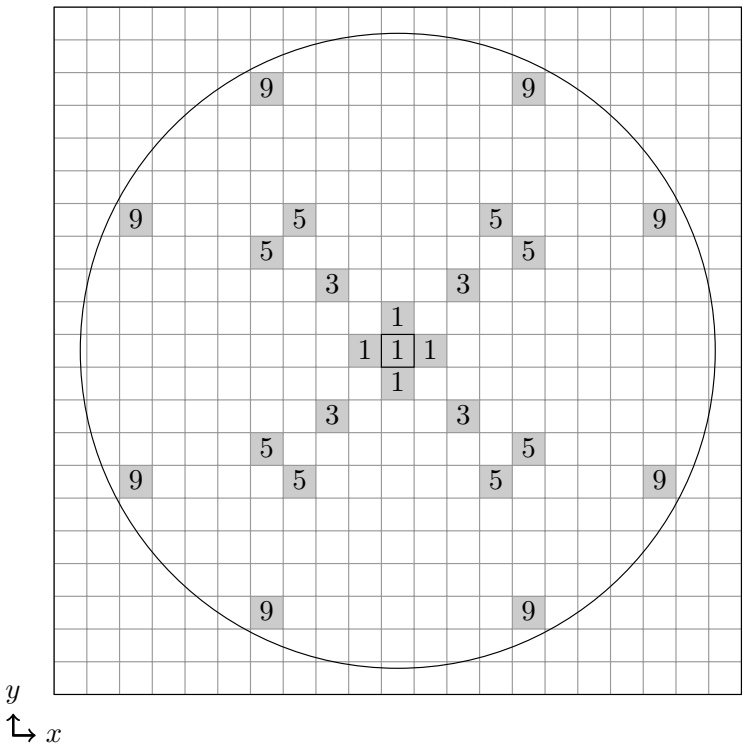


Figure 3.11 An example minimum search pattern for 45° slopes and a maximum vertical offset of 9 blocks. Numbers in cells are the z offset.

Determining the appropriate maximum z offset has historically been a point of concern. If the maximum offset is too high then there will be more precedence constraints and the optimization will take longer. However, if the maximum z offset is too low then the precedence constraints will not be accurately represented in the ultimate pit which could even have safety implications. The library developed in this chapter takes three steps to address this concern:

- Using a performant pseudoflow based solver with lazily generated precedence constraints and all the theoretical and practical improvements developed in this chapter makes it so that the solver is far less sensitive to the number of precedence constraints. Certainly the number of precedence constraints should be minimized, but solvers with less efficient implementations will have a multiplicative slowdown as more precedence constraints are used.

- Minimum search patterns also minimizes the impact of a higher maximum z offset. For example, a 45° minimum search pattern with isometric blocks only includes new precedence constraints at z offsets of 1, 3, 5, 9, 13, 17, 19, and 25, up to 25. That is, several maximum

z offsets are 'free' because the minimum search pattern does not need any additional blocks to accurately capture all required blocks.

- And finally a method for evaluating the *accuracy* of a given precedence pattern for a given block model is developed.

The accuracy of a given precedence pattern is a measure of how close the pattern comes to achieving the true set of precedence constraints, and efficiency is a measure of how wasteful the pattern is. A practitioner must decide on how to trade off computational effort versus accuracy, and always wants the most efficient precedence pattern. Efficiency, in this context, is already maximized by virtue of using minimum search patterns, but accuracy requires additional effort to define and measure. Note that the true set of precedence constraints is determined by simply connecting the base block to every overlying block that is within the provided slope constraints.

The accuracy of a precedence pattern is dependent on the size of the block model on which it will be used. Again continuing with the isometric block model and 45° slopes if the number of benches is unrealistically low then it may be acceptable to connect each base block to only the five blocks above in a cross pattern, the 1:5 pattern. However, this very quickly becomes unacceptable as the number of benches increases because when this is extended over several benches the resulting pit looks comically unrealistic; Figure 3.12.

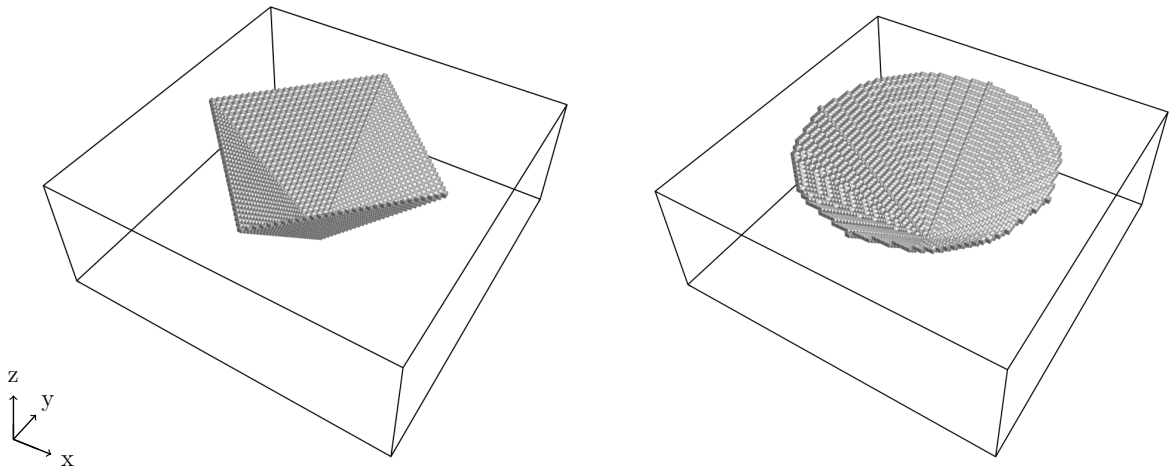


Figure 3.12 Left: The 'one-five' precedence pattern extended 30 blocks vertically. Right: the true set of antecedent blocks for a single base block

A numerical method of measuring accuracy is preferred. For this purpose Matthew's correlation coefficient is a useful measure [136, 137]. Matthew's correlation coefficient is defined as follows in Equation 3.1.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \times \frac{1}{2} + \frac{1}{2} \tag{3.1}$$

Where MCC is Matthew's correlation coefficient and TP, TN, FP, and FN are the counts of the true positive, true negative, false positive, and false negatives respectively. This quantity ranges from 0 to 1 with 1 representing perfect agreement between observed and predicted results and 0 when there is perfect disagreement. This measure is preferred over the more conventional accuracy measure in many application as the balance ratios of the confusion matrix categories are taken into account. That is, in our context, the number of true negatives can be very large and this really shouldn't be given as much weight as the false negatives or false positives.

To generate the confusion matrix an empty block model of the appropriate size is constructed and all blocks are classified as true negatives. The lowest central most blocks is identified and classified as a true positive. From this block the true antecedents are identified by evaluating all overlying blocks against the given azimuth slope pairs, initially classifying all of these blocks as false negatives. Then the pattern is repeatedly applied starting from the initial block reclassifying false negatives as true positives and true negatives as false positives. Finally, the counts of each type of block are calculated and Matthew's correlation coefficient is determined.

With this approach it is possible to numerically quantify the impact of a given maximum z offset for blocks models of a specific size, and also compare the minimum search patterns with the other precedence patterns. The results of a simple evaluation of this nature is included in Figure 3.13. This evaluation continues with a regular isometric block model with 45° slopes, and considers the Knight's move pattern discussed in Section 2.1.3.4. Along the y axis is the calculated Matthew's correlation coefficient for a block model with a given number of benches, across the x axis of the figure. Each line corresponds to a particular precedence scheme, the bolded line corresponds to the Knight's move pattern and the others are all minimum search patterns of various maximum z offsets.

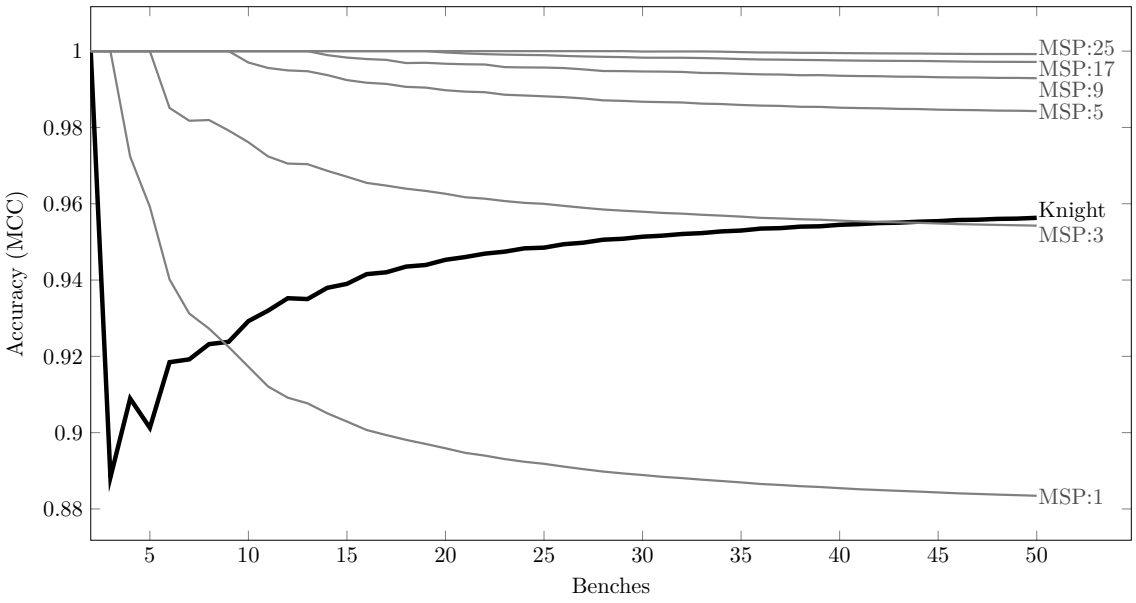


Figure 3.13 The slope accuracy of several minimum search patterns when used for block models with the indicated number of benches.

The performance of the Knight's move pattern is fundamentally different than the minimum search patterns because it connects the base block to some blocks with less than 45° slopes. This causes some false positives whereas the minimum search patterns only ever has false negatives. Therefore, the minimum search patterns will only ever get worse as the number of benches increases, although only very slightly.

Most realistic block models have around fifty to seventy benches and the pits very rarely reach the bottom of the block model. So it seems appropriate to use 17 as a starting max z offset which connects each base block to 45 overlying blocks, and potentially increase the maximum z offset to 25 (using 61 overlying blocks) in some circumstances.

3.2.3 Pseudoflow Solver

The pseudoflow algorithm as described in Section 3.1.5 translates relatively straightforwardly into an actual C++ implementation although some care must be taken. There are two main data types; nodes and arcs, which are stored in two different types of containers called the node pool and arc pool. The pseudoflow graph itself contains both pools and some bookkeeping information which keep track of how many nodes exist with a given label and the buckets of strong roots differentiated by label.

Each node contains its value, a pointer to the arc which leads towards the source or sink (this forms the normalized tree), the nodes current label, pointers which form a linked list of descendants, information regarding that node's precedence constraints, the original block index, and a pointer to the original root adjacent arc. This is a relatively large amount of information for each node so nodes only need to be generated as necessary in order to minimize memory use - additionally nodes are only ever created during the course of the algorithm (as new antecedents are required) and not removed. However, during testing it was found that preallocating all of the nodes had a positive impact on performance despite the higher memory use. This preallocation is preferred unless the library user desires the lighter implementation.

Arcs contain two pointers to their head and tail (null if this would go to the sink or source respectively), and the current flow along the arc. The capacity is not included because it is not necessary. Arcs are kept within a large object pool which also maintains a free list of available arcs that can be re-used. Arcs are created when merging a weak node with a strong root and removed during the split procedure.

Beside the node and arc pools the graph maintains the buckets of strong roots as an array of queues and the label counts within another array. These are relatively small and although vital to determining the next strong root to process are not hugely important. In general most components of the pseudoflow solver are kept as simple as possible in order to be as fast as possible. Facilities are included for reporting various statistics such as the complete elapsed time and how many merge and split operations occurred.

3.3 Computational Comparison

This chapter has focused on the ideas and implementation details behind an improved ultimate pit solver that professes to being both correct and more computationally efficient than commercially available alternatives. This claim must be supported by evidence. Five block models were collected ranging in size from 374,400 blocks to 16,244,739 blocks and imported into five different commercial software packages with ultimate pit optimizers. Each problem was also transformed into the pure max flow format in order to compare with Hochbaum's original implementation. In all cases MineFlow computed the ultimate pits in less time.

For each package and each problem five runs were completed. The solution times reported in Table 3.2 are given as the average of the middle three times, excluding the fastest and the slowest in an effort to minimize the effect of other processes. All computations were completed on a Windows 10 machine with a 3.70GHz Intel Xeon XPU E3-1245 v6 processor with 32 GB of available RAM. The times were measured as the complete time as reported by the package. This necessarily includes the time to read the input and write the output which can vary between packages, especially because most commercial packages use some proprietary binary format for their block models.

The precise number of mined blocks and total pit value varies by less than 1% between the commercial packages due to what appears to be discrepancies in exactly how precedence constraints are handled. The reported number of precedence constraints in the table is the total number as generated by MineFlow from an exhaustive search across all blocks. All of these precedence constraints are given to Hochbaum's `pseudo_fifo` program, because it is a general utility for solving the max flow min cut problem and does not generate precedence constraints. Notably package C also computed identical results to MineFlow for all datasets even though it generated its own precedence constraints. The commercial packages did not report the number of precedence constraints that they used.

The commercial packages are anonymized to respect the wishes of some of the software vendors. All five packages use a network flow based implementation however the precise details are not published or publicly available. One package uses the Push-Relabel algorithm and the other four report to use some variant of Pseudoflow.

For the smallest dataset MineFlow reports a solution time of zero seconds. This is because the entire problem solves in less than 500 milliseconds.

The 'Copper Pipe' dataset is interesting, because it is the same size as the McLaughlin dataset (approximately 3 million blocks) but often solves much slower in the commercial packages. Specifically with package D where the McLaughlin dataset solves in 34 seconds and the Copper Pipe dataset solves in four minutes and fifty seconds. This dataset, as the name suggests, consists of a large vertical porphyry which seems to pose problems for many solvers. This is potentially because many more nodes need to be explored to classify branches as weak compared to other datasets.

Table 3.2 The solution times obtained when solving for the ultimate pit for the five different datasets with seven different solvers. Times are given in seconds as the average middle three of five runs.

Dataset Name	Mined / Total Blocks	Precedence Constraints	Pkg. A	Pkg. B	Pkg. C	Pkg. D	Pkg. E	`pseudo_fifo`	MineFlow
Bauxite	74,412 / 374,400	5,349,104	25	22	4	9	9	3	0
Copper Study	357,304 / 1,827,500	28,321,632	179	99	14	85	43	15	2
Copper Pipe	198,078 / 2,754,000	43,877,152	399	183	64	290	109	24	3
McLaughlin	345,936 / 2,817,920	44,495,400	243	168	37	34	52	24	3
Gold Vein	602,150 / 16,244,739	264,007,172	721	764	169	752	556	81	9

3.3.1 MineLib Results

MineLib is a library of eleven publicly available test problem instances for three classical open pit mining problems including the ultimate pit limit problem [138]. A small wrapper around MineFlow was developed to adapt the MineLib format and compute the ultimate pit solution using MineFlow.

The high performance server, `isengard`, at the Colorado School of Mines was used to solve all eleven problem instances. The server has 48 Intel(R) Xeon(R) E5-2690 v3 @ 2.60GHz processors and 377 Gigabytes of RAM, although not all of this computing power was used exclusively for this comparison. In all instances the results as computed by MineFlow were identical to those reported by Espinoza et al. The problem instances and elapsed solving time, in milliseconds, are tabulated in Table 3.3.

Table 3.3 Summary information applying MineFlow to the Minelib '`upit`' problem instances.

Name	Number of Blocks	Number of Precedence Constraints	Elapsed solving time
`newman1`	1,060	3,922	2ms
`zuck_small`	9,400	145,640	17ms
`kd`	14,153	219,778	20ms
`zuck_medium`	29,277	1,271,207	95ms
`p4hd`	40,947	738,609	59ms
`marvin`	53,271	650,631	18ms
`w23`	74,260	764,786	106ms
`zuck_large`	96,821	1,053,105	190ms
`sm2`	99,014	96,642	40ms
`mclaughlin_limit`	112,687	3,035,483	291ms
`mclaughlin`	2,140,342	73,143,770	489ms

The precedence constraints for the MineLib problem instances are fully specified in a flat text file and therefore MineFlow is unable to take advantage of its ability to lazily generate precedence constraints. This does not seem to greatly increase computation time on these smaller problem instances.

3.4 Discussion

This chapter has focused on the development of a fast, memory efficient, implementation of Hochbaum's Pseudoflow algorithm. The algorithm has been slightly specialized for the ultimate

pit problem; taking advantage of the infinite capacities on precedence arcs and the ability to stop as soon as the minimum cut is identified. The implementation developed herein is more performant than the tested currently commercially available implementations.

Additionally this chapter discussed the importance of lazily generating precedence constraints and placing the responsibility for generating the constraints within the solver itself. Having the solver ask for precedence constraints as needed, instead of requiring them all up front, is an important optimization that is responsible for much of the speed improvements.

The notation developed in this chapter may also be used to introduce practitioners to the pseudoflow algorithm. And the implementation is open source and available with minimal restrictions for academics and commercial users alike.

Finally a means by which the accuracy of a given precedence arc template can be determined was developed.

CHAPTER 4

THE ULTIMATE PIT PROBLEM WITH MINIMUM MINING WIDTH CONSTRAINTS

A series of viable approaches to the ultimate pit problem with minimum mining width constraints are presented in this chapter. The minimum mining width is a fundamental operational constraint and is unfortunately commonly left as an afterthought for the design engineer drafting the polygonal pit designs. Incorporating it directly into the ultimate pit problem is of utmost importance in ensuring that pit designs accurately reflect how they will eventually be developed and produced. Disregarding operational constraints early in the mine planning process leads to overestimating the value of a mineral deposit and can cause costly suboptimal decisions.

Section 4.1 introduces the concept of minimum mining width constraints and addresses two flawed approaches for incorporating them into open-pit designs. These methods are natural and easy to conceptualize, but suffer from drawbacks when considered carefully.

Section 4.2 describes the formulation and the framework within which all of the subsequent approaches are developed. The precise form of minimum mining width constraints considered in this dissertation is documented along with relevant extensions and modifications.

Section 4.3 is a brief diversion which considers a two-dimensional simplified version of the ultimate pit problem with minimum mining width constraints and develops an extension to Lerchs and Grossmann's original dynamic programming algorithm. This extension incorporates minimum mining width constraints and addresses a minor error in the original description.

Section 4.4 presents the different solvers which take in an input problem and return a pit that satisfies minimum mining width constraints. This section describes straightforward solvers which are geometric, and do not consider economic block values, alongside solvers which are full optimization based approaches.

Section 4.5 discusses methods to reduce the problem size by identifying both inner and outer bounding pits. An inner bounding pit serves as a baseline solution with a positive contained value such that these blocks will be contained within the final pit limits. An outer bounding pit is a set of blocks that necessarily contains the optimal answer. Practically these methods are important when full 3D models from real mines are used.

The methods considered in Section 4.4 vary considerably in terms of solution quality and required time to compute the solution. Section 4.6 presents a comparison of the methods and identifies the current best method which is most applicable to large models with dozens of millions of blocks and hundreds of millions of implicit constraints.

4.1 Minimum Mining Width Preliminaries

Modern open-pit mines are developed using large machinery that require a minimum amount of operating area to perform their tasks in a safe and effective manner. This operating area should be incorporated into the design process at an early stage as a constraint so that any resulting designs are usable with only minor modifications. Ignoring these operational constraints will necessarily lead to designs which overestimate the value of the deposit and do not provide adequate information for downstream decision making and design.

There are two natural approaches to incorporating minimum mining width constraints that are often considered as potential solutions. The most common approach is to ignore minimum mining width constraints, solve for the ultimate pit, and then modify the result manually. Another approach is to try to reblock the model to use larger blocks that inherently satisfy the minimum mining width constraints. Neither of these approaches are ideal.

4.1.1 Manually Enforcing Minimum Mining Width Constraints

Many practitioners faced with creating an open-pit mine design that is operational do not have the tools to do so directly. However, they generally do have a conventional ultimate pit optimizer. It is natural to solve for the ultimate pit and then try and manually modify that result into an operational design.

The most common method of doing this is to display the ultimate pit on planar sections and then draw polygons around the mined blocks which are of an adequate size. Most CAD packages have some additional tooling for this purpose and will often add extra elements on screen, such as a user defined circle with a radius equal to the operating width that follows the cursor. This process, in cartoon form, can be seen in Figure 4.1.

The bench section displayed in Figure 4.1 is at a low elevation in the ultimate pit, which is where minimum mining width violations occur. The area currently being contoured on the left

side of the figure is of sufficient size so very few modifications are necessary, however the northern collection of blocks is likely to be excluded from the manually cleaned design regardless if that is the correct decision or not. This small spur of blocks could be included in the final design by mining additional blocks to the east or west, or some combination of expansion or contraction to achieve the highest value. Most practitioners manually incorporating minimum mining width constraints will not evaluate all of the possible alternatives and instead use their best judgment. Similarly the collection of mined blocks in the north east of the section which currently consists of three small collections of blocks is problematic. Some of the blocks could be included in a mineable polygon but it is unclear how best to proceed from only reviewing the display.

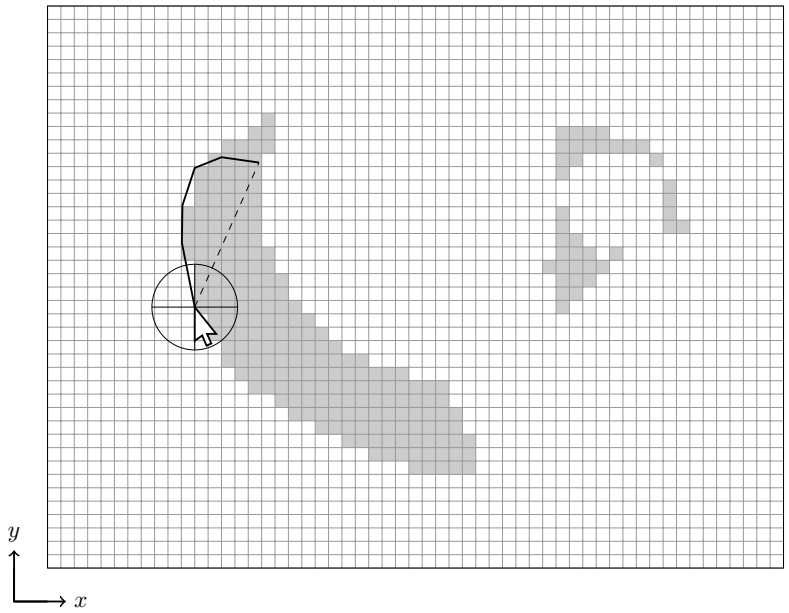


Figure 4.1 Manually contouring a block section. Shaded blocks are included in the ultimate pit.

Many open-pit designers will lean towards removing unmineable collections of blocks instead of incorporating additional waste. This is because when that freshly included waste is near the bottom of the pit it will often require extracting more waste higher up due to precedence constraints. However those blocks which are mined at the bottom of the pit are exactly the positive blocks which pay for the blocks above them. Excluding them will necessarily reduce profit which may have been better used to pay for additional waste blocks. Removing unmineable blocks at the bottom of the pit is not always the wrong thing to do. However, trying to incorporate minimum mining width constraints manually is unlikely to yield the optimal result.

Another issue with manually enforcing minimum mining width constraints is that it is very easy to miss non-obvious improvements to contained value. The example two-dimensional ultimate pit model with a two block minimum mining width in Figure 4.7 demonstrates this issue. In this example a practitioner might remove the lowest block (of value three) to satisfy a two block minimum mining width constraint, but this is overly conservative and misses out on expanding the pit at a higher elevation towards the right-hand side of the section. By expanding the pit an additional block, of value two, near the top right can also be mined for profit and help pay for the mining width. In realistic 3D models these sorts of possibilities for recouping value used to enforce minimum mining width constraints will almost certainly be missed unless more formal optimization methods are used.

Finally, this manual process takes a long time and must be repeated for each and every pit that is to be evaluated. If a series of calculated pits are to be used in the context of uncertainty analysis or as a part of a sensitivity study this manual process is unusable. A computerized optimization approach to enforcing minimum mining width constraints is desired.

One advantage of enforcing minimum mining width constraints manually is that the process can be combined with enforcing several other objectives that are hard to quantify or hard to model. An open-pit mine design is not complete without incorporating bench access into the design by adding necessary ramps and access roads. Additionally other operational or geometrical constraints might be challenging to capture in the optimization model but straight forward to incorporate by an experienced designer. In these situations enforcing minimum mining width constraints might be one of a myriad of other manual constraints that must be considered, and when considered in conjunction with these other objectives might not be a great concern. However, Even in these circumstances having minimum mining width constraints already satisfied will necessarily improve the subsequent manual design process.

4.1.2 Re-blocking

Block models, as discussed in Section 2.1.3.1, are the primary means of modeling open-pit mine designs and form the basis for the ultimate pit problem with or without minimum mining width constraints. Geologists, geotechnical engineers, and mining engineers will generally work together to balance the block sizes in order to accurately represent the deposit geology and

facilitate open-pit mine planning. These block sizes are invariably smaller than the required operating width, which can be quite large, and when solving with conventional methods that consider each block independently; the results will not satisfy minimum mining width constraints. A natural suggestion for addressing this issue is to use larger blocks that, when mined independently, would form an operational design.

These larger blocks could be used by the geologists and geostatisticians in the earlier modeling phase as well, however due to volume variance and dilution concerns this might not be desired. An alternative is to take the block model with smaller blocks and re-block it to a model with larger blocks that combines blocks together using the appropriate means. In a re-blocking procedure the economic block value variables should generally be summed together to form the larger block, whereas any grade variables may need to be averaged. However there are several downsides to using a re-blocked model.

Minimum mining width constraints are planar in nature and rely on ensuring that equipment and operations have enough space laterally to perform their tasks. There is no vertical component to a minimum mining width constraint, but re-blocking an input block model to contain blocks that are laterally expansive but still only as tall as the bench height will cause two problems. First it is very difficult to correctly represent precedence constraints on blocks that have abnormal aspect ratios. A bench may be 10 to 20 meters tall, but a typical operating width could be on the order of 50 meters. While it is possible to create precedence constraints between blocks of this shape, recreating the usual pit slopes of 40-50 degrees will not lead to desirable pits. Base blocks will be connected to blocks that are immediately above them, but blocks that are offset in the x and y directions can only be included after several benches without enforcing pit slopes that are far too shallow.

Additionally, it is not clear exactly where to start the re-blocking in order to achieve the best result. There is no innate reason to start re-blocking from the frontmost, leftmost block, and this may not be the best location. A different origin for the re-blocked model might lead to a better result, and this choice should be evaluated. In computing the term 'aliasing' is often used to describe the jagged appearance of curved or diagonal lines on low-resolution displays which is similar to this problem.

Re-blocking to blocks that are larger than the bench height in the z direction is possible and somewhat avoids the issue with representing precedence constraints. However the aliasing issues remain, but now increased to the z dimension. Finally, block models with larger blocks can incorporate undesirable dilution, and are limited to rectangular operating areas.

If at all possible it is better to avoid re-blocking to satisfy operational constraints. An approach that instead enforces small collection of blocks in operational shapes is preferred. In this fashion precedence constraints remain unchanged, there is no unnecessary dilution, and the operating areas can be constructed in the most appropriate shape for any given mine.

4.2 The Ultimate Pit Problem with Minimum Mining Width Constraints

A suitable formulation of the ultimate pit problem with minimum mining width constraints must satisfy three required criteria. These criteria are:

1. The minimum mining width constraints must not interfere with the aims of the original ultimate pit optimization. The results must still satisfy precedence constraints and must still maximize the total value of the mined blocks.

2. The minimum mining width constraints must adequately capture and model the real world concept of ensuring adequate operating area for heavy machinery in all areas of the open-pit mine. They should be flexible enough to accommodate the typical operating widths that are encountered in real world problems.

3. And finally, the minimum mining width constraints must be suitable for problems that are of a real world size. That is, they should be applicable to situations where the total number of blocks is in the tens or even hundreds of millions and there may be hundreds of millions of precedence constraints, and now hundreds of millions of minimum mining width constraints.

These criteria guide the formulation towards being as simple as possible and to being an extension to the original ultimate pit problem. The main formulation in this chapter is the maximum satisfiability formulation in Deutsch 2019 [121], but is presented as an integer programming problem in order to facilitate certain extensions and several of the solution methods in the subsequent sections.

The main idea is to represent minimum mining width constraints as arbitrary sets of blocks that when mined together form a valid operational area. This allows for complete flexibility in specifying operational areas, that can even vary throughout the mined area. Additionally, as these mining width sets are generally of a very similar form they can be defined implicitly, and do not need to be explicitly generated. This is similar to how precedence constraints are handled.

Typical operational areas consist of small contiguous collections of blocks. It is generally more efficient to specify minimum mining width constraints with a 'template' of blocks instead of on an individual per-block basis. Several example templates are shown in Figure 4.2. These templates collect several blocks in the x and y dimensions of the block model into small mineable groups that, when all mined together, represent a valid operational area.

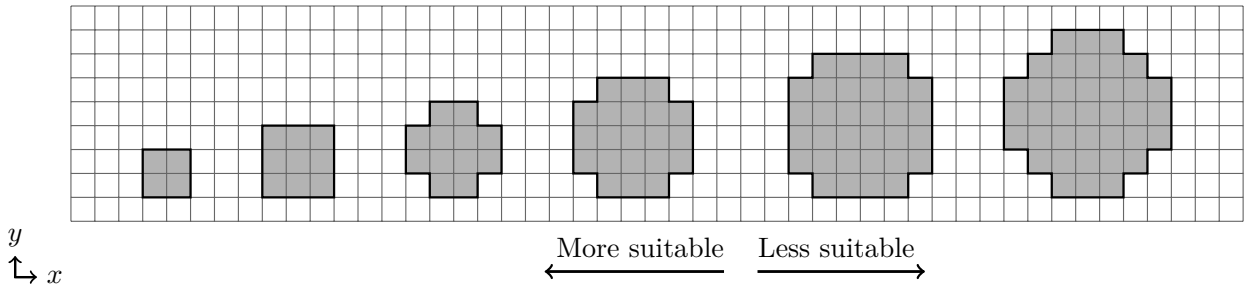


Figure 4.2 Example minimum mining width constraint templates

As indicated in Figure 4.2 the templates corresponding to smaller operational areas are generally more suitable. This is because incorporating minimum mining width constraints has a substantial impact on runtime. Larger mining width sets, on the order of 50 or more blocks per set, are more difficult to satisfy and should generally be avoided. This is still in keeping with the third original criteria, as most open-pit mines use blocks of a middling size - for example a mine modeled with 10 by 10 meter blocks may have an associated operating area diameter of 40 or 50 meters. The theory and algorithms developed herein do not enforce any arbitrary limit though, this is primarily a practical consideration.

Figure 4.3 shows two ultimate pit limits. The pit on the left is the conventional ultimate pit which does not satisfy minimum mining width constraints. The pit on the right satisfies a roughly 5x5 minimum mining width (the fourth template from the left in Figure 4.2). The outlined areas are shown in more detail in Figure 4.4.

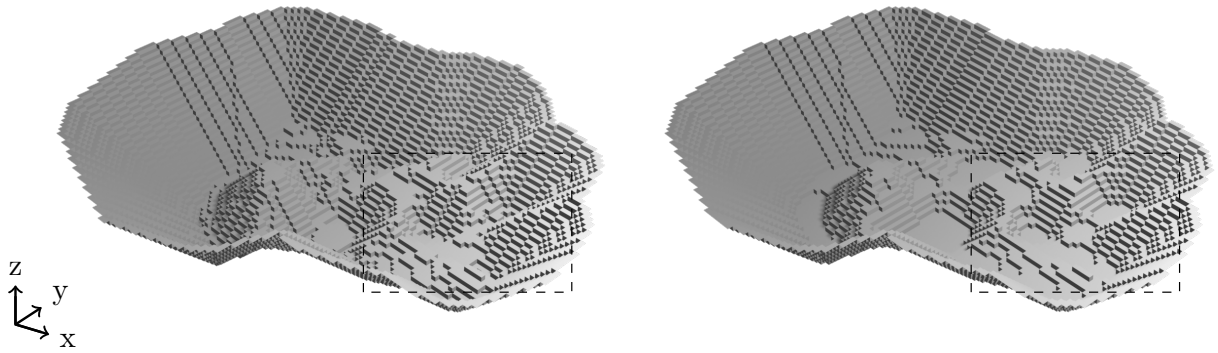


Figure 4.3 Left: the original ultimate pit. Right: the ultimate pit calculated with minimum mining width constraints.

Note that the pits are very similar. This is by design and should always be the case. The object of this exercise is to reduce the objective function value as little as possible compared to the original ultimate pit while satisfying operational constraints and removing the need to satisfy these constraints manually.

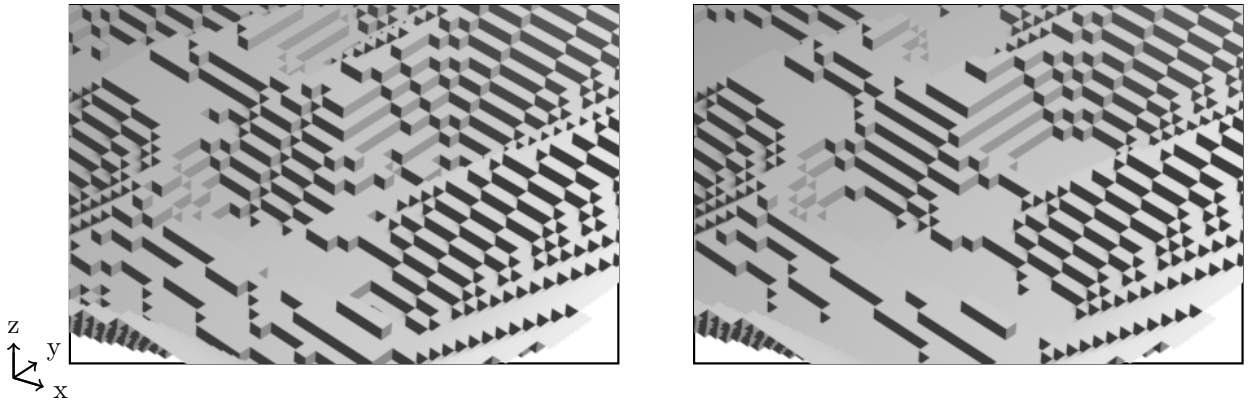


Figure 4.4 Inset from Figure 4.3. Left: the original ultimate pit. Right: the ultimate pit calculated with minimum mining width constraints. Most changes required by the minimum mining width constraints are at the bottom of the ultimate pit.

The framework of integer programming is used to describe a mathematical formulation for the ultimate pit problem with minimum mining width constraints in the following section. This formulation is then considered and extended in the subsequent sections to more closely align with reality, and to be suitable for solving with large real-world datasets.

4.2.1 The Main Integer Programming Formulation

The foundational formulation for the ultimate pit problem with minimum mining width constraints is as follows. Auxiliary variables are used, one for each mining width set, in order to ensure the resulting ultimate pit is operationally feasible.

Sets:

- $b \in \mathcal{B}$, the set of all blocks.
- $\hat{b} \in \hat{\mathcal{B}}_b$, the set of *antecedent* blocks that must be mined if block b is to be mined.
- $w \in \mathcal{W}$, the set of all mining widths.
- $\bar{b} \in \bar{\mathcal{B}}_w$, the set of blocks that are within mining width w.
- $\bar{w} \in \bar{\mathcal{W}}_b$, the set of mining widths of which block b is a member.

Parameters:

- v_b, the economic block value of block b.

Variables:

- X_b, 1 if block b is mined, 0 otherwise.
- M_w, 1 if mining width w is satisfied, 0 otherwise.

The Ultimate Pit Problem with Minimum Mining Width Constraints:

$$\text{maximize} \quad \sum_{b \in \mathcal{B}} v_b X_b \tag{4.1}$$

$$\text{s.t.} \quad X_b - X_{\hat{b}} \leq 0 \qquad \forall b \in \mathcal{B}, \hat{b} \in \hat{\mathcal{B}}_b \tag{4.2}$$

$$M_w - X_{\bar{b}} \leq 0 \qquad \forall w \in \mathcal{W}, \bar{b} \in \bar{\mathcal{B}}_w \tag{4.3}$$

$$X_b - \sum_{\bar{w} \in \bar{\mathcal{W}}_b} M_{\bar{w}} \leq 0 \qquad \forall b \in \mathcal{B} \tag{4.4}$$

$$0 \leq X_b, M_w \leq 1 \qquad \forall b \in \mathcal{B}, \forall w \in \mathcal{W} \tag{4.5}$$

$$X_b \text{ integer} \qquad \forall b \in \mathcal{B} \tag{4.6}$$

Function 4.1 is the objective function which is the same as for the conventional ultimate pit problem. The aim is to maximize the undiscounted value of the mined blocks. Inequality 4.2 specifies the precedence constraints which are again the same as the conventional ultimate pit problem. The purpose of these constraints is to enforce geotechnically stable designs and preclude underground mining.

Inequality 4.3 contains the first set of new constraints, which are called the *assignment* constraints. These constraints restrict the value of M_w to be zero for a given width w if not all of the blocks within that width are mined. That is, M_w can only be one if *all* of its associated blocks are also one. By themselves the assignment constraints don't do anything to modify the results from the original ultimate pit problem. Inequality 4.4 then specifies the *enforcement* constraints which, when combined with the assignment constraints, enforce operational areas in the resulting pit model. The enforcement constraints are on a per block basis, and are interpreted as: block X_b can be a one if and only if *at least* one of its associated auxiliary variables are also one.

Inequality 4.5 precludes unreasonable variable values, and Inequality 4.6 enforces integrality to prevent partially mining blocks.

4.2.1.1 Mathematical Nature of the Enforcement Constraints

Even with the added complexity of the enforcement constraints (Inequality 4.4) it would be very convenient if the model retained a nice mathematical structure similar to the original ultimate pit problem. That is, it would be nice if the constraint matrix were still totally unimodular or if a network optimization approach was still applicable.

The values within the constraint matrix are contained within $\{-1, 0, 1\}$, but even very simple systems are not totally unimodular. The tiny model with three blocks and three mining widths illustrated in Figure 4.5 demonstrates this.

The expanded enforcement constraints for this model follow in Inequalities 4.7 to 4.9.

$$X_1 - M_1 - M_2 \leq 0 \tag{4.7}$$
$$X_2 - M_1 - M_3 \leq 0 \tag{4.8}$$
$$X_3 - M_2 - M_3 \leq 0 \tag{4.9}$$

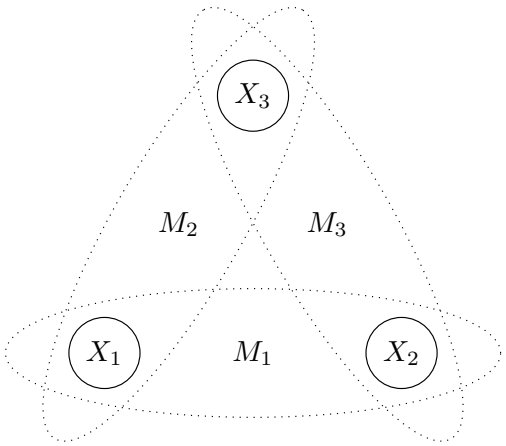


Figure 4.5 Small example mining width configuration with three variables and three minimum mining width constraints consisting of two variables each

The submatrix corresponding to the M variables is isolated in Equation 4.10 and has a determinant of 2.

$$\begin{vmatrix} -1 & -1 & 0 \\ -1 & 0 & -1 \\ 0 & -1 & -1 \end{vmatrix} = 2 \tag{4.10}$$

Thus this formulation does not contain a totally unimodular constraint matrix. The addition of the enforcement constraints, which are necessary to enforce operational mining areas, has ruined the nice mathematical structure of the ultimate pit problem and additional effort to minimize this is warranted.

The enforcement constraints are *covering* constraints which are essentially toggled on and off by their associated block variable. In a logical context they can be considered as *or* constraints, which is how they were developed in the original maximum satisfiability formulation. The constraint is satisfied if the block is not mined, $X_b \leftarrow 0$, *or* one of the associated mining width auxiliary variables is mined, $M_w \leftarrow 1$.

4.2.1.2 Reducing the Number of Enforcement Constraints and Auxiliary Variables

There are a lot of enforcement constraints. In Inequality 4.4 there is one enforcement constraint for every block in the original model which is on the order of tens of millions of constraints for a realistic model. Each of these constraints contains several non zero columns,

although realistically most models should contain on the order of a few dozen up to maybe fifty. However, that is still far too many for a conventional solver to handle.

The first simplification is that enforcement constraints and associated minimum mining widths are only required on *positive* valued blocks in the original model. Non positive valued blocks are only mined when a positive block below them is mined. If those positive valued blocks satisfy a minimum mining width then typically the non positive blocks above them will also.

Finally, the enforcement constraints can be implemented in a deferred or lazy fashion as in a realistic model there may be on the order of several thousand violating blocks in the ultimate pit to begin with. Depending on the approach used this can vastly reduce the actual number of enforcement constraints and auxiliary variables required.

4.2.1.3 Mathematical Nature of the Assignment Constraints

The assignment constraints (Inequality 4.3) are identical to the precedence constraints. These constraints form a totally unimodular substructure similar to the precedence constraints in the original ultimate pit problem. In Sections 4.4.6 and 4.4.7 this property is exploited.

4.2.1.4 Alternative Assignment Constraints

During development an alternative version of assignment constraints was considered. This alternative version is documented in Inequality 4.11 which could be used to replace Inequality 4.3 in the original formulation.

$$M_w + X_b - \sum_{b' \in \overline{\mathcal{B}}_w b' \neq b} \frac{X_{b'}}{|\overline{\mathcal{B}}_w| - 1} \leq 1 \quad \forall w \in \mathcal{W}, \bar{b} \in \bar{\mathcal{B}}_w \tag{4.11}$$

Where $|\overline{\mathcal{B}}_w|$ denotes the number of blocks in mining width set w. One interpretation of this constraint is that: The only way M_w and X_b are both 1 is if all $X_{b'}$ are also 1. This second type of assignment constraint is more complicated, and requires values in the constraint matrix that are not in $\{-1, 0, 1\}$. There is no change in the number of constraints with this method. There are more non zeros in the constraint matrix, however the outcome is the same and integer solutions are identical.

The differences arise when considering a linear relaxation of both types of assignment constraints. This second form, Inequality 4.11, leads to a looser linear relaxation with a higher

economic value that is calculated much quicker. In practice a tighter linear relaxation is preferred during a typical branch and bound approach to solving the integer programming problems, but the increase in speed can be substantial.

One example problem with 13 million rows and 724 thousand columns contained 38 million non zero entries with the original formulation, and 185 million non zero entries with this secondary formulation. The objective value achieved in the linear relaxations was 28.295 million and 28.417 million respectively. The total time required to achieve these results was 7 hours and 41 minutes with the original formulation compared to 21 minutes with this alternative.

Note that fractional values in the constraint matrix can be avoided by multiplying each constraint by $|\overline{\mathcal{B}}_w| - 1$. This is recommended to avoid any numerical instability issues associated with adding fractions together.

4.2.1.5 Precluding Other Inoperable Block Configurations

This formulation for the ultimate pit problem with minimum mining width constraints does not preclude all unmineable configurations of blocks. For example the single unmined block on the left in Figure 4.6, or the non ideal 'peanut' combination of two mining widths on the right of Figure 4.6.

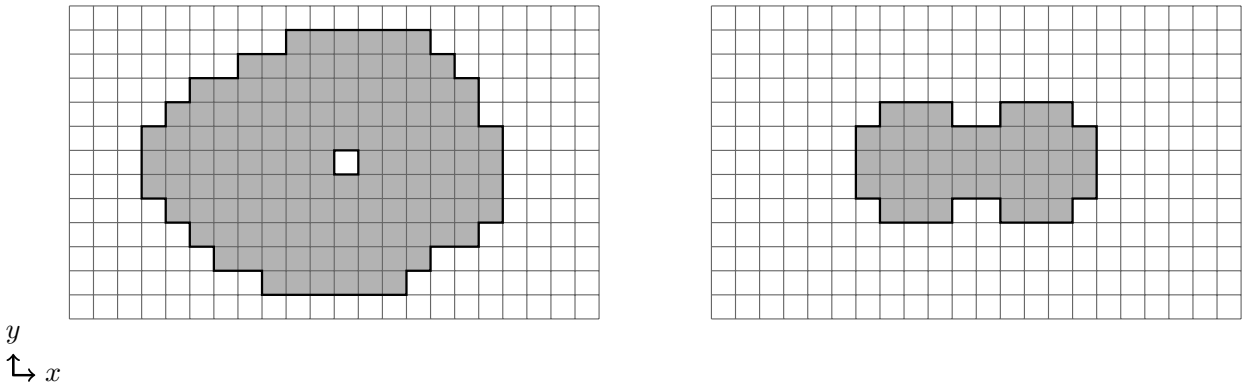


Figure 4.6 Left: An inoperable configuration of blocks permitted by the original formulation. Right: Another inoperable configuration

Neither of these block configurations are realistic, and should potentially be excluded from a truly operational ultimate pit design. One method of excluding these configurations would be to not only require the mined blocks to satisfy minimum mining widths - but also the unmined

blocks. This somewhat unconventional approach would require that any unmined blocks are a part of a suitable collection of other unmined blocks that is large enough.

Another approach would be to add additional constraints on the M_w variables themselves that preclude certain undesirable configurations. For example two nearby minimum mining width sets could only be mined if all of the minimum mining width sets between them were also mined.

4.2.1.6 Hierarchical Minimum Mining Width Encoding

Especially with rectangular minimum mining width sets there is a strong element of self-similarity in this problem. As an example, if a 4×4 minimum mining width area was desired for a regular block model this could be encoded hierarchically. Minimum mining width sets could be constructed as per usual for 2x2 collections of blocks, with each associated auxiliary variable given four assignment constraints for the four relevant blocks. Then a second layer of minimum mining width constraints could be defined for 3x3 collections of blocks, that are assigned if the 4 overlapping 2x2 collections of blocks were mined. Finally, a third layer of minimum mining width constraints for the desired 4×4 groups of blocks could be assigned in terms of the four contained 3x3 collections of blocks.

In such an encoding there are more auxiliary variables than the conventional formulation, however for large sections there will be fewer constraints. The number of assignment constraints, ND, with direct 4×4 mining width sets is given in Equation 4.12.

$$\text{ND} = (n_x - 3) \times (n_y - 3) \times 16 \tag{4.12}$$

This is for a single bench with n_x blocks in the x direction and n_y blocks in the y direction. With the hierarchical encoding the number of assignment constraints, NH, is given in Equation 4.13.

$$\text{NH} = (n_x - 1) \times (n_y - 1) \times 4 + (n_x - 2) \times (n_y - 2) \times 4 + (n_x - 3) \times (n_y - 3) \times 4 \tag{4.13}$$

In practice the complexity and added auxiliary variables overshadow the very slight reduction in the number of assignment constraints, and this hierarchical encoding is not currently recommended.

4.2.1.7 Computational Complexity

It would be convenient if a polynomial-time algorithm existed for the ultimate pit problem with minimum mining width constraints similar to the original ultimate pit problem. Unfortunately, in Appendix A it is shown that any 3-SAT problem can be transformed into the ultimate pit problem with minimum mining width constraints. This proves that this problem is $\mathcal{NP}$-complete, and that there is currently no known polynomial-time algorithm for this problem.

4.3 Two-Dimensional Ultimate Pit with Minimum Mining Width Constraints

In this section a brief diversion is made to the two-dimensional ultimate pit problem. Although not practically useful, the two dimensional ultimate pit problem is a useful teaching tool and is used to illustrate many of the principles of this fundamental mine planning problem. Extending this problem, and associated dynamic programming algorithm, to consider minimum mining width constraints is therefore a valuable exercise.

The two-dimensional ultimate pit problem operates on a single vertical section through a block model. The block model contains solely economic block value information and has very simple precedence constraints where each block is only allowed to be mined if the three blocks above it are also mined. These precedence constraints correspond to a 45° pit slope if the block sizes in the x and z direction are equivalent, and in general the slope angle is $\arctan(s_z/s_x)$. Minimum mining width constraints are defined by requiring an integer number of blocks along the pit bottom. Graphically the two-dimensional ultimate pit problem with minimum mining width constraints is shown in Figure 4.7. Three different solutions for mining widths of one, two, and three blocks are shown along with the input data.

An algorithm for solving this simplified two-dimensional problem is developed in this section. The value of such an algorithm is mostly pedagogical as it cannot be applied to real-world problems. The algorithm described in this section is a direct extension of the 2D algorithm introduced by Lerchs and Grossmann in 1965 [20], discussed previously in Section 2.2.7.

Fundamentally the two-dimensional dynamic programming ultimate pit algorithm operates from left to right and top to bottom along the section calculating for each block the best possible contribution of all the preceding columns given that the current block is contained within the pit. At the end of the scan the optimal pit can be extracted by following the traceback information

from the top right. In this section minimum mining width constraints are incorporated by disallowing pit bottoms from going back 'up' until they have mined the requisite blocks 'across.' There is no restriction on mining 'down' another bench, except that it resets any progress on satisfying the minimum mining width.

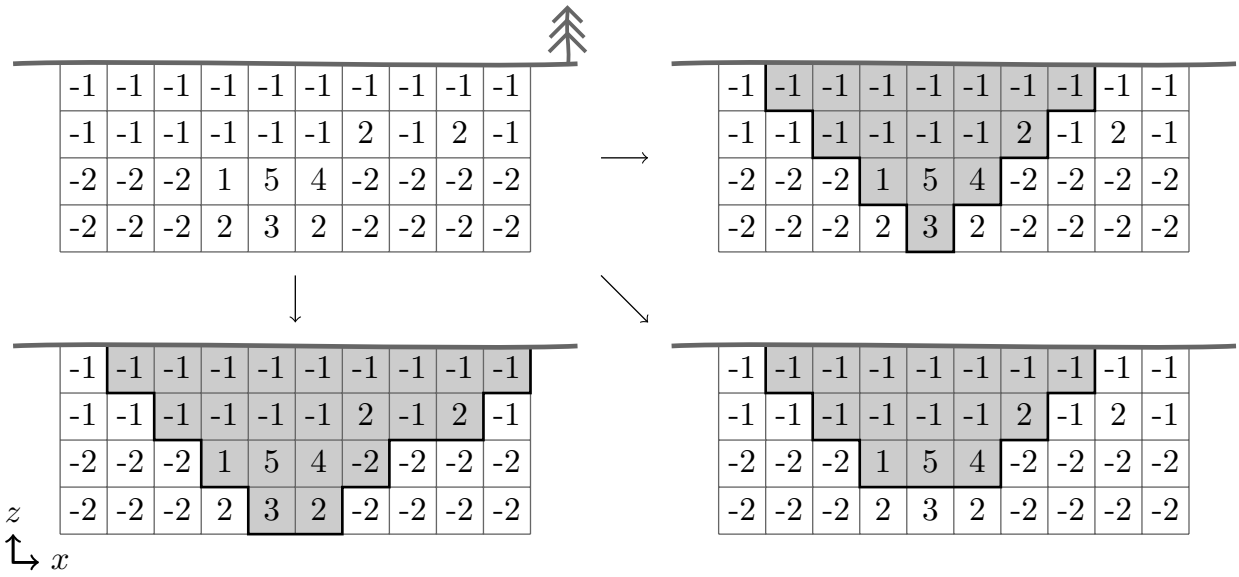


Figure 4.7 A graphical depiction of the two-dimensional ultimate pit problem with minimum mining width constraints. The input economic block value section (top left) is shown with three different solutions with mining widths of size one (the original ultimate pit problem, top right), two (bottom left), and three (bottom right)

4.3.1 Formal Description

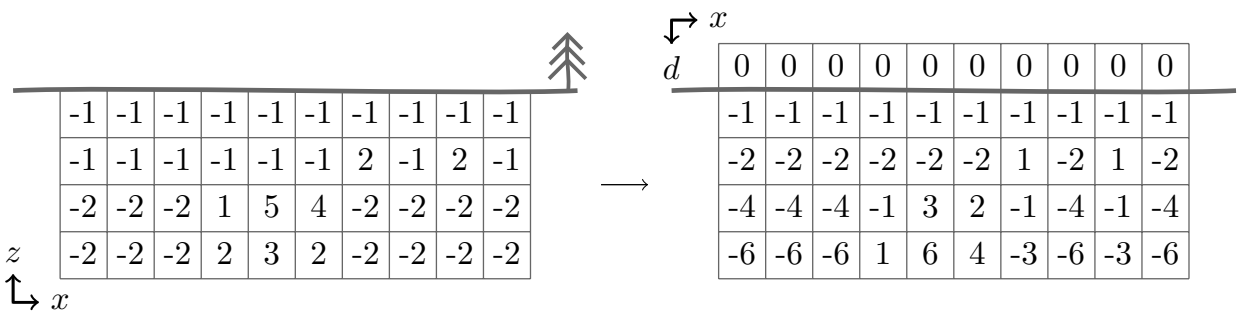
Given a regular 2D economic block value model $EBV_{x,z}$, where x and z are indices in the X and Z dimensions $x \in \{0, 1, ..., n_x - 1\}$, $z \in \{0, 1, ..., n_z - 1\}$ construct a depth-based cumulative value model $C_{x,d}$, $x \in \{0, 1, ..., n_x - 1\}$, $d \in \{-1, 0, 1, ..., n_z - 1\}$ as follows:

$$C_{x,-1} = 0 \quad \forall x \in \{0, 1, ..., n_x\} \tag{4.14}$$

$$C_{x,d} = \sum_{z=n_z-1}^{n_z-1-d} EBV_{x,d} \quad \forall x \in \{0, 1, ..., n_x\}, d \in \{0, 1, ..., n_z\} \tag{4.15}$$

Equation 4.14 appends the row of air blocks along the 'top' of the model which helps facilitate the traceback step and helps identify the maximum valued pit contour. The row of air blocks is indexed with a depth of -1. Equation 4.15 fills in the rest of the cumulative value model which

corresponds to the value of mining the entire column of blocks above each block. This algorithm uses depth, indexed with d, instead of the z coordinate because this simplifies the description of the algorithm and is a more efficient memory order in this application. An example input transformation is shown in Figure 4.8.



-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	2	-1	2	-1
-2	-2	-2	1	5	4	-2	-2	-2	-2
-2	-2	-2	2	3	2	-2	-2	-2	-2

0	0	0	0	0	0	0	0	0	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-2	-2	-2	-2	-2	-2	1	-2	1	-2
-4	-4	-4	-1	3	2	-1	-4	-1	-4
-6	-6	-6	1	6	4	-3	-6	-3	-6

Figure 4.8 Example input transformation from an economic block value model (left) to the cumulative value model (right). Note the extra row of 'air' blocks and the change of indices.

The size of the cumulative value model and the mining width, n_w, are then used to define a 3D volume for the dynamic programming iterations denoted $V_{x,d,w}$. This volume is indexed with $x \in \{0, 1, ..., n_x - 1\}$ for the x coordinate along the section, $d \in \{-1, 0, 1, ..., n_z - 1\}$ for the depth with -1 being the special row of air, and $w \in \{0, 1, ..., n_w - 1\}$. The w dimension of this volume serves as a counter which prevents the pit contour from going 'up' unless the minimum mining width requirement is met. A w index of 0 indicates that no blocks have yet been mined horizontally. Only when $w = n_w - 1$ is the growing pit permitted to go up a bench.

The two starting cases are:

$$V_{0,-1,0} = 0 \tag{4.16}$$

$$V_{0,0,0} = C_{0,0} \tag{4.17}$$

Where Equation 4.16 corresponds to not mining the first column, and Equation 4.17 corresponds to beginning the pit in the top left and starting to mine immediately. From this point forward the iteration proceeds down along the depth, d, fastest, then 'across' the mining widths w, and finally along columns x slowest. The special air row is handled as follows:

$$V_{x,-1,0} = \max\{V_{x-1,-1,0}, V_{x-1,0,n_w-1}\} \tag{4.18}$$

Read Equation 4.18 as: the value is the maximum of the value if the pit continues not mining ($V_{x-1,-1,0}$), or if the pit stops mining provided that w is $n_w - 1$. Note that it is important to handle the air row in this fashion where it is clearly delineated between not mining (value stays the same), and stopping mining (where the value becomes that of the best pit to the left). This idea was missed in the original description from Lerchs and Grossmann, without minimum mining width constraints, and could lead to incorrect results when the dataset consists of two disparate pits. The main cases are then split between three cases for the different possible values of w.

$$V_{x,d,0} = C_{x,d} + \max_{w'=0}^{w'<n_w} \left\{ V_{x-1,d-1,w'} \right\} \quad d \neq -1 \tag{4.19}$$

Read Equation 4.19 as we can drop down one bench from any width, provided we reset our width to zero. This is the only way to go down a bench.

$$V_{x,d,w} = C_{x,d} + V_{x-1,d,w-1} \quad w \geq 1, w < n_w - 1 \tag{4.20}$$

Equation 4.20 simply assigns the value over one block on the same bench and increases the width counter potentially allowing the pit to go upwards by one bench later via Equation 4.21.

$$V_{x,d,n_w-1} = C_{x,d} + \max \begin{cases} V_{x-1,d,w-1} \\ V_{x-1,d,w} \\ V_{x-1,d,w} \quad , d < n_w - 1 \end{cases} \tag{4.21}$$

The three cases in Equation 4.21 can be understood as follows:

- Continue mining horizontally along the current bench and 'finish' the mining width constraint.
- Continue mining horizontally along the current bench, even though the mining width has already been satisfied.
- Decrease the depth of the pit by one bench (by considering the value of the block at $d + 1$), and maintaining the completed mining width constraint.

The approach requires just under $n_x \times (n_z + 1) \times n_w$ iterations, as some cells of the full volume are invalid: either because they are unnecessary (for example when $d = -1$ and $w > 0$), or correspond to unreachable blocks ($d > (x - w)$), and a few other edge cases. Each iteration looks

at a constant number of previous blocks which is typically less than or equal to three, except when $w = 0$ when it is required to look at n_w other cells. This requirement to look at n_w other cells when $w = 0$ does not increase the algorithmic complexity of the algorithm. This implies an algorithmic complexity of $\mathcal{O}(n_x(n_z + 1)n_w)$, essentially $\mathcal{O}(n^3)$ where n is $\max\{n_x, n_z + 1, n_w\}$. If one considers the size of the mining width n_w as an input to the problem then the algorithmic complexity of this approach is technically pseudo-polynomial, as it depends on the value of the input mining width constraint.

Practically this algorithm has only marginally more value that the original 2D dynamic programming algorithm discussed in Section 2.2.7. It could ostensibly be applied in the same manner: on sections of a full 3D block model and subsequently smoothed as described in Johnson, 1971 [73]. Realistically the full 3D approaches described in the subsequent sections are to be preferred.

4.4 Solution Methods

The primary objective of this chapter is to provide solution strategies which are practical and applicable to very large block models and realistic datasets. It is insufficient to propose a single technique or solution methodology and expect it to work well across all possible inputs, or expect it to perform well in all scenarios. Therefore a wide ensemble of techniques are evaluated.

4.4.1 The Ultimate Pit

One possible strategy is to relax Constraints 4.3 and 4.4 and just solve for the ultimate pit without any mining width constraints at all. If the resulting pit happens to satisfy the mining width constraints already, either by mining nothing at all, or because the deposit is tabular in nature, then no additional work is necessary. The addition of minimum mining width constraints necessarily reduces the value by excluding certain configurations of blocks, but if those configurations do not occur then the optimal answer will not change.

The conventional ultimate pit is defined to be the *smallest* maximum valued closure of the precedence graph, but there may be larger pits with the same value. One option would be to compute the largest ultimate pit as well, and see if that one has better luck at satisfying minimum mining width constraints. The pseudoflow algorithm as implemented in Chapter 3 computes the

smallest closure, and it is not very straightforward to modify it to compute the largest - unlike in the Lerchs and Grossmann algorithm where it suffices to modify a few strict inequalities in the move toward feasibility procedure. One way to compute the largest ultimate pit is as follows:

- Count the number of nonnegative blocks, call this q
- Multiply each block value by q, and if the value is nonnegative add 1.

This procedure amounts to increasing all values by some very small epsilon that is large enough to mine blocks that were previously zero, but not so large as to change the results. It may be important to take necessary steps to avoid overflow with this technique as the flow values become very large. The minimum cut of this modified dataset is then the largest minimum cut of the original problem.

Of course these approaches are naïve, and are not expected to find the optimal result except in very unlikely situations. These approaches are excluded from the computational comparison because they are not guaranteed to generate feasible pits, although the original *Ultimate-Pit* is documented for comparison.

4.4.2 Floating 'Fat' Cones

The floating cone procedure originally described by Pana in 1965 is easily extended to handle minimum mining width constraints [58, 108]. The floating cone algorithm was described in Section 2.2.5.

To contend with minimum mining width constraints one can use the operating width as the base element of each cone instead of a single block. All minimum mining width sets which contain a positive valued block are a potential cone bottom and are considered in turn. If the full mining width set and all of its antecedent blocks have a net positive value they are extracted, and the floating process is continued. Once all of the mining width sets have been considered the process is repeated until a full scan is completed with no positive valued cones identified and no change to the overall pit.

The result necessarily satisfies the minimum mining width constraints - it is the union of satisfying pits. And the result necessarily has a positive value - only positive valued subsets were incorporated. However this approach does nothing to address the two fundamental flaws of

floating cone methods: overmining and undermining. The optimal results are not expected with this approach even for simple datasets, but it is included in the computational comparison.

One area of latitude within the floating cone algorithm is the order in which the cone bottoms are considered. For the evaluation the five following orderings were evaluated:

- In the *Float-Bottom-Up* solver the cone bottoms are sorted, by their coordinates, ascending through x, y, z.
- In the *Float-Top-Down* solver the cone bottoms are sorted, by their coordinates, descending through x, y, z.
- In the *Float-Random* solver the cone bottoms are shuffled randomly.
- In the *Float-Value-Ascending* solver the cone bottoms are sorted by their contained economic value from lowest to highest.
- In the *Float-Value-Descending* solver the cone bottoms are sorted by their contained economic value from highest to lowest.

All of the floating cone solvers suffer from poor runtime and are unable to find the optimal solution in many scenarios. The poor runtime is generally because for each cone bottom a large portion of the block model has to be scanned, and even if the cone is not removed in the first pass it will have to be re-examined again if any other cones are removed that may impact it. A lot of time is wasted checking negative valued cones with many blocks to see if they are still negative.

One minor wrinkle with the 'value ascending' and 'value descending' solvers is that simply reordering the cone bottoms by value once at the beginning of each pass is not totally accurate. When a pit is removed the value of many other cone bottoms may change which invalidates the initial ordering. One approach is to store the potential cone bottoms in a heap which is sorted by contained value. As cones are removed any affected cones can have their new value calculated and pushed into the heap as well. When a cone comes up for possible extraction its value is first checked if it is current, and only then possibly extracted. In practice this modification is overkill for a slow, heuristic approach with explicit downsides.

4.4.3 Geometric Methods

Geometric methods begin with the ultimate pit, and then heuristically work to modify that pit to satisfy minimum mining width constraints. These methods do not consider the economic block value of any blocks, and instead are essentially operating with a new objective function that minimizes the number of blocks changed from the conventional ultimate pit to some new satisfying pit. The idea being that minimizing the number of blocks changed is a good analogue for reducing the value as little as possible.

The *Subset* solver computes the ultimate pit and then creates a new pit consisting of only those mining widths that were satisfied. As previously mentioned this has the unfortunate consequence of removing precisely those blocks with a net positive value that support the blocks above them. However this approach is very straightforward to implement and efficient to calculate, requiring only a simple linear pass over the end results following pit optimization. In some respects this approach is similar to the approach adopted by many pit designers faced with computing a satisfying pit manually.

Similar to how the subset only removes blocks from the ultimate pit in order to create a feasible solution, the superset solvers only add blocks to the ultimate pit. In this way the ultimate pit is contained entirely within the final pit. Unlike the subset approach, however, there is no straightforward way to ensure that the superset pit is as small as possible without resorting to expensive enumeration or another optimization problem. The *Superset-Width* solver evaluates each possible width that could be incorporated into the solution that has at least one block already mined. Those widths are sorted ascending by how many blocks are required and greedily incorporated in sequence until the result fully satisfies. Similarly the *Superset-Cone* solver performs a similar process but looks not only at the blocks in the width, but counts all of the blocks in the entire cone of extraction. This takes longer but generally includes fewer blocks.

The next class of geometric solvers are based on ideas borrowed from mathematical morphology [117]. There are two fundamental operators in mathematical morphology which have straightforward analogues for this problem.

- *Erosion* operates by retaining only the blocks of the current solution such that *all* of the widths of which they are a member are fully mined. This is essentially a complete

contraction of the entire pit, even along the pit walls in the higher benches. The resulting pit following erosion does not necessarily satisfy minimum mining width constraints.

- *Dilation* operates by incorporating all blocks into the current solution such that *at least one* of the blocks within a containing width is mined. This is a complete expansion of the entire pit, and necessarily satisfies minimum mining width constraints.

These operators can be combined into a cleaning heuristic. In mathematical morphology the *closing* is the result from applying dilation followed by erosion. This tends to 'close' holes within the input because the dilation will incorporate new blocks that may not be removed by the subsequent erosion. The *opening* is the result from applying erosion followed by dilation. This operation removes small groups of blocks that are not large enough to survive the erosion operation. Both of these can be applied to ultimate pit models and yield new 'cleaned' pits that will have fewer (with closing), or no (with opening) minimum mining width violations.

These operators are very quick, requiring only a handful of linear scans over the blocks and widths. However, the closing is not guaranteed to satisfy minimum mining width constraints and like all other purely geometric approaches does not consider block value when deciding which blocks to remove or incorporate into the solution. An additional drawback of the mathematical morphology approaches is that they are not able to take advantage of the optimization described in Section 4.2.1.2 where the total number of enforcement constraints is reduced by only incorporating them on the positive blocks. Both erosion and dilation require that all of the mining width sets, even on non-positive blocks, are explicitly specified.

For these reasons the mathematical morphology based heuristics are not incorporated into the computational comparison in Section 4.6, however they were implemented. For the unreduced bauxite dataset on the left in Figure 4.3 the ultimate pit contains 74,412 blocks of which 608 do not satisfy minimum mining width constraints. The pit following closing contains 75,627 blocks (an increase of 1,215) of which 485 are unsatisfying (a decrease of 123 blocks). This increase in mining decreases the contained value from \$28,416,592 to \$27,673,437, by 2.6%. The pit following opening contains 72,629 blocks (a decrease of 1,783) and fully satisfies minimum mining width constraints because opening ends with a dilation operation. This reduction in pit size decreases the contained value to \$26,913,569 which is a reduction of 5.3%.

4.4.4 Meta-heuristic Methods

The appeal of meta-heuristic methods for solving the ultimate pit problem with minimum mining width constraints is that they can generally obtain reasonable solutions in a relatively short time. There is, unfortunately, no guarantee that the result is optimal. However for the very largest problems, that likely remains out of reach for all methods considered in this dissertation. Meta-heuristics are discussed in more detail in Section 2.5.5.

Many meta-heuristics require subroutines for the following operations: generating random satisfying solutions, efficiently modifying a given solution, and combining two solutions together into a third solution. Additionally it may be useful to be able to perform a 'hill-climbing' step on a solution to obtain a very similar solution with a better objective value. Each of these operations for the ultimate pit problem with minimum mining width constraints are discussed in this section.

Random satisfying pits are generated by randomly assigning values to the width variables and then applying a 'flood fill' or breadth first search to satisfy precedence constraints. The resulting pits satisfy both minimum mining width constraints and precedence constraints by construction. The optimal solution could be generated with this procedure although it is unlikely. Most real world problems have tens of thousands of width variables that are zero or one for a large solution space. Experience suggests that the optimal solution consists of fewer ones than zeros, so the random distribution used to initialize the width variables should be skewed.

For completeness, and to verify that the more sophisticated solutions considered in this dissertation are worthwhile, several of these random solvers are included in the computational comparison. Each random solver is given 20 minutes to generate and evaluate as many random pits as possible with the given proportion of zeros. The final pit is just the best out of those generated within the time limit. In practice this approach is primarily used within any other meta-heuristics to generate initial solutions.

One method for slightly modifying or mutating given solutions is to select a nearby width, for example one such that at least one of its contained blocks is mined, and then mine the rest of it along with any necessary antecedents. This grows the pit a small amount by incorporating additional blocks and if used within the context of simulated annealing or a genetic algorithm is useful. One advantage of this modification is that it does not require regenerating the entire pit

and all constraints remain satisfied. The opposite form of mutation, where blocks are removed, is more difficult. Removing blocks, even if done by removing an entire width at a time, may yield a solution that no longer satisfies precedence or minimum mining width constraints. Alternatively a 'shrinking' approach could assign the value of a width variable to zero - but this requires completely regenerating the pit from the remaining widths.

A simple crossover operation is to stochastically take width assignments from each parent solution and generate the resulting pit. This is a uniform crossover. Again, the main variables of interest are the auxiliary width variables, and not the block variables which are harder to work with directly and can instead be derived from any given width assignment.

Hill-climbing is achieved by evaluating very nearby solutions in a matter similar to the floating cone algorithm. Any width variables which are currently assigned zero can be evaluated with a linear search to identify the contained value of the unmined blocks. If that value is positive then they are included in the solution. This suffers from the same overmining and undermining issues present in all floating cone approaches but does yield a higher valued solution, and can be used to improve a solution within its local neighborhood.

These techniques are implemented in the *Evolutionary* solver. This genetic algorithm maintains a population of elites (high valued solutions from previous generations), immigrants (fresh randomly generated solutions), cross-overs (combinations of two elites), and mutants (small changes to elites). At each iteration the candidate solutions, which are pits formed from width assignments, are partially sorted and the lower quality solutions are discarded. Then the appropriate number of cross-overs, mutants, and immigrants are generated.

Meta-heuristics generally suffer from having a large parameter space which all have a non-negligible impact on performance and the resulting solution quality. A genetic algorithm requires the user to specify the appropriate population size, crossover rate, number of generations, and several other parameters. Simulated annealing requires an appropriate cooling schedule which is non-obvious and difficult to get correct. For the computational comparison in Section 4.6 reasonable parameters were sought, but the solutions were found to be of lower quality than the optimization based techniques which follow. Additional effort may be warranted to determine better parameters to use, higher quality unit operations, or more advanced techniques for population control and improved convergence.

4.4.5 Commercial Solvers

For completeness the entire problem can be given to a commercial branch and bound integer programming solver such as CPLEX or Gurobi, [139, 140]. These solvers first apply a 'presolve' operation to transform the model into an equivalent model such that the presolved model has the same feasibility and bounded properties, and such that the objective value of the presolved model is identical to the original model. Then the solution to the linear programming relaxation is sought typically with primal simplex, dual simplex, or by an interior point method such as the barrier algorithm with crossover. Finally branch and bound is used to identify the optimal solution by fixing certain variables to integer values and solving many more linear relaxations.

These solvers are the products of extensive work and research into the general integer programming paradigm and perform well across all manner of inputs. However, in this context, they are not designed to handle this specific problem and, unless the problem is very small, do not perform well. The *Gurobi* solver is present in the computational comparison with a time limit of 20 minutes.

4.4.6 Lagrangian Relaxation Guided Search

The primary integer programming formulation developed in Section 4.2.1 is very similar to the original ultimate pit problem by construction. The main complication stems from the addition of the enforcement constraints in Constraint 4.4. It seems natural, therefore, to remove those constraints by dualizing them into the objective and penalizing any transgressions following the tenets of Lagrangian relaxation introduced in Section 2.5.3. One notable aspect of this approach is that the second half of the minimum mining width constraints (the assignment constraints – Constraint 4.3) do not need to be dualized because they are exactly the same as the precedence constraints.

This approach removes, in some sense, the troublesome enforcement constraints but introduces several other problems. It is now necessary to determine appropriate dual multipliers, or penalties, on the dualized enforcement constraints which can prove difficult and computationally expensive. Additionally, this approach takes aim at the linear relaxation of the original problem and is not guaranteed to determine the optimal integer solution. However when carefully implemented with necessary extensions this approach proves to be quite effective even for very large problems.

4.4.6.1 Lagrangian Formulation

The original formulation and Lagrangian relaxation with the enforcement constraints dualized into the objective function are as follows.

Sets:

- $b \in \mathcal{B}$, the set of all blocks.
- $\hat{b} \in \hat{\mathcal{B}}_b$, the set of *antecedent* blocks that must be mined if block b is to be mined.
- $w \in \mathcal{W}$, the set of all mining widths.
- $\bar{b} \in \bar{\mathcal{B}}_w$, the set of blocks that are within mining width w.
- $\bar{w} \in \bar{\mathcal{W}}_b$, the set of mining widths of which block b is a member.

Parameters:

- v_b, the economic block value of block b.
- λ_b, the dual multiplier for the enforcement constraint associated with block b. The best values for these 'parameters' are sought. Note $\lambda_b \geq 0 \; \forall b \in \mathcal{B}$.

Variables:

- X_b, 1 if block b is mined, 0 otherwise.
- M_w, 1 if mining width w is satisfied, 0 otherwise.

Original Formulation:

$$\text{maximize} \quad \sum_{b \in \mathcal{B}} v_b X_b \tag{4.22}$$

$$\text{s.t.} \quad X_b - X_{\hat{b}} \leq 0 \qquad \forall b \in \mathcal{B}, \hat{b} \in \hat{\mathcal{B}}_b \tag{4.23}$$

$$M_w - X_{\bar{b}} \leq 0 \qquad \forall w \in \mathcal{W}, \bar{b} \in \bar{\mathcal{B}}_w \tag{4.24}$$

$$X_b - \sum_{\bar{w} \in \bar{\mathcal{W}}_b} M_{\bar{w}} \leq 0 \qquad \forall b \in \mathcal{B} \qquad \lambda_b \tag{4.25}$$

$$0 \leq X_b, M_w \leq 1 \qquad \forall b \in \mathcal{B}, \forall w \in \mathcal{W} \tag{4.26}$$

Lagrangian Relaxation:

$$\text{maximize} \quad \sum_{b \in \mathcal{B}} (v_b - \lambda_b) X_b + \sum_{w \in \mathcal{W}} \left(\sum_{\bar{b} \in \overline{\mathcal{B}}_w} \lambda_{\bar{b}} \right) M_w \tag{4.27}$$

$$\text{s.t.} \quad X_b - X_{\hat{b}} \leq 0 \qquad \forall b \in \mathcal{B}, \hat{b} \in \hat{\mathcal{B}}_b \tag{4.28}$$

$$M_w - X_{\bar{b}} \leq 0 \qquad \forall w \in \mathcal{W}, \bar{b} \in \bar{\mathcal{B}}_w \tag{4.29}$$

$$0 \leq X_b, M_w \leq 1 \qquad \forall b \in \mathcal{B}, \forall w \in \mathcal{W} \tag{4.30}$$

Function 4.27 is the objective function which contains the simplified and collected dualized enforcement constraints. Inequality 4.28 are the unchanged precedence constraints, and Inequality 4.29 are the unchanged assignment constraints. Finally Inequality 4.30 are the unchanged variable bounds. Note that this problem was, and remains, a maximization problem and because the enforcement constraints were of the form $AX \leq 0$, their dual multipliers, λ_b, are restricted to be non-negative.

In order to better facilitate the following steps the objective, Function 4.27, slightly changes the indices and range on the summation in Inequality 4.25. This is to collect the λ_b variables on a per mining width basis, which is more clear. This small algebraic manipulation does not change the results.

4.4.6.2 Interpretation

First, recognize that the resulting Lagrangian relaxation model, when the λs are fixed, is of precisely the same structure as the original ultimate pit problem *without* minimum mining width constraints. We can take the dual of the relaxed Lagrangian which can be solved with the pseudoflow algorithm, and the highly efficient MineFlow implementation developed in Chapter 3. In the new network the M variables become nodes, and the assignment constraints become additional precedence constraints.

This relaxation has a nice interpretation. The original enforcement constraints are on a per block basis, and require each mined block to be a part of a completely mined out width. When an enforcement constraint in the original formulation is binding then its associated dual has a positive value, now dubbed λ_b. This value has units which are equivalent to those of the objective

function and can be understood to be the 'price' of any given enforcement constraint. That is, it is exactly the cost associated with satisfying minimum mining width constraints for that block.

This cost must be *paid* by the original block value. So the new 'block values' in our relaxation are $v_b - \lambda_b$. If λ_b is too large; the price of satisfying enforcement constraints exceeds some unknown threshold, then the block will not be mined and X_b will be zero.

This payment goes towards the width variables of which that particular block is a member. This is the second part of the new objective function (Function 4.27). Mining widths originally contain no inherent value, but they adopt the value that is provided by any of their contained blocks in order to pay for any of the negative blocks which they contain.

Thus positive valued blocks are able to *pay* to extract some of their neighboring blocks in order to satisfy minimum mining width constraints, albeit indirectly. An example helps clarify this.

4.4.6.3 Example

Consider the very small ultimate pit problem with minimum mining width constraints in Figure 4.9. This example has seven blocks, six precedence constraints, two mining width sets (each consisting of two blocks), and four assignment constraints. For this example consider only a single enforcement constraint on block X_2 which is $X_2 - M_1 - M_2 \leq 0$.

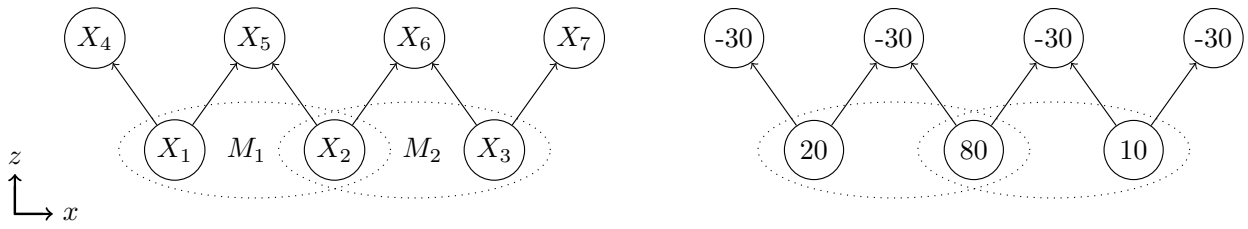


Figure 4.9 A very small example ultimate pit problem with minimum mining width constraints. Left: the seven block variables and two auxiliary variables. Right: The block values.

Applying the Lagrangian relaxation to the enforcement constraint and dualizing the resulting model yields the flow models in Figure 4.10. On the left the dual multiplier, λ_2, for block X_2 is zero, and for the right $\lambda_2 = 15$. When $\lambda_2 = 0$ the result is the original ultimate pit, and when $\lambda_2 = 15$ the 'value' of the central block is reduced by 15 and that value is routed 'back' to the mining widths. Solving this flow model for the ultimate pit now mines the block to the left and the necessary overlying block (X_1 and X_4) which is the optimal solution.

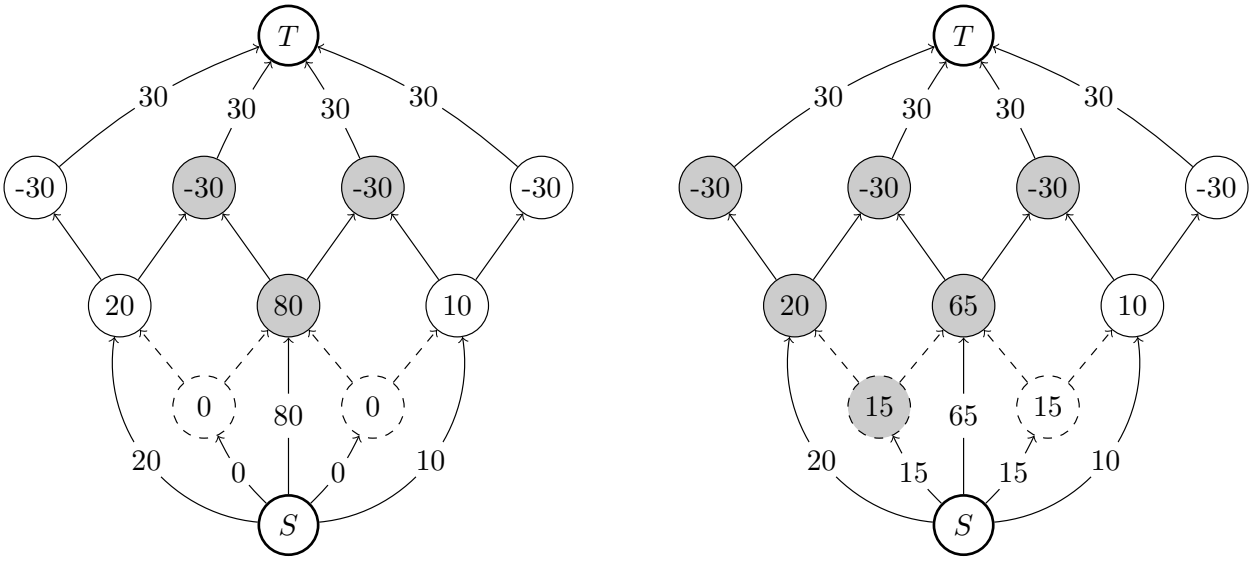


Figure 4.10 The small example's corresponding flow problems. Left: With a dual multiplier of 0. Right: With a dual multiplier of 15.

The value of 15 for the dual multiplier was somewhat arbitrarily chosen, and deserves more careful attention. In this example there are two breaking points. When $\lambda_2 \leq 10$ the solution to the flow problem is the original ultimate pit, and does not satisfy minimum mining width constraints. When $\lambda_2 > 20$ the solution to the flow problem is too large corresponding to mining all seven blocks, which is not the best solution either. This corresponds to not using enough funds from X_2 to pay for minimum mining width constraints and using too many funds respectively. This problem is compounded for larger and more complicated models where determining the correct dual multipliers is a serious challenge.

Another concern made clear through this example is an apparent imbalance in the change in value. The block value associated with X_2 is reduced by 15, but the total available value increases by 30. 15 for each mining width. This is because the enforcement constraint ($X_2 - M_1 - M_2 \leq 0$) when dualized into the objective function has a coefficient of 1 on both M variables. Theoretically this is the correct thing to do, but it seems slightly off. This is a consequence of having *or* constraints, where no particular minimum mining width constraint is preferred over another.

Finally, when the full linear relaxation of this model is solved and the dual on the enforcement constraint is calculated it has a value of 10. This is the most correct dual multiplier to use, at it corresponds to the exact cost of the enforcement constraint at optimality. If the constraint were

not in place, then no M variables would be 1 and the left hand side of the example (the blocks with values 20 and -30) would not be mined. 10 dollars is 'lost' to create the more operational pit design with a mining width of two blocks. However, if this dual multiplier is plugged into the relaxation precisely, the identified pit will be too small (because pseudoflow always identifies the *smallest* maximum valued closure), and does not satisfy the minimum mining width constraints.

4.4.6.4 The Lagrangian Relaxation Guided Search

Applying Lagrangian relaxation to the enforcement constraints inspires a new approach to solving the ultimate pit problem with minimum mining width constraints. The Lagrangian relaxation guided search follows in Algorithm 4.

Following the developments in Chapter 3 it is now possible to solve very large ultimate pit problems rapidly, especially when only a few block values are changed between iterations. This is necessary for this Lagrangian relaxation inspired approach, because the number of 'blocks' has increased substantially, each mining width set becomes another 'block', and the number of precedence constraints has also increased, each assignment constraint is another precedence constraint.

At each iteration if the computed pit already satisfies minimum mining width constraints it might not be the optimal solution because the dual values might be too high, and certain blocks might oversatisfy. In this case the associated dual multipliers on the oversatisfying blocks should be decreased to guide the solution towards the situation where each minimum mining width constraint is only *just* satisfied. If the dual multiplier is already zero, no action is necessary as this corresponds to the situation where the minimum mining width constraints are satisfied 'for free'.

If the computed pit does not satisfy minimum mining width constraints then there are some blocks that are mined, but not enough of the blocks near to them are mined. Therefore the dual multiplier on these blocks should be increased in order to reduce its value and use that value to fund neighboring blocks and satisfy the minimum mining width constraints. So that this iteration is not wasted it is prudent to evaluate nearby solutions that do satisfy minimum mining width constraints, either by computing subsets, supersets, or other nearby solutions as discussed in the preceding sections. If any of these nearby feasible solutions satisfy and are better than the best so far, they are recorded as the current best solution.

Algorithm 4: Algorithm to compute an ultimate pit which satisfies minimum mining width constraints.

```
Initialize a new flow network where every block and every mining width set is a node;
for All positive blocks do
    Connect the source to this block with an arc whose flow and capacity are equal to the
     block value;
for All negative blocks do
    Connect this block to the sink with an arc whose flow and capacity are equal to the
     absolute value of the block value;
for All mining width sets do
    Connect the source to this node with an arc whose initial flow and capacity are equal
     to zero;
Initialize all dual multipliers, one for each positive block, to zero;
Initialize current best solution to the empty set;
while Maximum iterations have not been completed do
    Solve for the minimum cut using pseudoflow;
    if Minimum cut satisfies minimum mining width constraints then
        if Minimum cut is better than current best solution then
            Set the current best solution to this cut;
    else
        for Several nearby satisfying pits do
            if This pit is better than current best solution then
                Set the current best solution to this pit;
    for Each positive block do
        if The block is mined, but does not satisfy minimum mining width constraints then
            Increase the dual multiplier;
            Decrease the capacity on the arc from the source to this block by the increase;
            Increase the capacity on the arcs from the source to all containing mining width
             sets by the increase;
        if The block is mined, and oversatisfies minimum mining width constraints then
            Reduce the dual multiplier;
            Increase the capacity on the arc from the source to this block the decrease;
            Decrease the capacity on the arcs from the source to all containing mining width
             sets by the decrease;
    if No change in dual multipliers then
        Break;
Return the best solution;
```

4.4.6.5 Updating the Dual Multipliers

Each dual multiplier, on each positive block, is bounded theoretically from below by zero and heuristically from above by the original block value. It is natural to begin with dual multipliers of zero, because the problem in this case is the same as the original ultimate pit problem and if this satisfies no further work is required. Unsatisfying blocks must have their duals increased, but by how much is unclear. Each dual multiplier does not have an independent effect on the solution. The point of this approach is that blocks can work together to satisfy minimum mining width constraints and keep duals as low as possible. Working through the interrelationships between mining widths to identify the best possible solution is what makes this problem non-trivial.

The subgradient optimization approach, discussed in Section 2.5.3, tends to work well in practice and can be simplified for this specific problem. In Equation 2.17, b is zero so first compute the helper vector C which is AX^t. The helper vector encodes on a per block basis the mining width satisfaction, that is: when C_b is zero the block is either not mined or perfectly satisfied with exactly one containing width mined. When C_b is one, then the block does not satisfy mining widths, and finally when C_b is negative then the block oversatisfies.

$$C_b = X_b - \sum_{\bar{w} \in \bar{W}_b} M_{\bar{w}} \tag{4.31}$$

The new dual multiplier is then:

$$\lambda_b^{t+1} \leftarrow \min\{\max\{\lambda_b^t + \frac{v_b \times s_t \times C_b}{||C||}, 0\}, v_b\} \tag{4.32}$$

Where s_t is the step size for iteration t. By construction $0 < s_t \leq 1$. In practice starting s_t around 0.4 and setting $s_{t+1} \leftarrow s_t \times 0.8$ every few iterations has given good results, although it is a parameter to consider and potentially investigate for specific problems.

4.4.6.6 Discussion

The Lagrangian relaxation guided search is motivated by duality theory and has a strong intuitive justification. It also works well in practice. The *Lagrangian-Subgradient* solver in the computational comparison implements this approach, and is very successful.

The efforts to improve the pseudoflow algorithm and develop a fast, flexible solver in Chapter 3 are highly relevant to the practical success of this approach. Many problems require solving hundreds or even thousands of large constructed ultimate pit problems, and a fast ultimate pit solver is very relevant. However there are additional avenues to explore that could potentially improve the practical performance of this method.

Subgradient optimization is a well suited approach for determining and updating the dual multipliers however, in this problem it can be prone to oscillation. When a block doesn't satisfy minimum mining width constraints, its dual is increased – but if this increase is too large then it is likely to oversatisfy on the next iteration, and therefore must be reduced. To a certain extent this is mitigated by using a steadily reducing step size, but it may be possible to smooth the dual multipliers more aggressively, potentially by averaging the dual multipliers over the last few iterations.

The subgradient optimization approach elects to update *all* of the dual multipliers for every block that are currently unsatisfying or oversatisfying on each iteration. In certain datasets this may be contributing to the oscillation and slower convergence. It may be worthwhile to update a subset of the dual multipliers on some steps, and instead elect to leave certain dual multipliers fixed for a while.

One idea, that was not evaluated, is to use a meta-heuristic to 'optimize' the dual multipliers. A population of dual multipliers could be maintained, and combined through, for example, a particle swarm optimization methodology that may yield higher quality duals faster. Particle swarm optimization could take advantage of the subgradient in addition to the objective value. This is a potential area for future research.

4.4.7 The Bienstock Zuckerberg Algorithm

In Chapter 5 the Bienstock-Zuckerberg algorithm is used to solve the general block scheduling problem with a wide range of constraints. In the block scheduling problem each block can generally be assigned to one of several different destinations, and mined in one of several different time periods. Additionally there are different classes of constraints to enforce capacity requirements, blending, and handle various other concerns. However, the block scheduling problem is a superset of the ultimate pit problem in terms of formulation complexity. So by

removing all auxiliary constraints and setting the number of destinations and time periods to one the original ultimate pit problem is revealed.

The BZ algorithm and information on how to incorporate the minimum mining width constraints are discussed in Section 5.3, and are not repeated here. The *Bienstock-Zuckerberg* algorithm with the integerization procedure from Aras, Dağdelen and Johnson is included in the computational comparison. Note that this implementation of the BZ algorithm uses Gurobi to solve the master problem and MineFlow to solve the subproblem.

4.4.8 Combined Approach

One of the distinct advantages of the BZ algorithm with the integerization approach from Aras, Dağdelen and Johnson is that it is very easy to incorporate other heuristic solutions. Any heuristic solution can be orthogonalized with the current set of orthogonal columns at any stage, such that the new orthogonal columns necessarily span the heuristic solution. Therefore, the Lagrangian relaxation guided search can be incorporated directly into the Bienstock-Zuckerberg algorithm.

This approach is included in the computational comparison as the *Combined* solver. The combined solver uses the Lagrangian relaxation guided approach to generate an initial set of orthogonal column for the BZ algorithm.

4.5 Bounding the Problem

All of the solvers developed in Section 4.4 are highly sensitive to the size of the input problem. The ultimate pit problem with minimum mining width constraints is $\mathcal{NP}$-complete, as shown in Appendix A, and any efforts to make the problem smaller and easier to solve are warranted. The effectiveness of bounding ultimate pit problems has been demonstrated before, including by Barnes in 1982 [60].

Two types of bounds are considered. An inner bound is a set of blocks which is necessarily contained within the optimal answer, and an outer bound is a set of blocks such that the optimal pit is necessarily contained within. Procedures for identifying both of these bounds are presented.

4.5.1 Inner Bound

An algorithm to compute an inner bounding pit for the ultimate pit problem with minimum mining width constraints follows in Algorithm 5.

Algorithm 5: Algorithm to compute an inner bounding pit which satisfies minimum mining width constraints.

```
Initialize the set of current positive blocks to be the set of all positive blocks;
while The set of current positive blocks is not empty do
    Solve for the minimum cut using pseudoflow;
    Remove all positive blocks which do not satisfy minimum mining width constraints;
Return the last minimum cut;
```

In practice this algorithm very rarely terminates with the empty set for realistic models. There generally exists some set of positive valued blocks that satisfy the minimum mining width constraints and are themselves a valid self-supporting ultimate pit. These blocks necessarily are within the final solution and can be removed to reduce the problem size substantially.

4.5.2 Outer Bound

The outer bound can be computed by solving a single constructed ultimate pit instance following the Lagrangian relaxation approach discussed in Section 4.4.6. Simply set each dual multiplier to the maximum possible value for each block ($\lambda_b \leftarrow v_b$), and solve. The resulting pit necessarily satisfies minimum mining width constraints, and could never be any larger. This substantially reduces the problem size in all case studies considered.

Note that the original ultimate pit is *not* necessarily contained within this bound. Only the ultimate pit which satisfies minimum mining width constraints is.

4.6 Solution Comparison

Several different approaches to the ultimate pit problem with minimum mining width constraints were proposed in this chapter. A computational comparison was executed in order to draw broad initial conclusions about the proposed approaches effectiveness and suitability in real world applications. This comparison is not comprehensive, and there are several areas where it is very difficult to make a proper and fair evaluation of the solvers. It is important to note that the comparison presented herein might speak more to the quality of the author's implementations

than anything else. As demonstrated in Chapter 3, it is generally possible to apply additional software engineering effort and some theoretical knowledge in order to tremendously decrease the runtime of any given solver.

The solvers considered in this comparison are summarized in Table 4.1. All of the solvers, except for Gurobi, were implemented by the author in approximately 9,000 lines of C++ code. Reasonable effort has been expended to develop the solvers with an emphasis on speed and minimizing memory use while accurately implementing the relevant ideas. Where appropriate, reasonable parameter values were sought through experimentation, such as the step size schedule in the Lagrangian search or the population sizes and crossover rate for the evolutionary solver. In solvers where convergence is not guaranteed a time limit of 20 minutes was imposed.

Table 4.1 The solvers used in the computational comparison

Solver Name	Section	Brief Description
Ultimate-Pit	Section 4.4.1	Solve for the ultimate pit, and hope it satisfies.
Float-Bottom-Up	Section 4.4.2	Floating cone heuristic from the bottom up.
Float-Top-Down	Section 4.4.2	Floating cone heuristic from the top down.
Float-Random	Section 4.4.2	Floating cone heuristic in a random order.
Float-Value-Ascending	Section 4.4.2	Floating cone heuristic by width value ascending.
Float-Value-Descending	Section 4.4.2	Floating cone heuristic by width value descending.
Subset	Section 4.4.3	Largest subset of ultimate pit.
Superset-Width	Section 4.4.3	Superset of ultimate pit, by evaluating widths.
Superset-Cone	Section 4.4.3	Superset of ultimate pit, by evaluating cones.
Random-3	Section 4.4.4	Random search solver, 75% zeros.
Random-15	Section 4.4.4	Random search solver, 94% zeros.
Random-31	Section 4.4.4	Random search solver, 97% zeros.
Evolutionary	Section 4.4.4	Evolutionary algorithm solver.
Gurobi	Section 4.4.5	The Gurobi Version 9.5.2 commercial IP solver.
Lagrangian-Subgradient	Section 4.4.6	The Lagrangian relaxation guided solver.
Bienstock-Zuckerberg	Section 4.4.7	The Bienstock-Zuckerberg algorithm.
Combined	Section 4.4.8	The Combined Lagrangian and BZ algorithm.

Six datasets were collected ranging from a very small 2D synthetic dataset to a large realistic 3D block model from a real proposed gold mine, Table 4.2. The same dataset can be used with many different minimum mining width templates or precedence constraints. However, the primary focus of this chapter is on the minimum mining width constraints - so precedence constraints are held constant at 45° with nine benches of connections. In the computational

comparison the dataset name has a suffix appended to indicate the minimum mining width size. That is, `bauxitemed_2x2`, is the BauxiteMed dataset with 2×2 minimum mining width constraints. Similarly `biggold_5x5c` is the BigGold dataset with a 5×5 minimum mining width template with the corners removed. The five different mining width templates considered are the suitable templates shown in Figure 4.2.

Table 4.2 The datasets collected for the computational comparison

Dataset Name	Block Model Size	Brief Description
Sim2D76	$75 \times 1 \times 40$	Small synthetic 2D dataset simulated with sequential Gaussian simulation.
BauxiteMed	$120 \times 120 \times 26$	3D bauxite dataset estimated with ordinary Kriging.
MclaughlinGeo	$140 \times 296 \times 68$	Well known Mclaughlin Dataset.
CuCase	$170 \times 215 \times 50$	A laterally expansive copper dataset simulated with sequential Gaussian simulation.
CuPipe	$180 \times 180 \times 85$	A steeply dipping porphyritic copper dataset.
BigGold	$483 \times 333 \times 101$	A large finely estimated gold dataset.

The high performance server used to run all experiments is called `isengard`, and is a large Linux machine made available remotely for all Colorado School of Mines students. The server has 48 Intel(R) Xeon(R) E5-2690 v3 @ 2.60GHz processors and 377 Gigabytes of RAM, although not all of this computing power was used exclusively for this comparison. All of the solvers, excluding Gurobi, are single-threaded, which is a useful potential avenue for future research. Even the Lagrangian guided solver could be multi-threaded by evaluating different paths and delegating the local search subroutine to another thread.

4.6.1 Problem Bounding

Inner bounds and outer bounds were computed for the raw datasets following the procedures described in Section 4.5. Relevant statistics are tabulated in Table 4.3, including the compute time required to prepare the smaller equivalent problem. For the original ultimate pit problem without minimum mining width constraints bounding is less necessary because solving for the ultimate pit is already very fast. These steps are highly effective at making the problems more manageable, and are a necessary part of the overall process for realistic datasets.

The effectiveness of the bounding procedure necessarily decreases as the size of the minimum mining width template increases. It is much more difficult to identify a large inner bound when the subset step in the bounding procedure removes many more blocks.

The large reduction in problem size is expected for realistic datasets constructed with regular block models. Many practitioners use large models that extend beyond the ultimate pit limits with predominately negative valued blocks. These blocks are identified during the outer bounding process and quickly removed.

The bounding procedure described in Section 4.5 is also generally very quick. Even with the largest model considered with the largest minimum mining width constraint template the process completes in less than two minutes. This is because the bounding procedure leverages the improvements to ultimate pit optimization developed in Chapter 3, and the bounds are computed by solving several specially constructed ultimate pit problems.

Recall that the number of enforcement constraints is the same as the number of positive blocks, which is a useful metric for the size of the problem, and can be used to evaluate the effectiveness of the bounding procedure.

Table 4.3 Time taken and reduced problem size following bounding the ultimate pit problem with minimum mining width constraints for real datasets.

Test Case	Original Problem				Compute Time	Bounded Problem			
	Blocks	Widths	Precedence Constraints	Enforcement Constraints		Blocks	Widths	Precedence Constraints	Enforcement Constraints
sim2d76_3x1	3,000	2,920	8,697	681	0s	250	119	535	87
sim2d76_5x1	3,000	2,840	8,697	681	0s	366	185	873	113
sim2d76_8x1	3,000	2,720	8,697	681	0s	681	285	1,385	147
bauxitemed_2x2	374,400	368,186	7,116,016	37,671	2s	40,404	10,638	424,941	8,973
bauxitemed_3x3	374,400	362,024	7,116,016	37,671	2s	62,831	17,392	734,628	11,811
bauxitemed_4x4c	374,400	355,914	7,116,016	37,671	2s	72,219	20,586	875,792	12,689
bauxitemed_5x5c	374,400	349,856	7,116,016	37,671	2s	88,199	27,268	1,123,962	13,982
cucase_2x2	1,827,500	1,808,300	39,562,848	34,455	9s	96,254	12,848	1,033,338	7,068
cucase_3x3	1,827,500	1,789,200	39,562,848	34,455	14s	196,365	26,935	2,810,325	10,268
cucase_4x4c	1,827,500	1,770,200	39,562,848	34,455	12s	214,397	31,246	3,155,245	10,559
cucase_5x5c	1,827,500	1,751,300	39,562,848	34,455	14s	258,797	45,432	4,006,969	12,892
cupipe_2x2	2,754,000	2,723,485	62,282,528	77,113	13s	489,199	41,433	9,436,699	33,672
cupipe_3x3	2,754,000	2,693,140	62,282,528	77,113	18s	855,937	69,401	17,757,880	46,259
cupipe_4x4c	2,754,000	2,662,965	62,282,528	77,113	22s	900,816	76,676	18,797,251	47,054
cupipe_5x5c	2,754,000	2,632,960	62,282,528	77,113	24s	992,042	91,177	20,919,081	48,043
mclaughlingeo_2x2	2,817,920	2,788,340	62,835,432	32,426	13s	69,552	3,789	606,531	2,107
mclaughlingeo_3x3	2,817,920	2,758,896	62,835,432	32,426	20s	215,797	12,405	3,092,202	4,396
mclaughlingeo_4x4c	2,817,920	2,729,588	62,835,432	32,426	17s	235,683	16,085	3,437,890	4,810
mclaughlingeo_5x5c	2,817,920	2,700,416	62,835,432	32,426	34s	316,585	28,894	5,014,208	6,847
biggold_2x2	16,244,739	16,162,424	378,804,772	93,103	61s	1,073,896	74,574	20,870,865	47,368
biggold_3x3	16,244,739	16,080,311	378,804,772	93,103	99s	2,412,996	167,064	51,178,640	74,973
biggold_4x4c	16,244,739	15,998,400	378,804,772	93,103	97s	2,661,465	208,861	57,162,354	78,611
biggold_5x5c	16,244,739	15,916,691	378,804,772	93,103	101s	3,450,204	293,460	75,676,445	86,024

4.6.2 Solvers Performance vs Gurobi

Gurobi was able to solve eight of the 23 datasets to optimality within 20 minutes each. The largest of these, `bauxitemed_4x4c`, contains 1,104,542 rows, 92,805 columns, and 2,335,746 nonzeros in the constraint matrix. These datasets are of interest in evaluating the proposed solution methods because the optimal answer is known.

The small 2D datasets; `sim2d76_3x1`, `sim2d76_5x1`, `sim2d76_8x1` are so small that all solvers, even the general purpose solver Gurobi, finish in under a second so the only relevant metric for comparison is the solution quality, or objective function value, which is summarized in Table 4.4. The ultimate pit value is included, even though the ultimate pit does not satisfy minimum mining width constraints, in order to provide additional information.

The most relevant takeaway from this comparison is that for very small models the Lagrangian guided search solver, the evolutionary meta-heuristic approach, and the BZ solver are all able to find the optimal answer, whereas the other primarily geometric approaches often perform quite poorly. One interesting outcome was that for `sim2d76_8x1` the floating cone approach was able to identify the optimal solution.

Table 4.4 Value achieved on the three very small 2D datasets by each solver.

Solver	`sim2d76_3x1`	`sim2d76_5x1`	`sim2d76_8x1`
Ultimate-Pit	691	2,791	6,301
Gurobi	**482**	**2,424**	**5,114**
Lagrangian-Subgradient	**482**	**2,424**	**5,114**
Evolutionary	**482**	**2,424**	**5,114**
Bienstock-Zuckerberg	**482**	**2,424**	**5,114**
Combined	**482**	**2,424**	**5,114**
Float-Bottom-Up	234	2,262	**5,114**
Float-Top-Down	234	2,215	2,237
Float-Random	234	2,262	**5,114**
Float-Value-Ascending	234	2,262	1,632
Float-Value-Descending	234	2,262	1,632
Subset	90	671	4,181
Superset-Width	455	0	0
Superset-Cone	238	2,188	668
Random-3	0	0	0
Random-15	152	1,837	4,471
Random-31	152	1,053	4,471

The five additional larger models that Gurobi was able to solve to optimality are summarized in Table 4.5.

Table 4.5 Value achieved and time taken on the five smallest 3D datasets by solver.

Dataset / Solver	Objective	Percent	Elapsed	Speed-up
`cucase_2x2`				
Gurobi	127,299	–	39s	–
Combined	124,869	98.1%	1m 2s	0.6x
Lagrangian-Subgradient	118,010	92.7%	38s	1.0x
Bienstock-Zuckerberg	124,040	97.4%	31s	1.3x
Evolutionary	66,000	51.8%	41s	1.0x
Floating Cone	99,095	77.8%	2s	19.5x
Geometric	69,806	54.8%	0s	–
`mclaughlingeo_2x2`				
Gurobi	6,660,127	–	46s	–
Combined	6,657,377	99.9%	23s	2.0x
Lagrangian-Subgradient	6,651,721	99.8%	18s	2.5x
Bienstock-Zuckerberg	6,618,472	99.4%	2s	23x
Evolutionary	4,727,968	71.0%	12s	3.8x
Floating Cone	5,613,584	84.3%	1s	46x
Geometric	5,959,546	89.5%	0s	–
`bauxitemed_2x2`				
Gurobi	34,587	–	18s	–
Combined	34,587	100.0%	7s	2.6x
Lagrangian-Subgradient	34,202	98.9%	5s	3.6x
Bienstock-Zuckerberg	34,357	99.3%	1s	18x
Evolutionary	21,313	61.6%	28s	0.6x
Floating Cone	24,251	70.1%	1s	18x
Geometric	0	–	0s	–
`bauxitemed_3x3`				
Gurobi	181,683	–	5m 45s	–
Combined	178,059	98.0%	51s	6.7x
Lagrangian-Subgradient	171,745	94.5%	38s	9.1x
Bienstock-Zuckerberg	170,034	93.6%	24s	14.4x
Evolutionary	80,108	44.1%	51s	6.8x
Floating Cone	157,769	86.8%	5s	69x
Geometric	47,370	26.1%	0s	–
`bauxitemed_4x4c`				
Gurobi	294,136	–	14m 59s	–
Combined	286,069	97.3%	1m 51s	8.1x
Lagrangian-Subgradient	275,330	93.6%	1m 7s	13.4x
Bienstock-Zuckerberg	283,954	96.5%	28s	31.8x
Evolutionary	121,727	41.4%	1m 6s	13.6x
Floating Cone	250,662	85.2%	5s	179.8x
Geometric	98,890	33.6%	0s	–

In these instances the time taken and objective function value achieved is relevant, however to save space several of the solvers are combined together and only the best result is reported.

There are a few important takeaways from these larger datasets that are still small enough for branch and bound. In all cases the Lagrangian relaxation guided solver, the Bienstock-Zuckerberg algorithm, and the combined approach are better than the geometric and other meta-heuristic approaches developed in this chapter. The evolutionary solvers effectiveness has decreased rapidly as the problem size has increased, in all cases even under performing relative to the floating cone based algorithm. This is expected, because the evolutionary solver does not fundamentally take advantage of the structure of the problem in the way that the Lagrangian relaxation guided solver, BZ algorithm, or the floating cone algorithm does.

The Lagrangian relaxation guided solver is able to achieve within 8% of optimal for all problems in terms of objective function value. It is also generally faster, especially for the larger datasets where the branch and bound algorithm present in Gurobi begins to suffer from its exponential complexity.

The BZ solver is able to achieve within 7% of optimal for all problems, and in many cases is much closer. BZ consistently outperforms the Lagrangian relaxation guided solver in runtime for these small problem, but achieves a lower objective value in two of the five datasets.

The combined approach is the best of those considered. This is to be expected, as any benefits from the Lagrangian relaxation guided solver are adopted directly as possible solutions into the set of orthogonal columns within the broader context of the BZ algorithm.

4.6.3 Solver Performance On Large Datasets

The fifteen remaining datasets were too large for Gurobi, but the available results are summarized in Table 4.6. Where possible the linear relaxation value as computed by Gurobi, and the BZ algorithm, is reported. In the nine cases that Gurobi was able to solve the linear relaxation it matched with the value reported by the prototype BZ algorithm. This gives an approximate cost of the minimum mining width constraints when compared to the ultimate pit value, but in general the linear relaxation of the ultimate pit problem with minimum mining widths takes advantage of partially mining blocks to contribute to the enforcement constraints while minimizing excess waste. It is not clear what the actual gap between the linear relaxation

objective and unknown optimal integer objective is.

The best value achieved by the Lagrangian relaxation guided solver, the BZ algorithm using the Aras procedure for integerization, and the combined approach is reported alongside how long it took to achieve the result. A time limit of 20 minutes was enforced for all solvers.

The `cupipe` dataset exhibits strange behavior. Gurobi is unable to even solve any of the linear relaxations after 20 minutes, but the Lagrangian search and BZ algorithm terminate after only a handful of iterations. Owing to the steeply vertically dipping nature of the ore body it turns out that the ultimate pit very nearly satisfies the minimum mining width constraints. For the `cupipe_2x2` dataset only five blocks are initially unsatisfied, and their duals are quickly determined. In the `cupipe_4x4c` dataset it appears that the optimal solution is to mine nothing, which due to the initial inner bounding process implies that the solved inner subset pit was optimal, or at least very nearly so. The other datasets consist of many hundreds or thousands of unsatisfying blocks in the ultimate pit which interact in complicated overlapping ways that the algorithms must untangle.

The `biggold_4x4c` and `biggold_5x5c` are the only cases where limiting the 20 available minutes to a single approach, instead of the combined solver, was able to achieve a higher objective value. For these two datasets the Lagrangian solver obtained a slightly higher value.

Table 4.6 Computational summary for large realistic datasets.

	Ultimate	Linear	Bienstock Zuckerberg IP		Lagrangian Solver		Combined Solver	
Dataset	Pit	Relaxation	Objective	Elapsed	Objective	Elapsed	Objective	Elapsed
bauxitemed_5x5c	831,293	678,949	536,357	5m 29s	542,531	3m 46s	556,534	14m 7s
cucase_3x3	567,613	508,270	446,474	20m	429,250	6m 6s	459,757	20m
cucase_4x4c	656,038	591,676	473,996	20m	476,178	10m 2s	514,271	20m
cucase_5x5c	1,236,046	1,166,025	744,101	20m	871,772	20m	1,008,365	20m
cupipe_2x2	25,757	10,312	10,312	6s	10,312	5s	10,312	14s
cupipe_3x3	78,320	42,961	33,340	13s	33,340	6s	33,340	17s
cupipe_4x4c	80,315	34,377	0	28s	0	40s	0	1m 6s
cupipe_5x5c	212,776	141,888	85,801	3m 44s	60,323	7m 37s	92,027	11m 5s
mclaughlingeo_3x3	25,265,407	24,302,490	23,993,483	52s	24,020,371	3m 18s	24,088,683	4m 35s
mclaughlingeo_4x4c	35,481,524	34,377,200	33,923,420	1m 44s	33,986,639	5m 34s	34,110,326	9m 53s
mclaughlingeo_5x5c	83,756,662	82,076,376	80,747,963	8m 56s	80,785,866	18m 39s	81,064,677	20m
biggold_2x2	1,032,842	870,006	806,384	1m 22s	794,218	4m 4s	822,791	10m 11s
biggold_3x3	11,515,028	11,014,858	9,814,845	20m	10,089,255	20m	10,245,597	20m
biggold_4x4c	18,205,567	17,615,751	14,194,367	20m	16,256,546	20m	16,232,940	20m
biggold_5x5c	22,847,235	–	9,071,733	20m	19,202,412	20m	19,143,637	20m

4.7 Discussion

This chapter has focused on the development, and analysis, of a novel, efficient, formulation of the conventional ultimate pit problem with minimum mining width constraints. Several methods, ranging from geometric heuristics to full blown iterative optimization approaches based on Lagrangian relaxation, were developed and compared with general purpose commercial solvers. The BZ algorithm with operational constraints was also evaluated, although the details are left to Chapter 5. A strong emphasis was placed on ensuring that the developed approaches and techniques were readily applicable to real world datasets, across a wide range of deposits and mining operations, and in realistic applications.

Both the Lagrangian relaxation guided search and the BZ algorithm are able to compute high quality results in a reasonable amount of time on large datasets which exceed the capabilities of more general purpose solvers. These developments will allow open-pit mine planning engineers to make better decisions throughout their mine planning efforts and avoid the costly and error-prone manual process of incorporating operational constraints.

Additionally, a small, but pedagogically useful, algorithm for the two-dimensional ultimate pit problem with minimum mining width constraints was developed. This algorithm is not applicable to real world problems, but helps to elucidate several aspects of this problem and may have further applications in future developments.

CHAPTER 5
THE BLOCK SCHEDULING PROBLEM WITH OPERATIONAL CONSTRAINTS

The block scheduling problem contends with many more constraints than the ultimate pit problem, and is far more complicated. This chapter presents initial work and provides formulations for incorporating minimum mining width constraints into the block scheduling problem, additionally this chapter provides some initial insight into how these larger problems can be solved with the Bienstock-Zuckerberg algorithm. The current best practice approach to solving the direct block scheduling problem, the Bienstock-Zuckerberg algorithm, is applied and discussed.

Section 5.1 introduces the block scheduling problem and describes two of the formulations which were suggested by Johnson in 1968 [44] and are the most commonly used. The first formulation uses so-called *at* variables, which are natural, but are not always the most efficient choice. The second uses *by* variables which have nice mathematical properties, especially with precedence and sequencing constraints. Finally, this section introduces some of the operational constraints that are most relevant in block scheduling.

Section 5.2 describes how minimum mining width constraints and minimum pushback width constraints can be incorporated into the block scheduling problem with auxiliary variables in a similar fashion to the ultimate pit problem with minimum mining widths discussed in Chapter 4. Formulations are provided for both *at* and *by* variables.

Section 5.3 discusses how the Bienstock-Zuckerberg algorithm may be applied to solve direct block scheduling problems with operational constraints. Relevant concerns regarding integerization, variations to the conventional Bienstock-Zuckerberg algorithm, and implementation details are discussed.

Finally, Section 5.4 presents two brief case studies applying the outlined approach to a very small dataset and the realistic, well known McLaughlin gold mine dataset. The impact of relevant operational constraints on the NPV of the open-pit mining project is evaluated.

5.1 Block Scheduling Preliminaries

The block scheduling problem is more complex than the ultimate pit problem and contains more variables, more constraints, and does not afford a nice network flow based solution. Where

the ultimate pit problem has a single variable which is set to 1 if a block is mined and 0 otherwise, in the block scheduling problem a set of variables are required for each block in order to specify *when* that block will be mined and how it will be routed. This increases the dimensionality of the solution space but allows for an objective function which aims to maximize NPV instead of simply the contained undiscounted economic value.

Often, yet another dimension is added to each block variable to indicate which process that block is routed to. In the ultimate pit problem the economic block value of any given block assumes that the block goes to the highest value process available. In the block scheduling problem this routing is often left as a choice to be determined by the optimization procedure alongside realistic constraints on the destinations. For example, generally only so many blocks are allowed to be routed to the mill within a particular time period.

Additionally, some block scheduling problems consider stockpiling, uncertainty, interactions with other mining complexes, and more. One of the fundamental challenges of the direct block scheduling problem is that each formulation is typically customized for a particular mining application, and it is difficult to specify a general purpose formulation that is usable in all scenarios. However, there are some common elements, and many constraints are of a similar mathematical form. In the following two sections the two essential forms of most block scheduling problems are presented.

5.1.1 At Variables

The first main type of block scheduling formulation uses variables which are called *at* variables, and are specified to be 1 if a block is mined *at* a particular time and sent to a specific destination. This is the most natural method for specifying many of the other types of constraints in the block scheduling problem. A typical block scheduling problem using *at* variables may begin with the following:

Sets:

- $b \in \mathcal{B}$, the set of all blocks.
- $\hat{b} \in \hat{\mathcal{B}}_b$, the set of *antecedent* blocks that must be mined if block b is to be mined.
- $t \in \mathcal{T}$, the *ordered* set of all time periods (often yearly).

- $d \in \mathcal{D}$, the set of all possible destinations (mill, roaster, leach, dump, etc.).

Parameters:

- p_b, the tonnage of block b.
- $v_{b,t,d}$, the economic value of mining block b and sending it to destination d *at* time period t.
- g_b, the grade of block b
- $\hat{M}_t$, the total maximum tonnage of mining capacity in time period t.
- $\hat{D}_{t,d}$, the maximum tonnage allowed to be routed to destination d in time period t.
- $\check{D}_{t,d}$, the minimum tonnage required to be routed to destination d in time period t.
- $\underline{G}_{t,d}$, the minimum average grade blending requirement at destination d in time period t.

Variables:

- $X_{b,t,d}$, 1 if block b is mined and sent to destination d in time period t, 0 otherwise.

The Simple Multi-Destination Block Scheduling Problem with *at* variables

$$\text{maximize} \quad \sum_{b\in\mathcal{B}}\sum_{t\in\mathcal{T}}\sum_{d\in\mathcal{D}} v_{b,t,d}X_{b,t,d} \tag{5.1}$$

$$\text{s.t.} \quad \sum_{t\in\mathcal{T}}\sum_{d\in\mathcal{D}} X_{b,t,d} \leq 1 \qquad \forall b \in \mathcal{B} \tag{5.2}$$

$$\sum_{d\in\mathcal{D}} X_{b,t,d} \leq \sum_{d\in\mathcal{D}}\sum_{t'\leq t} X_{\hat{b},t',d} \qquad \forall b \in \mathcal{B}, \hat{b} \in \hat{\mathcal{B}}_b, t \in \mathcal{T} \tag{5.3}$$

$$\sum_{b\in\mathcal{B}}\sum_{d\in\mathcal{D}} p_b X_{b,t,d} \leq \hat{M}_t \qquad \forall t \in \mathcal{T} \tag{5.4}$$

$$\sum_{b\in\mathcal{B}} p_b X_{b,t,d} \leq \hat{D}_{t,d} \qquad \forall t \in \mathcal{T}, d \in \mathcal{D} \tag{5.5}$$

$$\sum_{b\in\mathcal{B}} p_b X_{b,t,d} \geq \check{D}_{t,d} \qquad \forall t \in \mathcal{T}, d \in \mathcal{D} \tag{5.6}$$

$$\sum_{b\in\mathcal{B}} (\underline{G}_{t,d} - g_b) p_b X_{b,t,d} \leq 0 \qquad \forall t \in \mathcal{T}, d \in \mathcal{D} \tag{5.7}$$

$$X_{b,t,d} \in \{0,1\} \qquad \forall b \in \mathcal{B}, \forall t \in \mathcal{T}, d \in \mathcal{D} \tag{5.8}$$

Where Equation 5.1 is the objective function using the pre-computed discounted block values. The constraints in equation 5.2 are the *reserve* constraint which enforces that each block is mined at most once.

Equation 5.3 define the precedence constraints. Because this formulation uses *at* variables a precedence constraint between block b and $\hat{b}$ can be read as, before the base block b is mined to *any* destination all of its antecedent blocks must have been mined to some destination in some earlier or equivalent time period. The $t' \leq t$ limit on the second summation on the righthand side of Equation 5.3 is what requires the set of time periods to be an ordered set.

Equation 5.4 contains the mining capacity constraints on a per period basis. Simply ensuring that the total mined tonnage in a period, regardless of which destination is chosen, is below some threshold.

Equations 5.5 and 5.6 are the maximum and minimum capacity thresholds on a per destination basis for each time period. The minimum capacity is generally used to ensure that the mill receives a sufficient quantity of ore within a period to remain operational. Some destinations, such as a waste dump, may not have a minimum capacity.

Equation 5.7 is an example of a blending constraint that enforces a lower bound on the average grade requirements at each process destination in each period.

Finally, Equation 5.8 enforces integrality on the *at* variables, and precludes meaningless variable values.

5.1.2 By Variables

A common reformulation used to simplify the precedence constraints, and to facilitate decomposition approaches such as the BZ algorithm, is to replace *at* variables with *by* variables. A *by* variable is 1 if and only if the block is mined "by" time period $\breve{t}$, (i.e no later than period $\breve{t}$). Similarly it is possible to apply this concept to the destinations, although the "by" name is a bit out of place in that context. For this the possible destinations must be ordered, and a *by* variable may be defined such that it is 1 if and only if block b is mined by time period $t-1$, or it is mined in time period $\breve{t}$ with destination $\breve{d}$ such that the 'actual' destination d is $d \leq \breve{d}$.

The full details associated with reformulating the objective and all relevant constraints is documented in several places [101–103], and are shown in more detail in Section 5.3.2. The

primary outcomes are that an *at* variable($X_{b,t,d}$) can be replaced by at most two *by* variables as in equations 5.9 to 5.11.

$$X_{b,t,d} = Z_{b,t,d} - Z_{b,t,d-1} \qquad \forall b \in \mathcal{B}, t \in \mathcal{T}, d = 2, ..., |\mathcal{D}| \tag{5.9}$$
$$X_{b,t,1} = Z_{b,t,1} - Z_{b,t-1,|\mathcal{D}|} \qquad \forall b \in \mathcal{B}, t = 2, ..., |\mathcal{T}| \tag{5.10}$$
$$X_{b,1,1} = Z_{b,1,1} \qquad \forall b \in \mathcal{B} \tag{5.11}$$

With this reformulation the precedence constraints are greatly simplified to being between two *by* variables at a time, with no summations, however additional 'precedence' constraints are required to enforce the appropriate *by* variable values. Following all of these machinations the condensed and simplified form for the block scheduling problem using matrix notation is as follows in Equations 5.12 to 5.15.

The Condensed General Block Scheduling Problem with *by* variables

$$\text{maximize} \quad \vec{c}Z \tag{5.12}$$
$$\text{s.t.} \quad Z_i - Z_j \leq 0 \qquad \forall (i,j) \in I \tag{5.13}$$
$$HZ \leq \vec{h} \tag{5.14}$$
$$Z \in \{0,1\} \tag{5.15}$$

Where Z is the $(|\mathcal{B}| \times |\mathcal{T}| \times |\mathcal{D}|) \times 1$ column vector of *by* variables, as described earlier. There is one *by* variable for each block, time period, and destination. Where $\vec{c}$ are the objective coefficients on the *by* variables which can be determined from the original values v as follows:

$$c_{b,t,d} = \begin{cases} v_{b,t,d} - v_{b,t,d+1} & \text{if} \quad d < |\mathcal{D}| \\ v_{b,t,d} - v_{b,t+1,1} & \text{if} \quad t < \mathcal{T}, d = |\mathcal{D}| \\ v_{b,t,d} & \text{if} \quad t = |\mathcal{T}|, d = |\mathcal{D}| \end{cases} \tag{5.16}$$

Each *by* variable Z is typically indexed with b, t, d for block, time, and destination, but in Equation 5.13 a shorthand notation is adopted where i and j simply refer to two different *by* variables. $\mathcal{I}$, then, is the total set of all precedence constraints between blocks *and* all of the constraints that are required from turning *at* variables into *by* variables. $\mathcal{I}$ contains the following three sets of precedence constraints in Equations 5.17 to 5.19.

$$Z_{b,t,|\mathcal{D}|} - Z_{\hat{b},t,|\mathcal{D}|} \leq 0 \qquad \forall b \in \mathcal{B}, \hat{b} \in \hat{\mathcal{B}}_b, t \in \mathcal{T} \tag{5.17}$$

$$Z_{b,t,d} - Z_{b,t,d+1} \leq 0 \qquad \forall b \in \mathcal{B}, t \in \mathcal{T}, d < |\mathcal{D}| \tag{5.18}$$

$$Z_{b,t,|\mathcal{D}|} - Z_{b,t+1,|\mathcal{D}|} \leq 0 \qquad \forall b \in \mathcal{B}, t < |\mathcal{T}| \tag{5.19}$$

Equation 5.17 are the original precedence constraints which connect base blocks (b) mined by a given period (t) mined to any destination ($d = |\mathcal{D}|$) to the antecedent block ($\hat{b}$) also mined by that period and also to any destination. Equation 5.18 links the 'by' destinations together, and Equation 5.19 does the same with the time periods.

In the condensed block scheduling problem Equation 5.14 are all of the capacity constraints, blending, uncertainty constraints, and everything else that does not amount to a simple precedence constraint. The H matrix is an $(|\mathcal{B}| \times |\mathcal{D}| \times |\mathcal{T}|) \times |h|$ matrix typically with coefficients equal to grades or tonnages, corresponding to some right hand side $\vec{h}$ which is the column vector of right hand sides. As all of these variables are based on the 'by' variables it may be first necessary to apply the rules in Equations 5.9 to 5.11 to translate the constraints which are more familiar to mine planning engineers.

Finally Equation 5.15 enforces the resource constraints which limit blocks to be mined not at all, or exactly once.

5.1.3 Example Direct Block Scheduling Results

The outcome from either the *at* variable or *by* variable formulation for the block scheduling problem is ultimately a plan which indicates both when blocks are mined, and to which destination they are routed. To illustrate some of the most common operational concerns, which must be addressed by incorporating operational constraints, consider the small synthetic dataset in Figure 5.1

For this simple 2D dataset there are two possible destinations, the mill and the dump although these are are not differentiated in the figures. There are three time periods with a straightforward mining capacity of 300 blocks which are indicated by the three pit contours. The blocks within the smallest pit contour are mined in period 1, between the smallest and second in period 2, and between the second and the largest in period 3. Although this dataset is only a synthetic 2D dataset it illustrates three operational concerns.

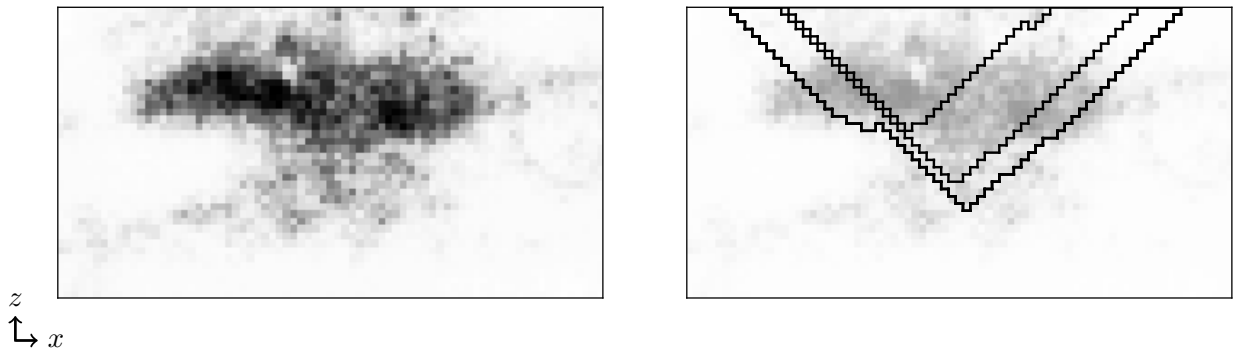


Figure 5.1 Synthetic 2D scheduling dataset. Left: the economic value of each block, darker is higher. Right: A block schedule consisting of three phases

5.1.3.1 Sink Rate

The sink rate of an open pit mine is a limit on the vertical rate of advance that can realistically be achieved in any given year. Typically an open pit mine may be limited to a maximum of six to twelve benches a year which is a consequence of, among other things, the number of shovels in use and the relative cost of developing access to those benches. Incorporating a sink rate into a block schedule is simple with either *at* or *by* variables. Either force the variables corresponding to blocks of unattainable z values to be zero or don't generate them in the first place.

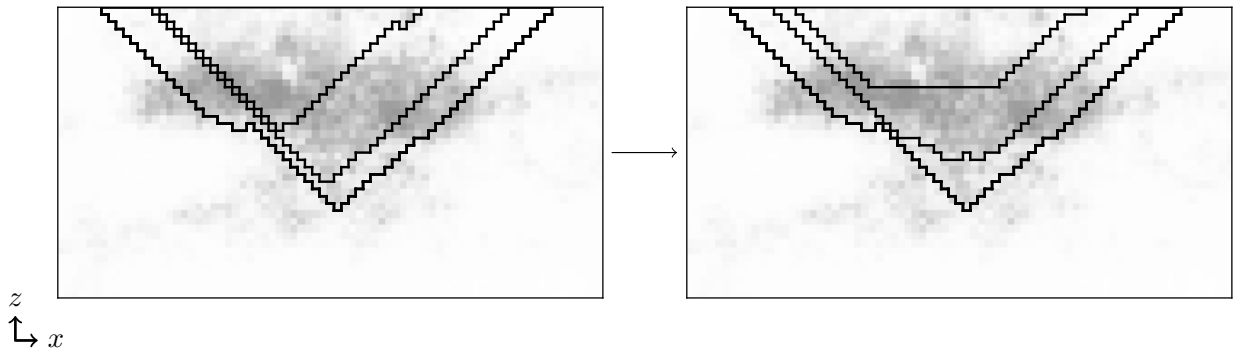


Figure 5.2 Example block schedule for the synthetic 2D scheduling dataset that satisfies a maximum sink rate operational constraint

In the synthetic example a block schedule that incorporates an eleven bench sink rate is shown in Figure 5.2. A useful byproduct of enforcing a maximum sink rate is that it can prevent minimum mining width violations in certain circumstances, although this is not guaranteed.

5.1.3.2 Minimum Mining Width Constraints

The final pit limits in a operationally feasible block schedule should also satisfy minimum mining width constraints, for all of the same reasons discussed in Section 4.1. Incorporating minimum mining width constraints into block scheduling problems is discussed in detail in Section 5.2.

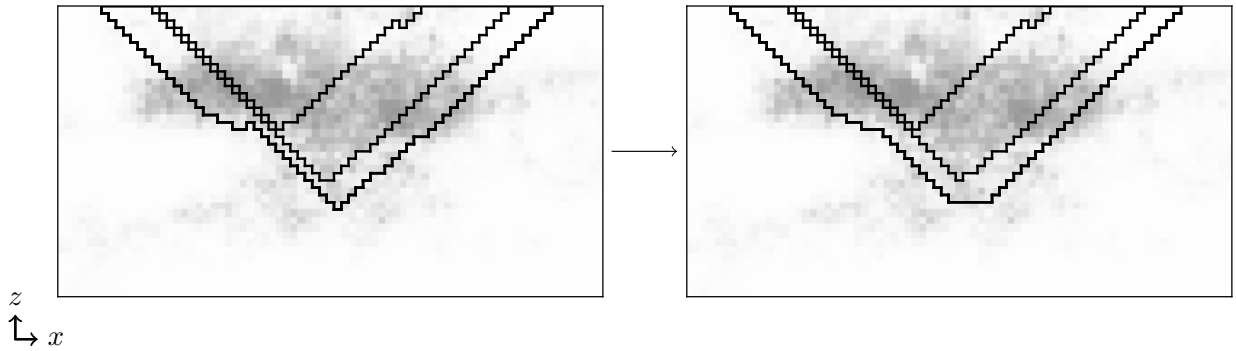


Figure 5.3 Example block schedule for the synthetic 2D scheduling dataset that satisfies a minimum mining width constraint of six blocks

In Figure 5.3 the pit on the right satisfies minimum mining width constraints in the final pit limit, but each phase individually does not satisfy a minimum mining width. It is possible to require a suitable operating area at the bottom of each phase and this may be desirable in some circumstances.

5.1.3.3 Minimum Pushback Width Constraints

A common characteristic of block schedules that do not consider operational constraints are very small changes between some of the pit walls in between phases. This is evident even in the synthetic 2D example, specifically along the west wall between phase 1 and 2, where the distance between the pit walls is only a single block. This is a very poor plan from an operational perspective because in order to start mining in an area large equipment must be relocated to that area first, among other preparations, which can take a long time and incur substantial operating costs. Because shovel movement is not typically considered directly in a long-range plan it should at least be handled implicitly by precluding these sorts of configurations.

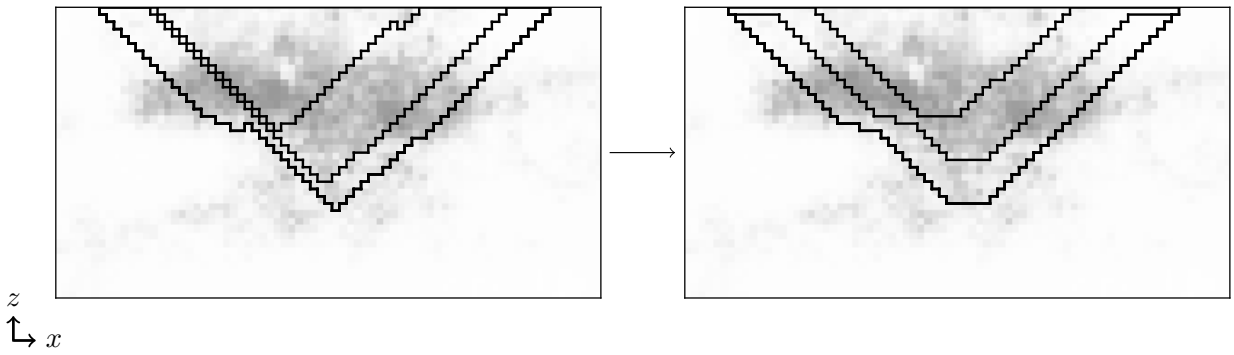


Figure 5.4 Example block schedule for the synthetic 2D scheduling dataset that satisfies a minimum pushback width constraint of six blocks

In Figure 5.4 a minimum pushback width of six blocks is specified. A consequence of how this is currently modeled is that it inherently satisfies minimum mining width constraints as well. In this synthetic example the optimal answer was, somewhat unexpectedly, to push back the west wall of the second phase instead of snapping the walls together. This just reiterates the importance of using optimization methods to incorporate operational constraints when possible, because the best set of changes may be nonobvious.

Unfortunately this constraint does not translate easily to current optimization approaches in 3D. The first concern is that there needs to be a very large number of constraints, on all blocks in all phases which can overwhelm all commercial solvers and those developed within this chapter. However, the second concern is that properly modeling large minimum pushback width constraints can be very difficult because the block templates do not fit nicely together, and interact destructively with the precedence constraints and the general shape of nested pits. An initial formulation for minimum pushback width constraints is presented in the following sections, but this does not work well in 3D.

5.1.3.4 Additional Operational Constraints

The most impactful operational constraint considered beyond the scope of this dissertation is bench access. Main haulage ramps, drop cuts, and access roads are part of a very important set of operational constraints with impacts on economic viability and safety. This dissertation does not develop any ideas or formulations on this topic.

Additionally, besides enforcing minimum pushback width constraints, shovel movement and scheduling is beyond the scope of this dissertation. This extends also to the type of plans that satisfy minimum mining widths and pushback constraints but still have mining areas separated by long distances. Integer programming formulations which consider this type of operational constraint would potentially be very complicated, as they may have to incorporate a connected components analysis in some form.

Finally, many operational constraints that are more within the realm of short range planning are not considered. For example, the destinations chosen for each block should satisfy some form of minimum mining width constraint as well. As mentioned in Section 2.1.4 this particular operational constraint is generally handled during the short range grade control process, in part because blast movement of the material should be considered. Incorporating more operational constraints into a block scheduling problem necessarily reduces value, and will generally increase computation time, and may lead to currently unmanageable levels of complexity.

5.2 Width Constraints in Block Scheduling Problems

The prior formulation for minimum mining width constraints, Section 4.2.1, naturally extends to the block scheduling problem with either *at* or *by* variables. The fundamental concept of using an auxiliary variable to represent a set of contiguous blocks which must be treated similarly can be used directly for both minimum mining width constraints and minimum pushback width constraints.

5.2.1 Minimum Mining Width Constraints with *at* Variables

Minimum mining width constraints in the block scheduling problem require that all areas of the pit must be a part of an operationally feasible area. However, they do *not* need to all be mined within the same phase.

For each width $w \in \mathcal{W}$ with member blocks $\bar{b} \in \bar{\mathcal{B}}_w$, as defined in Section 4.2.1, define an auxiliary variable M_w. Then accounting for destinations ($\mathcal{D}$) and time periods ($\mathcal{T}$) define the assignment constraints as in Equation 5.20. And the enforcement constraints as in Equation 5.21.

$$M_w - \sum_{d \in \mathcal{D}} \sum_{t \in \mathcal{T}} X_{\bar{b},t,d} \leq 0 \qquad \forall w \in \mathcal{W}, \bar{b} \in \bar{\mathcal{B}}_w \tag{5.20}$$

$$X_{b,t,d} - \sum_{\bar{w} \in \bar{\mathcal{W}}_b} M_{\bar{w}} \leq 0 \qquad \forall b \in \mathcal{B}, t \in \mathcal{T}, d \in \mathcal{D} \tag{5.21}$$

Each assignment constraint, Equation 5.20, is no longer as simple as in the ultimate pit problem because a width, w, may be assigned a value of 1 if its contained block $\bar{b}$ is mined to any destination in any time period. The enforcement constraints remain of a similar mathematical form, however there are many more of them. It may be possible to reduce the number of enforcement variables by requiring them only on blocks of the last phase, $t = |\mathcal{T}|$, but this is only applicable in certain datasets, and may lead to minimum mining width violations in some cases.

5.2.2 Minimum Pushback Width Constraints with *at* Variables

With minimum pushback width constraints the number of auxiliary variables must increase, because a block must be mined as a part of a operationally feasible area which is all mined within the same period. Therefore the auxiliary variable will have two indices as $M_{w,t}, w \in \mathcal{W}, t \in \mathcal{T}$ and the assignment and enforcement constraints follow in Equations 5.22 and 5.23.

$$M_{w,t} - \sum_{d \in \mathcal{D}} X_{\bar{b},t,d} \leq 0 \qquad \forall w \in \mathcal{W}, \bar{b} \in \bar{\mathcal{B}}_w, t \in \mathcal{T} \tag{5.22}$$

$$X_{b,t,d} - \sum_{\bar{w} \in \bar{\mathcal{W}}_b} M_{\bar{w},t} \leq 0 \qquad \forall b \in \mathcal{B}, t \in \mathcal{T}, d \in \mathcal{D} \tag{5.23}$$

5.2.3 Minimum Mining Width Constraints with *by* Variables

The *by* formulation also simplifies the assignment constraints when enforcing minimum mining width constraints. It can be useful to think of a *by* variable as, this block is mined in this period or any of the preceding periods, or, this block is mined to this destination or any of the previous destinations. So the assignment constraints and enforcement constraints simply fall out from this understanding in Equations 5.24 and 5.25.

$$M_w - Z_{\bar{b},|\mathcal{T}|,|\mathcal{D}|} \leq 0 \qquad \forall w \in \mathcal{W}, \bar{b} \in \bar{\mathcal{B}}_w \tag{5.24}$$

$$Z_{b,|\mathcal{T}|,|\mathcal{D}|} - \sum_{\bar{w} \in \bar{\mathcal{W}}_b} M_{\bar{w}} \leq 0 \qquad \forall b \in \mathcal{B} \tag{5.25}$$

The *by* formulation therefore simplifies the assignment constraints back to being simple two 'block' precedence constraints and vastly reduces the number of enforcement constraints. Only one enforcement constraint, on the last destination / time period, is necessary because this variable will be 1 if and only if the block is mined to any destination in any time period.

5.2.4 Minimum Pushback Width Constraints with *by* Variables

Similar to *at* variables the number of auxiliary variables for pushback width constraints with *by* variables is increased. However the same idea of using the built in 'or' interpretation of the *by* variables is no longer possible, because this would not properly enforce minimum pushback width constraints on earlier periods. A possible formulation for minimum pushback width constraints follows in Equations 5.26 to 5.28.

$$M_{w,1} - Z_{\bar{b},1,|\mathcal{D}|} \leq 0 \qquad \forall w \in \mathcal{W}, \bar{b} \in \bar{\mathcal{B}}_w \tag{5.26}$$

$$M_{w,t} - Z_{\bar{b},t,|\mathcal{D}|} + Z_{\bar{b},t-1,|\mathcal{D}|} \leq 0 \qquad \forall w \in \mathcal{W}, \bar{b} \in \bar{\mathcal{B}}_w, t > 1 \tag{5.27}$$

$$Z_{b,t,|\mathcal{D}|} - \sum_{\bar{w} \in \bar{\mathcal{W}}_b} M_{\bar{w},t} \leq 0 \qquad \forall b \in \mathcal{B}, t \in \mathcal{T} \tag{5.28}$$

5.3 Applying the Bienstock-Zuckerberg Algorithm

The Bienstock-Zuckerberg algorithm, Section 2.5.4, is the best current known approach to solving the linear relaxation of the general block scheduling problem with *by* variables, and can handle problems that are too large for conventional general purpose linear programming solvers by taking advantage of the large network substructure. Additionally, there are at least two approaches to constructing an integer feasible solution once the linear relaxation is identified. The TOPOSORT heuristic from Chicoisne et al [104], and Aras' procedure which modifies the BZ algorithm and adds additional steps [103]. The TOPOSORT heuristic is limited to upper bounded capacity constraints - and can not handle arbitrary constraints such as blending or lower bounds on capacity. Aras' procedure does not restrict the form of any of the side constraints.

The BZ algorithm is generally very performant because it dualizes all of the more general constraints, Equation 5.14, and uses the pseudoflow algorithm to solve a sequence of constructed subproblems. The relevant duals are determined by solving a linear master problem which uses variables corresponding to aggregated collections of many blocks. These aggregated variables are

constrained to be orthogonal, such that each original variable is in exactly one aggregate.

The BZ master problem follows in Equations 5.29 to Equation 5.32.

$$\begin{aligned} \text{maximize} \quad & \vec{c}V\lambda && && (5.29)\\ \text{s.t.} \quad & \lambda_i - \lambda_j \leq 0 && \forall (i,j) \in J && (5.30)\\ & HV\lambda \leq \vec{h} && && (5.31)\\ & 0 \leq \lambda \leq 1 && && (5.32) \end{aligned}$$

In this matrix notation based representation V is the $|\lambda| \times |Z|$ orthogonal 0,1 matrix which specifies which Z variables are within each orthogonal pit variable λ. The helper list of precedence constraints J in Equation 5.30 is the subset of original precedence and *by* constraints as in Equations 5.17 to 5.19 that are necessary for the orthogonal aggregates. Equation 5.31 are all of the original capacities, and other side constraints. The duals on this set of constraints π are then used in the subproblem.

The BZ sub problem follows in Equations 5.33 to 5.35.

$$\begin{aligned} \text{maximize} \quad & \vec{c}Z - \pi\left(HZ - \vec{h}\right) && && (5.33)\\ \text{s.t.} \quad & Z_i - Z_j \leq 0 && \forall (i,j) \in I && (5.34)\\ & Z \in 0,1 && && (5.35) \end{aligned}$$

This subproblem can then be solved by taking the dual (yet again) and using pseudoflow. The solution is then incorporated into V which creates many more orthogonal aggregates.

5.3.1 Operational Constraints in the BZ Algorithm

The formulation for minimum mining width constraints (Section 5.2.3) is well suited for incorporation into the BZ algorithm. The auxiliary variables M_w fit nicely with the original $Z_{b,t,d}$ variables, and the assignment constraints (Equations 5.24) are just another set of precedence constraints, that can be incorporated into both the master problem and subproblem alongside all the others. A minor concern arises with the enforcement constraints.

In Munoz et al's review of the Bienstock-Zuckerberg algorithm they state:

> The BZ algorithm is very effective when the number of rows in H and the number of [additional] variables is small relative to the number of [Z] variables. [102].

This also arises from one of the central tenets in the BZ algorithm. Owing to the totally unimodular nature of the main precedence constraint submatrix there will be at most $|\vec{h}|$ unique fractional values in the final optimal result which, in the worst case, would all need individual orthogonal aggregations [101]. This is typically not a major concern, because there are generally relatively few rows in the H matrix corresponding to the dual multipliers on the capacity, blending, and similar side constraints. However, the enforcement constraints upset this balance substantially – because there is an enforcement constraint on every block in every time period for the minimum pushback width constraint, and even the negative valued blocks require enforcement constraints owing to potential interactions with the side constraints. This increases the likelihood of many unique fractional values in the linear relaxation solution which may lead to slower convergence.

Fortunately some of the earlier evaluated examples considered in Section 4.6 do not exhibit this worst case. The ultimate pit problem is a special case of the block scheduling problem where the number of time periods and destinations are one, and there are no side constraints. The number of unique values in this special case is exactly two (zero and one), but with the addition of even hundreds of thousands of enforcement constraints the number of fractional values does not increase substantially. The `biggold_3x3` dataset, for example, has 1,411 binding enforcement constraints at optimality of the 270,000 original enforcement constraints and only 953 unique fractional values for the 285,000 partially mined blocks.

5.3.2 Example BZ Subproblem with Operational Constraints

The subproblem in the BZ algorithm with *by* variables for multiple time periods, multiple destinations, and operational constraints can become very large and must be constructed carefully. There are multiple sets of precedence constraints consisting of those inherent in the *by* variable reformulation, those required to enforce geotechnical stability, and those owing to the assignment constraints. In order to further explain how these precedence constraints must be constructed and the overall nature of the subproblem, a small example follows in this section.

The example block model in Figure 5.5 consists of eight blocks: three on the lower bench and five on the upper bench. Each of the three lower blocks depends on three blocks in the upper bench indicated by the directed arcs. The numbers within the blocks are their respective block indices. In addition, two sets of two blocks for operational constraints (blocks one and two, and blocks two and three) are indicated with the dashed ellipses.

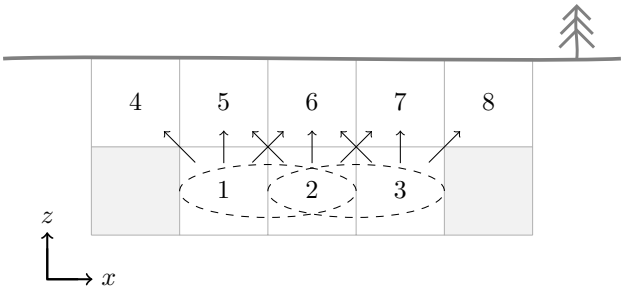


Figure 5.5 A small example block model used to illustrate the BZ subproblem

If each block is allowed to be mined in one of two time periods and routed to one of three destinations, there are ultimately $8 \times 2 \times 3 = 48$ individual block nodes in the subproblem. If the operational constraints are initially ignored the subproblem follows in Figure 5.6. Now for each block there are six nodes which are notated with a three digit number such that the first digit is the original block index, the second digit is the time period index and the third digit is the destination index.

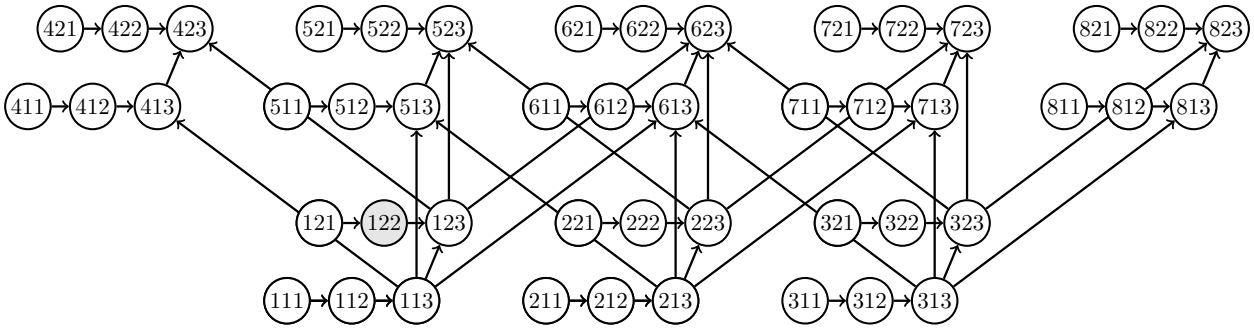


Figure 5.6 The base precedence constraints and block nodes in the BZ subproblem. Source and sink arcs are omitted.

For example if block one is to be routed to the second destination in the second phase that means that node 122 (the gray node in Figure 5.6) must be mined. This *requires* nodes 123, 423,

523, and 623 to also be mined, which is essentially saying: If block one is to be mined in period two, then blocks four, five and six *must* also have been mined by period two, to any destination. This does not preclude mining those other blocks in earlier time periods or to other destinations, the precedence constraints just say they must be mined by at least the same time period to any destination.

One important aspect of the subproblem highlighted by the example is that the by variable reformulation for multiple destinations increases the size of the subproblem unnecessarily. That is, the nodes corresponding to destinations of lower index can be combined together into a single node that takes the value of the maximum valued destination. This is an important optimization which reduces the size of the subproblem substantially, and is further discussed in several references [101–103].

When operational constraints are included additional nodes and precedence constraints are required. For this example if minimum mining width constraints are enforced on the final pit limits there are two additional nodes that must connect to 123 and 223, and 223 and 323 respectively as shown in Figure 5.7. Additionally in Figure 5.7 the destinations are collapsed into a single node each.

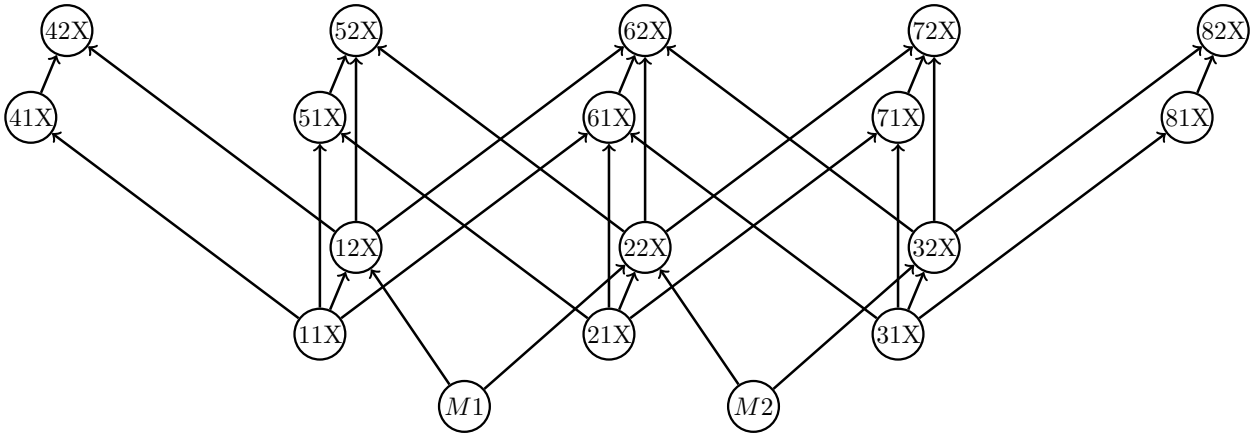


Figure 5.7 BZ Subproblem with collapsed destination nodes and two minimum mining width constraints. Source and sink arcs are omitted.

5.3.3 Integerization

The Bienstock-Zuckerberg algorithm solves the linear relaxation of the direct block scheduling problem which allows for partially mining blocks and creates inoperable schedules that are not directly usable for downstream applications. Therefore it is necessary to create integer feasible schedules through some additional means.

Chicoisne et al propose the TOPOSORT heuristic which uses the linear relaxation result as a guide to round the partially mined blocks to integer values while respecting some of the original constraints. Additionally they propose a local search heuristic that is used in combination with the TOPOSORT heuristic to obtain integer feasible solutions.

Aras describes a procedure for computing an integer feasible solution following the application of their modified BZ algorithm [103]. In practice their approach works well, and the gap between the LP solution as computed by BZ and the IP is generally very small.

One potential avenue for future work is to use the BZ algorithm as part of a branch and bound integerization process. Intelligently selecting the variables to restrict to integer values in the branching process may allow for higher quality integer solutions although the sheer number of variables could lead to problems. The orthogonal columns could be retained across levels of the tree to allow for the master problem to be solved more efficiently without having to start over from the beginning. This remains to be explored.

5.3.4 Implementation Details

The BZ algorithm can be implemented relatively efficiently in a computer program on top of two major components: a solver for linear programs that provides dual values on relevant constraints, and a flow based solver for solving the constructed ultimate pit problem instances. The difficulty of the implementation is in ensuring that all relevant bookkeeping information is routed correctly and the master and sub problems are constructed correctly. A prototype implementation of the BZ algorithm using the Gurobi C++ application programming interface to solve the master problem and MineFlow to solve the constructed subproblems was developed.

None of the reviewed discussions about the BZ algorithm describe the data structure used to store the orthogonalized columns. A naïve approach is not recommended as incorporating new columns from the subproblem and computing the new orthogonalized pits can be a laborious

process. The partition refinement data structure, [141, 142], is one high quality data structure for this component of a BZ implementation and can easily be extended to maintain the value of each orthogonalized pit and all of the information required to create the master problem's constraints.

The subproblem ultimate pit instances remain the same size throughout the decomposition process, and relatively few block values are modified by the duals from the previous master problem solution. Therefore it is important to use a solver that can re-use the previous solution's information to more rapidly compute the next.

However the size of the master problem does grow rapidly as additional columns are incorporated and orthogonalized. In order to prevent the number of columns from reaching unmanageable levels Bienstock and Zuckerberg propose a coarsification process which, when necessary, replaces the collection of orthogonal aggregates with some smaller set that spans the current solution by having only one orthogonal aggregate for each unique value of λ. Special care must be taken to prevent cycling. Munoz et al suggest only applying this coarsification process on iterations where the value of the objective (Equation 5.29) strictly increases [102].

Interestingly, this coarsification process did not yield improved convergence in the cases considered here. Instead when coarsification was applied the process took many additional iterations that obviated any runtime improvements realized by solving the master problem more quickly with fewer variables. This process should be considered sparingly perhaps only when the master problem reaches truly unmanageable levels, or only on those columns that are not a part of the current best LP solution. Another possible explanation is that the problems considered herein were not sufficiently sophisticated to necessitate the coarsification process. Problems with more variables, more constraints, or more difficult types of constraints may benefit from the coarsification process.

When possible it is generally worth seeding the master problem with a more useful initial set of orthogonal columns. This includes splitting up all of the blocks by time period, destination, and potentially even by bench. In some cases this lead to as much as a 50% reduction in run time compared to beginning with all of the blocks in one single column. Determining the best strategies for initializing, splitting up, and merging these columns is a ripe area for future research especially in the context of complicated side constraints.

5.3.4.1 MineLib Results

The MineLib library of test problem instances contains eleven constrained pit limit problems, or '`cpit`' problems, which are a special case of the general direct block scheduling problem [138]. The only side constraint considered in the constrained pit limit problems are resource constraints as in Equations 5.5 and 5.6. These problems were used to verify the developed prototype BZ implementation. The problem instances and results are tabulated in Table 5.1 and Table 5.2.

Table 5.1 Summary information of the MineLib '`cpit`' problem instances.

Name	Number of Blocks	Number of Precedence Constraints	Number of Phases	Number of Side Constraints
`newman1`	1,060	3,922	6	12
`zuck_small`	9,400	145,640	20	40
`kd`	14,153	219,778	12	12
`zuck_medium`	29,277	1,271,207	15	30
`p4hd`	40,947	738,609	10	20
`marvin`	53,271	650,631	20	40
`w23`	74,260	764,786	12	36
`zuck_large`	96,821	1,053,105	30	60
`sm2`	99,014	96,642	30	60
`mclaughlin_limit`	112,687	3,035,483	15	15
`mclaughlin`	2,140,342	73,143,770	20	20

The linear relaxation objective value as computed by the prototype BZ algorithm deviate slightly, by less than 1%, from the reported results in Espinoza et al [138]. Upon closer inspection this is caused by the provided solution files from MineLib not always adhering to the capacity constraints precisely. This may be due to inexact tolerances or numerical instability, as many of the values in the MineLib dataset are large when considered in the context of floating point numbers, especially when accounting for how the aggregation process sums many value and tonnage coefficients together. Overall the results from MineLib and the prototype BZ implementation for the linear relaxation are in agreement and these small discrepancies do not have a big impact.

The prototype BZ implementation was extended to compute the IP feasible results as well, and the results are summarized in Table 5.2. In all eleven cases the prototype BZ implementation was able to find schedules with higher objective value than those reported by MineLib.

Table 5.2 IP results from applying the prototype BZ implementation to the Minelib 'cpit' problem instances.

Name	MineLib Objective	Prototype BZ Objective	Percent Improvement	Elapsed time
newman1	23,483,671	24,176,579	3.0%	9s
zuck_small	788,652,600	789,066,986	0.1%	40s
kd	396,858,193	402,485,039	1.4%	16s
zuck_medium	615,411,415	618,075,337	0.4%	1m 22s
p4hd	246,138,696	247,089,680	0.4%	39s
marvin	820,726,048	822,163,289	0.2%	38s
w23	392,226,063	393,068,316	0.2%	2m 26s
zuck_large	56,777,190	56,846,147	0.1%	14m 24s
sm2	1,645,242,774	1,647,879,436	0.2%	12m 32s
mclaughlin_limit	1,073,327,197	1,075,862,841	0.2%	3m 4s
mclaughlin	1,073,327,197	1,075,930,704	0.2%	10m 37s

The prototype BZ implementation uses a combination of Aras's approach and the TOPOSORT heuristic to solve for the final integer feasible result. The orthogonal aggregates are pre-seeded by bench, phase, and a few nested pits calculated without any side constraints before calculating the linear relaxation result. At this stage the TOPOSORT heuristic is applied to compute a high quality satisfying result, but is not taken as the final answer. This solution is orthogonalized into the original aggregates and the whole set of columns is handed off to Gurobi to compute the final integer feasible solution. In the largest MineLib example, `mclaughlin`, this final IP had over 30,000 columns which is far less than the original $2,140,342 \times 20 = 42,806,840$ nodes. In practice this combined approach performs well.

5.4 Case Studies

The proposed methodology is applied to a small 3D example with three phases and the well known McLaughlin gold deposit with three destinations and ten phases. Note that although the name is shared, this McLaughlin dataset is subtly different than the one included in MineLib.

5.4.1 Small 3D Example

This small example was first used in Dağdelen 1985 [27]. The model is a synthetic, small, high grade copper deposit with 5,400 blocks arranged in a $30 \times 30 \times 6$ regular block model. Each block measures $100 \times 100 \times 45$ feet. The model contains 64 blocks of ore with an average grade of 3.7%

copper content. A three time period schedule is sought with capacities of 19 ore tons in the first period, 21 ore tons in the second, and 24 ore tons in the third.

Dağdelen's schedule, as computed with nine rounds of discounting block values, achieves a NPV of $139,569 without operational constraints. The prototype BZ implementation, without operational constraints, achieves a NPV of $140,660. With minimum mining width constraints corresponding to 2×2 blocks, the NPV is reduced to $138,527, and with 3×3 minimum mining width constraints the NPV is reduced to $134,536. Planar sections through the schedules are given in Figure 5.8.

5.4.2 McLaughlin Dataset

The McLaughlin mining complex is simple, with three possible destinations for each block: a mill, a leach pad, and a waste dump. The economic parameters used in this case study are carried over from Aras, [103], and tabulated in Table 5.3.

Table 5.3 Economic parameters used in the McLaughlin case study. Same as Aras 2018 [103].

Parameter	Value
Gold price	1,250 $/oz
Mill cost	12 $/t
Leach cost	6 $/t
Mill recovery	90%
Leach recovery	70%
Discount rate	12.5%

The first step is to compute the original ultimate pit limits using MineFlow. For the ultimate pit limit the discount rate is not used and each block is assumed to be routed to the highest value destination. No capacity constraints or other side constraints are considered, and for this first calculation no operational constraints are included. The naïve ultimate pit is shown in Figure 5.9, it mines only 258,054 of the 2,847,636 input blocks and achieves an undiscounted value of $2.2 billion with these parameters and assumptions. Constant 45° precedence constraints using eight benches of arcs were used.

The ultimate pit with minimum mining width constraints was also calculated using the bounding procedure and the methodology developed in Chapter 4. For this case study the ultimate pit satisfying a 5x5 minimum mining width reduces the contained undiscounted value by

$10 million and reduces the overall size of the pit by 3,225 blocks. This reduction in value represents bringing the unrealistic original value closer to an actually attainable value.

The blocks within the mining width feasible ultimate pit were extracted and used for the direct block scheduling procedure using the prototype Bienstock-Zuckerberg algorithm built on Gurobi and MineFlow. The process capacities, which form the main side constraints, for each period were taken from Aras 2018 and are tabulated in Table 5.4.

Table 5.4 Process capacity by time period. Same as Aras 2018 [103].

Time Period	Mill Capacity (Tons)	Leach Capacity (Tons)
1	1,500,000	1,500,000
2	1,750,000	1,750,000
3	2,000,000	2,000,000
4	2,750,000	2,750,000
5	3,000,000	3,000,000
6	3,000,000	3,000,000
7	2,750,000	2,750,000
8	2,000,000	2,000,000
9	1,750,000	1,750,000
10	1,500,000	1,500,000

With these 20 side constraints and no operational constraints the overall NPV of the project as computed with the prototype BZ implementation is $1,485,402,500. The Linear relaxation has a value of $1,489,951,000 for a gap of roughly half of one percent. Adding a 5x5 minimum mining width constraint reduces the NPV to $1,484,989,000 but increases the compute time from five minutes and eight seconds to 53 minutes and 24 seconds. Cross sections of both of these results are shown in Figure Figure 5.10.

In this dataset the negative valued waste blocks are relatively low compared to the ore blocks, and it is more economical to expand the pit in most areas to satisfy minimum mining width constraints. Additionally, once the pit has already been expanded to satisfy minimum mining width constraints additional areas become economic to recoup some of that cost. That is not generally to be expected in all datasets.

5.5 Discussion

This chapter has extended the operational constraints described in Chapter 4 to the direct block scheduling problem for both *at* and *by* variables. Schedules can now be constructed that satisfy minimum mining width using the Bienstock-Zuckerberg algorithm and associated methods. Additionally a prototype BZ implementation which takes advantage of recent advances in BZ understanding was developed and applied to the MineLib series of problems computing higher valued solutions in all eleven cases. Further efforts on this solver, with and without operational constraints, is warranted.

Many other applications to large scale open-pit mine planning and scheduling problems are possible with some of the efforts described in this chapter.

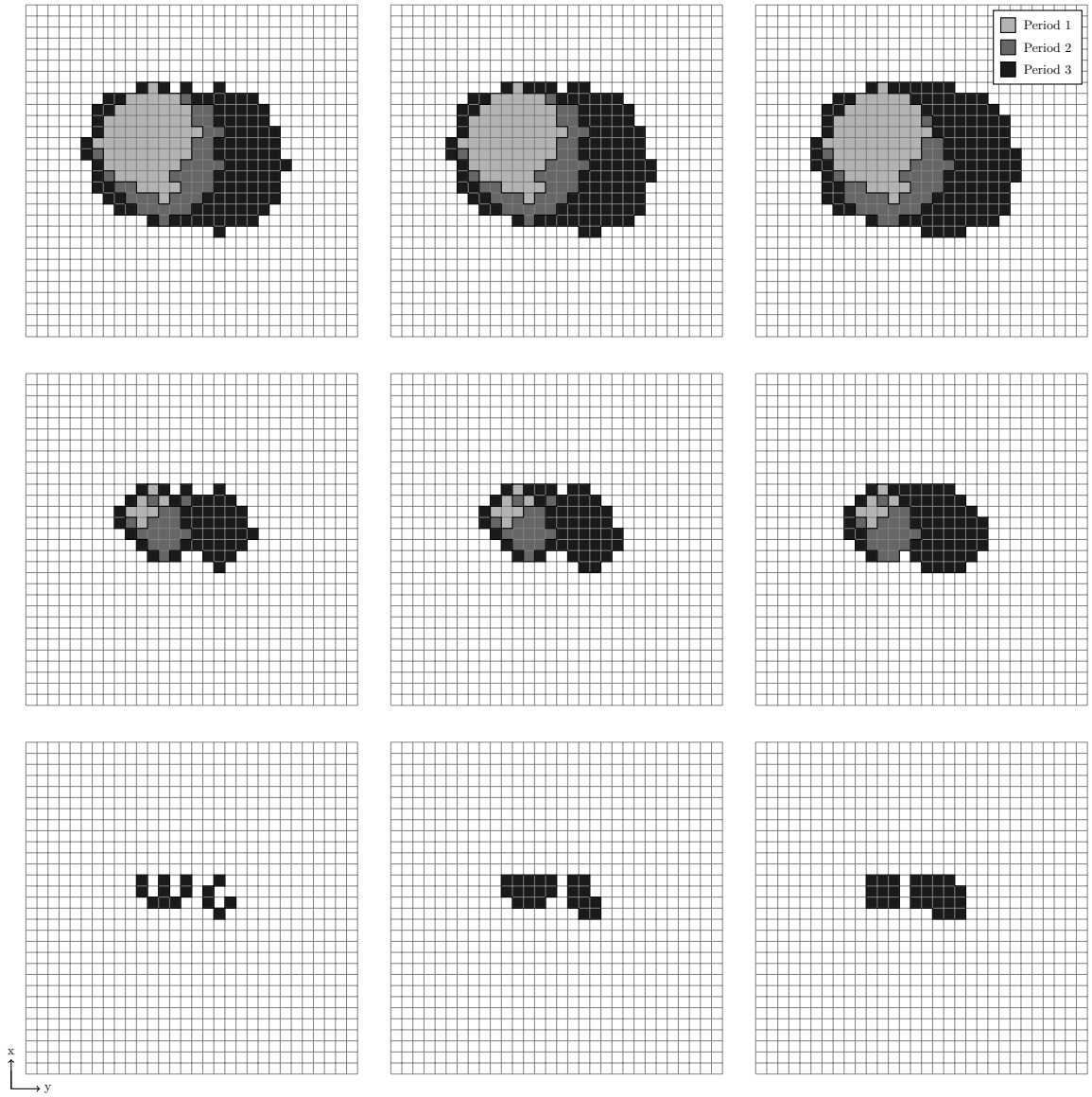


Figure 5.8 Planar sections through three schedules computed with Dağdelen's data. Left: No operational constraints, Middle: 2×2 minimum mining width constraints, Right: 3×3 minimum mining width constraints. Top: Bench five, Middle: Bench three, Bottom: Bench one

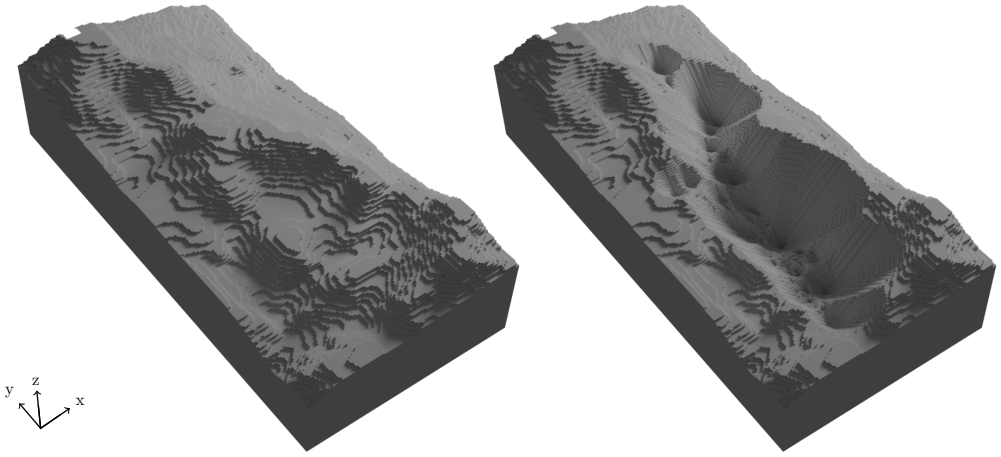


Figure 5.9 A 3D overview of the McLaughlin area of interest. Left: Original topography. Right: Naïve ultimate pit.

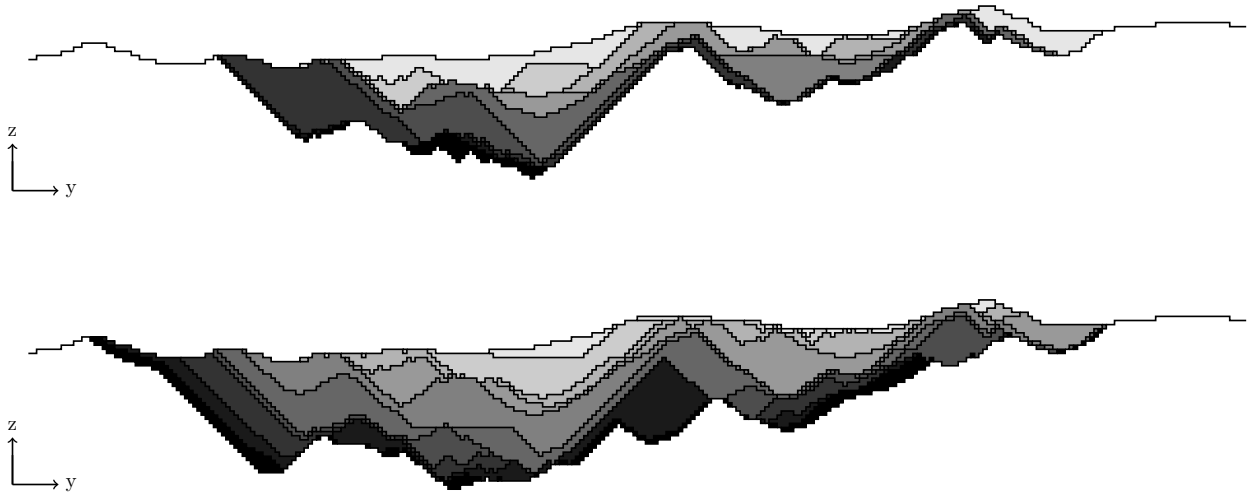


Figure 5.10 Cross sections through the McLaughlin block schedules, lighter blocks are mined earlier. Top: No operational constraints, Bottom: 5x5 minimum mining width.

CHAPTER 6
CONCLUSIONS

The main contribution of this dissertation is a methodology and program for solving the ultimate pit problem with minimum mining width constraints, however contributions were also made to the original ultimate pit problem and the block scheduling problem. The specific structure of the ultimate pit problem allows for modest optimizations to the conventional pseudoflow algorithm. The ultimate pit problem with minimum mining width constraints is now within reach even for very large models with dozens of millions of blocks and hundreds of millions of constraints. High quality solutions can be calculated very rapidly using the bounding procedures developed herein along with the Lagrangian relaxation guided solver. And finally, flexible formulations for the block scheduling problem with operational constraints alongside an initial prototype BZ solver capable of solving realistic models were presented. Each of the main contributions are summarized here, followed by ideas for future work, and final comments.

6.1 An Improved Ultimate Pit Solver – MineFlow

The ultimate pit problem remains a relevant problem in long range open-pit mine planning, either as a standalone problem in the early stage of the project or as a subproblem in more complicated optimization procedures or design applications. Solving for the ultimate pit as quickly as possible is a worthwhile goal that benefits both academia and industry alike. MineFlow, developed in Chapter 3, is a strong contender for the fastest ultimate pit solver currently available, taking advantage of several important optimizations that are possible specifically in the ultimate pit problem where only the minimum cut is desired and all arcs are of a similar form. The notation developed in this chapter should also be of moderate pedagogical value for those interested in solving for ultimate pits with pseudoflow. The software, which is readily available online or from the author, should continue to prove to be an important and useful tool for open-pit mine planning.

Additionally this chapter saw the development of useful ideas on how best to generate and evaluate precedence constraints with a heavy emphasis on efficiency and accuracy. The importance of changing the conventional paradigm from generating *all* precedence constraints

before solving into starting the solve immediately and only generating precedence constraints as necessary was also highlighted.

Future work on this topic could be to evaluate the recent work from Chen et al in 2022 that describe an algorithm for determining the maximum flow in near-linear time. Although there may be challenges with creating a workable implementation, some of the ideas may translate into the currently more practical methods. Additionally there may be room for further developments by switching between a depth first and breadth first strategy for incorporating precedence constraints that leads to normalized trees of higher quality with less splitting operations. This could have a tangible impact on the overall solution time.

6.2 Ultimate Pit Problem with Minimum Mining Width Constraints

Incorporating minimum mining width constraints directly into the ultimate pit optimization process is of the utmost importance in the early stages of long-range open-pit mine planning. Ignoring these constraints leads to unrealistic pits which overestimate the value of any given mineral deposit. These unrealistic pit values can lead to costly suboptimal decisions, and unwelcome surprises during the manual design and refinement process later.

This chapter saw the development of a concise, simple, and powerful formulation for the ultimate pit problem with minimum mining width constraints and several viable solution approaches. An extensive computational comparison was completed in order to validate that the work described in this chapter is actually applicable to a wide range of realistic datasets and deposits.

Additionally, methods were developed to generate inner and outer bounding pits which vastly reduce the size of the problem. This is necessary because it is shown that the ultimate pit problem with minimum mining width constraints is $\mathcal{NP}$-complete, which is a valuable result for future researchers looking to incorporate operational constraints into their open-pit mine planning problems. This result helps to protect future researchers from spending fruitless efforts trying to develop a polynomial time approach specifically for this problem.

Future work on the ultimate pit problem with minimum mining width constraints could be focused on improved heuristics, or even a completely different formulation that has different useful characteristics. The Lagrangian relaxation guided approach contains a step which evaluates

'nearby' satisfying pits in an effort to generate higher valued solutions that do not immediately fall out of the iterative process. This step deserves additional effort to improve its speed and its ability to generate high quality nearby solutions.

The Bienstock Zuckerberg algorithm, when combined with the Lagrangian relaxation guided solver, proved to generate the best result in all cases evaluated.

Finally, although the commercial branch and bound based optimizers were unable to make much headway on the larger models, their abilities on the smaller models are promising and perhaps combining the approaches developed in this chapter with the commercial solvers would be useful. For example, the approaches developed herein could be used to provide so called MIP starts, or additional bounding information, which could lead to a higher quality results faster.

6.3 The Block Scheduling Problem with Operational Constraints

The block scheduling problem is far more complicated than the single time period ultimate pit problem, but also more realistic and potentially more useful. Formulations for incorporating operational constraints, including minimum mining width constraints and minimum pushback width constraints, were developed in Chapter 5 for the most common variable types. Additionally, a prototype Bienstock Zuckerberg based solver was developed using Gurobi for the master problem and MineFlow for the sub problem. This solver takes advantage of the nature of the operational constraints and provide operationally feasible block scheduling solutions rapidly.

Future work for the block scheduling problem with operational constraints may include efforts to create even more operationally realistic schedules that account for such concerns as bench access. Formulations for these additional operational constraints are expected to be quite complicated. Additionally, methods for further enhancing the Bienstock Zuckerberg algorithm could be considered. Strategies for managing the orthogonal columns more effectively to balance the time spent solving the master problems versus the overall convergence rate should be investigated.

6.4 Final comments

In this dissertation a computationally efficient and flexible approach to incorporating minimum mining width constraints into the ultimate pit problem was presented alongside modest

improvements to the pseudoflow algorithm. These approaches were evaluated on real world datasets and their suitability was demonstrated. An initial exploration of operational constraints in the direct block scheduling problem was completed. The ultimate pit problem with minimum mining width constraints was shown to be $\mathcal{NP}$-complete. Finally, several smaller contributions: a useful notation for the pseudoflow algorithm, an algorithm for the two dimensional ultimate pit problem with minimum mining width constraints, a means to evaluate the accuracy of a given precedence pattern, and a dynamic programming algorithm for selecting evenly spaced pushbacks following parametric analysis, complete the dissertation.

The work completed herein can be used to assist long range open-pit mine planning engineers to more efficiently and responsibly use the Earth's natural resources.

REFERENCES

[1] Michael G Nelson. 6.1 evaluation of mining methods and systems. In Peter Darling, editor, *SME Mining Engineering Handbook*, volume 1. SME, 2011.

[2] Andrew Wetherelt and Klaas Peter van der Wielen. 10.1 introduction to open-pit mining. In Peter Darling, editor, *SME Mining Engineering Handbook*, volume 1. SME, 2011.

[3] MINING.com Editor. Ranked: World's 10 biggest underground mines by tonnes of ore milled, december 2021. URL https://www.mining.com/featured-article/ranked-worlds-10-biggest-underground-mines-by-tonnes-of-ore-milled/.

[4] Benjamin C Koskiniemi. Hand methods. *Open pit mine planning and design*, pages 187–195, 1979.

[5] Alexandra M Newman, Enrique Rubio, Rodrigo Caro, Andrés Weintraub, and Kelly Eurek. A review of operations research in mine planning. *Interfaces*, 40(3):222–245, 2010.

[6] Yves Lizotte. The economics of computerized open-pit design. *International Journal of Surface Mining, Reclamation and Environment*, 2(2):59–78, 1988.

[7] R Evans, CJ Moran, and D Brereton. Beyond npv—a review of valuation methodologies and their applicability to water in mining. *Proc, Water in Mining*, pages 97–103, 2006.

[8] David Whittle. 10.2 open-pit planning and design. In Peter Darling, editor, *SME Mining Engineering Handbook*, volume 1. SME, 2011.

[9] Mark E Gershon. Optimal mine production scheduling: evaluation of large scale mathematical programming approaches. *International journal of mining engineering*, 1(4): 315–329, 1983.

[10] K Dağdelen. Open pit optimization-strategies for improving economics of mining projects through mine planning. In *17th International Mining Congress and Exhibition of Turkey*, pages 117–121, 2001.

[11] J Whittle. Beyond optimization in open pit design. In *Proceedings Canadian conference on computer applications in the mineral industries*, pages 331–337, 1988.

[12] William A Hustrulid, Mark Kuchta, and Randall K Martin. *Open pit mine planning and design, two volume set & CD-ROM pack*. CRC Press, 2013.

[13] Mario E Rossi and Clayton V Deutsch. *Mineral resource estimation*. Springer Science & Business Media, 2013.

[14] William Lowrie and Andreas Fichtner. *Fundamentals of geophysics*. Cambridge university press, 2020.

[15] Clayton V Deutsch, André G Journel, et al. Geostatistical software library and user's guide. *Oxford University Press*, 8(91):0–1, 1992.

[16] Edward H Isaaks and Mohan R Srivastava. *Applied geostatistics*. Oxford University Press, 1989.

[17] Andre G Journel and Charles J Huijbregts. *Mining geostatistics*. Blackburn Press, 1976.

[18] M Jamshidi and M Osanloo. Determination of block economic value in multi-element deposits. In *6th International Conference in Computer Applications in the Minerals Industries. Istanbul, Turkey*, 2016.

[19] T Tholana, C Musingwini, and MM Ali. A stochastic block economic value model. In *Proceedings of the Mine Planners Colloquium 2019: Skills for the Future—Yours and Your Mine's*, pages 35–50. The Southern African Institute of Mining and Metallurgy Johannesburg . . . , 2019.

[20] Helmut Lerchs and Ingo Grossmann. Optimum design of open-pit mines. In *Operations Research*, volume 12, page B59, 1965.

[21] James W Gilbert. *A mathematical model for the optimal design of open pit mines*. PhD thesis, University of Toronto, 1966.

[22] Michael P Lipkewich and Leon Borgman. Two-and three-dimensional pit design optimization techniques. *A decade of digital computing in the mineral industry*, pages 505–523, 1969.

[23] T Chen. 3d pit design with variable wall slope capabilities. In *14th symposium on the application of computers and operations research in the mineral industries (APCOM), New York*, 1976.

[24] Louis Caccetta and Lou Giannini. Generation of minimum search patterns in the optimum design of open pit mines. *AIMM Bull. Proc.*, 293:57–61, 07 1988.

[25] Reza Khalokakaie, Peter A Dowd, and Robert J Fowell. Lerchs–grossmann algorithm with variable slope angles. *Mining Technology*, 109(2):77–85, 2000.

[26] Seyed-Omid Gilani and Javad Sattarvand. A new heuristic non-linear approach for modeling the variable slope angles in open pit mine planning algorithms. *Acta Montanistica Slovaca*, 20(4):251–259, 2015.

[27] Kadri Dağdelen. *Optimum multi period open pit mine production scheduling.* PhD thesis, Colorado School of Mines, 1985.

[28] Boleslaw Tolwinski and Robert Underwood. A scheduling algorithm for open pit mines. *IMA Journal of Management Mathematics*, 7(3):247–270, 1996.

[29] Georges Matheron. Le paramétrage des contours optimaux. *Technique notes*, 401:19–54, 1975.

[30] R Vallet. Optimisation mathematique de l'exploitation d'une mine a ciel ouvert ou le problem de l'enveloppe. *Annales des Mine de Belgique*, pages 113–135, 1976.

[31] Kadri Dağdelen and Dominique François-Bongarçon. Towards the complete double parameterization of recovered reserves in open pit mining. *Proceedings of 17th international APCOM symposium*, pages 288–296, 1982.

[32] Kadri Dağdelen. Cutoff grade optimization. *Preprints-Society of Mining Engineers of AIME*, 1993.

[33] Kenneth F Lane. *The economic definition of ore: cut-off grades in theory and practice.* Mining Journal Books London, 1988.

[34] Jeff Whittle. A decade of open pit mine planning and optimization-the craft of turning algorithms into packages. *Proceedings of the APCOM 99 symposium*, 1999.

[35] I Isaaks, E. Treloar and T Elenbaas. Optimum dig lines for open pit grade control. In *Proceedings of Ninth International Mining Geology Conference 2014*, pages 425–432. The Australasian Institute of Mining and Metallurgy: Melbourne, 2014.

[36] K.P. Norrena Neufeld, C.T. and C.V. Deustch. Guide to geostatistical grade control and dig limit determination. *Guidebook Series*, 1:63, 2005.

[37] M Tabesh and H Askari-Nasab. Automatic creation of mining polygons using hierarchical clustering techniques. *Journal of Mining Science*, 49(3):426–440, 2013.

[38] Matthew Deutsch. A branch and bound algorithm for open pit grade control polygon optimization. *Proc. of the 19th APCOM*, 2017.

[39] Louis Caccetta and Stephen P Hill. An application of branch and cut to open pit mine scheduling. *Journal of global optimization*, 27(2-3):349–365, 2003.

[40] Matthew Deutsch, Eric Gonzalez, and Michael Williams. Using simulation to quantify uncertainty in ultimate-pit limits and inform infrastructure placement. *Mining Engineering*, 67(12), 2015.

[41] AD Mwangi, Zh Jianhua, H Gang, RM Kasomo, and MM Innocent. Ultimate pit limit optimization methods in open pit mines: A review. *Journal of Mining Science*, 56(4): 588–602, 2020.

[42] Karo Fathollahzadeh, Mohammad Waqar Ali Asad, Elham Mardaneh, and Mehmet Cigla. Review of solution methodologies for open pit mine production scheduling problem. *International Journal of Mining, Reclamation and Environment*, 35(8):564–599, 2021.

[43] David Muir. Labeling lerchs-grossmann revisited with billion block model. preprint on webpage at researchgate.net/publication/ 340316084_Labeling_Lerchs-Grossmann_revisited_with_Billion_Block_Model, 02 2020.

[44] Thys B Johnson. *Optimum open pit mine production scheduling.* PhD thesis, California Univ Berkeley Operations Research Center, 1968.

[45] Robert Underwood and B Tolwinski. The lerchs grossmann algorithm from a dual simplex viewpoint. *26th Proceedings of the Application of Computers and Operations Research in the Mineral Industry*, pages 229–235, 1996.

[46] Dorit S Hochbaum. A new-old algorithm for minimum-cut and maximum-flow in closure graphs. *Networks*, 37(4):171–193, 2001.

[47] Alexander Schrijver et al. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

[48] Jean-Claude Picard. Maximal closure of a graph and applications to combinatorial problems. *Management science*, 22(11):1268–1272, 1976.

[49] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network flows.* Cambridge, Mass.: Alfred P. Sloan School of Management, Massachusetts ..., 1988.

[50] Lester Randolph Ford and Delbert Ray Fulkerson. *Flows in networks.* Princeton university press, 1962.

[51] Steven Skiena. *Implementing discrete mathematics: combinatorics and graph theory with Mathematica.* Addison-Wesley Longman Publishing Co., Inc., 1991.

[52] Andrew V Goldberg and Robert E Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988.

[53] Boris V Cherkassky and Andrew V Goldberg. On implementing the push—relabel method for the maximum flow problem. *Algorithmica*, 19(4):390–410, 1997.

[54] Dorit S Hochbaum. The pseudoflow algorithm: A new algorithm for the maximum-flow problem. *Operations research*, 56(4):992–1009, 2008.

[55] Dorit S Hochbaum and James B Orlin. Simplifications and speedups of the pseudoflow algorithm. *Networks*, 61(1):40–57, 2013.

[56] Matthew Deutsch, Kadri Dağdelen, and Thys Johnson. An open-source program for efficiently computing ultimate pit limits: Mineflow. *Natural Resources Research*, Mar 2022. ISSN 1573-8981. doi: 10.1007/s11053-022-10035-w. URL https://doi.org/10.1007/s11053-022-10035-w.

[57] Gary D Bond. *A mathematical analysis of the Lerchs and Grossmann algorithm and the nested Lerchs and Grossmann algorithm.* PhD thesis, Colorado School of Mines, 1996.

[58] MT Pana. The simulation approach to open pit design. In *APCOM SYMPOSIUM*, volume 5, pages 127–138, 1965.

[59] Thomas R Carlson, John D Erickson, DT O'Brain, and Milton T Pana. Computer techniques in mine planning. *Mining Engineering*, 18(5):53–56, 1966.

[60] Randal J Barnes. Optimizing the ultimate pit. Master's thesis, Colorado School of Mines, 1982.

[61] FJ Gauthier and RG Gray. Pit design by computer at gaspe-copper-mines-limited. In *Canadian Mining and Metallurgical Bulletin*, volume 64, page 70. Canadian inst Mining Metallurgy petroleum, 1971.

[62] Robert Underwood and Boleslaw Tolwinski. A mathematical programming viewpoint for solving the ultimate pit problem. *European Journal of Operational Research*, 107(1):96–107, 1998.

[63] A Wright. Moving cone ii-a simple algorithm for optimum pit limits design. *Proceedings of the 28rd APCOM*, pages 367–374, 1999.

[64] R Khalokakaie. Optimum open pit design with modified moving cone ii methods. *Journal of engineering Tehran university*, 4:297–307, 2006.

[65] Reza Kakaie et al. A new algorithm for optimum open pit design: Floating cone method iii. *Journal of Mining and environment*, 2(2):118–125, 2012.

[66] JM Berlanga, R Cardona, and MA Ibarra. Recursive formulae for the floating cone algorithm. *International Journal of Surface Mining, Reclamation and Environment*, 3(3): 141–150, 1989.

[67] Gokhan Turan and Ahmet Hakan Onur. Optimization of open-pit mine design and production planning with an improved floating cone algorithm. *Optimization and Engineering*, pages 1–25, 2022.

[68] G Ares, C Castañón Fernández, ID Álvarez, D Arias, and AB Díaz. Open pit optimization using the floating cone method: A new algorithm. *Minerals*, 12:1–20, 2022.

[69] Dorit S Hochbaum and Anna Chen. Performance analysis and best implementations of old and new algorithms for the open-pit mining problem. *Operations Research*, 48(6):894–914, 2000.

[70] DCW Muir. Pseudoflow, new life for lerchs-grossmann pit optimisation. *Orebody Modelling and Strategic Mine Planning, AusIMM Spectrum Series*, 14, 2007.

[71] Bala G Chandran and Dorit S Hochbaum. A computational study of the pseudoflow and push-relabel algorithms for the maximum flow problem. *Operations research*, 57(2):358–376, 2009.

[72] Matthew Deutsch and Clayton V Deutsch. An open source 3d lerchs grossmann pit optimization algorithm to facilitate uncertainty management. *CCG Annual Report*, 15, 2013.

[73] Thys B Johnson and William R Sharp. *A Three-dimensional dynamic programming method for optimal ultimate open pit design*, volume 7553. Bureau of Mines, US Department of the Interior, 1971.

[74] Y Zhao and YC Kim. New graph theory algorithm for optimal ultimate pit design. *Transactions-society of mining engineers of aime*, pages 1832–1832, 1991.

[75] Yixian Zhao. *Algorithms for optimum design and planning of open-pit mines*. PhD thesis, The University of Arizona, 1992.

[76] Ernest Koenigsberg. The optimum contours of an open pit mine: An application of dynamic programming. *17th Application of Computers and Operations Research in the Mineral Industry*, pages 274–287, 1982.

[77] Shenggui Zhang and AM Starfield. Dynamic programming with colour graphics smoothing for open-pit design on a personal computer. *International Journal of Mining Engineering*, 3 (1):27–34, 1985.

[78] F.L. Wilke and E.A. Wright. Ermittlung der günstigsten endauslegung von hartgesteinstagebauen mittels dynamischer programmierung (determining the optimal ultimate pit for hard rock open pit mines using dynamic programming). *Erzmetall*, 37: 138–144, 1984.

[79] E Alaphia Wright. The use of dynamic programming for open pit mine design: some practical implications. *Mining Science and Technology*, 4(2):97–104, 1987.

[80] Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

[81] EA Dinic. Algorithm for solution of a problem of maximum flow in a network with power estimation. *Soviet Math*, 11:1277–1280, 1970.

[82] Valerie King, Satish Rao, and Rorbert Tarjan. A faster deterministic maximum flow algorithm. *Journal of Algorithms*, 17(3):447–474, 1994.

[83] Andrew V Goldberg and Satish Rao. Beyond the flow decomposition barrier. *Journal of the ACM (JACM)*, 45(5):783–797, 1998.

[84] James B Orlin. Max flows in o (nm) time, or better. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 765–774, 2013.

[85] Jan van den Brand, Yin-Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 919–930. IEEE, 2020.

[86] Tuncel M Yegulalp and JA Arias. A fast algorithm to solve the ultimate pit limit problem. In *23rd International Symposium on the Application of Computers and Operations Research in The Mineral Industries*, pages 391–398. AIME Littleton, Co, 1992.

[87] Ravindra K Ahuja and James B Orlin. A fast and simple algorithm for the maximum flow problem. *Operations Research*, 37(5):748–759, 1989.

[88] Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. *arXiv preprint arXiv:2203.00671*, 2022.

[89] C Meagher, R Dimitrakopoulos, and D Avis. Optimized open pit mine design, pushbacks and the gap problem - a review. *Journal of Mining Science*, 50(3):508–526, 2014.

[90] Roussos Dimitrakopoulos, CT Farrelly, and M Godoy. Moving forward from traditional optimization: grade uncertainty and risk effects in open-pit design. *Mining Technology*, 111 (1):82–88, 2002.

[91] R Dimitrakopoulos and S Ramazan. Uncertainty based production scheduling in open pit mining. *SME transactions*, 316, 2004.

[92] R Dimitrakopoulos, L Martinez, and S Ramazan. Optimising open pit design with simulated orebodies and whittle four-x: A maximum upside/minimum downside approach. *Australasian Institute of Mining and Metallurgy Publication Series. Perth, Australia, Australasian Institute of Mining and Metallurgy*, pages 201–206, 2007.

[93] M Godoy and R Dimitrakopoulos. A multi-stage approach to profitable risk management for strategic planning in open pit mines. In *Orebody Modelling and Strategic Mine Planning – Uncertainty and Risk Management International Symposium 2004*. The Australasian Institute of Mining and Metallurgy, 2004.

[94] Ady AD Van-Dúnem. *Open-pit mine production scheduling under grade uncertainty*. Colorado School of Mines, 2016.

[95] Barry King, Marcos Goycoolea, and Alexandra Newman. Optimizing the open pit-to-underground mining transition. *European Journal of Operational Research*, 257(1): 297–309, 2017.

[96] Kazuhiro Kawahata. New algorithm to solve large scale mine production scheduling problems by using the lagrangian relaxation method, a. *2000-2009-Mines Theses & Dissertations*, 2006.

[97] Natashia Boland, Irina Dumitrescu, and Gary Froyland. A multistage stochastic programming approach to open pit mine production scheduling with uncertain geology. *Optimization online*, pages 1–33, 2008.

[98] Marcos Goycoolea, Daniel Espinoza, Eduardo Moreno, and Orlando Rivera. Comparing new and traditional methodologies for production scheduling in open pit mining. In *Proceedings of APCOM*, pages 352–359, 2015.

[99] Beyime Tachefine and François Soumis. Maximal closure on a graph with resource constraints. *Computers & operations research*, 24(10):981–990, 1997.

[100] Atsushi Akaike. *Strategic planning of Long term production schedule using 4D network relaxation method*. PhD thesis, Colorado School of Mines, 1999.

[101] Daniel Bienstock and Mark Zuckerberg. A new lp algorithm for precedence constrained production scheduling. *Optimization Online*, pages 1–33, 2009.

[102] Gonzalo Muñoz, Daniel Espinoza, Marcos Goycoolea, Eduardo Moreno, Maurice Queyranne, and Orlando Rivera Letelier. A study of the bienstock–zuckerberg algorithm: applications in mining and resource constrained project scheduling. *Computational Optimization and Applications*, 69(2):501–534, 2018.

[103] Canberk Aras. *A new integer solution algorithm to solve open-pit mine production scheduling problems*. Colorado School of Mines, 2018.

[104] Renaud Chicoisne, Daniel Espinoza, Marcos Goycoolea, Eduardo Moreno, and Enrique Rubio. A new algorithm for the open-pit mine production scheduling problem. *Operations Research*, 60(3):517–528, 2012.

[105] Amina Lamghari, Roussos Dimitrakopoulos, and Jacques A Ferland. A variable neighbourhood descent algorithm for the open-pit mine production scheduling problem with metal uncertainty. *Journal of the Operational Research Society*, 65(9):1305–1314, 2014.

[106] W Brian Lambert, Andrea Brickey, Alexandra M Newman, and Kelly Eurek. Open-pit block-sequencing formulations: a tutorial. *Interfaces*, 44(2):127–142, 2014.

[107] Jorge Amaya, Daniel Espinoza, Marcos Goycoolea, Eduardo Moreno, Thomas Prevost, and Enrique Rubio. A scalable approach to optimal block scheduling. In *Proceedings of APCOM*, pages 567–575, 2009.

[108] Jeff Whittle. 5.3 open pit optimization. In Bruce Kennedy, editor, *Surface Mining*, pages 470–475. Society for Mining, Metallurgy and Exploration, Incorporated, Littleton, 1990. ISBN 0873351029.

[109] Christopher Wharton and Jeff Whittle. The effect of minimum mining width on npv. In *Optimizing with Whittle*, pages 173–178. Whittle Programming Pty. Ltd Perth, Western Australia, 1997.

[110] Peter Stone, Gary Froyland, Merab Menabde, Brian Law, Reza Pasyar, and PHL Monkhouse. Blasor - blended iron ore mine planning optimisation at yandi, western australia. In Roussos Dimitrakopoulos, editor, *Orebody Modelling and Strategic Mine Planning*, pages 285–288, 2007.

[111] M Zhang. An automated heuristic algorithm for practical mining phase design. In *17th International Symposium on Mine Planning and Equipment Selection*, 2008.

[112] M Zhang. Applying simulated annealing to practical mining phase design. *34th Application of Computers and Operations Research in the Mineral Industry. CIM, Vancouver*, pages 266–273, 2009.

[113] Y Pourrahimian, H Askari-Nasab, and DD Tannant. Production scheduling with minimum mining width constraints using mathematical programming. *18th International Symposium on Mine Planning and Equipment Selection*, 2009.

[114] Christopher Cullenbine, R Kevin Wood, and Alexandra Newman. A sliding time window heuristic for open pit mine block sequencing. *Optimization letters*, 5(3):365–377, 2011.

[115] Mohammad Tabesh, Clemens Mieth, and Hooman Askari-Nasab. A multi–step approach to long–term open–pit production planning. *International Journal of Mining and Mineral Engineering*, 5(4):273–298, 2014.

[116] KRJA Systems DBA Maptek. Vulcan 10 help documentation, 2016.

[117] Jean Serra. *Image analysis and mathematical morphology*. Academic press, April 1982.

[118] Guillermo Juarez, Ricardo Dodds, Adriana Echeverría, Javier Ibanez Guzman, Matías Recabarren, Javier Ronda, and E Vila-Echague. Open pit strategic mine planning with automatic phase generation. In *OREBODY MODELLING AND STRATEGIC MINE PLANNING SYMPOSIUM. Proceedings... AusIMM, Perth (WA)*, pages 24–26, 2014.

[119] Xiaoyu Bai, Denis Marcotte, Michel Gamache, D Gregory, and A Lapworth. Automatic generation of feasible mining pushbacks for open pit strategic planning. *Journal of the Southern African Institute of Mining and Metallurgy*, 118(5):514–530, 2018.

[120] Iain Farmer and Roussos Dimitrakopoulos. Schedule-based pushback design within the stochastic optimisation framework. *International Journal of Mining, Reclamation and Environment*, 32(5):327–340, 2018.

[121] Matthew Deutsch. Open-pit mine optimization with maximum satisfiability. *Mining, Metallurgy & Exploration*, 36(4):757–764, 2019.

[122] David Muir. Labeling lerchs-grossmann with minimum mining width constraints. preprint on webpage at researchgate.net/publication/ 343498297_Labeling_Lerchs-Grossmann_with_Minimum_Mining_Width_Constraints, 08 2020.

[123] Pierre Nancel-Penard and Nelson Morales. Optimizing pushback design considering minimum mining width for open pit strategic planning. *Engineering Optimization*, pages 1–15, 2021.

[124] Juan L Yarmuch, Marcus Brazil, Hyam Rubinstein, and Doreen A Thomas. A mathematical model for mineable pushback designs. *International Journal of Mining, Reclamation and Environment*, 35(7):523–539, 2021.

[125] Juan L Yarmuch, Marcus Brazil, Hyam Rubinstein, and Doreen A Thomas. A model for open-pit pushback design with operational constraints. *Optimization and Engineering*, pages 1–17, 2021.

[126] Ronald L Rardin and Ronald L Rardin. *Optimization in operations research*, volume 166. Prentice Hall Upper Saddle River, NJ, 1998.

[127] W Brian Lambert and Alexandra M Newman. Tailored lagrangian relaxation for the open pit block sequencing problem. *Annals of Operations Research*, 222(1):419–438, 2014.

[128] Leonid Vitalievich Kantorovich. A new method of solving of some classes of extremal problems. In *Dokl. akad. nauk sssr*, volume 28, pages 211–214, 1940.

[129] George B Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity analysis of production and allocation*, 13:339–347, 1951.

[130] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, 1984.

[131] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[132] William H Press, William T Vetterling, Saul A Teukolsky, and Brian P Flannery. *Numerical recipes*. Citeseer, 1988.

[133] Ingo Rechenberg. Evolutionsstrategie. *Optimierung technischer Systeme nach Prinzipien derbiologischen Evolution*, 1973.

[134] Ernst Althaus and Kurt Mehlhorn. Maximum network flow with floating point arithmetic. *Information processing letters*, 66(3):109–113, 1998.

[135] Free Software Foundation. The gnu multiple precision arithmetic library, 2023. URL https://www.gmplib.org.

[136] Brian W Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451, 1975.

[137] Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC genomics*, 21:1–13, 2020.

[138] Daniel Espinoza, Marcos Goycoolea, Eduardo Moreno, and Alexandra Newman. Minelib: a library of open pit mining problems. *Annals of Operations Research*, 206(1):93–114, 2013.

[139] IBM ILOG CPLEX. V12. 6: User's manual for cplex. *International Business Machines Corporation*, page 564, 2015.

[140] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL https://www.gurobi.com.

[141] Ravi Sethi. Scheduling graphs on two processors. *SIAM Journal on Computing*, 5(1):73–82, 1976.

[142] Robert Paige and Robert E Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[143] Robert J Fowler, Michael S Paterson, and Steven L Tanimoto. Optimal packing and covering in the plane are np-complete. *Information processing letters*, 12(3):133–137, 1981.

[144] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

APPENDIX A

THE ULTIMATE PIT PROBLEM WITH MINIMUM MINING WIDTH CONSTRAINTS IS NP-COMPLETE

Efficient algorithms exist to solve many different computational problems such as finding the shortest path through a graph, sorting large arrays, and solving the ultimate pit problem. In Section 2.2.3 a straightforward means by which the ultimate pit problem can be transformed into a max-flow / min-cut problem was described. This transformation can then be used with a wide range of algorithms, such as the pseudoflow algorithm, to obtain solutions quickly. These algorithms can obtain the solution in a number of steps which can be expressed as a polynomial in terms of the size of the input, and are therefore reasonably fast even as the problem size increases. Problems of this sort are said to be in $\mathcal{P}$, and there exist algorithms to solve them with a deterministic Turing machine in polynomial time.

It would be convenient if the ultimate pit problem with minimum mining width constraints could also be solved in polynomial time, however the reduction described in this appendix shows that this is not currently possible. Following this result it can only be declared that there is currently no known polynomial time algorithm for this problem, which is a useful theoretical result with some practical ramifications. Specifically this result places the ultimate pit problem with minimum mining width constraints among the $\mathcal{NP}$-complete problems which can only be solved by a non-deterministic Turing machine in polynomial time. It is not yet known if there is a algorithm that could solve this class of problems in polynomial time. This is a long-standing open problem commonly referred to as the $\mathcal{P}$ vs. $\mathcal{NP}$ problem, which is not considered here.

It is now known that the heuristic methods described in Chapter 4 are not going to be obviated by a clever reformulation of the problem into a pre-existing graph problem or something similar. If a clever trick existed to solve our problem in polynomial time it would also be able to solve all these other, much more heavily researched, problems as well.

The argument presented herein centers around showing that it is possible to transform an arbitrary 3-SAT problem into the ultimate pit problem with minimum mining widths. With this polynomial time reduction one can confidently say that if there were a very fast algorithm for the

ultimate pit problem with minimum mining widths there would also have a very fast algorithm for 3-SAT. One could take a 3-SAT problem, transform it using this process, solve that efficiently, and report back the answer.

The reduction described in this appendix modifies the reduction given by Fowler, Paterson, and Tanimoto in 1981 for the *planar geometric covering problem* where the goal is to determine whether some set of geometric objects can completely cover another set of points in the plane [143]. The primary difference between Fowler et al's reduction and this reduction is that in Fowler et al the goal was to limit the number of geometric shapes used, which roughly correspond to mining width sets. In the ultimate pit problem with minimum mining width constraints there is no restriction on the number of mining width sets. This reduction retains the character of their idea by embedding the entire problem in a plane of negative valued blocks but must contend with additional complications.

Section A.1 describes the 3-SAT problem which is a special case of the well known satisfiability problem. Section A.2 then strips away all of the nonvital elements of the ultimate pit problem with minimum mining width constraints. Specifically the precedence constraints are completely removed and the problem is transformed into a decision problem instead of an optimization problem. Finally, Section A.3 provides the polynomial time reduction from 3-SAT which yields the desired result.

A.1 3-SAT

3-SAT is a special case of the Boolean satisfiability problem; or SAT, which was the original problem shown to be $\mathcal{NP}$-complete [144]. Satisfiability is the problem of determining whether there is an assignment of values (true or false) to a set of Boolean variables which satisfies *all* the clauses of a particular formula in conjunctive normal form. Formulas in conjunctive normal form are expressed as a conjunction of disjunctions, or an 'and' of 'or's. In the 3-SAT special case these disjunctions consist of exactly three distinct literals. SAT formulas, with disjunctions of any length, can be transformed easily into 3-SAT instances although the details are not relevant here.

An example 3-SAT formula, ϕ, follows in Equation A.1. Each Boolean variable is indexed from the set of n Boolean variables X, as $x_1, x_2, ..., x_n$. In this small example n is equal to five. Each clause, indexed from the set of m clauses $\mathcal{C}$, as $C_1, C_2, ..., C_m$, is a disjunction of three

literals formed from those variables. This example has eight clauses the first of which is given as $C_1 = (x_1 \vee x_2 \vee \neg x_3)$. This implies that in order for clause one to be satisfied at least one of the following is true: x_1 is assigned true, x_2 is assigned true, or x_3 is assigned false. The $\vee$ symbol stands for 'or' and the $\neg$ symbol is the negation operator. The remaining seven clauses are joined with C_1 with the $\wedge$ operator which means 'and.'

$$\begin{aligned}\phi =&(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4) \wedge \\ &(x_1 \vee \neg x_2 \vee \neg x_5) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge \\ &(x_1 \vee \neg x_3 \vee x_5) \wedge (x_1 \vee \neg x_4 \vee x_5) \wedge \\ &(x_2 \vee x_4 \vee x_5) \wedge (\neg x_3 \vee x_4 \vee \neg x_5)\end{aligned} \tag{A.1}$$

The 3-SAT instance in Equation A.1 is satisfiable. For example assigning the following values to each variable, where 1 is true and 0 is false, satisfies all eight clauses. $x_1 \leftarrow 1$, $x_2 \leftarrow 0$, $x_3 \leftarrow 1$, $x_4 \leftarrow 1$, $x_5 \leftarrow 0$.

A great many problems can be specified with this seemingly restrictive set of rules including the well known vertex cover problem and graph coloring problem. It is straightforward to transform many problems into 3-SAT problems so a fast 3-SAT algorithm is highly sought after. However, being able to transform arbitrary problems into 3-SAT, or more general satisfiability problems clearly does not mean that the input problem is difficult. Instead if a problem is meant to be shown to be $\mathcal{NP}$-complete it must be shown that any arbitrary 3-SAT problem can be turned into an instance of that problem in polynomial time.

3-SAT is not an optimization problem and does not aim to maximize or minimize some objective function, although such variants do exist. Instead 3-SAT is a decision problem which only results in a yes or a no; the formula is satisfiable or not satisfiable. It is straightforward to transform an optimization problem into a series of decision problems. The general idea is to first solve the problem without the objective constructing a feasible solution, then if the solution exists proceed by introducing clauses which force the new objective value to exceed the previous objective value by some amount (if maximizing). This constructs a new decision problem which asks if a better result exists. If there is no satisfying solution then an upper bound on the objective has been determined which can be used to refine the working decision problem until the optimal solution is found.

A.2 The Simplified Operational Ultimate Pit Decision Problem in the Plane

It is sufficient to show that a simplified version of the ultimate pit problem with minimum mining width constraints is $\mathcal{NP}$-complete because any algorithm capable of solving the unsimplified version would also have to solve the simplified version. Therefore it is possible to completely ignore precedence constraints and turn the problem into a two dimensional planar problem of identifying mineable groups of blocks. The valid mining width sets are restricted to 3x3 sets of blocks[2] and the problem is formulated as a decision problem.

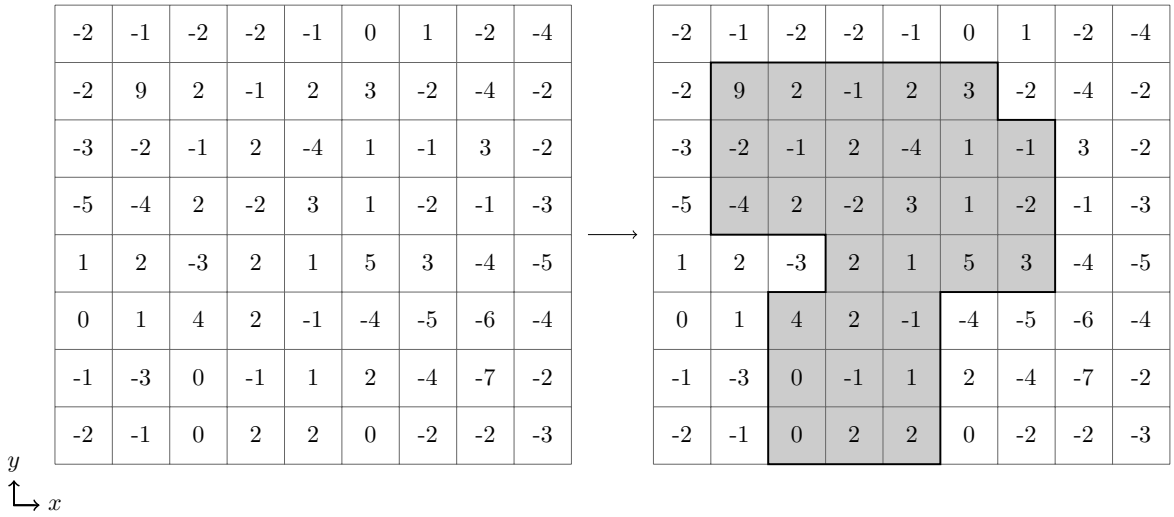


Figure A.1 Example simplified operational ultimate pit decision problem in the plane. Numbers in blocks are the EBV. If the requested lower bound is less than 28 the set of shaded blocks on the right is a valid selection corresponding to a 'yes' answer to the decision problem.

Given a 2D planar cross section through a regular block model of size n_x by n_y with integral economic block values for each block as $v_{x,y}$, and a single scalar lower bound on total value V; we seek an assignment $X_{x,y}$ of either 1 or 0 to each block such that the total value exceeds the lower bound (Equation A.2) and each mined block is a part of at least one 3x3 square of mined blocks. A small 9×8 example is shown in Figure A.1.

$$\sum_{x=0}^{n_x-1} \sum_{y=0}^{n_y-1} v_{x,y} X_{x,y} \geq V \tag{A.2}$$

[2]It is possible that the 2x2 and 2x3 cases are also NP-complete - however that is not shown in this appendix.

A.3 The Ultimate Pit Problem with Minimum Mining Width Constraints is $\mathcal{NP}$-Complete

Theorem 1. The Ultimate Pit Problem with Minimum Mining Width Constraints is $\mathcal{NP}$-complete.

Proof. A polynomial-time reduction of 3-SAT to the simplified ultimate pit problem with minimum mining width constraints is given. A 3-SAT formula with N variables and M clauses is encoded into a 2D grid of size $\mathcal{O}(M) \times \mathcal{O}(N)$ with mining width sets of size 3×3. Additionally a value V is provided such that this value can be achieved if and only if it is possible to satisfy the input formula.

At a high level the reduction involves representing each of the input variables as a 'wire' formed from large positive valued blocks embedded in a predominantly negative valued block model slice. Each wire is constructed as a loop consisting of an even number of high value blocks which have exactly two possible maximum valued solutions. The two parities correspond to assigning a value of true or false to the input variable in the satisfiability formula.

Each wire is then carefully attached to specific clause points following the input 3-SAT formula. The clause points are constructed such that there are exactly seven maximum valued solutions corresponding to one, or more, of the three Boolean variables being true. That is, only the solution where all three variables are false has a lower value than the seven others.

For clauses where a particular variable appears in its negated form it is necessary to flip the parity of the wire before it enters the clause point. It is also necessary to cross wires over one another in the plane strictly maintaining the parity of each wire. The following sections describe each of these individual components before showing how to connect them all together and complete the construction.

In all the following examples and figures the numbers within blocks are the economic block value. The hatched blocks have a very large negative value, for example -9,999, which completely removes the possibility of mining those blocks. They could also be removed from the problem. The dark shaded blocks are the mined blocks within each solution.

A.3.1 Wires

In Figure A.2 we see an example wire. It is straightforward to see that each of the two maximum valued solutions have the same value (in this case 344), and vitally these are the *only* two solutions which have a value of 344. Any other configuration of mined blocks would necessarily mine additional negative valued blocks and reduce the value, so if a solution is sought with a value of at least 344 the answer will be 'yes' with one of these two outputs. It is possible to extend the wire either horizontally or vertically by inserting additional rows and columns provided they follow the pattern.

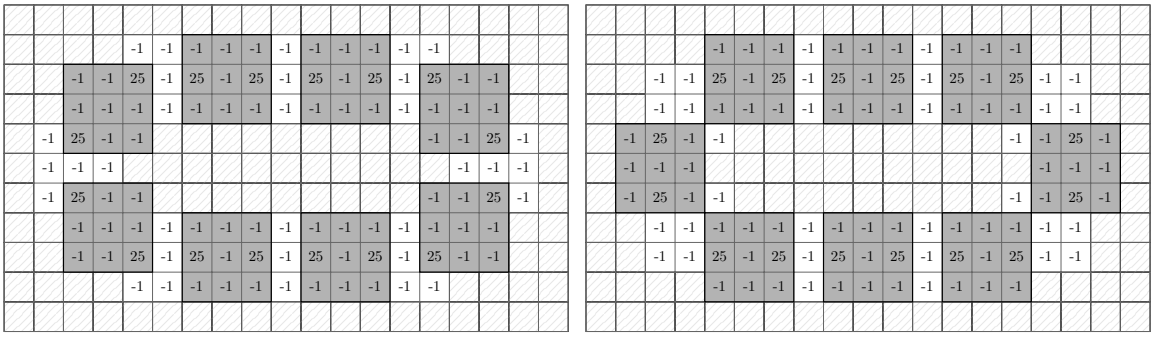


Figure A.2 The wire concept in the $\mathcal{NP}$-completeness proof. Each of the two solutions (left and right) have the same value and correspond to the two possible assignments (true and false).

A useful schematic representation for this wire is given in Figure A.3. In this representation the high value blocks are represented as nodes. Arcs are present between nodes if it is possible to place a 3x3 mining width and mine both nodes, and in these examples the bolded arcs correspond to mining width sets that are mined in a particular solution.

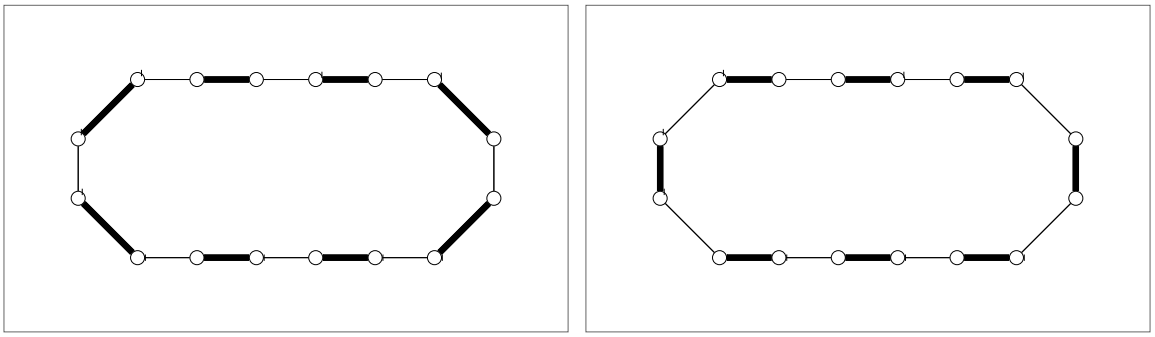


Figure A.3 The wire in Figure A.2 as a schematic instead of explicit block values. Bolded arcs correspond to mined mining width sets in the equivalent solutions.

A.3.2 Negation

Wires are constructed such that they always have exactly two equivalent parities with identical objective function values. These parities correspond to assigning the relevant input variable from the original 3-SAT problem a true or false value. However, when a variable is negated in a clause it must 'appear' to that clause as the other parity. This section, therefore, demonstrates how to negate a wire and have the parity appear different on either side of the negation.

A negation in a wire can be constructed by increasing the length of a wire by incorporating a wiggle. This is best understood by example as in Figure A.4. The top wire in is the same as in Section A.3.1 just extended horizontally. The bottom wire incorporates a wiggle to swap the perceived parity of the wire.

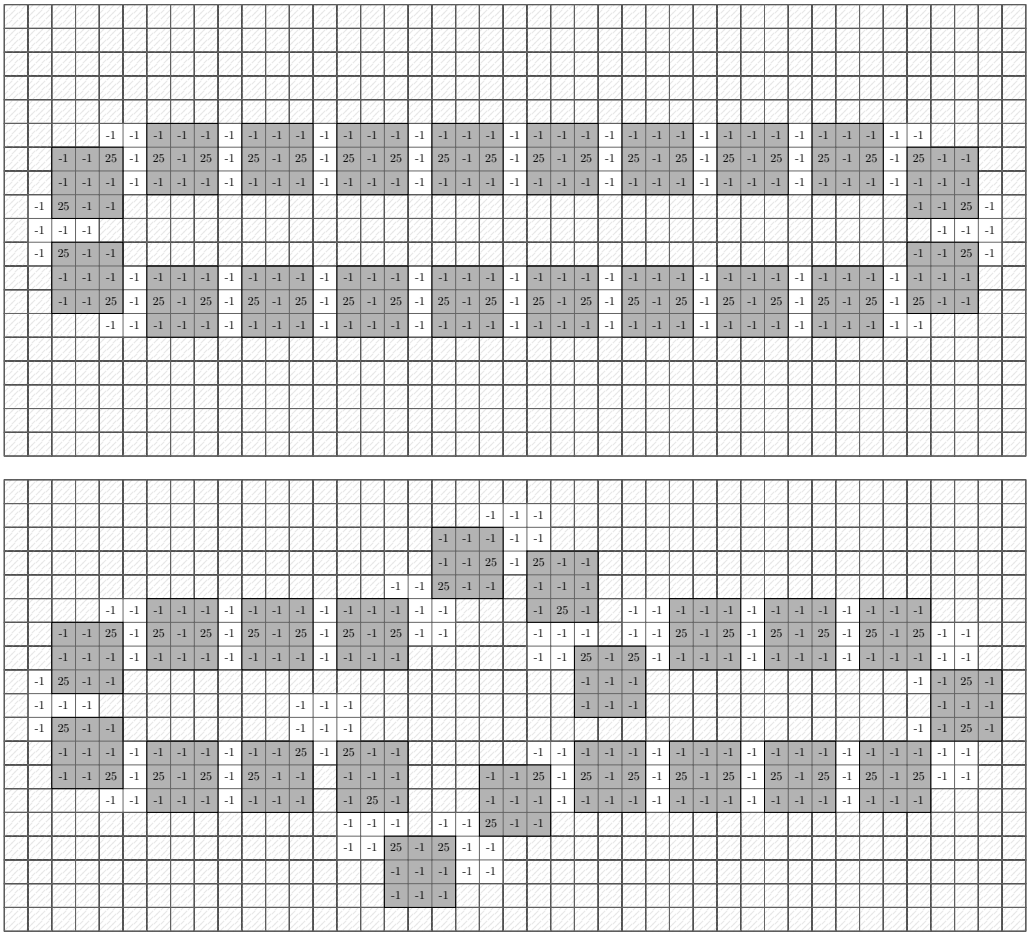


Figure A.4 A negation 'wiggle' incorporated into a wire. The top wire has exactly two parities (only one is shown) that appears the same on both sides. The bottom wire also has exactly two parities (again only one is shown), but the parities appear different on either side of the wiggle.

A.3.3 Crossovers

It is also necessary to cross wires over one another in order to connect them to various clause points and correctly encode the 3-SAT problem into the ultimate pit problem with minimum mining width constraints. The crossover is quite simple. The only concern is to ensure that the parity of the wires can not change which is achieved by using higher negative valued blocks at the point of the crossover and precluding certain possibilities of 'sharing' any negative value between mining width sets.

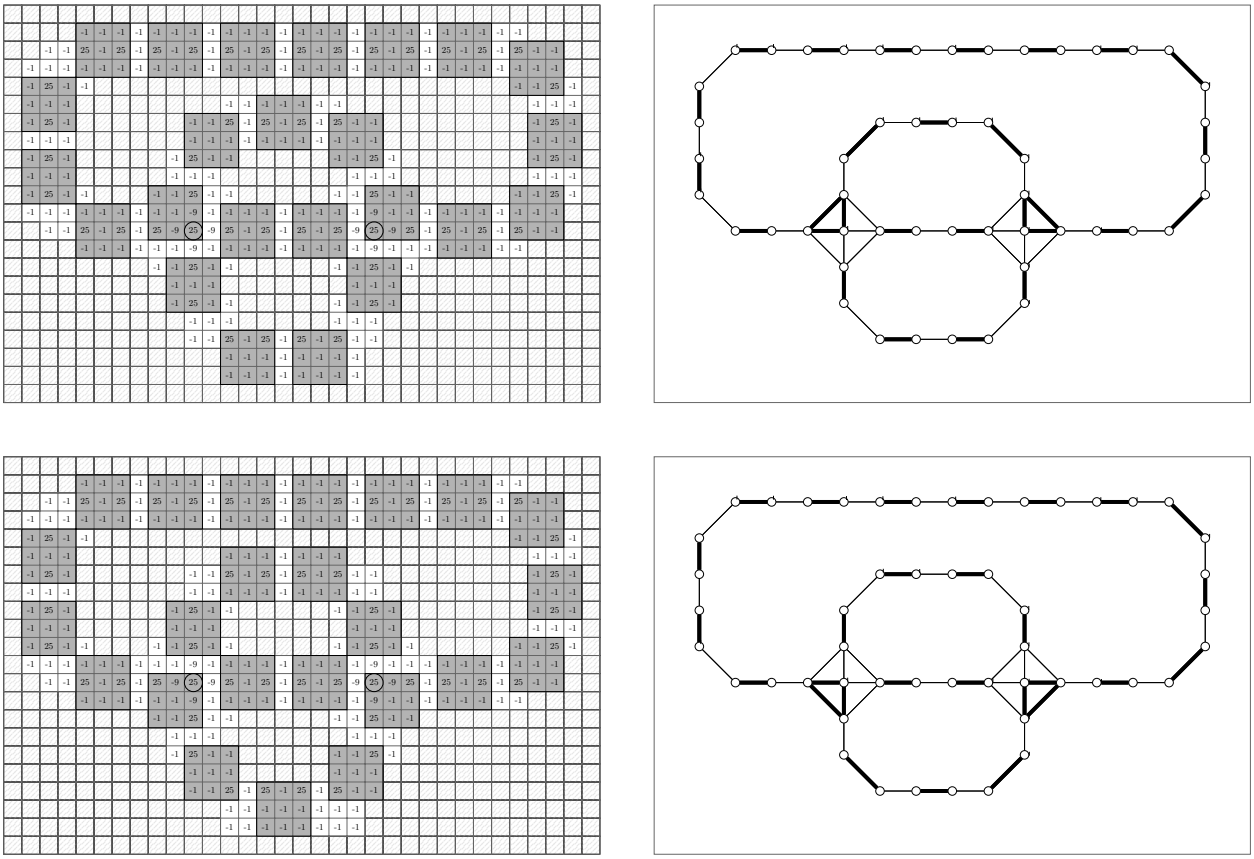


Figure A.5 A configuration demonstrating how to cross wires over other wires. The extra negative valued blocks next to the circled crossover points ensure that the parity of each wire is retained. Only two of the four possible solutions are shown alongside the relevant schematic.

The configuration in Figure A.5 has four equal valued solutions, one for each assignment of true or false to each of the two variables. Only two of the solutions are shown, alongside the relevant schematic representations. The other two solutions are similar.

A.3.4 Clause point

The clause point is the most complicated part of the construction and deviates the most from Fowler et al's construction. The general idea is to bring three wires corresponding to three variables into close proximity and overlap them in such a fashion that there are *exactly* seven maximum valued solutions corresponding to the cases where at least one of the wires is 'true'. It is essential that none of these configurations is preferred over the others, and that the parity of each wire must not switch in the overlapping section. This is achieved by using specially chosen block values that increase or decrease blocks within overlapping mining widths.

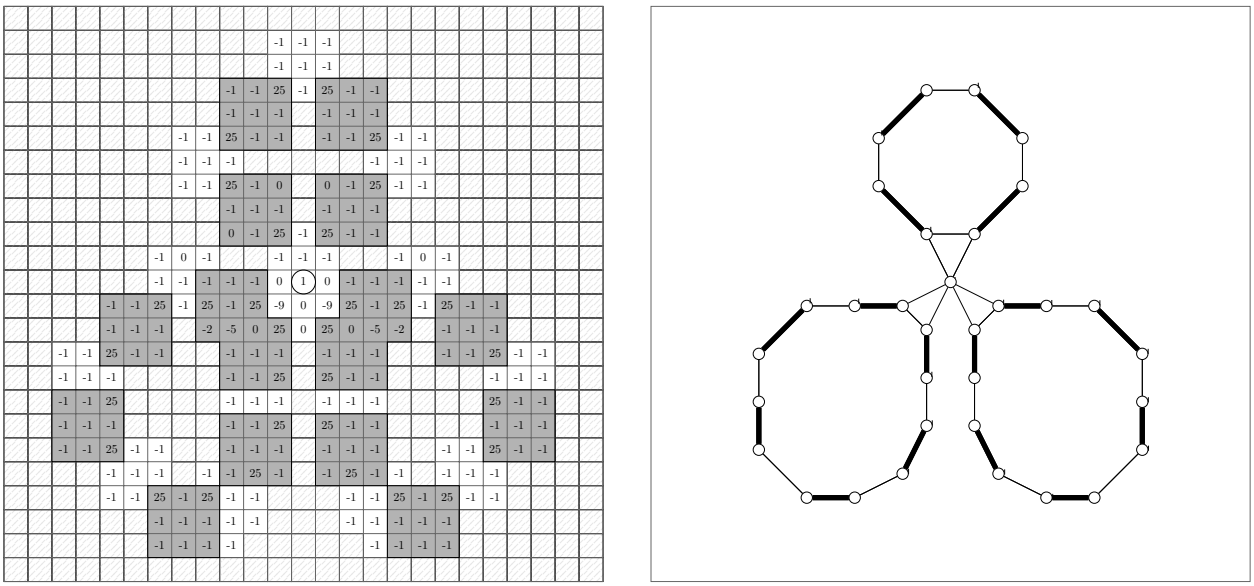


Figure A.6 An example clause point consisting of three small wires brought close together around the circled clause point. The shown solution is *suboptimal*, and does not capture the one unit of value available in the circled clause point.

An example clause point is shown in Figure A.6. The solution in Figure A.6 is not an optimal solution; only obtaining a total value of 687 in this case. There are seven optimal solutions which obtain a total value of 688, capturing the extra point of value available in the circled node. These solutions are shown in Figure A.7.

A commercial integer programming solver, in this instance Gurobi [140], was used verify that the example configuration has the necessary characteristics. Similar to the wire example in Section A.3.1 the wires associated with the clause point can be extended simply and used in the full construction that follows.

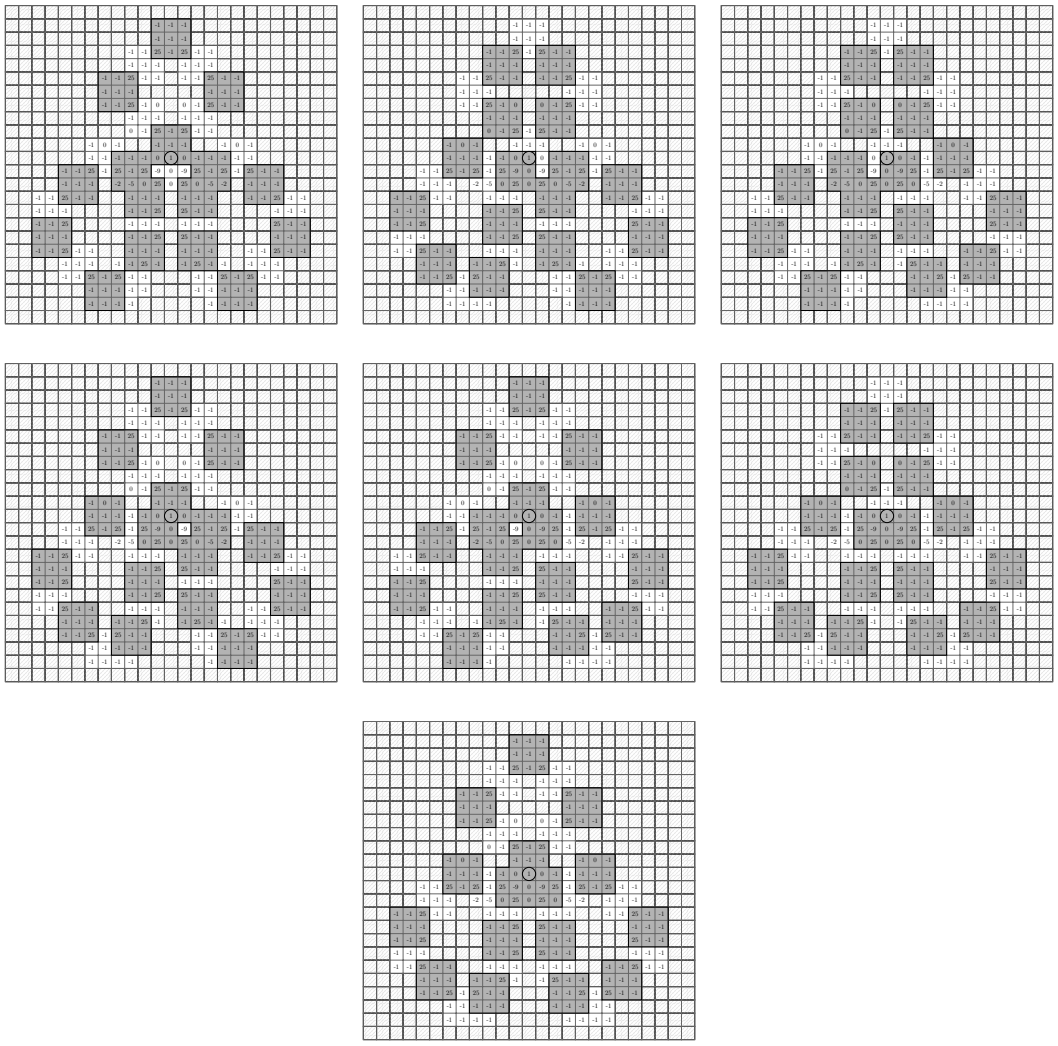


Figure A.7 The seven optimal solutions for the simple clause point example corresponding to the three cases where one variable is true (top row), three cases where exactly two variables are true (middle row), and when all variables are true (bottom).

A.3.5 Entire Construction

In the full construction multiple copies of the preceding components are combined together to fully transform the input 3-SAT problem into an ultimate pit problem with minimum mining width constraints. There are two sections in the full construction: the variable bus which traverses along the top of the region containing the wires for each variable, and the clause region which contains the M clause points. For each clause point the three relevant variables are diverted from the bus, crossed over any intervening variables, possibly negated, and then used to form a clause point.

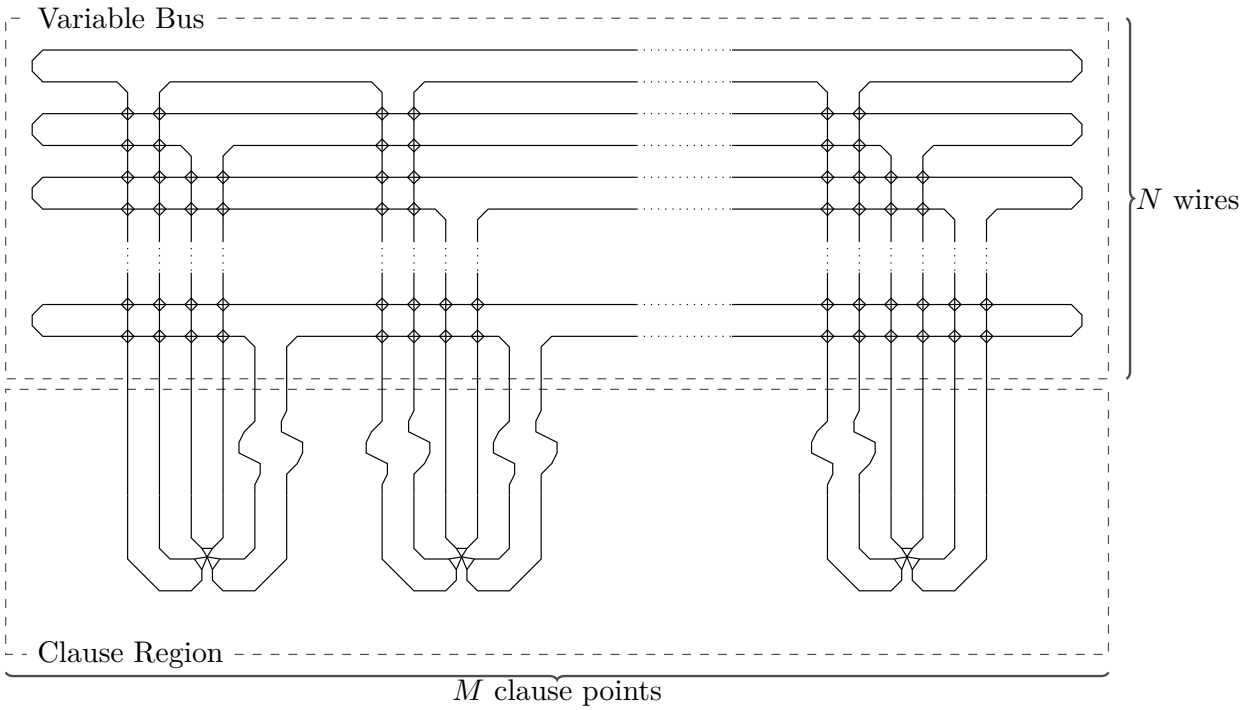


Figure A.8 An overview of the construction used to transform a 3-SAT problem into an ultimate pit problem with minimum mining width constraints.

In Figure A.8 the entire construction is shown in schematic form. The variable bus contains the N wires associated with the N input variables. The height of the variable bus region is linear with respect to the number of input variables as each wire only needs enough space to turn around on both ends and not interfere with the other wires.

The clause region then contains the M clause points associated with the M clauses, and is a constant height in order to accommodate the clause point itself and a possible wiggle to negate the incoming wires. The width of this region is linear with respect to the number of input clauses, and the entire area occupies a $\mathcal{O}(M) \times \mathcal{O}(N)$ region that is predominantly large negative values.

The precise arithmetic required to determine the lower bound V is tedious, but poses no major problems. Each component of the construction could be modified to fit into a regular 2D grid with block values adjusted to maintain a very low optimal value such that the clause points themselves are the only possible sources of value to the solver. In any case the entire construction can be constructed in $\mathcal{O}(M \times N)$ time, even though it is quite intricate. This completes the proof that the ultimate pit problem with minimum mining width constraints is $\mathcal{NP}$-Complete. ■

APPENDIX B

SELECTING EVENLY SPACED PUSHBACKS FOLLOWING PARAMETRIC ANALYSIS

Applying parametric analysis in open pit mine planning, with or without minimum mining width constraints,[3] suffers from a specific challenge called the 'gap problem'. Unfortunately not to be confused with one of the other homonymous gap problems in open-pit mining, this particular gap problem refers to the tendency for the contained tonnage of nested pits to vary in an unpredictable, nonlinear manner. Given an economic block model a series of nested pits can be calculated by incorporating a Lagrangian relaxation of a total tonnage constraint $\Sigma X_b \leq T$, as originally proposed by Lerchs and Grossmann, or a number of other methods described previously in Section 2.1.4. The nested pits are then commonly used to help with approximate scheduling and a wide range of other tasks. However, crucially, most tasks do not rely on using all of the nested pits and instead a smaller subset of nested pits is desired.

In general the desired subset of nested pits are used to approximate yearly production volumes often called pushbacks. The process of selecting pushbacks has historically been accomplished by hand by observing the pit-by-pit graph, or a spreadsheet, and manually choosing which pits to carry forward in downstream analysis. This appendix presents a novel, automated, dynamic programming approach to selecting pushbacks that minimizes the squared tonnage differences between subsequent pushbacks. This removes the tedious and error-prone manual process and permits for extended automated analysis especially in cases where uncertainty is to be explicitly understood.

In Figure B.1 and Table B.1 we see the nonlinear nature of a typical pit by pit graph and the output of the algorithm developed in this appendix. The example problem has 40 pits, along with the 'mine-nothing' case with zero tonnage. In this example we are selecting for five pushbacks, however this is an input parameter. The pit numbers 3, 11, 33, 39 and 40 are approximately evenly spaced by tonnage but not by pit number.

[3] If minimum mining width constraints are used then the pits will not necessarily be nested entirely within one another. This is not addressed by this procedure.

Table B.1 Table of pit numbers and tonnages following parametric analysis. Tonnage differences between both pits and selected pushbacks are included. Shaded rows indicate pits serving as pushbacks.

Pit Number	Tonnage	Tonnage Increase	Tonnage Increase By Pushback
0	0	0	-
1	72514	72514	-
2	72516	2	-
3	73622	1106	73622
4	74433	811	-
5	75278	845	-
6	76333	1055	-
7	76361	28	-
8	77661	1300	-
9	77753	92	-
10	78594	841	-
11	147011	68417	73389
12	148862	1851	-
13	149680	818	-
14	150488	808	-
15	151107	619	-
16	152791	1684	-
17	163523	10732	-
18	164470	947	-
19	168442	3972	-
20	170581	2139	-
21	171515	934	-
22	171569	54	-
23	173986	2417	-
24	175824	1838	-
25	177175	1351	-
26	184820	7645	-
27	187615	2795	-
28	189550	1935	-
29	191013	1463	-
30	191623	610	-
31	195092	3469	-
32	199770	4678	-
33	202118	2348	55017
34	208858	6740	-
35	211074	2216	-
36	212883	1809	-
37	215546	2663	-
38	245463	29917	-
39	259233	13770	57115
40	357304	98071	98071

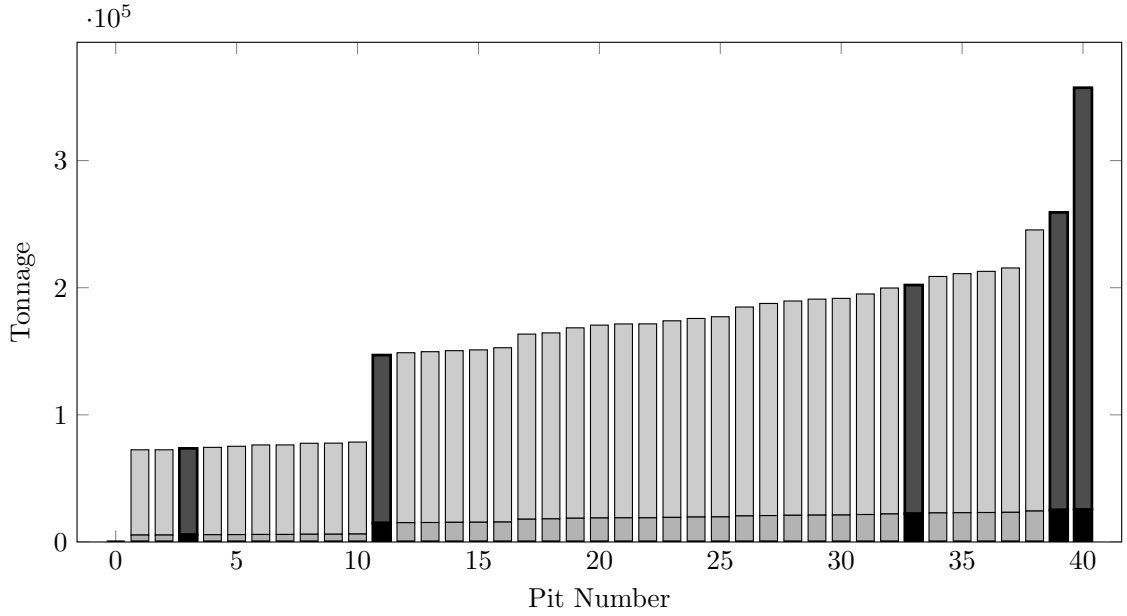


Figure B.1 An example pit by pit graph with the selected, evenly spaced, pushbacks indicated with darker bars.

Although the tonnage differences between pushbacks do vary quite widely, from as low as 55,017 to up to 98,071, there is no better subset of pits to serve as pushbacks. Any change in the selected pushbacks will increase the overall tonnage differences between pushbacks and leads to a worse result which is less appropriate for downstream processes.

Before discussing the algorithm in detail, it is important to understand where the problem even comes from. The 'gaps' in Figure B.1 between pushbacks 10 and 11, and pushbacks 39 and 40 are the main concern. These gaps form because large areas of the deposit suddenly become economic all at once when a critical economic threshold is reached.

The algorithm in this appendix does suffer from some drawbacks which preclude its use as an end all solution. Specifically the pushbacks selected by this process do not, in general, satisfy minimum pushback width constraints and the procedure does not consider any geometric or operational considerations. The algorithm also can not overcome situations where the gaps are so large, or the pit-by-pit graph is so nonlinear, that evenly spaced pits do not exist in the input set. However, the pushbacks selected by this approach are guaranteed to be as evenly spaced as possible with respect to tonnage and are a far cry better than manually selected pushbacks.

An early analogue of this algorithm first appeared in Maptek Vulcan Version 10 with the Automated Pit Designer [116]. Users of that tool would run conventional nested pit analysis using a push-relabel based pit optimizer resulting in several dozen nested pits. The tool would then select a handful of those pits that were approximately evenly spaced by tonnage to create high level initial designs. The pushback selection subroutine found extensive use as a practical way for planners to filter their nested pits and also to define the so called "selected case" schedule. However this early analogue was unable to guarantee an optimal solution, being developed by the author of this dissertation following an ad-hoc divide and conquer procedure that was relatively unsophisticated.

B.1 Algorithm Description

Formally, we are given the Tonnage, T_p, for each nested pit p in the set of nested pits $\mathcal{P}, |\mathcal{P}| = N$, and an integer n which is the requested number of pushbacks to select. We seek a subset of pit numbers $\hat{\mathcal{P}} \subset \mathcal{P}, |\hat{\mathcal{P}}| = n$ such that the cumulative squared tonnage differences between neighboring pushbacks is minimized. There is some flexibility with how this objective is defined – but the squared tonnage differences performs well in practice and is well suited to an optimal solution by dynamic programming.

We define the following additional pieces of notation:

- $i \in \{0, 1, ..., N-1\}$; the index of the last selected pushback.
- $j \in \{0, 1, ..., N-1\}$; the index of the current pit number in consideration.
- $r \in \{0, 1, ..., n\}$; the number of remaining pushbacks to select.
- $S(i, j, r)$; the objective value provided we make the optimal selection for the remaining r pushbacks from our current pit number j given our last selected pushback was i.

$S(i, j, r)$ is governed by the following recurrence relation and two base cases:

$$S(i,j,r) = \min\left((T_j - T_i)^2 + S(j, j+1, r-1),\ S(i, j+1, r)\right) \tag{B.1}$$
$$S(i,j,r) = (T_{N-1} - Ti)^2 \quad \text{when } r = 0 \tag{B.2}$$
$$S(i,j,r) = (T_j - Ti)^2 + S(j, j+1, r-1) \quad \text{when } r = (N-j-1) \tag{B.3}$$

Equation B.1 encodes the objective. That is, for a given configuration we can select this pit number as the next pushback increasing our objective and decreasing the number of remaining pushbacks r by one. Or we can choose to not select this pit number; the second term in the min statement. Equation B.2 applies when we have no more pushbacks to select and adds the difference between the ultimate pit and the penultimate pushback. Equation B.3 applies when we *must* select all of the remaining pit numbers to serve as pushbacks.

A recursive implementation of equations B.1 to B.3 would be correct, but very inefficient. The dynamic programming implementation and traceback step follows in Algorithm 6. This routine runs for $\text{Tet}(N-2) - \text{Tet}(N-2-n)$ steps where $\text{Tet}(x)$ is the x^{th} tetrahedral number: $\text{Tet}(x) = (x(x+1)(x+2)) \,/\, 6$. This is on the order of $\mathcal{O}(n^3)$ however the algorithm is practically very fast as N is typically less than a few hundred and n is typically less than twenty.

A minor modification of the algorithm allows the routine to output all optimal sets of pushbacks ranging from 3 to $n+2$. This modification is useful when the desired number of pushbacks is not known or will be used as a sensitivity parameter for downstream analysis.

B.2 Conclusions

Nested pit analysis remains a useful tool for long-range open pit mine planners early in the design process. Selecting a subset of those nested pits to serve as pushbacks for downstream scheduling and design has generally been a tedious and error-prone manual process. This novel algorithm removes this manual step from the planning process and solves a long standing inconvenience faced by mine planners.

Algorithm 6: Algorithm to select evenly spaced pushbacks

Input : The monotonically increasing pit number tonnages T of size N, and n the number of pushbacks to select less 2

Output: The set of evenly spaced pushbacks $\hat{\mathcal{P}}$ of size $n+2$

$K \leftarrow$ `Tet(`$N-2$`) - Tet(`$N-2-n$`)`;
Initialize Sv, Tb, Sp as arrays of size K;
$si \leftarrow 0$;

for $r \leftarrow 1$ **to** n **do**
  $q \leftarrow N-1-r$;
  $tsi \leftarrow si -$ `Tri(`$q+1$`)` **if** $r \neq 1$ **else** -1;
  for $j \leftarrow q$ **to** 1 **do**
    $tni \leftarrow si - j$ **if** $j \neq q$ **else** -1;
    for $i \leftarrow j-1$ **to** 0 **do**
      if $r = 1$ **then**
        $sv \leftarrow (T_j - T_i)^2 + (T_{N-1} - T_j)^2$;
      else
        $sv \leftarrow (T_j - T_i)^2 + Sv[tsi]$;
      if $j = q$ **then**
        $nv \leftarrow -1$;
      else
        $nv \leftarrow Sv[tni]$;
      if $nv = -1$ ***or*** $sv \leq nv$ **then**
        `// Select this pit number as a pushback`
        $Sv[si] \leftarrow sv,\ Tb[si] \leftarrow tsi,\ Sp[si] \leftarrow 1$;
      else
        $Sv[si] \leftarrow nv,\ Tb[si] \leftarrow tni,\ Sp[si] \leftarrow 0$;
      $si \leftarrow si + 1$;
      $tni \leftarrow tni + 1$;
    if $r \neq 1$ **then**
      $tsi \leftarrow tsi + j + 1$;

$\hat{\mathcal{P}} \leftarrow \{0\}$;
$ti \leftarrow K - 1$;
$pn \leftarrow 1$;
while $ti \neq -1$ **do**
  if $Sp[ti] = 1$ **then**
    $\hat{\mathcal{P}} \leftarrow \hat{\mathcal{P}} \cup \{pn\}$;
  $ti \leftarrow Tb[ti]$;
  $pn \leftarrow pn + 1$;
$\hat{\mathcal{P}} \leftarrow \hat{\mathcal{P}} \cup \{N-1\}$;
return $\hat{\mathcal{P}}$;

Procedure `Tet(`n`)`
  return $(n * (n+1) * (n+2))$ / 6;

Procedure `Tri(`n`)`
  return $(n * (n+1))$ / 2;

APPENDIX C

SPRINGER NATURE PERMISSION

Chapter 3 is heavily based on the paper "An Open-Source Program for Efficiently Computing Ultimate Pit Limits: MineFlow" by Matthew Deutsch, Kadri Dağdelen, and Thys Johnson which was published during the course of preparing this dissertation. Permission to reproduce this material was obtained from the Springer Nature and Copyright Clearance Center and the complete agreement is included as a supplementary file titled `appendixc_springer_license.pdf`.