Optimisation of Dragline Block Lengths

Iain Rudge

School of Mechanical Engineering

The University of Queensland

supervised by

Dr. Michael Kearney (UQ)

November 2, 2017

Dr. Ross McAree
Head of School
School of Mechanical Engineering
The University of Queensland
St Lucia, QLD 4072

Dear Dr McAree,

In accordance with the requirements of the degree of Bachelor of Engineering in the division of Mechatronics

I present the following thesis entitled "Optimisation of Dragline Block Lengths". This work was performed under the supervision of Dr Michael Kearney of the School of Mech Mining at UQ.

I declare that the work submitted in this thesis is my own, except as acknowledged in the text and footnotes, and has not been previously submitted for a degree at The University of Queensland or any other institution.

Yours sincerely,

Iain Rudge

Acknowledgements

Abstract

Contents

Acknowledgements **v**

Abstract **vi**

List of Figures **x**

List of Tables **xi**

1 Introduction **2**

1.1 Problem Definition 3

1.2 Motivation 3

1.3 Scope 5

1.4 Thesis Structure 5

2 Background **7**

2.1 Mining Background 7

2.1.1 Strip Mining 8

2.1.2 Draglines 8

2.1.3 Blocks 8

2.2 Operations Research Background 8

2.2.1 Linear Programming 8
2.2.2 Dynamic Programming 9
2.2.3 Metaheuristics 10

3 Literature Review **11**

4 Design and methodology **12**
4.1 Modelling Data 12
4.1.1 Types of Input 13
4.2 Cost of Overburden Movement 14
4.2.1 Scope 14
4.2.2 Assumptions 14
4.3 Cost of a Block 15
4.3.1 Assumptions 15
4.3.2 Justification 15
4.4 Cost of a Strip 16
4.4.1 Assumptions 16
4.4.2 Justification 16

5 Implementation **17**
5.1 Modelling 17
5.1.1 Spoil modelling 18
5.1.2 Action Cost 19
5.2 Optimisation at the Block Level 19
5.2.1 Mixed Integer Programming 20
5.2.2 Mixed Integer Programming with Validity 29

5.2.3 Dynammic Programming 33

5.3 Optimisation of the Strip 38

5.3.1 Resource Constrained Dynamic Programming 38

5.3.2 Resource Constrainted Dynamic Programming with Sequence Generation 41

6 Results **44**

6.1 Cost of Block 44

7 Conclusion **46**

Appendices **47**

A Code Listings **48**

List of Figures

5.1 Block Analysis for real world Data 28
5.2 Block Analysis for generated Data 29
5.3 Block cost comparison on an empty spoil 32
5.4 Block analysis on real world data 37

List of Tables

1 Variables used in the report 1

Table 1: Variables used in the report

Variable Name	**Variable Description**
Terra(x,y)	The function representing the height of the ground relative to a fixed point (x,y). It is assumed that this function will be a continuous function that is differentiable
Coal(x,y)	A density function representing the density of coal prevalent in a point (x,y) in the mine. This function is not assumed to be continuous or differentiable, however will most likely be represented as a piecewise function.
MaxBL	A constant, relating the maximum acceptable size of a block's length, this will be dependant on the draglines maximum reach and the movement patterns of the dragline.
MaxBW	A constant, relating the maximum acceptable size of a block's width, this will be dependant on the draglines maximum reach and the movement patterns of the dragline.
MinBL	A constant, relating the minimum acceptable size of a block's length, this will be dependant on the draglines maximum reach and the movement patterns of the dragline.
MinBW	A constant, relating the minimum acceptable size of a block's width, this will be dependant on the draglines maximum reach and the movement patterns of the dragline.
MineL	A constant that is the total length of the mine, it is a necessary variable as it will dictate one of the key constraints in the model.
MineW	A constant that is the total width of the mine, it is a necessary variable as it will dictate one of the key constraints in the model.
Spoil(n,m)	This function is used to represent the amount of spoil generated by block (n,m). It can be stated that the function will be reliant on both Coal(x,y) and Terra(x,y).
SpoilCap(n,m)	The amount of spoil that can be dumped into the exhausted block (n,m) this will be related in some manner to the amount of material removed from the block previously.
L(n,m)	The length of the nth block in the mth strip, this is one of the variables that our model aims to solve for. This is one of the outputs of the model
W(n,m)	The width of the nth block in the mth strip, this is one of the variables that our model aims to solve for. This is one of the

Chapter 1

Introduction

Draglines are massive machines used in open cut mining, often having boom-lengths of up to 100m these massive machines will move through a mine in a series of strips. Each strip in a mine can be broken down into fundamenatal blocks which have an internal movement pattern, depending on the machine the allowable sizes of these blocks will vary, as will the internal patterns. The purpose of this thesis is to develop a method of calculating block lengths for each strip in a mine such that if the time taken varies with the length of the block, then the minimal time spent per strip is found. However this method is not only useful for the calculation of an optimal path, the planning of blocks in a mine is an untapped topic of research, while products such as MineMax and polymathian exist for pit mines and underground mines, no software packages exist for the planning and optimisation of a strip mine with regards to overburden removal with draglines. (http://www.carlsonsw.com/solutions/mining-solutions/surface-mining/) While packages like Carlson mining solutions does deliver a strip miningm modeller it does not generate the block dimensions within a strip, nor how these should or could be planned.

As No methods exist in the space therefore the topic of strip generation in mining is valid for the exploration of throughout this thesis, while here only one strip is considered at a time the block dimension calculations are independant from strip to strip and as such will be able to be calculated as the mine is further explored.

1.1 Problem Definition

The aim of this thesis is the optimisation of dragline block lengths throughout a strip, where a block is the region of movement for a fixed movement pattern. While movement patterns that vary could be considered they are not within the scope of this project. The problem therefore can be defined as the calculation of feasible and optimal sequences of blocks within a strip. As no algorithmic methods exist for such a task the comparison and validation of results will be compared to examples from current pracetices in mining. Simply put the thesis is the minimization of time taken to remove overburden from a strip using a dragline.

1.2 Motivation

The optimisation of strip mining within Australia has been the focus of many academic groups over decades of research [**?**]. The reasoning for this is based firmly in economics. The majority of Australia's exports are derived from mining, with iron ores accounting for 20.2% of Australian exports in 2014[**?**] and coal accounting for an additional 11.6% in the same year[**?**]. These two industries alone accounted for $104 007 million of national exports, [**?**] which is a combined total of 32.8% of the nations exports at the time. Therefore the core motivation for this project is to increase the efficiency of a dragline, as doing such will be financially beneficial, it can be found that saving the industry 1% of time will save $35 million [**?**].

Draglines are massive machines used in strip mining to remove overburden, allowing target minerals to be exposed and extracted[**?**]. Automating and optimising these procedures will lead to massive reductions in cost and time [**?**]. The optimisation and automation of draglines has been a growing field. Initially the application of automation in draglines was seen as a challenging task, as the machinery must operate in harsh, complex three dimensional systems. [**?**]. In response to this challenge groups such as the 22 year old [**?**] group, Shared Autonomy for

Improving Mining Equipment Productivity, were formed to improve the autonomy and functionality of draglines.

The first attempts at automation were made in the 1980s [**?**], the approach taken was to record the actions taken by an operator and then replay these actions in a loop. However this project was unsuccessful as the operator bases its movements on an instantaneous state[**?**] rather than just a goal state. This attempt at innovation however was not in vain as other faculties began investigating methods of automation in dragline mining.

In 1993, driven by previous work a model derived from vision-based robotic control [**?**], was successfully implemented on a scaled model of a dragline. However, the methods of sensing and control initially provided difficulty in the application of this model to full scale machines [**?**]. During this it was found that in a single cycle a dragline, would spend 80% of the cycle swinging in free space. [**?**] This became the focus for many groups as it was seen as a potential area to reduce cycle time. The automation of a dragline and its components is a still developing field[**?**], however is not the main focus of this study. As the automation of draglines began to reduce the variance in operator it became apparent that other methods could be applied to optimise the motion and paths taken of the dragline.[**?**]. This research into path planning and modelling is a field that was and is still applied today. Topics such as motion planning, active control and path planning have all been applied to further improve performance in mining.[**?**] In the year 2002 a two week test on a Bucyrus-Erie 1350 system showed that the machine was able to match or exceed the performance of an operator, a reduction in swing time was made and the system ran consistently at all points. [**?**] This marked an important step in the improvement of efficiency in draglines as it meant that additional modelling and path planning, as well as optimisation regarding movements could be considered with little to no operator variance[**?**].

The motivation of this thesis is then in part based on the work of others in the automation of draglines, as it will allow former methods to be applied, and has allowed for more specific optimisations to be undertaken.In particular the optimisation of the movements of a dragline is a major task attempted to boost the

efficiency of a mining system. Current models use a variety of methods, however the one that is of interest to this thesis is the use of mixed integer linear programming[**?**] in order to optimise the motions of a dragline. However with this application, as with other applications the run time of the simulation does not scale well. Therefore the motivation of this thesis is to determine an efficient technique that is capable of calculating the optimal lengths of blocks in a mining strip. These lengths can then be used in the previously stated models to increase runtime efficiency by reducing the amount of calculations required.

In conclusion the motivation for this thesis is multi layered, at its core the main motivation is the financial benefit of optimisation in the utilisation of draglines, however many other researchers have been working on this task for decades, and as such any individual method is not feasible. Therefore this thesis seeks to help improve the runtime efficiency of current models by supplying a time efficient alternative in determining the optimal solutions for block size in conventional drag lines.

1.3 Scope

1.4 Thesis Structure

Introduction Introduces the motivation and goals of the thesis while also properly defining the problem , variables and notation used throughout the thesis.

Background A foccused review of neccisarry information and background knowledge that is referenced and used throughout the thesis in the development of the algorithm, as well as methods and information that was considered throughout.

Literature Review A review of prior art and techniques that can be compared to the methods outlined in this thesis.

Design and Methodology A systems level description of the design of the al-

gorithm, including the assumptions and considerations made for each design choice.

Implementation A detailed description of the implementation of each section, including discarded methods and results of the sections.

Results Results of testing the software against prior art as well as results generated for the final strip mine,.

Discussion A justification of assumptions and decisions made as well as a final comparrison to other methods mentioned

Conclusion A summary of the thesis with a focus on the future of the project.

Chapter 2

Background

2.1 Mining Background

Mining is a key part of this thesis, while not an in depth knowledge of mining is required, basic information is needed for problem definitions and clarity of comminication. The focus of the thesis is the removal of overburden from strip mines using draglines, as this is a specific topic and not a general discussion on optimisation of mining machines in general only infomration on draglines and the enviroment in which they operate should be considered.

2.1.1 Strip Mining

2.1.2 Draglines

2.1.3 Blocks

2.2 Operations Research Background

The field of operations research is a mathematical discipline specailising in the application of advanced analytical methods to make better decisions for specified problems. Often considered as a subsection of applied mathmatics operations research employs techniques from mathmatical modelling and optimization theory to calculate optimal or near optimal solutions to complex decision making problems. Naturally this branch of mathematics lends itself to the problem of optimal block dimensions in a strip mine, with the use of optimal decisions through either precise calculation or policy the best possible solution can be found using the field of operations research. The scope of this field is diverse, with many potential solutions to a problem, however the generation of the matematical model will aid in selecting a method that will obtain results quickest.
Mathematical modelling is a vital section of operations research, however the problem can be written and formulated in many different ways, as such the specific formulation styles of a technique will also be mentioned in that section, modelling in general can follow some simple rules yet no overall method or technique exists for the creation of a mathematical model, it is instead reliant on understanding of the system.

2.2.1 Linear Programming

Linear programming is a technique used to obtain the optimal outcome of a model such as the minimal time or highest profit. This method can only be aplied to a mathematical model with a linear objective function subject to linear equality and

linear inequalitiy constrants.
Often the model is described as groups of variables and contstraints with an overall objective function. This method of optimisation is often solved using complex optimisation engines such as gurobi rather than through algorithms implemented specifically for the implementation. The process of optimisation is done by defining a feasible region of valid solutions as a convex n^{th} dimensional polytope, where n is the dimensionality of the variables associated. Here this polytope is defined as an intersection of a finite amount of half spaces defined by linear inequalities. Within this polytope the optimal feasible solution will be found on a vertex satisfying all constraints and the objective function, this method can be calculated using matrix manipulation in algorithms such as the modified simplex algorithm.
The standard variables ina linear program are continous, however a subtechniqe of linear programming is integer programming which adds the constraint that some variables must be full positive integers. This problem is typically harder to solve as the optimal linear solution may not be similar to the optimal integer solution. An integer solution is often calculated by calculating an optimal linear solution and branching on rounding decisions for the integer variables to create a decision tree. This tree is branched such that the difference between the linear solution and chosen integer solution is minimised, this is the origin of the increased difficulty of integer programming in comparison to linear programming.

2.2.2 Dynamic Programming

Dynamic programming is a method of optimisation in which a subproblem is solved and the solution stored, this is done for all subproblems until the full problem can be constructed from, if a subproblem has already been solved and stored then it is called from memory instead of through calculation. The process of storing sub problems for later use is called memotization and is vital for use in dynamic programming as dynammic programming often encounters similar subproblems when working recursivly.
typically a dynamic program refers to recursivly breaking down decisoions into steps through a stage. The value of each decision in a stage is a combination of the

intial myopic value of the choice as well as the value of all other solutions that occur after this decision is made. This requires that some state in the problem is altered when a decision is made, then the affects of this decision must also be considered for all subsequent choices. This creates and expanding tree of subproblems and is why the method of memotisation is so vital to dynamic programming as often the same subproblem will be encountered multiple times.

2.2.3 Metaheuristics

A metahueristic is a higher level opimisation technique used to find, generate or select a hueristic that may provide a good solution to an optimisation problem, typically when working with incomplete information , unaccurate information or limited computational power. Unlike Linear programming and dynamic progamming a metahueristic does not garuntee a globally optimal solution , due to metahuerisitics typically relying on stochastic optimisation to generate solutions.
The genetic algorithm is such an example of a metahueristic where canddate solutions are evolved to find a better solution, this is done by calculating the value of all candidates. After this is complete the more preffereable solutions are combined with one another in such a way that a new series of candidate solutions is generated. This is done iteraitvly until the change between generations is suffciently small, at this point either a local optimal or global optimal solution can be found. Typically the genetic algorithm is used for determining a policy for good solutions, rather than linear programming or dynamic programming which will calculate the true optimal solution.

Chapter 3

Literature Review

Chapter 4

Design and methodology

4.1 Modelling Data

The modelling of data in this thesis is the vital first step to the optimization of a dragline, any simplifications made in this step will carry through all steps of the algorithm and effect the accuracy of the final results, therefore the manipulation of data to suit the system must be considered very carefully. As the dimensionality of the data decreases the runtime will as well, however the acheivable accuracy will decrease as the system will become more general. The method selected for the modelling of the geological data is a one dimensional density function used to represent the amount of overburden required to be removed from the mine along a specific strip. While this is not ideal in terms of a truly optimal solution it will allow the program to run rapidly, many assumptions along the way will result in loss of optimality at many stages, while an optimal solution is the desired result any feasible or near optimal solution is also acceptable.
Alternate methods that were considered for the modelling where a data driven approach or a two dimensional aproach, while a data driven aproach lends itself to machine learning of meta hueristics, these methods are often slow and not garunteed to lead to an optimal solution

One dimensional density functions are also useful as they can be generated through many methods, either through a pregenerated mine for testing or through geological survey data. In the thesis the sample data was generated by generating typical mine shapes and layouts of interest, such as a ramp and step function; within these samples a noise function was applied to give a better and more realistic set of data to test on. Survey data was supplied in a .xyz format, by assuming a constant depth of cut and a homogenius density of overburden the one dimensional density function was taken along the centre of the proposed strip path. This density data was represented as an Linked list for ease of use in python, an additional reason for this decisionis that the mine will always be mined in order and as such the list will simply be traveresed from start to end.

4.1.1 Types of Input

The intended use of the program is to rapidly calculate optimal, near optimal or feasible block dimensions fora single strip inside a mine, however one of the key features of the program is its relaticly rapid runtimes, allowing on the fly adjustments and the ability to make corrections and compensations for errors. This input however is harder to predict and could be considered outside the scope of the project. As the thesis focuses more on the algorithms and underlying princples of the software rather than the user interface and user experience it is considered that the input to the project is a 1 dimensional array representing the spoil and another representing the mine at the current point in time, this will allow for expansion if required for on the fly adjustments. The program can also have the specifics of the dragline changed so that any possible dragline can be considered, while this is simple to change there is no user interface for the program and as such if it were to see further use then a graphc user interface would need to be constructed.

4.2 Cost of Overburden Movement

The cost of a block is vital to the solution to the problem, however the cost of a block is calculated as a summation of many individual movements, therefore the proper consideration for these models should be undertaken, ideally the cost of overburden movement will be able to calculate the time of movement for the action of moving overburden from point n in the mine to s in the spoil strip. Therefore two inputs would be the only required information for this function.

4.2.1 Scope

The function to calculate the cost of an action is required to work for all $n \in N, s \in S$ if this movement is at all valid, therefore we can assume that only valid enteries will be passed into this solution. Therefore for any valid combination the function should return a cost of movement, while the cost funtion is not required to be linear it must be able to be used in a Linear programming engine such as Gurboi, and as such must be Linearly related to block usage for objective funtions. This is the most pressing constraint as otherwise linear programming will be infeasible.

4.2.2 Assumptions

The assumptions made in this section is that the fucntion can calculate a reasonable model of the cost of movement through a continous function of one input. This will allow for the cost of a block to be calculated based on swingtime, however the movement of the dragline is not taken into account as this would be consistent for any action pattern and therefore can be negated.

4.3 Cost of a Block

The cost of a block is vital in the calculation of the cost of the entire strip, each block will have a cost dependant on its location within the mine, its size and the state of the spoil prior to exscavating the block. The function must be able to be calculated quickly as it will be called many times.

4.3.1 Assumptions

The assumptions are the at the cost of movement through a block can be ignored, and the swingtime will be representative of the cost of the block, furthermore that the cost of a block is able to be calculated in a reasonably accurate in a one dimensional enviroment. Further it is assumed that the block is mined sequentially with no consideration to the specific movement pattern of the mine, all overbudern will be removed from the current section before the next section is considered and the reach of the dragline will be considrered from teh location of this block.

4.3.2 Justification

The modelling of a block in this simplistic method is valid as the most important attribute with the block model is that the results are analagous to other models, as long as the cost of a block vs length is similar to that of other models the strip calculation method is still applicable as it can be appliedwith any other block modelling method. However the model selected is a reasonable approximation of overburden removal for each block as is should represent the most basic of dragline movement patterns throughout the block. For more complex patterns this model must be updated, however typically more complex models will require a higher dimensionality of data and therefore this model would have to be reconsidered.

4.4 Cost of a Strip

The cost of a strip must be calculated in a way such that the cost of all total blocks is considered and minimised, the cost of blocks as well as a constant for starting a new block is the only cost associated with the cost of a strip. This assumption means that the cost of a strip is independant on all other actions that preceed it but that the spoil is updated as the strip is mined, furthermore the length of the strip must be defined before the mining has begun such that the desired strip length is already defined as the allowable length of the mine.

4.4.1 Assumptions

The strip must be removed entirely and the length of the strip cannot be increased as this may be unallowable and cannot be decreased as the savings in time are offset by the loss of potential profit, furthermore it is assumed that there is some additional cost in time for the starting of a new block,associated with the setup of machinery and relocation of equipment.

4.4.2 Justification

The suggested model is a valid application of ordering of blocks in a way to reduce the ost of actions independantly of the block model, this means that if the block model is changed the method of optimising the

Chapter 5

Implementation

5.1 Modelling

Modelling of a real world system has two motivators, the first is to gather better understanding of the system through mathematics and data analysis, the other is to allow operations such as optimisation and simulation onto the system. However there are challenges that can affect the accuracy of the model, as such careful deliberation should be taken to make good assumptions and simplifications. The assumptions and simplifications that are made will be explained and justified for each relevant algorithm, sub-model and constraint; all of which are needed to properly communicate the model in such a way that is independent of the optimisation techniques used.

The models defined in this section of the report will be used in all optimisation methods attempted, the accuracy of these models is vital to the accuracy of any formulation dependant on them and as such proper considerations and justifications must be made. The models that are discussed here are inherit to the system, they are often constants or functions within larger problem formulations, as such they are simple to change but yet key to the accuracy of the thesis.

The models discussed here are broken into four major sections, proper modelling of spoil distribution, modelling of movement patterns within a block, modelling of

data unique to a dragline and finally the modelling of an action. Each of these categories aligns with a specific section of a formulation, and as such can be developed independently of each other.

5.1.1 Spoil modelling

The Spoil or overburden is the strictest constraint on the feasblitly of a strip, as such the modelling of spoil distribution is vital to the accuracy of a model. The spoil channel can be discretised into sections with a given maximum capacity per section, following this the swell rate of the spoil and expansion factor due to inefficient spoil placement should also be considered. These three models are the required knowledge to properly consider and implement spoil capacity constraints on a formulation, each model can be seen below.

5.1.1.1 Spoil Capacity

The capacity of a section of spoil is a vital constraint in the formulation of a block model, the capacity of the spoil is therefore modelled as the maximum amount of spoil that could stability be held at that section without collapse of the spoil channel, the stable angle for loose overburden to be stacked is 36 degree, with the spoil channel 20m lower than that of the mine therefore by using an approximate model of a cone to represent each spoil section, then the spoil capacity is $0.33 \times W^2/2 \times H$ is the maximum spoil capacity.

5.1.1.2 Swell Factor

Swell factor is the amount attributed to the swelling of overburden as it decompresses from its compacted pre mined state, this is found in textbooks and for a typical soil type of compact dirt can be modelled as an increase in volume of 30%. The modelling taken from prior resources such as textbooks is sufficient for the purposes of this thesis.

5.1.2 Action Cost

The cost of an action is here defined as the time taken to move overburden from a section in the mine to a section on the spoil, this should be based either on experimental data or through a full model of the system. However modelling the entire system proves difficult in one dimension, as a result methods were taken from a literature review. It was found that after a review of several papers the most intuitive solution was also the most accurate, that is the time taken to move the dragline is a linear relationship with the angle that the dragline is moving.
The calculation of the angle is simple, the distance from the draglines current point to the point in the spoil can be found using basic triginomitry and as such the cost of an action is defined below.

5.2 Optimisation at the Block Level

In order to model a sequence of blocks it is vital to first develop an accurate model describing the cost and spoil of a block, several methods were considered, each of which was an optimisation problem seeking to minimise the time taken to process the block. The calculation and modelling of a block must be rapid, as this modelling function will be used thousands of times in the calculation of a strip, while storing block costs into memory was originally considered the feasibility and cost of a block varies with three inputs, its start and end locations, and the current state of the spoil. Therefore the amount of unique blocks that can be considered is huge and storing results in memory loses efficiency
A variety of methods were formulated and the results were compared against one another for both run time and accuracy, in order to verify the accuracy of a model its output is compared to techniques that are outlined in sections **??**, **??** and **??**. This comparison was done on both real world data and standard data sets that were generated for the purpose of testing. The comparison will focus on the run times of each solution as well as the accuracy of the method proposed.

5.2.1 Mixed Integer Programming

Mixed Integer Programming is a somewhat obvious choice for this model, as this technique is already applied in a 3 dimensional environment as outlined in section**??**, however some changes must be made for this model to be used with the modified density data, as a result while the concept of a MIP was still considered, the problem itself had to be reformulated. Formulation for the model is found in the following section, while developing a MIP it is vital that all constraints and objectives are linearly related to data and variables, otherwise the solver algorithms will not work. The process of formulating an appropriate model for the block cost is based on the physical movements of a dragline and the constraints that will affect it, while some elements of the formulation were adjusted or ignored, the end formulation is the most accurate model with the data available.

5.2.1.1 Model Formulation

Sets

$$S \quad \text{Set of all Spoil Sections } s \in S \tag{5.1}$$

$$N \quad \text{Set of all Mine Sections } m \in M \tag{5.2}$$

Here the sets are used to represent the discretised mine and spoil sections, while these sets have no effect on the system they are used for communication of data and variables. The sizes of sets S and M must be the same as the spoil channel is the results of a prior strip in a mine. The discretisation of the Mine into sections was chosen as the move from a continues model to a discrete model made the modelling of the problem much easier. While some information is lost in this process it is negligible as survey data itself is discrete. The size of the sections themselves are dependant on both survey data and desired accuracy, for the testing of this thesis each section was a metre in length, however if more precision is required then the resolution could be increased. However the increase in detail and resolution will result in slower calculations of block dimensions as more potential solutions are

presented. One metre was selected because it was the resolution of the real world data supplied however as seen in the comparison of results a lower resolution will dramatically speed up the run time of the problem, this could be used in first pass solutions to get an approximation of good results.

Data

$$SF \quad \text{Swell Factor of Spoil} \tag{5.3}$$

$$SEf \quad \text{Packing Efficiency of Overburden} \tag{5.4}$$

$$SpoilCap \quad \text{Maximum capacity of spoil} \tag{5.5}$$

$$n_j \quad \text{Density of overburden at section } j \quad \forall m_j \in M \tag{5.6}$$

$$s_j \quad \text{Density of overburden at section } j \quad \forall s_j \in S \tag{5.7}$$

$$C_{jk} \quad \text{Cost of movement from } m_j \text{ to } s_k \tag{5.8}$$

$$Dvol \quad \text{Volume of overburden moved in one action} \tag{5.9}$$

The data relevant to the formulation of the MIP is modelled and calculated in section 4.1, in this section models and justifications can be made for each design choice that is made in the accuracy of the data however in this section the way that the data is used is much more important. The data used is relatively simple, often represented as either a constant or an array of constants. The use of data 5.3 to represent the swell of spoil is vital to the proper modelling of spoil distribution and will be used whenever interacting with the spoil output of the system, this data is simply a constant value, as suggested in the prior section. Similarly data from 5.4 is used as a n approximate model for the expansion of the spoil due to inefficiencies in stacking methods, this constant can vary depending on the model used however is simply represented in the formulation of this problem and is not a key data point for initial testing, rather it is important for the accuracy of the spoil channel distribution, which itself is also dependant on data 5.5 or the maximum capacity of the spoil at any given section s, here we assumes that this capacity is constant throughout the spoil channel, however this could be easily adapted if needed for varying capacities. This capacity is used in constraint 5.14 as the right hand side of the equation. Sensitivity analysis could be done on this constraint to estimate the

potential benefits of increasing the capacity of spoil available, this would have the same effect as increasing the efficiency of the spoil stacking methods, as described by a lower value of data5.4.

The amount of material moved per action can be found in data 5.9, while a very simplistic piece of information it is still necessary to include as it represents the amount of overburden moved per action through the size of the draglines bucket. Sensitivity analysis could also be done to relate how the changing size of this constant affects the mining time of a block, increasing this constraint should reduce the cost of the block as it will reduce the amount of actions required to remove the overburden from a mine section. This cost is found in data 5.8however this data is an $m \times s$ array that contains the movement cost from each point $m \in M$ to each spoil section $s \in S$, the movement cost is used in the objective function to minimize the total swing time of the dragline inside a block. This cost data can be calculated as a function $\arctan(\frac{abs(n-s)}{Reach})$ however this may also be pre calculated and put into an array to reduce function calls, the latter method was selected for the implementation of this data.

The representation of the mine5.6 is an $1 \times m$ array where the value at each point is the average amount of overburden to be removed at that point, this value will not change as we will never come back to or refer to a section once the overburden has been removed. Unlike this is data 5.7 which represents the amount of overburden currently in the spoil. Once a block is calculated this spoil value is updated so that future blocks use the spoil output of the previous block to have a consistent and proper model for the spoil distribution throughout the strip. Therefore while this data will not change during the calculations performed on the block, once the solution is calculated this data is updated to reflect on the resultant spoil.

Variables

$$X_{jk} \quad \text{Overburden moved from } m_j \text{ to } s_k \tag{5.10}$$

$$Y_{jk} \quad \text{Actions moving from } m_j \text{ to } s_k \tag{5.11}$$

The variables for this model are both related to the movement of overburden from a mount m in the mine to a point s in the spoil. Variable 5.10 refers to the capacity of dragline volume that is moved from m to s. This is necessary for the calculation of spoil capacity constraints as well as the calculation of removal of overburden from the mine at point m. However the assumption is made that movement actions taken at any capacity will still incur the same costs. It is for this reason that variable 5.11 is used, representing the ceiling of actions taken from point m to point s for all points in the mine. This variable is exclusivly used in the objective function, participating only in constraints that allow it to be linked to 5.10.

Objective

$$\min \left(\sum_{m \in M} \sum_{s \in S} (C_{n,s} \times Y_{n,s}) \right) \tag{5.12}$$

The objective function of the Mixed integer linear program is simple, the minimisation of time taken to complete all actions. As some actions will cost more than others it becomes obvious that the cheapest possible movement will be selected when available, it should be noted that while cost5.8 is not linear, its relationship with movement actions 5.11 is, allowing linear programming to be used. The minimization of this is obvious, as smaller movement actions will take less time than a larger swing action, they will be prefered where feasible.

Constraints

$$\sum_{s} X_{m,s} \times Dvol = M_m \quad \forall m \in M \tag{5.13}$$

$$\sum_{m \in M} X_{m,s} \times SEf \times SF + S_s \leq SpoilCap \quad \forall s \in S \tag{5.14}$$

$$X_{m,s} \leq Y_{\mu,s} \quad \forall m, s \in M, S \tag{5.15}$$

$$X_{m,s} \geq 0 \qquad \forall m \in M, s \in S \tag{5.16}$$

$$Y_{m,s} \in \mathbb{N} \qquad \forall m \in M, s \in S \tag{5.17}$$

The constraints on the model could be considered as the most interesting part of the model, with no constraints the model would simply choose to perform no actions, as this is the minimal allowable solution. Therefore the minimum amount of work done by the dragline must be constrained, the strict constraint found in equation 5.13 states that all overburden must be removed from a section m for all sections in the mine. The purpose of this constraint is to set the exact amount of work done by the dragline, it must remove all overburden from the block, but cannot remove any additional material from the block, while the latter is less likely as this would increase the cost of operation it is still best to have as strict a constraint as possible.

The constraint described in equation 5.14 constrains the solution the most of any constraint in not only the Block model but the entire thesis, it is the feasibility constraint that the spoil capacity cannot be exceeded, it should be noted that this constraint also takes into account the state of the spoil prior to this block, allowing past blocks actions to carry over. This constraint will invalidate many solutions and cause the cost of action to go up if all local spoil sections are at or close to capacity, this constraint is what drives up the cost in the calculation of a block.

Constraint 5.15 is the linking constraint between X5.10 and Y5.11, it ensures that Y is the ceiling of X. As the objective is a minimisation problem, then Y will seek to be the lowest allowable integer value greater than or equal to X, this linking constraint ties constraints 5.13 and 5.14 which are both dependant on X to the objective function 5.12 which is dependant on Y. Similarly constraints 5.16 simply state that X must be a positive real number and Y must be an integer greater

than or equal to zero, these constraints are simply to properly set up the variables for the model.

5.2.1.2 Implementation

[Put Code Here]

```
1    m = G.Model()
2    X = {(n,s):m.addVar() for n in self.N for s in self.S}
3    Y = {(n,s):m.addVar(vtype = G.GRB.INTEGER) for n in self.N for s
     in self.S}
4    m.setObjective(G.quicksum((self.MoveCost(n,s)*Y[n,s]) for n in
     self.N for s in self.S),G.GRB.MINIMIZE)
5
6     CreateCieling = {(n,s):X[n,s]<=Y[n,s] for n in self.N for s in
     self.S}
7
8     RemoveOnlyallowed = {
9     (n):m.addConstr(G.quicksum(X[n,s]*self.Dragline.get_BucketVol()
     for s in self.S)==self.Mine[n]) for n in self.N
10    }
11
12    spoilcapacity = {
13    (s):m.addConstr(G.quicksum(X[n,s]*self.Dragline.get_BucketVol()*
     self.swell*self.expand for n in self.N)+self.Spoil[s]<=self.
     spoilcap)
14    for s in self.S
15    }
16
17    m.optimize()
```

The implementation of the model is done in python, with use of the gurobi optimisation engine linear programming becomes a very simple method. Lines 2 and 3 are the deceleration of variables relating to the variables found at equations 5.10, 5.11 corresponding to them exactly. From here the objective is declared and set to minimise , matching the objective 5.12 stated in the formulation section. The constraints are also similar with each constraint 5.15,5.14,5.13 all represented in

code. For additional information on supporting code please refer to Appendix A. Once the formulation is complete gurobi will perform preprocessing on a matrix representation of the model, before using the modified simplex algorithm to solve the linear problem, while this algorithm could be written specifically for this thesis it was seen as unnecessary as the gurobi optimisation engine is much more efficient at this task.

Once this optimisation algorithm has been completed the spoil is updated and the cost is returned, this is done by simply updating the relevant sections of the array and can be seen completed in the below code snippet.

```
self.cost = m.ObjVal
self.potential_spoil = self.Spoil[:self.step]+[sum(M2S[n,s].x*\
self.Dragline.get_BucketVol()*self.swell*self.expand for n in self.N)
   +self.Spoil[s]\
 for s in self.S]+self.Spoil[self.etemp:]
```

This entire process is then wrapped into one function so that the cost of a block can be calculated though one function call, vital to the application of this model in the optimisation of the entire strip.

```
  def BlockCost(self,start,end,Spoil,output=0):
    if end>len(self.Mine):
      end = len(self.Mine)
      # print("Adjusting End Position")
    self.set_Spoil(Spoil)
    if output ==1:
      print("Spoil Set")
    self.set_Block(start, end)
    if output ==1:
      print("Block Set:\t" ,[start,end])
    self.Calc_Valid()
    if output ==1:
      print("Column Generation Done")
    stime = time.time()
    try:
      self.MIP(output)
      etime = time.time()-stime

    except:
```

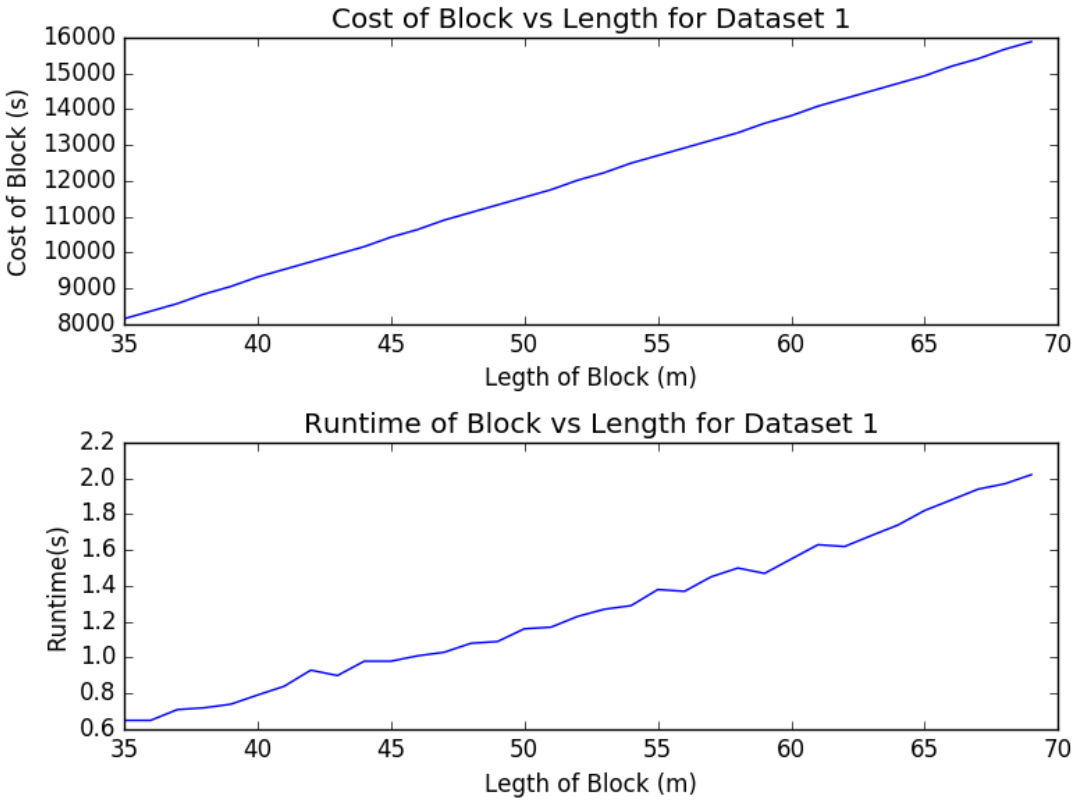
```
20        etime = time.time()-stime
21        print('\t MIP Infeasible')
22
23        return(np.inf,Spoil,etime)
24      if output ==1:
25        print("MIP Complete")
26      return (self.cost,self.potential_spoil,etime)
27
```

The only unique or interesting part of this function is that if the block is not feasible then the function will return a cost of infinity, this will mean that the block will not be used in the optimal or feasible soluion of a sequence of blocks in a strip. Each call of the function take in the start, end and state of the spoil, allowing the block to be calculated to return the cost and updated spoil, as well as the time taken to calculate the MIP. Both cost and resulting spoil are vital to the calculation of the strip, and are the two desired outputs of any model describing the actions taken within a block.

5.2.1.3 Results

A set of 10 generated cases were selected for comparing my algoriths and models against each other, the first five cases are on an open spoil and increase in complexity, the final five cases have spoil channels of increasing complexity. Where possible the results of potential blocks captured in the same locations of each mine are considered. In this section of results, prior to comparison to other such methods the type of solution recieved will be inspected. It should be noted that Dataset 1 is the real world data supplied and all other datatypes are generated for the purposes of additional testing. The above figure **??** shows that the block is peicewise linear when calculated on an open spoil, that is the blocks cost can be represented as a series of linear functions which may appropriatly approximate the block, this interestingly could be used as a potential alternative to the solution of a MIP problem. However the block will consistently increase in cost as the length increases, similarly as the size of the block increases as does the time taken to calculate and simulate its values. As can be seen in figure **??** the dragline behav-

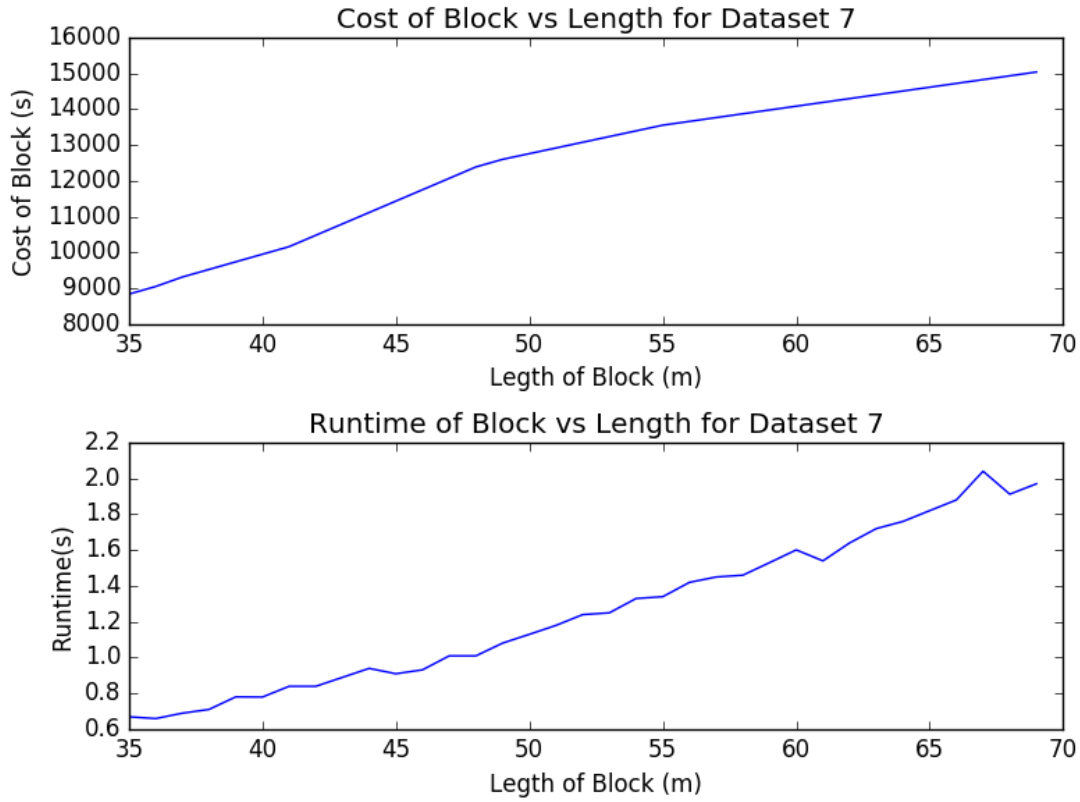
Figure 5.1: Block Analysis for real world Data



ior is similar although the cost will plateau at some points given the additional constraints of the spoil. This result suggests that while the cost of a block is linear under no or weak constraints any linearisation or simplification would not hold for blocks of more constrained or complex spoil channels. Therefore the use of a mixed integer linear program is still the best approach as it is capable of calculating the cost of a block rapidly and ensuring an accurate and optimal solution to the cost of a block.

The steroetypical increase in the cost of a block can be dependant on both the length of the block and the state of the spoil, however in this formulation the reach of the dragline is not properly considered. While this is acceptable for data sets with no or weak spoil constraints it will not function properly formore constrained blocks as the current model will incur a large cost to generate an infeasible solution

Figure 5.2: Block Analysis for generated Data



where the suggested spoil section used is outside the reach of the dragline.

5.2.2 Mixed Integer Programming with Validity

This model is an updated version of the Mixed Integer model with only one added array of data and one additional constraint, however this single constraint makes a huge differences to the complexity and accuracy of the model. The constraint that a dragline cannot move to or reach every point in a block or in the spoil will constraint the areas where overburden can be distributed in the spoil section. This constraint was not initially obvious as typically the dragline will place spoil in local areas rather than in sections that would be out of its reach. However as the availability of spoil room is reduced the cost associated with this action is

seen as preferable rather than an infeasible solution. This was only found after an implementation of the strip optimisation was complete, and as such can be seen as an updated model.

5.2.2.1 Model Formulation

Additional Data

$$Z_{jk} = \begin{cases} 1 & \text{if movement from } n \text{ to } s \text{ is valid} \\ 0 & \text{Otherwise} \end{cases} \qquad \forall n \in N s \in S \tag{5.18}$$

This data is an $m \times s$ array of binary values, such that if the dragline is able to feasibly reach the section of spoil when mining the section m of overburden then the arrays value is 1 and 0 otherwise. This array is currently based on the feasible maximum and minimum reach of the dragline, however other factors such as the pit edge while starting a new block, or having to move over sections in the spoil may cause this array to become more complex. Updating and improving this data should be of high importance for an accurate model, however as it is not integral to how the algorithm works, some simplifications can be made.

Additional Constraints

$$Y_{n,s} \leq \infty \times Z_{n,s} \qquad \forall n \in N, s \in S \tag{5.19}$$

This constraint is a big M constraint, that is it will limit the value of Y unless the value of Z is one for that combination of m and s. This constraint is simply implemented and constrains the feasible solutions of the model, it should be noted that this is setting most of the potential values of Y and X before any optimisation is considered, doing this has a dramatic effect on the reduction of runtime. As less options are available, less calculations must be considered and as such the program can run faster.

5.2.2.2 Implementation

The code for this section is the exact same as previously, however one additional constraint is added, as can be seen below it is very simple to add additional constraints into the gurobi model.

```
1
2 OnlyifValid = {
3     (n,s):m.addConstr(Y[n,s]<= np.inf*self.isValid[n-self.start,s-
    self.start-self.Dragline.get_reach()]) for n in self.N for s in
    self.S
4     }
```

However more interestingly is the calculation of valid blocks, as mentioned previously, the simplistic model for a valid block is if it is within the reach of the dragline, while this is not an accurate model it is sufficient for the demonstration of the block planning algorithm and with more time could be developed to be more fitting for application.

```
1 def Calc_Valid(self):
2   self.isValid = np.zeros((self.end-self.start+1,self.end-self.start
     +2*self.Dragline.get_reach()+1))
3   for i in range(self.end-self.start):
4     for j in range(i-self.Dragline.get_reach(),i+self.Dragline.
    get_reach()):
5       self.isValid[i,j+self.Dragline.get_reach()] = 1
```

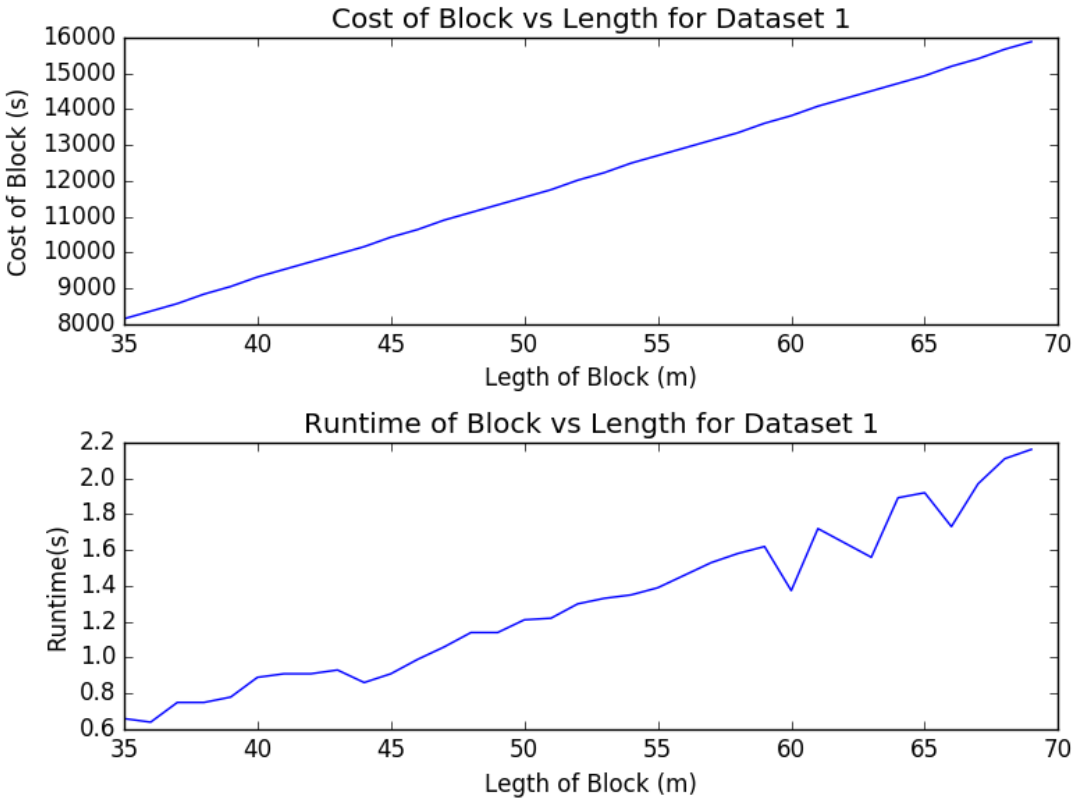
Here the array size is set up, we only consider areas within the block and spoil and as such this will reduce the size of the array and allow for more rapid calculation of results. Once this is done for every possible combination of blocks in this window a check is done to ensure that it is a valid movement.

5.2.2.3 Results

The results of the real world data should be the same as that of the prior formulation, this is because the additional constraint will only typically be applied when

the spoil channel is crowded and potential solutions are restricted. When not in this situation the spoil will be placed in the lowest cost location possible, which is typically within the reach of the dragline. Despite the exact same cost for the block throughout this dataset the time taken per calculation is increased, this is due to both the additional constraint and the need to calculate the set of valid positions for each location. This increased runtime suggests that a combination of both techniqes could be valid, using the faster method on open spoil and the slower onamore constrianed system. The results suggested in figure **??** suggest different

Figure 5.3: Block cost comparison on an empty spoil



solutions to those provided in figure **??**, this is due to the proper consideration of spoil location and capacity as well as the added constraint on reachability causes the cost of the block to be more expensive. This additionally will have a differeing spoil to that of the prior model and the individual actionsmay be drastically different. This model is more robust as it works in a constrained or underconstrained

enviroment. However other formulation methods should be considered as the run time of a block increases with both the complexity and the length of the block.

5.2.3 Dynammic Programming

Dynamic programming is an obvious approach to the problem, as the use of memeotisation will deecrease the amountof calculations required to find the opitmal solution, unlike a MIP the dynamic program will recusrivly consider all potential solutions which while thorough could be suffciently fast for the purposes of block cost optimisation.

5.2.3.1 Model Formulation

States

$$S \quad \text{State of spoil at current stage} \tag{5.20}$$

The state of a formulation is the current resources of remaining in a resource constrained dynamic program. Here this is the spoil channel, which will change with every action and therefore is updated at every stage, this state alongside the current stage should be stored in the memory so that the process of memotisation can occur. This technique uses the principle of optimality to reduce the amount of calculations required, that is if we have already calculated the optimal path for the block from this stage with the current states then the solution is already known and that branch of exploration can be completed. However the only state in this formulation is the amount of spoil in each section at the current stage, the use of this state is vital as it is used for the constraint on the actions available at each point. The spoil is a state as it will change with each action, nothing else will change depending on the action taken and therefore all other dynamic variables are simply stages.

Stages

$$H \quad \text{Overburden remaining at current block} \tag{5.21}$$

$$M \quad \text{current mine section } m \tag{5.22}$$

$$D \quad \text{Position Of Dragline} \tag{5.23}$$

The stages of the model are something that we do not have control over, each action moves us from one stage to the next. This formulations two stages are interlinked, however it could also be seen that the position of the dragline is dependant on the current mine section. Therefore the model has effectively two stages, the overburden left at the current section and the current section within the mine. Once all overburden is removed from the stage H the position M is updated. If the position is sufficiently updated then the draglines position also will update. For any given block these stages will occur in the same order no matter what. This is the differentiation between states and stages, the dragline will always move through the block and remove all overburden however the actions taken to remove the overburden will not always be the same. Here the states are selected as the minimum required representation of the block in order to properly model its behaviours and movement process.

Transition Function

$$S_{stage+1}(s) = S_{stage+1}(s) \times SEf \text{where } s \text{ is the spoil section in action} \tag{5.24}$$

The transition value allows the next state to be calculated based on the current state and the action chosen. Here the action chosen is the moment of overburden from the current block to some point on the spoil s, this action will succeed in adding additional overburden to the relevant section on the spoil, updating this is simple and from here the only additional piece of information required is the constraint that this transition function cannot create a state that exceeds the spoil capacity, this is done by simply checking the suggested next state and adding a large cost if this constraint is violated.

Value Function

$$C_A = \arctan(\frac{abs(D - A)}{Reach}) \tag{5.25}$$

$$V_t(S_t) = \min(C_A + V_t(S_{t+A})) \tag{5.26}$$

The value function calculates the cost associated with a given action based on the cost of the action and the resulting states for all future stages, the cost of an action is simply the swing time from the current block to the spoil selected as an action. This cost is to be minimised however will also affect the state of the next action, as such this will recursively consider all potential courses of action for each action taken. Therefore the more actions that can be made the more computation time is required to calculate the cost of a block.

5.2.3.2 Implementation

The algorithm mentioned is performed recursively, at each iteration the position of the dragline and current block are updated, as they are stages they are very simple to update. Then the memory is checked to see if a solution already exists for this point. The process taken is simply outlined by the formulation above, but with minor changes to input saturation to take into account any limits on the system.

```
1  _Bucket ={}
2 M = MineGen(100)
3 S = [0 for i in range(130)]
4
5 def Bucket(Cur,End,SP,M,Scoops,H,D-):
6     global _Bucket
7     SPtup = str(SP)
8     SPmax = 150
9     Reach = range(D-20,D+20)
10    if D < 20:
11        Reach=range(0,D+20)
12    if D-Cur<=5:
13        D = D+5
14    if H <=0:
15        Cur = Cur+1
```

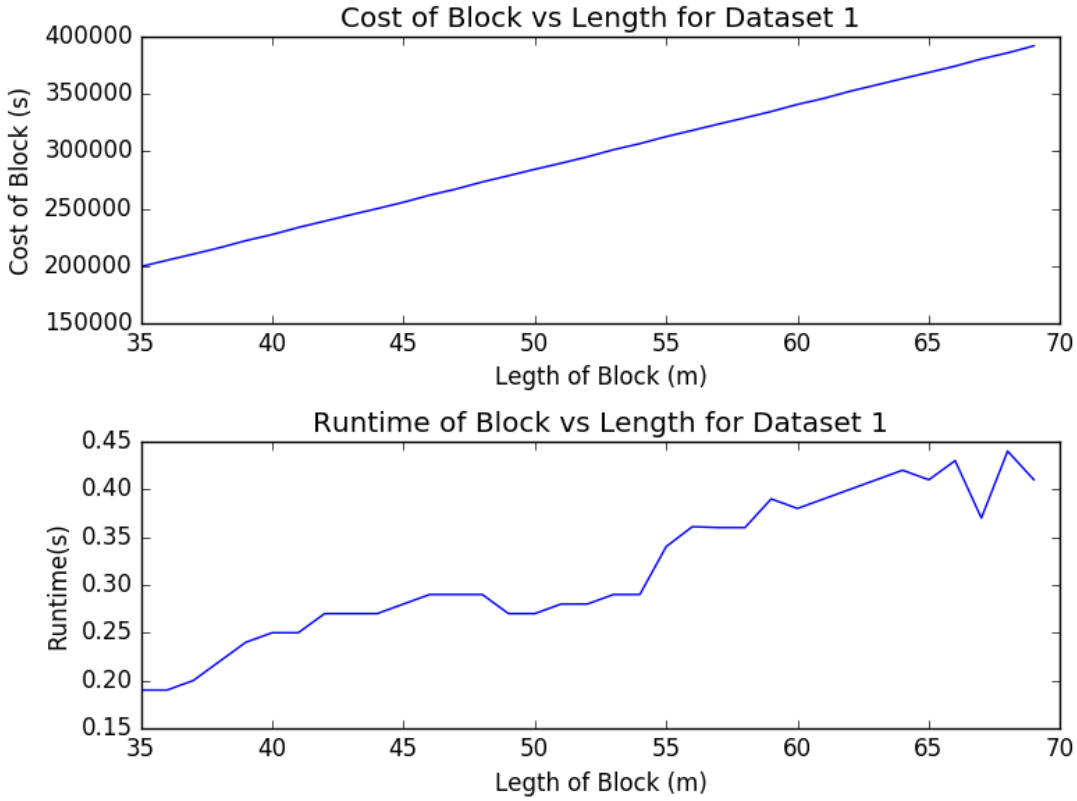
```
16          Scoops = 0
17          H = M[Cur]
18      if Cur == End:
19          return (0,0,0,0,[],0)
20      if (Cur,End,Scoops,SPtup) not in _Bucket:
21          _Bucket[Cur,End,Scoops,SPtup] = min((costspoilNL(Cur,A,H,D)+
22              Bucket(Cur,End,SP[:A]+[SP[A]+28]+SP[A+1:],M,Scoops+1,H
    -10,D)[0],Cur\
23                      ,A,Scoops,SP[:A]+[SP[A]+28]+SP[A+1:],D) for A in
    Reach if\
24      SP[A]+28 <= SPmax)
25      return _Bucket[Cur,End,Scoops,SPtup]
26
27  def GetData(Start,End,Mine,Spoil):
28      try:
29          A = Bucket(Start,End,Spoil,Mine,0,Mine[0],Start+20)
30          Cost = A[0]
31          print("New Spoil Aquired")
32          Flag = True
33          NewSpoil = _Bucket[max(_Bucket.keys())][4]
34          Position = A[5]
35          print("Cost Of Block:",A[0])
36      except:
37          print("Failed")
38          Flag = False
39          Cost = 100000000000000
40          NewSpoil = Spoil
41          Position = 0
42      return(Cost,NewSpoil,Flag,Position)
```

However in the memeotisation it is vital to include the state of the spoil, unfortunately this is likely to be unique and as such will not occur very often, this reduces the amount of memory calls made and the runtime of the program will increase, however if the state is not in the memory then the optimal solution of the problem has no guarantee to be returned, and instead a feasible solution will be supplied.

5.2.3.3 Results

The cost of a block for real world data does not match the results suggested by the prior two methods, this is due to the incorrect use of memotisation will mean that optimal solutions cannot be found and the solutions supplied are feasible at best, however it should be noted that this formulation will generate the cost of a block drastiacally quicker than the two prior models, however the assumption can be made that any small saving in code time is negated by the huge loss in time taken to mine the block an dother ineffcient planning. However when the state of the

Figure 5.4: Block analysis on real world data



spoil was introduced to the memotisation, the run time of the solution increased massivley, a hard timeout of 5 minutes was applied and no solution was found, further investigation showed that in five minutes A funciton call was performed 10^7 times. This amount of fundtion calls implies that the uniquness of the spoil

will limit the usability of the memotisation, and in fact only 8% of those function calls were able to be avoided due to memotisation. This presents the problem that the optimal solution will take too long to calculate yet the feasible solution is incredibly fast to calculate. The poor performance of the dynamic programming technique is due in no small part to both the huge amount of possibilities but also the uniqueness of the states, the combination of these two factors suggests that the approach may not be appropriate for this problem, however some of the methods learnt here could be refined for other more fitting areas of the implementation.

5.3 Optimisation of the Strip

5.3.1 Resource Constrained Dynamic Programming

Resource constrained dynamic programming is the initially obvious solution to the problem, as the mine is explored blocks in a valid range are considered and the current position is updated. The amount of blocks required is not known in advance and therefore a resource constrained dynamic program will find all feasible solutions to this problem alongside the optimal solution.

5.3.1.1 Model Formulation

States

$$S \quad \text{State of spoil at current stage} \tag{5.27}$$

Similarly to the formulation of the cost of a block the spoil capacity at any point is the resource constraining the solution, this is fed into the block cost algorithm in order to properly obtain the optimal solution to the problem, as the spoil is updated creating blocks will become more difficult

Stages

$$M \quad \text{current mine section } m \tag{5.28}$$

The stages of the model are something that we do not have control over, each action moves us from one stage to the next. This formulation is the current location of the start of the next block in the model, we consider this as a way of finding our end condition , however the stage is dependant on the actions taken prior, a longer block will result in a stage further on in the mine than usual. The use of a location as a state is vital to the understanding of the formulation.

Transition Function

$$S_{stage+1}(s) = S_{stage+1}(s) \times SEf \text{where } s \text{ is the spoil section in action} \tag{5.29}$$

The transition value allows the next state to be calculated based on the current state and the action chosen. Here the action chosen is the movement of overburden from the current block to some point on the spoil s, this action will succeed in adding additional overburden to the relevant section on the spoil, updating this is simple and from here the only additional piece of information required is the constraint that this transition function cannot create a state that exceeds the spoil capacity, this is done by simply checking the suggested next state and adding a large cost if this constraint is violated.

Value Function

$$C_A = \text{Cost of block length A} \tag{5.30}$$

$$V_t(S_t) = \min(C_A + V_t(S_{t+A})) \tag{5.31}$$

The value function calculates the cost associated with a given action based on the cost of the action and the resulting states for all future stages, the cost of an action is simply the swing time from the current block to the spoil selected as an action. This cost is to be minimised however will also affect the state of the next action,

as such this will recursively consider all potential courses of action for each action taken. Therefore the more actions that can be made the more computation time is required to calculate the cost of a block.

5.3.1.2 Implementation

```
1  def DP(self,position,end,spoil):
2    self.count +=1
3    print("Resource Constrained Dynamic Program \t Called:\t",self.
   count,'  times')
4    print("Distance Remainging  :\t" ,end-position)
5    if position>end-self.Dragline.get_minBlock():
6      position = end
7      # print("FUCK")
8      return (10000000000000000000000000,0,[])
9
10   if position == end:
11     # print(value, sep, end, file, flush)
12     return (0,0,[])
13   # if end <= position:
14     # self.dict[position,end,str(spoil)] = 100000000
15     # return self.dict[position,end,str(spoil)]
16   if (position,end,str(spoil)) not in self.dict:
17     self.dict[position,end,str(spoil)] = min((10000+self.Blk.
   BlockCost(position,position+Action,spoil)[0]+
18       self.DP(position+Action,end,self.Blk.BlockCost(position,
   position+Action,spoil)[1])[0]
19       ,Action,self.Blk.BlockCost(position,position+Action,spoil)
   [1])
20     for Action in range(self.Dragline.get_minBlock(),self.Dragline.
   get_maxBlock()))
21   return self.dict[position,end,str(spoil)]
```

5.3.2 Resource Constrainted Dynamic Programming with Sequence Generation

The strongest constraint in the formulation of the strip is that the sum of all blocks must be equal to the length of the strip, that is $\sum B = L_s$, however in the prior formulation this constraint is not considered, this constraint is applied by generating all sequences of potential blocks that will equal the length of the mine. This set of potential solutions is much less than the potential solutions without this constraint and therefore the application of such a constraint will reduce the exploration of the dynammic program, sequence generation is done recursively as seen in the below code snippet.

```
def GenBranchs(self,current,end,min,max,past):
  # count = count+1
  # print(count)
    if current == end :
      past+''
      self.Prunedstr.append(past)
      past = ''
      return 1
    if current>end:
      past = ''
      return -1
    for i in range(min,max):
      pred = past + ','+str(i)
      self.GenBranchs(current+i, end, min, max,pred)
    return 0
```

Once this is complete the potential solutions are a set of unique solutions to the problem, from here each potential solution is considered by using a forward dynamic program as formulated in the prior section, this modified method is seen below.

```
def PrunedCost(self):
    self.best = np.inf
    self.bestStr = []
    counter = 0
    Flagg = 0
```

```
6     for solution in tqdm(self.Prunelst):
7       # counter+=1
8       # print('Percent Complete:\t',counter/len(self.Prunelst)
    *100,'%')
9       for Block in range(len(solution)):
10        position = sum([solution[i] for i in range(Block)])
11        # print(position,Block,solution)
12        combo = ''
13        if Block > 0:
14          combo = str([solution[i] for i in range(Block)])
15        if combo in self.prunedict:
16          oldspoil = self.prunedict[combo][1]
17          blockcalc = self.Blk.BlockCost(position,position+solution[
    Block],oldspoil)
18          cost = self.prunedict[combo][0]+blockcalc[0]+10000
19          # if cost > self.best:
20            # Flagg = 1
21            # break
22        else:
23          oldspoil = self.Spoil
24          blockcalc = self.Blk.BlockCost(position,position+solution[
    Block],oldspoil)
25          cost = blockcalc[0]+10000
26          # if cost > self.best:
27            # Flagg = 1
28            # break
29        newcombo = str([solution[i] for i in range(Block)])
30        self.prunedict[newcombo] = [cost,blockcalc[1]]
31      self.Prunefinal[str(solution)] = self.prunedict[str([solution[i
    ] for i in range(Block)])]
```

5.3.2.1 Results

The implementation of the dynamic program without any preprocessing of potential solutions was unable to reach any final solution for a mine of length 500m, the result of this attempt was a five hour long simulation in which 10^16 state combinations were explored before the program caused the computer to crash, this was

found to be a reoccuring problem and was a key motivator for a method to reduce the amount of potential solutions considered.

Most solutions to the formualtion can overshoot the problem by one block or undershoot it by less than one block, the set of solutions that can exacly equal the length of the mine is a small subset of the total solution space. The set of these sequences was generated and for a 500m strip was found to contain 10^9 uniqur solutions to the problem. However the runtime of this sequence generation is $O(2^n)$ where n is the set of possible lengths the block could be. This runtime is not ideal for high resolution data however when compared to the previous method a solution is still found within three hours. The time taken to generate all possible sequences for a strip of 500m at a resolution of 1m is 900 seconds, however at a resolution of only 5m this is reduced to 200 seconds and 10^6 solutions. Therefore a quick solution could be used as a hueristic for potential better solutions to reduce the amount of time taken. The time spent in the dynammic program however is still dramatically reduced as the runtime of the program is $O(|A|S^2|)$ and as the statespace is reduced by a magnitude of 7 the states processed is also hugely reduced.

Chapter 6

Results

The results of this thesis are intended to show evidence of a working system, and in particular the performance of subsystems. The validation and explanation of such subsystems is vital to the validity of the work done in this thesis.

6.1 Cost of Block

In the implementation of this thesis three methods were selected to represent the cost of a block, from these three the Mixed integer linear program with validity constraints was selected to be the model used in the final iteration of the project. The model was then compared to the results generated by preexisting algoritms[**?**] such that a validation could be made, the comparison of these two methods is seen below in table **??**.

Length of Block	Operation Cost	True Cost	Error (%)
32	7466	7377	1.206
37	8578	8383	2.326
42	9743	9314	4.606
47	10908	10229	6.638
52	12020	11554	4.033

As table **??** shows the error between the true values and the calculated values increases as the length of the block increases, this could be in part due to differences and assumptions made in the modelling of the spoil chanel and spoil capacity. However it is also reasonable that this error is due in no small part to the assumptions made when reducing a three dimensional problem into two dimensions an as such could be seen as a potential validation of the model of block costing, furthermore the typical cost of a block for any enviroment seems to have similar results through all blocks, with the more complex datasets (5,6,7,8,9,10) having a larger cost in general than the simpler counterparts. As seen in table **??** the results for sets 1,3,4,5 were all very similar, however the results for the more complex blocks are often larger as the range of feasible solutions is lower. This suggests that the cost of a block is dependant on location, legnth and spoil, and as such methods such as machine learning or linear simplifications may not be valid.

Length	Set 1	Set 2	Set 3	Set 4	Set 5
32	7466.6	5772.1	7201.9	7784.4	7466.6
37	8578.7	6884.1	8261	9108.2	8472.8
42	9743.7	8419.8	9055.3	10696.9	9214.1
47	10908.7	10008.5	10114.4	12285.5	10008.5
52	12020.7	10855.7	11173.5	13768.3	11226.4
Length	Set 6	Set 7	Set 8	Set 9	Set 10
32	7625.5	8261	7519.6	7360.7	9002.3
37	8737.5	9320	8525.7	8261	10326.2
42	10061.4	10485.1	9373	9214.1	11491.2
47	11173.5	12073.7	10167.3	9743.7	12020.7
52	12020.7	13079.8	10961.6	11120.5	12762.1

However the cost of a block is not the only output generated by the model, the state of the spoil after the block has been removed is also returned. Spoil constraints on the overall solution requew

Chapter 7

Conclusion

Appendix A

Code Listings

Bibliography