OPTIMUM DESIGN OF OPEN PIT MINES

LUCIANO MARIO GIANNINI

This thesis is presented in fulfilment of

the requirements for the award of

Doctor of Philosophy

of

Curtin University of Technology

1990

DEDICATION

To my mother and father, who made it all possible, to my wife Irene, for her interest, encouragement and patience, and to my children Fiona, Jeremy, Natasha and Leanne, whom it is all ultimately for.

SUMMARY

A fundamental problem in open pit mine planning is that of determining the optimum ultimate pit limits of the mine. These limits are that pit contour which is the result of extracting a volume of material which maximizes the difference between the value of extracted ore and the total extraction cost of ore and waste whilst satisfying certain practical operational requirements, such as, safe wall slopes. The determination of the optimum pit contour provides information which is essential in the evaluation of the economic potential of the mineral deposit.

A number of optimization techniques have been proposed for determining the optimum pit contour. Of these techniques, those based on graph theory, linear programming and dynamic programming are mathematically rigorous, but only those based on graph theory are more suited to solving the three-dimensional problem. Unfortunately, direct application of these techniques to large ore-bodies may cause considerable difficulties because of the exceptionally high demand on computer storage and time requirements. Indeed, 25 years of research effort has not satisfactorily resolved these computational problems.

A major contribution of the work presented in this thesis is the successful implementation of a system of techniques to solve the graph theoretic model, particularly when applied to large ore-bodies. A measure of this success is the fact that pits, as much as seven times larger may be designed with a given amount of computer storage, at a fraction of the time required by current software packages. The solution strategy presented involves the application of a modified Dinic's Maximum Flow algorithm, together with an efficient 'data reducing' technique. Computational results of these

techniques applied on data from gold producing mines in Western Australia are used to demonstrate the success of this strategy.

The relationships between the rigorous pit optimization techniques are also considered in this work. In particular, the Lerchs-Grossman graph-theoretic method is shown to be stepwise equivalent to a modified version of the Dual-Simplex Linear Programming technique and not as efficient as the Network Flow method.

CERTIFICATION

I certify that this thesis is entirely my own work, and that it has not previously been submitted, in whole or in part, for any academic award at Curtin University of Technology or elsewhere.

Luciano Mario Giannini

April, 1990

ACKNOWLEDGEMENTS

I wish to express my sincerest gratitude to my friend and colleague Dr Lou Caccetta, my thesis supervisor, through whom I have cultivated a strong affinity for research in discrete mathematics. My thanks also goes to Dr K Vijayan, associate supervisor, for his assistance and valuable comments and suggestions.

I am deeply indebted to Peta Kelsey who very professionally and skillfully converted algorithms in Chapter 5 to an efficient pit optimization software package and tested it on data from actual producing mines.

I am also indebted to Dr Spero Carras who introduced me to the real problems of open pit mining and for the interest and support he has given to the project, and to Carras Mining and staff for providing the data.

A special thank you is due to Julie Aizlewood who skillfully typed this dissertation.

Lastly, but by no means least, I wish to thank the School of Mathematics and Statistics for its support, and June Yates and Anita Littlewood for their help in the typing associated with the diagrams.

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 An Overview

For many years now optimization techniques have been successfully implemented to solve problems arising in the mining industry. Indeed, since the 1960's, numerous publications have appeared in the literature concerned with the application of these techniques to mining optimization problems - see, for example, the APCOM Symposium Series (19 volumes since 1961) published by the Society of Mining Engineers of AIME, New York or the book: *Computer Methods for the 80's in the Mining Industry* (Weiss 1979). These applications include:

- Ore-body modelling and ore reserve estimation.
- The optimum design of open pits.
- The determination of mine production schedules.
- The determination of optimal operating layouts.
- The determination of optimal blends.
- The determination of equipment maintenance and replacement policies.

A feature of these problems is their complexity which is due, in part, to the variability of the material being mined and the difficulty of estimating or predicting this variability. The frequent unavailability of accurate data and, when data is available, the logistic problem concerned with the processing of this data also contributes significantly to the problem's complexity. Thus the operations research analyst is usually provided with some very challenging problems. In this thesis we restrict our attention to the problems arising in open pit design.

The initial problem facing the mining engineer is that of developing a model of the ore-body. This model is based on geological exploration and bore hole analysis. Usually, the initial feasibility study establishes whether or not a potentially profitable ore-body exists. If the study indicates the existence of such an ore-body, then a more accurate (and more expensive) model is built in order to evaluate the full economic potential of the deposit. This model must convey, accurately, the quantity and quality of the ore, the mining costs (extraction and transportation) and the market value of the ore. Once this information is obtained one addresses the problem of mine planning.

Mine planning involves the determination of an extraction sequence over a particular time horizon, typically the life of the mineral deposit. The **optimum ultimate pit limit** of a mine is defined to be that contour which is the result of extracting the volume of material which provides the total maximum profit whilst satisfying certain practical operational requirements. A minimum requirement would be that the pit should have safe and workable wall slopes. These restrictions are dependent upon the geological structure of the material being mined as well as the mining equipment used. Other requirements that may be stipulated are:

- Minimum pit bottom width,
- Incremental pit generation, and
- Tonnage specification.

The phrase **pit limit problem** is usually associated with the case when the only constraints are the maximum allowable wall slopes. We will continue with this convention and use the phrase **constrained pit limit problem** when additional constraints (such as those mentioned above) are considered.

The pit limit problem has long been considered a fundamental problem in mine planning. The ultimate pit limit plan provides information which is essential in the evaluation of the economic potential of a mineral deposit, and in the formulation of long-, intermediate-, and short-range mine plans.

The long-range plan establishes the initial investment plan and the planning of surface facilities such as treatment plants, waste dumps, tailing ponds, and other elements complementary to the mining operation. A truly workable pit plan must allow for access, haulage, enough operating room to work equipment efficiently, the location of dumps at each stage, dewatering, and enough ore exposure to assure a proper mill feed even in the face of uncertainty.

Intermediate-range plans are a sequence of feasibile depletion schedules leading from the initial condition of the deposit to the ultimate pit limit. These plans are developed with the objective of obtaining optimum cash flows within the total reserves as outlined in the long-range plan and within a framework of restrictions dictated by physical, geological, operational, legal, and other policy considerations. Each plan usually varies in duration from 1 to 10 years, and provides management information necessary for forecasting future production and capital requirements.

The techniques of linear programming and mixed integer programming have been successfully used in both long- and intermediate-range planning. Albach (1967) uses a linear programming model for determining optimal multi-stage production plans when there are uncertainties on future demand and technical progress of a lignite open pit mine. Dagdalen and Johnson (1986) uses the lagrangian multiplier technique to propose a method for solving the multi-period production scheduling problem. Janssen

(1969) uses a linear programming model for resolving planning problems associated with the production of iron ore from a number of relatively small deposits of the Iron Ore Company of Canada. Wilke, Mueller and Wright (1984) employ a simulation algorithm in conjunction with linear programming to determine the long- and medium-term production schedules. They use simulation for handling geometrical and equipment restrictions, and linear programming for determining the optimum ore and waste movements as a function of time. Other studies involving these techniques in this context are Fraser (1971), Gershon (1982), Johnson (1969) and Kim (1979b).

Short-range plans are concerned with the operational state of the mine within the confines of the most recent medium-range plan. The planning period here is usually one year with stages of months, weeks or days. Production scheduling, operating equipment, and material handling procedures form significant components of the short-range mining activities. Production scheduling/planning provides projections of future mining progress and time requirements for the development and extraction of the resource. This is important to the overall mine design because of the substantial costs associated with labor, supplies and equipment. The main objective of production scheduling is meeting the production goal at minimum operating expense. Usually, the goal is to supply as much ore of a uniform grade and tonnage as the plant is capable of handling without jeopardizing the plant's operating efficiency. Uniform grade ore is obtained by the blending of different materials either from different locations within the mine or from different mines. The restrictions which the mining schedule must satisfy include: orderly extraction; mining equipment capacity; milling capacity; refining capacity; grades of millfeed and concentrates;

labor; and other physical, operating, legal and policy limitations. Production scheduling also allows one to analyse the production capacity of the existing equipment. This evaluation usually includes an examination of previous equipment performance levels and a projection of expected equipment.

A number of authors have considered the problem of production scheduling. Kim (1979b) contains a technical overview of production scheduling. Gershon (1982) contains an excellent review of the applications of linear programming in the mining industry as well as a Mine Scheduling Optimization System which determines, using a mixed integer programming model, the optimal operation of a mine, from mine to plant to market. This system deals with long-, intermediate-, and short-range mine planning. Roman (1973) uses dynamic programming for determining production schedules. Other studies dealing with mine production scheduling are Couzens (1979), Fytas and Calder (1986), Johnson (1969), Kahle and Scheaffer (1979), Kim (1979b), and Wilke, Mueller and Wright (1984).

Techniques for solving the pit limit problem are available and discussed further in Section 1.2. However, the complexity of the problem has limited the use of the more elaborate of these techniques and many open pit mines are still being designed manually. Considerable effort has recently been made, particularly the work presented in this thesis, on applying these techniques to the design of large open pit mines.

The constrained pit limit problem, because it incorporates a number of practical requirements, is important in mine planning and management. It is also closely related to the problem of production scheduling. We consider the constrained pit limit problems listed above, in Sections 4.3, 4.4 and 4.10. As we shall see in Section

4.10, the fixed total tonnage requirement constraint problem can be reduced to the pit limit problem using the lagrange multiplier technique.

1.2 The Pit Limit Problem

As discussed in Section 1.1, a fundamental problem in open pit mine planning is that of determining the ultimate pit limits of the mine (often referred to as **pit design**). The problem is to determine a pit contour which maximizes the difference between the value of the extracted ore and the total extraction cost of ore and waste whilst satisfying the safe wall slope restrictions.

We may express this pit limit problem analytically as follows. Let v, c and m be three density functions defined at each point (x,y,z) of a three-dimensional region containing the ore-body with

$v(x,y,z)$: mine value of ore per unit volume,

$c(x,y,z)$: extraction cost per unit volume,

$m(x,y,z)$: profit per unit volume;

$$m(x,y,z) = v(x,y,z) - c(x,y,z).$$

Let $\alpha(x,y,z)$ denote the set of angles specifying the wall slope restrictions at the point (x,y,z), with respect to a fixed horizontal plane, for a given set of orientations (azimuths). We describe this set α as the **variability of wall slope requirement**. Let $\mathcal{S}$ define a set of surfaces such that at no point does their slope, with respect to the fixed horizontal plane, for each of the given orientations, exceed the corresponding angle in α. Let $\mathcal{V}$ be the family of volumes of the ore-body, contained within the surfaces of $\mathcal{S}$. The pit limit problem is to find a volume $V \in \mathcal{V}$ which maximizes the integral

$$\int_V m(x,y,z)dx\ dy\ dz.$$

The surfaces in $\mathcal{S}$ are referred to as **feasible pit contours**, and that surface in $\mathcal{S}$ whose volume is V is called the **optimum pit contour**.

Since, in practical situations, there is no simple analytical representation for the functions v and c, numerical techniques must be used to solve the pit limit problem. This involves the discretization of the problem.

Generally, the ore deposit is divided into blocks. There are various block models one can use (Kim 1979a) but the regular 3-D fixed-block model is the most common and is best suited to the application of the computerized optimization methods of pit limit design. The model is based on the ore-body being divided into fixed-size blocks. The vertical dimension of each block usually corresponds to the bench height. Horizontal dimensions of the blocks are fixed and do not vary from location to location; they are dependent on the physical characteristics of the mine, such as pit slopes, dip of deposit and grade variability.

Drill-hole assays are given a definite spatial location. There are various methods (Gignac 1975) to assign to the centre of each block a grade representative of the whole block such as distance weighted interpolation, regression analysis, weighted moving averages and kriging. The block model is then economically evaluated by assigning a net value to each block. Separation between profitable material and non-profitable material is dependent on costs. The relevant costs associated with each block are:

- **Mining costs** (c_m). This includes costs of labor, equipment, loading and hauling the block from the mine. Generally, this cost of mining will increase with depth

because of the increasing hauling distance.

- **Processing costs** (c_p). This includes costs involved in the crushing and extraction of valuable minerals from the block.
- **General costs** (c_g). These include overheads, selling costs, royalities, etc.

Knowing the costs, it is then possible to fix a cut-off grade below which an unmined block will be considered as waste material.

The value v of a block (in dollars) is given by the equation

$$v = M \times P \times R$$

where

M : metal content (tonne); M = specific gravity $\times$ grade

P : market price (dollar/tonne)

R : processing recovery (percentage).

Assuming that the cost of mining a block does not depend on the sequence of mining, the net profit value m of a block is given by

$$m = v - c$$

where

$$c = c_m + c_p + c_g.$$

If a block is mined with a negative net profit value m, the block is considered as waste. If the mined block is such that $m > 0$, then it is considered as ore. We note, here, that open pit mining involves the removal of non-profitable material to recover profitable material.

On discretizing the ore-body into rectangular blocks and referencing the blocks with a suitable three-dimensional grid system (in Chapter 5 we set up such a grid for an optimization method), we let m_{ijk} be the net profit value of a block with centre at (i,j,k). Assuming that the desired wall slopes and pit outlines can be

approximated by removed blocks, the pit limit problem becomes one of finding

$$\max_{\gamma} \sum_{(i,j,k)\in\gamma} m_{ijk}$$

where γ is a set of coordinates of blocks the removal of which results in a feasible pit design. Note that another assumption being made here is that total undiscounted profit is to be maximized.

1.3 Pit Design Techniques

To date the optimization techniques proposed for the solution of the pit limit problem are (first publication dates are given in brackets):

- Dynamic programming (1965)
- Graph theory (1965)
- Linear programming (1969)
- Network flow (1976)
- Pit parameterization (1976)
- Heuristics (1965-1974)
- Manual methods.

The heuristic algorithms were developed in the late 60's and early 70's. A good review of these techniques may be found in Kim (1979a). These techniques were developed and used because of a number of reasons: lack of a rigorous mathematical technique, lack of understanding the rigorous methods by the mining industry or simply lack of computer power to accommodate the complex nature of the more rigorous methods. Manual methods (Koskiniemi 1979) are methods used by the mining engineer in designing 'optimum' pit limits from various sections and cross-sections of the

mineralization of the ore-body. These methods are extremely time consuming and are heavily dependent upon the skill of the engineer. From the author's experience, heuristics such as the Moving Cone method (Robinson 1975) as well as manual methods are extensively being used by the mining industry today. This suggests that there is a certain lack of confidence in the state of the art software packages available on the market. We take this up later in our discussion. However, both the heuristic and manual methods are capable of missing the optimum completely and thus resulting in considerable loss in revenue.

The dynamic programming method was first introduced by Lerchs and Grossman (1965). The method optimizes the pit contour on each two-dimensional cross section of the block model. Johnson and Sharp (1971) modified this method by applying dynamic programming on all the optimum two-dimensional pit outlines in each section with pit bottom restricted to each of the rows (levels) within the section. Unfortunately, the method does not completely cater for variability in wall slope requirements and in many cases fails to produce a feasible pit contour. Koenigsberg (1982) and Wright (1987) made serious attempts at producing true 3-D dynamic programming methods but these are subject to implementation problems because of an increase in the number of state variables and other problems discussed in Caccetta and Giannini (1990).

The graph theoretic method was also developed by Lerchs and Grossman (1969) who formulated the problem as one of finding, in a given weighted directed graph, a maximum weight subgraph satisfying a certain property. We discuss this method in detail in Chapter 2. The method is a true 3-D optimization method capable of handling the variability of the wall slope requirement within the pit. The time

complexity of their algorithm is not precisely known but it is shown in Chapter 3 that this is at least $O(n^3)$ where n is the number of blocks in the model. Moreover, the computer storage requirement (also discussed in Chapter 3) is extremely demanding, particularly for large pits. For these reasons, the major problem associated with this method is in its practical implementation. This is evident from the works of Chen (1976), Lipkewich and Borgman (1969) and Alford and Whittle (1986). The latter work (incorporating the recently developed 'state of the art' package) reports a run time of 3 hours for their implementation of this method on a small 30,000 block model, and of 100 hours for a 500,000 block model, requiring 100 megabytes of disk. In Chapter 5 we present a software system which is extremely efficient in terms of both computer storage and run time making it better suited for the smaller computer such as a personal computer.

Meyer (1969) formulated the pit limit problem as an integer linear programming problem and because of the large sparse coefficient matrix which results from such a formulation makes several assumptions to reduce the computational effort in implementing this technique. His assumptions however, severely limit the practical use of his model, and so this method has not found any use in pit design to this day. A brief outline of this technique is given in Chapter 3.

Picard (1976) showed how the pit limit problem is solved by modelling it as a maximum flow problem in a network. Despite the number of techniques available to solve the maximum flow problem there is no recorded evidence of its use by the mining industry as a practical solution strategy for the pit limit problem. In Chapter 5 we implement a specifically modified Dinic's maximum flow algorithm

to solve this problem.

In the pit parameterization algorithm (Francois-Bongarcon and Marechal 1976), the conventional search for an optimum pit contour is replaced by the determination of a parameterizing function, the isovalue curves of which give the solution under a given set of economic conditions. The essence of the algorithm lies in the development of the parameterizing function which is an approximation of the true pit contour. Certain implicit assumptions are made to satisfy the mathematical requirements. One such assumption is that the pit under consideration will be a convex pit. Although this algorithm has been commercially available, there is not much published about its performance. Moreover, very little interest (if any) is shown in this method by the mining industry.

1.4 Objectives and Outline

In this thesis we focus on the pit limit problem and its solution. In particular, we address the problem of the optimum design of *large* open pit mines. Since the development of the Lerchs-Grossman graph-theoretic method in 1965, a number of authors (see Section 2.5) have experienced significant problems in its implementation particularly when applied to large ore-bodies. The difficulty of these problems is perhaps best measured by the fact that 25 years of research effort has not resulted in a satisfactory resolution of these problems. A major contribution of this thesis is the successful implementation of a system of techniques (including Network Flow) to solve the Lerchs-Grossman graph-theoretic model for large ore-bodies.

The rigorous techniques of pit design, namely, Graph Theory, Linear and Dynamic Programming and Network Flow are discussed.

Their relationships are also considered. The implementation of the methods are analytically compared and we show that the Network Flow method, using a modified Dinic's Maximum Flow algorithm together with a 'data reducing' algorithm, is the most efficient technique to date. As a result of this efficiency, much larger pits (as much as seven times larger) may be designed with a given amount of computer storage and at a fraction of the time required by the current packages.

In Chapter 2, we give a more mathematical treatment of the Lerchs-Grossman graph-theoretic method than originally presented. We devise an algorithm for generating the graph model for the pit limit problem given any number of wall slope restrictions. This algorithm is also shown to generate the most economical graph for a given problem, in that, a minimum number of arcs are used in its construction for a required error tolerance in the wall slopes. Test cases for this algorithm and implementation strategy for the Lerchs-Grossman algorithm are also presented.

Chapter 3 looks at the linear programming model proposed by Meyer (1969) and shows that the Lerchs-Grossman graph-theoretic algorithm is, in fact, equivalent (step-by-step) to solving, using a modified dual-simplex technique, the dual of a linear programming formulation of the pit limit problem. With this equivalence, an alternate non graph-theoretical method of solving the pit limit problem is presented and an estimate of the time complexity of the Lerchs-Grossman algorithm is provided.

In Chapter 4 we discuss the 2-D and 3-D dynamic programming methods. We show how the 3-D method may be modified to allow for some variability in the wall slope restrictions and to properly handle air blocks. The theory of bounding the optimum pit for the

purpose of significantly reducing the size of the problem, is developed and a fast bounding algorithm based on the 3-D dynamic programming method is presented. Desirable practical features such as *minimum pit bottom width* are also discussed and implemented.

In Chapter 5 we discuss the network flow method of Picard and the selection of the Dinic's algorithm to determine the maximum flow in the network model of the pit limit problem. A detailed description of the implementation strategy of this method is also included. Using block model data from gold producing mines in Western Australia we show, through various runs of this implementation together with the bounding algorithm, under different block and wall specifications, that this technique is extremely fast and efficient. We discuss the performance of this technique and compare it with the claims of the current state of the art implementation of the Lerchs-Grossman technique available on the market. Pseudo code of the main modules in the software for the network flow technique (coded in the C-language) is given in the Appendix.

CHAPTER 2

THE GRAPH THEORETIC APPROACH

A **graph** may simply be thought of as an entity consisting of nodes or **vertices** joined by lines or **edges**. If the edges are directed they are known as **arcs** and the graph is then known as a **directed graph** or **digraph**. If a value is associated with each vertex in the graph, we refer to the latter as a (vertex) **weighted graph**. If some vertices and/or edges (arcs) are removed from a graph (digraph), the resulting graph (digraph) is known as a **subgraph (subdigraph)** of the original graph (digraph). A **path** in a graph is a finite sequence $v_o e_1 v_1 e_2 v_2 \ldots e_k v_k$ consisting of distinct vertices v_i $(i=0,1,\ldots,k)$ and distinct edges e_i $(i=1,\ldots,k)$ such that for each i, $1 \le i \le k$, the ends of e_i are v_{i-1} and v_i; v_o and v_k being the start and end of the path, respectively. A **cycle** is a path in which the start and end vertices are joined by an edge. A graph is **connected** if for each pair of vertices u and v in the graph there is a path with u and v as start and end vertices. Finally, a **tree** is a connected graph without cycles.

The graph theoretic method of solving the pit limit problem was developed by Lerchs and Grossman (1965) who formulated the problem as one of finding, in a given weighted directed graph, a maximum weight subgraph satisfying a certain property. A finite algorithm for determining such a subgraph was also presented in their paper. There are, however, serious computational problems associated with the application of their algorithm to real ore-bodies. How, for instance, is the graph, associated with the pit limit problem, constructed? When constructed, for even a small-sized model, the

graph requires considerable computer storage space and its manipulation by the algorithm requires considerable computational effort. A number of researchers have addressed these problems (see Section 2.5) but without a completely satisfactory resolution, particularly when the algorithm is applied to large ore-bodies. In Chapter 5 we present a system of techniques for resolving these problems.

The treatment of the method given by Lerchs and Grossman lacks a certain amount of mathematical rigor. In this chapter we present the full development of the graph theoretic method. We devise an important algorithm for generating the graph for the pit limit problem given any number of wall slope restrictions within the pit. This algorithm is also shown to generate the most economical graph for a given problem, in that, no redundant arcs are included in its construction. Test cases for this algorithm and an implementation strategy for the graph theoretic method are also presented in this chapter.

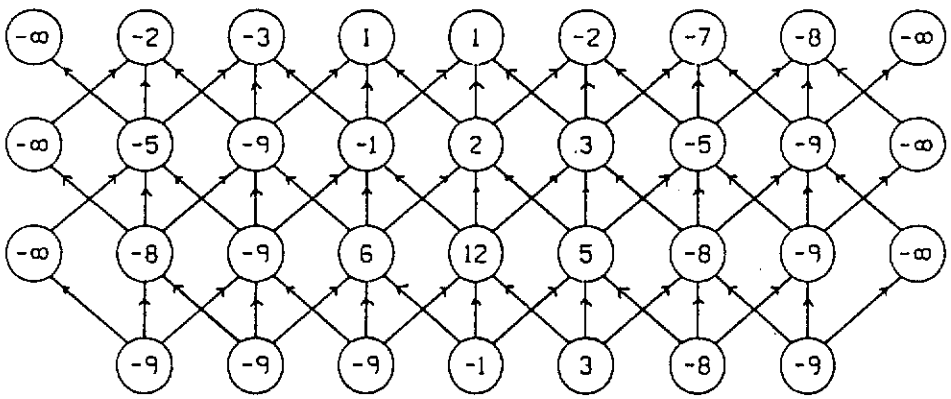
2.1 Pit Limit Problem Formulation

As described in Caccetta and Giannini (1988a), we may represent the block model of the ore-body as a weighted directed graph with the vertices representing the blocks and the arcs representing the mining restrictions on adjacent blocks. More specifically, our graph contains the arc (x,y) if the mining of block x is dependent upon the removal of block y. However, if the mining of block y is dependent upon the removal of block z and so arc (y,z) is in the graph, then arc (x,z) is redundant and is not in the graph. We will discuss the generation of such graphs in Section 2.4. Thus, for our

purposes we will refer to two blocks x and y as **adjacent** if either arc (x,y) or arc (y,x) is in the graph. The profit resulting from the mining of a block is represented by an appropriate vertex weight. Figure 2.1 illustrates the model for a cross-section with a 45° wall slope restriction. Note that the "$-\infty$" vertices are necessary to prevent mining outside the limits of the data.

-2	-3	1	1	-2	-7	-8
-5	-9	-1	2	3	-5	-9
-8	-9	6	12	5	-8	-9
-9	-9	-9	-1	3	-8	-9

(a)



(b)

Figure 2.1 (a) Cross-section of ore-body. (b) Graph representation.

We define a **closure** of a weighted digraph D as a set C of vertices of D such that if $x \in C$ and (x,y) is an arc of D, then $y \in C$. The **weight** $w(C)$ of C is the sum of the weights of the vertices

of C. A **maximum closure** is a closure of maximum weight. Note that a closure of D represents a feasible pit contour; its weight represents the profit realized by the resulting pit contour. Thus the problem of determining the optimum pit contour is equivalent to the graph-theoretical problem :

Find, in a weighted digraph D, a closure of maximum weight.

2.2 Properties of Closure

For the most part, our graph theory terminology follows that of Bondy and Murty (1977). Thus a graph G has vertex set $V(G)$ and edge set $E(G)$, a digraph D has vertex set $V(D)$ and arc set $A(D)$. All graphs considered in this thesis are simple (i.e. loopless and have no multiple edges). Also, all our graphs are (vertex) weighted.

Consider a weighted digraph D. Let D' be a **spanning subdigraph** of D (i.e. D' is obtained from D by removing some of the arcs). Let C and C' be maximum closures of D and D', respectively. Then

$$w(C) \leq w(C') \ . \tag{2.1}$$

The implication of inequality (2.1) is that if C' is a closure of D, then it must be a maximum closure of D. This suggests that to find the maximum closure of D we try and deal with some simple spanning subdigraph of D. Trees are a class of graphs whose properties are well documented, and have proven to be algorithmically attractive. The Lerchs-Grossman algorithm manipulates a special type of spanning tree.

Given a digraph D we can obtain an undirected graph G from D by replacing each arc by an edge; G is the **underlying graph** of D. As we do not consider concepts involving only the notion of

orientation, all terminology used in the following applies to both directed and undirected graphs. Thus a directed graph is a tree if its underlying graph is connected and has no cycles.

A **rooted tree** is a tree with one distinguished vertex called the root; we denote the root vertex by v_o. The graph obtained by deleting an arc $a_i = (x_i, y_i)$ in a rooted tree T has two components. The component T_i which does not contain the root of T is called a **branch** of T; the arc a_i is said to **support** the branch T_i. The end of a_i which is in T_i is called the root of T_i. We say a_i is a **p-arc** (plus arc) if y_i is the root of T_i; otherwise it is called an **m-arc** (minus arc). Denoting the total weight of the vertices of the branch T_i as $w(T_i)$, we say that T_i is a **strong branch** if a_i is a p-arc and $w(T_i) > 0$ or a_i is an m-arc and $w(T_i) \leq 0$; otherwise it is a **weak branch**. a_i is classified as **strong arc** or **weak arc** according to whether the branch it supports is strong or weak. A vertex u is a **strong vertex** if there exists at least one strong arc on the (unique) path joining u to the root vertex v_o. Figure 2.2 illustrates these concepts. Note that each arc a_i carries a label of the form $(\pm, w(a_i))$ to identify its status (+ for p-arc, - for m-arc) and the weight $(w(a_i))$ of the branch it supports; we can thus identify an arc as strong or weak by examining its label. This labelling will be very useful in our algorithm implementation. The arcs (c,d) and (e,f) are strong; all other arcs are weak. Thus vertices a, b, c, f, g, h and i are all strong whilst d and e are both weak. A rooted tree whose root is common to all strong arcs is called a **normalized tree**. The tree in Figure 2.2 is not a normalized tree. The concept of normalization is important because of the following characterization.

Theorem 2.1 *Every normalized tree T has a maximum closure whose vertices correspond to its set of strong vertices.*

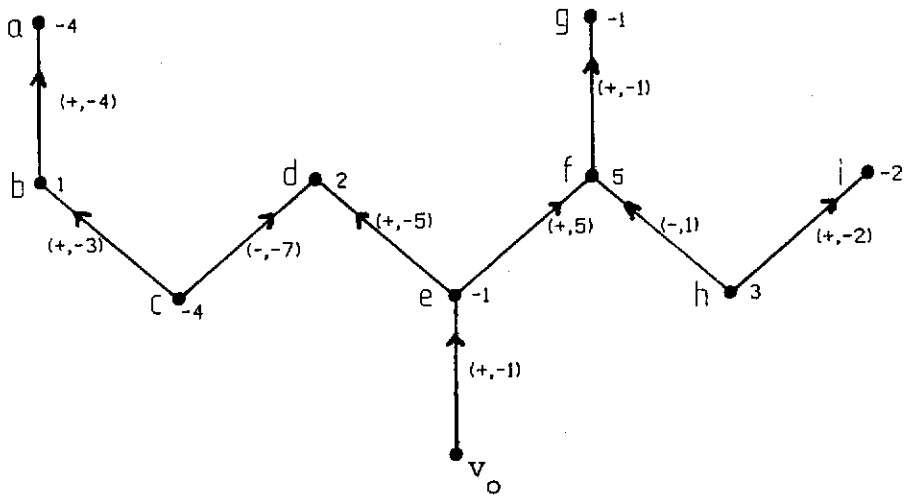


Figure 2.2 Graph theoretic concepts.

This result together with inequality (2.1) yields:

Theorem 2.2 *If, in a weighted digraph D, a normalized spanning tree T can be constructed such that the set Y of strong vertices of T is a closure of D, then Y is a maximum closure of D.*

The above results form the basis of the Lerchs-Grossman algorithm which, in effect, constructs a sequence of normalized trees $T^0, T^1, \ldots, T^n$, terminating when the set of strong vertices of T^n forms a closure of the graph representing the ore-body. In applying the algorithm, the digraph D representing the ore-body is augmented by adding a vertex v_o and all arcs (v_o, v_i), $v_i \in V(D)$. The vertex v_o is given a negative weight and therefore cannot be

part of any maximum closure.

Each normalized tree T^i constructed forms a spanning tree of the augmented graph with root v_o and is obtained from T^{i-1} by following certain well-defined rules. The algorithm is described in detail in Section 2.3. Since the proof of Theorem 2.1 given in Lerchs and Grossman (1965) is not complete, we include a detailed proof of this theorem.

Let T be a normalized tree with maximum closure $C^*(T)$. We have the following lemma.

Lemma 2.1 *If $v_i \in C^*(T)$, then every vertex of the branch $B(v_i)$ with root v_i is in $C^*(T)$.*

Proof. Suppose v_j is a vertex of $B(v_i)$ which is not in $C^*(T)$. Let

$$B_1 = V(B(v_i)) \cap C^*(T)$$

and

$$B_2 = V(B(v_i)) \backslash B_1 .$$

Since $C^*(T)$ is a closure, all arcs that join vertices of B_1 with vertices of B_2 have their terminal vertex in $C^*(T)$. Let $P(v_i, v_j)$ be the unique (v_i, v_j) - path in $B(v_i)$. There must necessarily be an arc (v_q, v_p) on $P(v_i, v_j)$ with $v_p \in B_1$ and $v_q \in B_2$ (v_p could be v_i and v_q could be v_j). Since no vertex of $B(v_i)$ except possibly v_i can be joined to the root v_o, the arc (v_q, v_p) must be an m-arc and hence carries a label $(-, w(v_q, v_p))$ with $w(v_q, v_p) > 0$ (since T is normalized). Now consider the branch $B(v_q)$ of T with root v_q. Let

$$B_1' = B_1 \cap V(B(v_q))$$

and

$$B_2' = V(B(v_q)) \backslash B_1' .$$

Since $v_i \notin B(v_q)$ and T is a tree, the unique path connecting a vertex of B_1' to v_i must pass through the arc (v_q, v_p). Hence all arcs joining a vertex of B_1' to a vertex of B_2' are p-arcs and hence carry a non-positive weight. Thus $w(B_1') \leq 0$ and so

$$\begin{aligned} w(B_2') &= w(B(v_q)) - w(B_1') \\ &\geq w(B(v_q)) \\ &> 0 \ . \end{aligned}$$

Hence the closure

$$\hat{C} = C^*(T) \cup B_2'$$

has a weight

$$\begin{aligned} w(\hat{C}) &= w(C^*(T)) + w(B_2') \\ &> w(C^*(T)) \end{aligned}$$

contradicting the assumption that $C^*(T)$ is a maximum closure of T. □

Lemma 2.1 is essentially Property 1 in Lerchs and Grossman (1965). It alone does not imply Theorem 2.1 since it does not tell us anything when $v_i \notin C^*(T)$. The following lemma is needed.

Lemma 2.2 *Let $B(v_k)$ be the branch of T with v_k joined to the root v_o. If $v_k \notin C^*(T)$, then $B(v_k)$ is a weak branch and there is a maximum closure which contains no vertex of $B(v_k)$.*

Proof. If no vertex of $B(v_k)$ is in $C^*(T)$, then we have nothing to prove. So suppose v_i is a vertex of $B(v_k)$ belonging to $C^*(T)$. Let P be the unique (v_i, v_k)-path in $B(v_k)$, and $v_q v_p$ the first edge of P with $v_q \notin C^*(T)$ and $v_p \in C^*(T)$. Since $C^*(T)$ is a closure, (v_q, v_p) is a p-arc and (since T is normalized) of non-positive weight. By

Lemma 2.1, every vertex of the branch $B(v_p)$ with root v_p is in $C^*(T)$. Since $C^*(T)$ is a maximum closure and (v_q, v_p) supports $B(v_p)$, $w(B(v_p)) = 0$.

Next, we establish that no vertex of the (v_q, v_k)-section of P is in $C^*(T)$. Suppose this is not the case and let v_α be the first vertex of the section which is in $C^*(T)$. By Lemma 2.1, every vertex of the branch $B(v_\alpha)$ with root v_α is in $C^*(T)$. But v_q is in $B(v_\alpha)$ and so $v_q \in C^*(T)$ contradicting our assumption that $v_q \notin C^*(T)$. Hence no vertex of the (v_q, v_k)-section of P is in $C^*(T)$. Letting

$$B_1 = V(B(v_k)) \cap C^*(T),$$

that is B_1 is the set of vertices of $B(v_p)$, and

$$\hat{C} = C^*(T) \backslash B_1 ,$$

we have that $\hat{C}$ is a closure of T with weight

$$\begin{aligned} w(\hat{C}) &= w(C^*(T)) - w(B_1) \\ &= w(C^*(T)) \end{aligned}$$

since $w(B_1) = w(B(v_p)) = 0$.

Hence $\hat{C}$ is a maximum closure of T not containing any vertices of $B(v_k)$. □

Theorem 2.1 follows immediately from Lemmas 2.1 and 2.2.

2.3 The Lerchs-Grossman Algorithm

In this section we present the Lerchs-Grossman algorithm and provide a proof of its convergence. As mentioned in the previous section, the digraph D representing the ore-body is augmented by adding a vertex v_o and all arcs (v_o, v_i), $v_i \in V(D)$. The vertex v_o is given a negative weight so that it cannot be part of a maximum closure.

Algorithm

Step 1 (Initialization). Construct a normalized tree T^0; T^0 is conveniently taken as the spanning tree whose arc set is $\{(v_o,v_i) : v_i \in V(D)\}$. Identify the set Y^0 of strong vertices of T^0 (which are those vertices with positive weight). Set $i = 0$ and proceed to Step 2.

Step 2 (Optimality test). Search for an arc (v_k,v_l) in D such that $v_k \in Y^i$ and $v_l \notin Y^i$ and proceed to Step 3. If no such arc is found, stop; Y^i is a maximum closure of D.

Step 3 (Update). Identify the unique (v_o,v_k)-path P in T^i. Suppose v_o is joined to v_m in P. Construct the tree T'^i by replacing the arc (v_o,v_m) of T^i with the arc (v_k,v_l) and proceed to Step 4.

Step 4 (Normalization). Normalize the tree T'^i (the procedure for this is discussed below). This yields a tree T^{i+1}. Identify the set Y^{i+1} of strong vertices of T^{i+1}. Set $i+1 = i$ and go to Step 2. □

To implement Step 2, one has to, at each iteration, scan the set of strong vertices and test whether or not the graph contains an arc directed from a strong to a weak vertex. The efficiency of executing this step certainly depends upon the manner in which the graph is stored and manipulated. We have found that a convenient data structure to achieve this is to represent the digraph D representing the ore-body (referred to as the parent graph) as an adjacency list in a direct-access file. The normalized tree may be represented as a linked-list with arc labels of the form $(\pm,w(a))$.

We now detail the procedures for implementing Steps 3 and 4. Assign to each arc a of T^i a label of the same form as above, namely $(\pm, w(a))$ so as to identify its status and the weight of the branch it supports. In the actual computer implementation T^i may be stored as an undirected graph - all the information we need is contained in the labels. We think of T^i as directed simply for convenience. The procedure for implementing Step 3 is as follows.

Procedure for the update step.

Suppose arc (v_k, v_l) of T^i satisfies the condition in Step 2, namely, $v_k \in Y^i$ and $v_l \notin Y^i$. Let v_k and v_l belong to the branches B_i and B_j supported by the arcs (v_o, v_m) and (v_o, v_n), respectively. Let $P(v_m, v_k)$denote the unique (v_m, v_k)-path in B_i, and $P(v_l, v_n)$ the unique (v_l, v_n)-path in B_j. We then update the tree T^i to T'^i as follows:

(i) The tree T'^i is given by

$$T'^i = T^i + (v_k, v_l) - (v_o, v_m). \tag{2.2}$$

(ii) The only arc labels that change are those of the arc (v_k, v_l) and the arcs on the paths $P(v_m, v_k)$ and $P(v_l, v_o)$. The new arc (v_k, v_l) in T'^i will carry the label given by

$$\text{label } (v_k, v_l) = (-, w(B_i)). \tag{2.3}$$

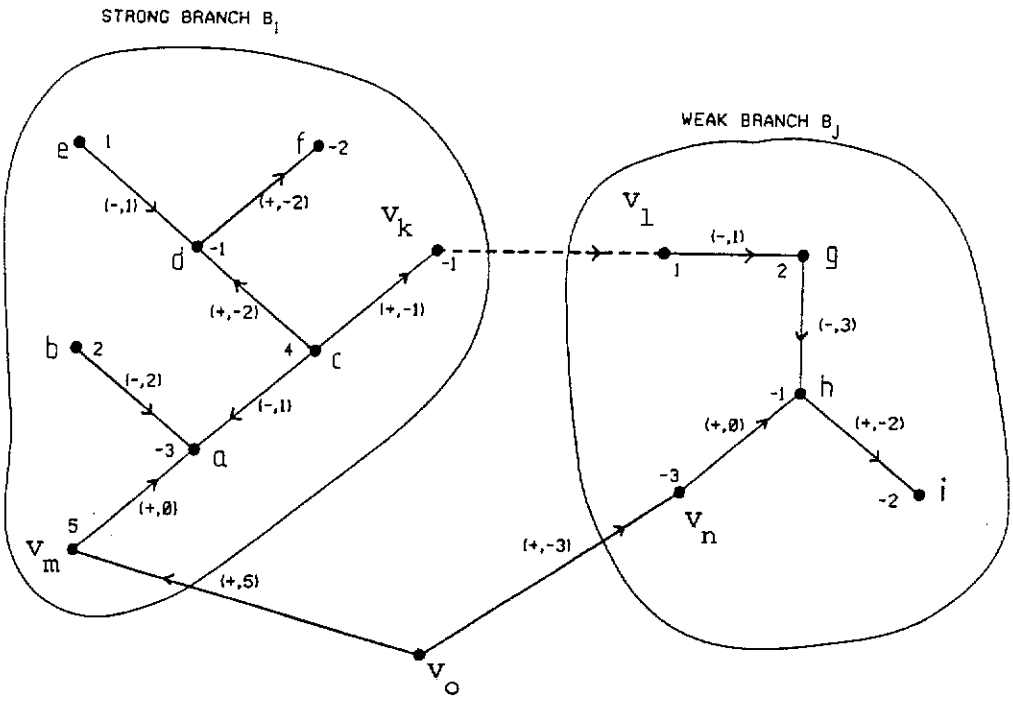
An arc a_i of $P(v_m, v_k)$ in T^i will have its label updated in T'^i as follows:

$$(\pm, w(a_i)) \rightarrow (\mp, w(B_i) - w(a_i)). \tag{2.4}$$

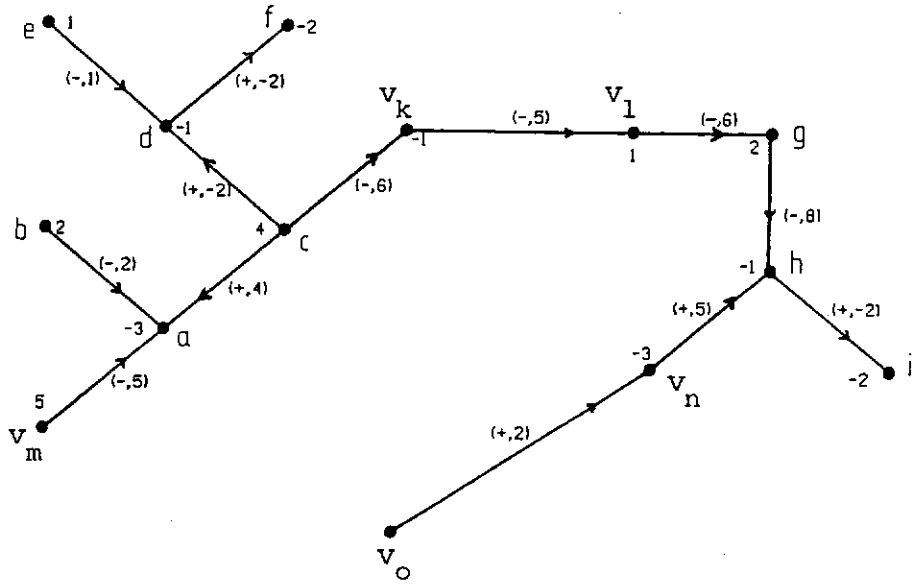
An arc a_j of $P(v_l, v_o)$ has its label updated as follows:

$$(\pm, w(a_j)) \rightarrow (\pm, w(a_j) + w(B_i)). \tag{2.5}$$

The example in Figure 2.3 illustrates the label changes in the update step; the arcs of the paths $P(v_m, v_k)$ and $P(v_l, v_o)$ are indicated in solid lines. Note that the branch of T'^i supported by



(a)



(b)

Figure 2.3 (a) The labels in T^i. (b) The labels in T'^i.

the arc (v_o, v_n) contains strong arcs, namely (c,a) and (v_n,h) which do not have v_o as one of the end vertices. Hence the tree T'^i is not normalized. The procedure for implementing Step 4, the normalization step, follows.

Procedure for normalization of a tree

The normalized tree T^{i+1} is obtained from T'^i by carrying out the following steps:

Step (i) Move along the unique path $P(v_m, v_o)$ and locate the first strong arc $a_s = (v_p, v_q)$. (We will show later, in Lemma 2.3, that the only strong arcs occuring in the Lerchs-Grossman algorithm are p-arcs). If no such arc is present, set $T''^i = T'^i$ (T'^i is already normalized) and go to Step (iv). Otherwise, delete arc a_s and join v_o to its branch root v_q.

Step (ii) For each arc a_j along the path $P(v_p, v_o)$, update its label as follows:

$$(\pm, w(a_j)) \rightarrow (\pm, w(a_j) - w(a_s)) \qquad (2.6)$$

where $w(a_s) = w(B(v_q))$; that is, the weight of the branch $B(v_q)$ with root v_q, in the new tree T''^i.

Step (iii) Locate the next strong arc $a_s = (v_r, v_t)$ on path $P(v_p, v_o)$ in T''^i. If not found, go to Step (iv). Otherwise, delete arc a_s, join v_o to its branch root v_t, set $p=r$ and $q=t$, and go to Step (ii).

Step (iv) Set $T^{i+1} = T''^i$, stop. T^{i+1} is the required normalized tree.

Figure 2.4 gives the result of the normalization step applied to the branch in Figure 2.3(b).

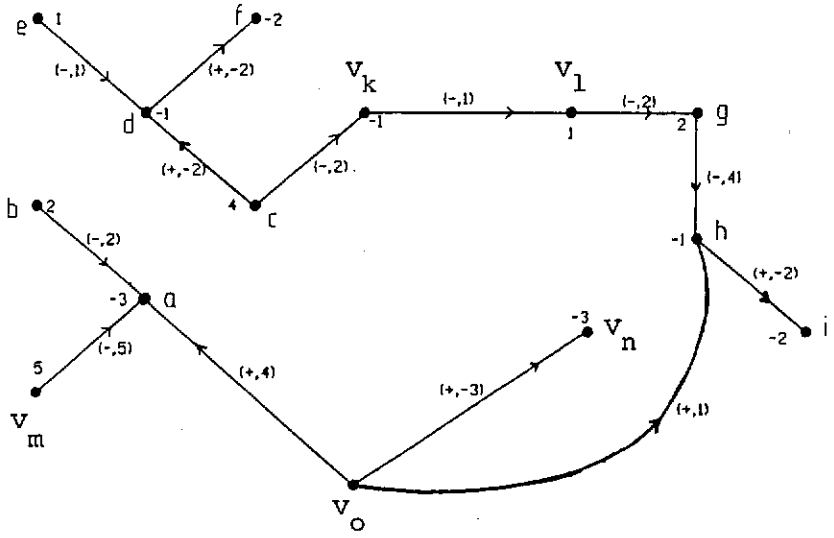


Figure 2.4 Normalization of the branch in Figure 2.3(b).

The following lemma is necessary to prove the convergence of the Lerchs-Grossman algorithm in Theorem 2.3.

Lemma 2.3 *All strong arcs arising in the Lerchs-Grossman algorithm are p-arcs.*

Proof. We adopt the same notation as in the foregoing discussion. Thus, (v_k, v_l) is an arc in D with $v_k \in B_i$, a strong branch rooted at v_m and $v_l \in B_j$, a weak branch rooted at v_n (as in Figure 2.3); B_i and B_j being branches of the normalized tree T^i in the Lerchs-Grossman algorithm. During the update step, the tree T'^i is constructed as in equation (2.2) with only the arcs on the path

$P(v_m, v_o)$ changing their labels. By equation (2.3), (v_k, v_l) enters T'^{i} as an m-arc with positive weight $w(B_i)$, and hence is weak. So, any strong arcs must occur on the paths $P(v_m, v_k)$ and $P(v_l, v_o)$ in T'^{i}.

Suppose a_s is a strong m-arc on path $P(v_m, v_k)$ in T'^{i}. Then its weight $w'(a_s)$ in T'^{i} must be non-positive. But by (2.4), $w'(a_s)$ is given by

$$w'(a_s) = w(B_i) - w(a_s).$$

Thus

$$w(B_i) - w(a_s) \leq 0,$$

and hence

$$w(a_s) \geq w(B_i) > 0.$$

But by (2.4), a_s is a p-arc in T^i with positive weight and hence, is a strong arc. This contradicts the normalization property of T^i. Thus, a strong m-arc cannot occur on $P(v_m, v_k)$.

Suppose a_s is a strong m-arc on the path $P(v_k, v_n)$ in T'^{i}. By (2.5)

$$w'(a_s) = w(B_i) + w(a_s)$$

and since $w'(a_s) \leq 0$,

$$w(a_s) \leq -w(B_i) < 0.$$

Thus a_s is also a strong m-arc in T^i (since a_s does not change status) contradicting its normalization property. Hence the result. □

Theorem 2.3 *The Lerchs-Grossman algorithm converges in a finite number of steps.*

Proof. As the number of trees in a finite graph is finite it suffices to show that no tree can repeat itself in the sequence $T^0, T^1, \ldots, T^n$ of normalized trees generated by the algorithm. We establish this by showing that, at each iteration, either

$$\text{(i)} \quad w(Y^i) > w(Y^{i+1})$$

or else

$$\text{(ii)} \quad w(Y^i) = w(Y^{i+1}) \text{ and } |Y^i| < |Y^{i+1}|.$$

We use the same notation as in the above discussion and lemma. In the Lerchs-Grossman algorithm we first consider the case $T'^i = T^{i+1}$; that is, T'^i is already normalized. Suppose $w(B_j) < 0$, then two possible cases can arise:

(a) Arc (v_o, v_n) is strong in T^{i+1}. Then,

$$w'(v_o, v_n) = w(B_i)+w(B_j) \qquad \text{(follows from (2.5))}$$
$$< w(B_i), \quad \text{since } w(B_j) < 0.$$

Hence, $w(Y^{i+1}) < w(Y^i)$.

(b) Arc (v_o, v_n) is weak in T^{i+1}. Then all vertices in B_i are no longer strong and hence $w(Y^{i+1}) < w(Y^i)$. If $w(B_j) = 0$, then (v_o, v_n) is strong in T^{i+1} and $w'(v_o, v_n) = w(B_i)$. Thus, $w(Y^{i+1}) = w(Y^i)$. But since all the vertices of B_j become strong and all the vertices of B_i remain strong, $|Y^{i+1}| > |Y^i|$.

Suppose now $T^{i+1} \neq T'^i$. Let P denote the unique (v_m, v_o)-path in T'^i. If a_1 is a strong arc in the B_i section of P, then its weight in T'^i is given by

$$w'(a_1) = w(B_i)-w(a_1). \qquad \text{(from (2.4))}$$

By Lemma 2.3, a_1 is a p-arc in T'^i and by (2.4), an m-arc in T^i, so $w(a_1) < 0$. Thus, $w'(a_1) < w(B_i)$ and condition (i) holds. If a_1 is a strong arc in the B_j section of P, then by (2.5),

$$w'(a_1) = w(B_i)+w(a_1).$$

By Lemma 2.3 and 2.5, a_1 is a p-arc in T^i, so $w(a_1) \leq 0$. This implies

$$w'(a_1) \leq w(B_i).$$

If $w'(a_1) < w(B_i)$, then condition (i) holds. On the other hand, if $w'(a_1) = w(B_i)$, then $w(Y^i) = w(Y^{i+1})$ and since $a_1 \in B_j$, every vertex of B_i is strong. Hence, since a_1 cannot be the arc (v_k, v_1) (as the latter is an m-arc), we must have $|Y^i| < |Y^{i+1}|$. □

We discuss the complexity of the Lerchs-Grossman algorithm in Chapter 3 where we prove that the algorithm is equivalent (step by step) to a modified version of the dual-simplex algorithm. However, it is interesting, at this stage, to note the time complexity of Step 2 of the algorithm; it is by far the most taxing in computer time. If D has n vertices and m arcs then there can be a maximum of n strong branches in a tree T at any time in the algorithm. Since for each strong branch in T there is a maximum of m arcs to be tested for connection to a weak branch, there can be a total of $O(mn)$ operations at this step. If follows that, not only do we need to have a method of generating the arcs of D to represent a specific block model with specific wall slope restrictions, but we also require a method which generates the minimum number of arcs in D, necessary to achieve the required restrictions. This is the topic of our next section.

2.4 The Minimum Search Pattern

Since the generation of the arcs in D representing the block model of an open pit mine is dependent upon physical measurement such as, length, angle and location, it is instructive to introduce the following notation and terminology. To identify blocks in the

ore-body we make use of the grid system shown in Figure 2.5; the coordinates $(\ell, i, j,)$ signifying a block whose centre is on level ℓ, iB units to the east and jW units to the north, where B and W are the block dimensions along the i- and j-axes, respectively. A **level** is the set of blocks in the mine at equal height above sea level. The top of a level is referred to as **bench** in the mining industry. The benches are separated by a **bench height** of H units (the height of a block).

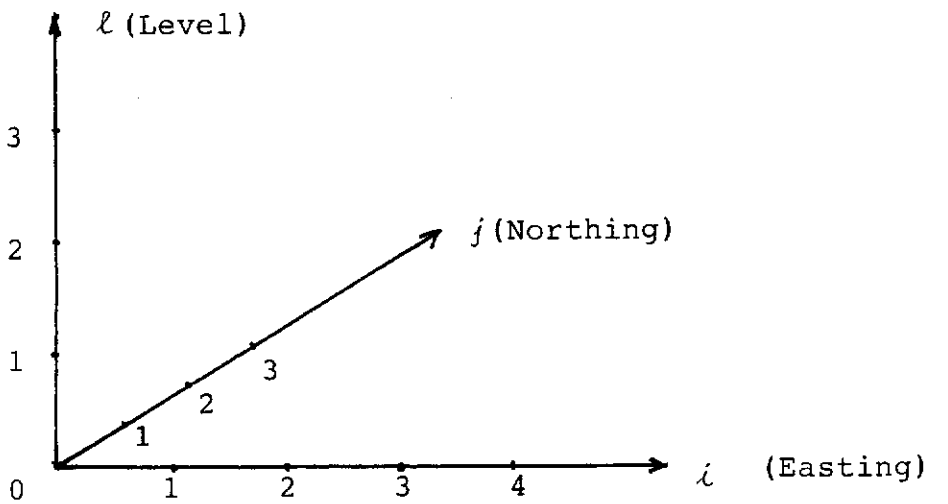


Figure 2.5 Grid system for a block model.

Consider the set R given by

$$R = \{(\theta_i, \alpha_i) : 1 \leq i \leq m\} \tag{2.7}$$

of azimuth and dip pairs and a vertex v representing (the centre of) a block in the block model of the ore-body. Suppose we have a set of vertices $S_v = \{v_1, v_2, \ldots, v_k\}$ above vertex v forming a specific geometric pattern G when joined to v. For each $v_i \in S_v$ we find a corresponding set S_{v_i} forming the same geometric pattern G when joined to v_i. Continue this procedure (referred to as *the*

application of a search pattern to a vertex) until the top level in the model is reached. Let $\mathcal{S}$ be the union of all the sets of vertices obtained by this procedure. If the removal of all the blocks in the model represented by the vertices in $\mathcal{S}$ results in a conical shaped pit with v as its apex and maximum slope angle of its wall at azimuth θ_i equal to α_i, for each i, then S_v is called a **search pattern** for v. A search pattern, relative to the vertex at the origin, with a minimum number of elements (that is, no redundant arcs are generated in the graph when the search pattern is repeatedly applied in the model) is called a **minimum search pattern** (MSP).

Prior to the work by Caccetta and Giannini (1988b), only two specific search patterns had been reported in the literature, namely the so-called "knight's move" pattern described in Lipkewich and Borgman (1969), and the one described in Chen (1976). The question of generality and, in particular, optimality had not been addressed.

The knights move pattern is a search pattern for a cubic block model of a pit with wall slope restriction of 45° throughout. This pattern generates a digraph with each vertex having out-degree 13 and for a vertex v, the set S_v includes five vertices from the level immediately above v and eight vertices from the second level above v; see Figure 2.8. The success of the knight's move pattern is dependent (as we shall see) on the fact it is the minimum search pattern for the block model described above; moreover, the 13 elements in the MSP do not extend past level 2 (see Figure 2.8). One should observe however, that the actual wall angles generated (that is those formed between the line joining the centre of a block on the wall to the origin and the horizontal) by repeatedly applying

the knight's move pattern, range from 41.8° to 45°. This range of angles is understandable since we are dealing with discrete quantities, namely blocks, to approximate the wall limits. In our development of a general algorithm for producing a minimum search pattern, we basically make use of the properties observed in the knight's move pattern and extend the idea further.

2.4.1 The MSP Algorithm

There are two parameters ε_1 and ε_2 incorporated in our algorithm; the mechanics behind these will be discussed in Section 2.4.2. The ultimate purpose of these parameters, which we may call **tolerance parameters**, is two-fold:

(i) If the desired wall angle is α degrees within the pit, say at azimuth θ, then the wall angle will be held approximately within the range $[\alpha-\varepsilon_2, \alpha+\varepsilon_1]$.

(ii) The highest level reached by the MSP is kept as low as possible. This feature is desirable since the number of levels is very restricted, particularly for a smaller pit.

The following is the input required by the MSP algorithm:

1. The number of levels the search pattern is required for (N).
2. The set of azimuth and dip pairs R defined in (2.7).
3. The block dimensions $(H \times B \times W)$.
4. The tolerance parameters (ε_1 and ε_2).

The output from the MSP algorithm is the set S of coordinates (ℓ, i, j) of the vertices in the required minimum search pattern. The minimum search pattern S is constructed by adding elements according to the following procedure. Vertex O is the vertex representing the block with centre at $(0,0,0)$.

Algorithm

Step 1 (Initialization). Inditialize S by adding the coordinates of the vertex directly above vertex O and tag it. Thus currently $S = \{(1,0,0)\}$. Set $\ell = 1$ and go to Step 3.

Step 2 (Application of search pattern). Apply the current search pattern S (see Remark 2.1) to each of the tagged vertices on levels 1 to $\ell-1$, inclusive.

Step 3 (Slope violation check). Check for slope violation (see Section 2.4.2) for each of the untagged vertices on level ℓ. If an untagged vertex with coordinates (ℓ, i, j) violates a slope restriction, add (ℓ, i, j) to S, and tag the vertex.

Step 4 (Stopping criteria). Increment ℓ by 1 and go to Step 2. Stop when ℓ exceeds N. □

Remark 2.1 As discussed earlier, the phrase "applying a search pattern S to a vertex v" means identifying those vertices forming the same geometric pattern G relative to v as the one which the vertices of S form relative to O. This is achieved by adding the coordinates of each of the vertices in S to those of v. Thus if v has coordinates (ℓ_0, i_0, j_0) and a vertex in S has coordinates (ℓ_1, i_1, j_1) then $(\ell_0+\ell_1, i_0+i_1, j_0+j_1)$ are the coordinates of the corresponding vertex in the geometric pattern G relative to v.

2.4.2 Slope Violation Criteria

We first present the model in the MSP algorithm used to define the slope violation criteria, and then explain the rationale behind

it. Suppose that at azimuth β, the wall angle restriction at the origin (in Figure 2.5) is α. Figure 2.6 shows the x-y axes on the vertical plane Q, through the origin, making an angle β to the right of the j-axis (Northing); the y-axis coinciding with the ℓ-axis and origin coinciding with O in Figure 2.5. $\overline{AB}$ is the line segment, of length H, representing the height of a block in the block model; OD, OE and OF are rays making angles $\alpha-\varepsilon_2$, α and $\alpha+\varepsilon_1$ (for some positive parameters ε_1 and ε_2), respectively. The line segment $\overline{AG}$ (through B) is perpendicular to the x-axis. We wish to find the length of $\overline{BG}$ such that the range $[\alpha-\varepsilon_2, \alpha+\varepsilon_1]$ is spanned by a single block height $\overline{AB}$ as shown.

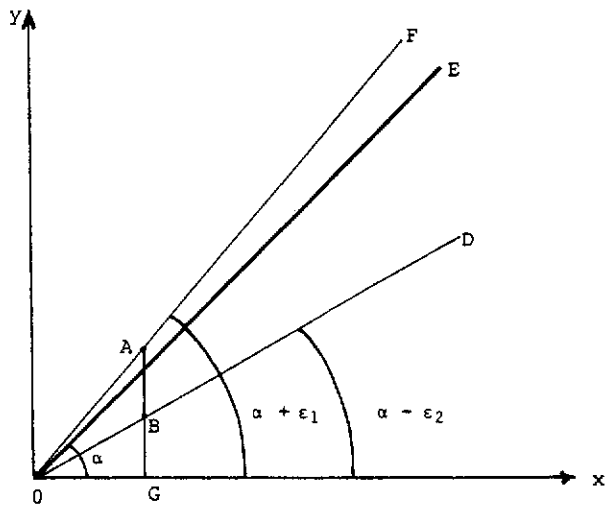


Figure 2.6 x-y axes on plane Q for calculation of n_c.

Denoting the length of the line segment $\overline{AG}$ as AG etc, we have,

$$tan(\alpha+\varepsilon_1) = \frac{AG}{OG} \tag{2.8}$$

and

$$tan(\alpha-\varepsilon_2) = \frac{BG}{OG} \ . \tag{2.9}$$

Eliminating OG from equations (2.8) and (2.9) and using the relation

$$AG = AB + BG$$

$$= H + BG,$$

we obtain

$$BG = \frac{H \tan(\alpha-\varepsilon_2)}{\tan(\alpha+\varepsilon_1) - \tan(\alpha-\varepsilon_2)} . \tag{2.10}$$

We define the **critical level** n_c by the equation

$$n_c = \lceil BG/H \rceil$$

$$= \left\lceil \frac{\tan(\alpha-\varepsilon_2)}{\tan(\alpha+\varepsilon_1) - \tan(\alpha-\varepsilon_2)} \right\rceil \tag{2.11}$$

and a **critical angle** α_ℓ, for level $\ell > n_c$, as follows. First we find the **slope violation line** L, passing through the point $(0,-H)$ and having slope $\tan(\alpha+\varepsilon_1)$ (see Figure 2.7). The equation for L is

$$y = x \tan(\alpha+\varepsilon_1) - H. \tag{2.12}$$

We then find the x-coordinate x_p of the point of intersection P of L with the line

$$y = \ell H . \tag{2.13}$$

From equations (2.12) and (2.13), we have

$$\ell H = x_p \tan(\alpha+\varepsilon_1) - H .$$

hence

$$x_p = \frac{(\ell+1)H}{\tan(\alpha+\varepsilon_1)} . \tag{2.14}$$

Then, α_ℓ is the angle which the line OP makes with the x-axis. Thus

$$\tan \alpha_\ell = \frac{\ell H}{x_p}$$

$$= \frac{\ell}{\ell+1} \tan(\alpha+\varepsilon_1). \qquad \text{(from (2.14))}$$

Hence

$$\alpha_\ell = \arctan\left\{\frac{\ell}{\ell+1} \tan(\alpha+\varepsilon_1)\right\} . \tag{2.15}$$

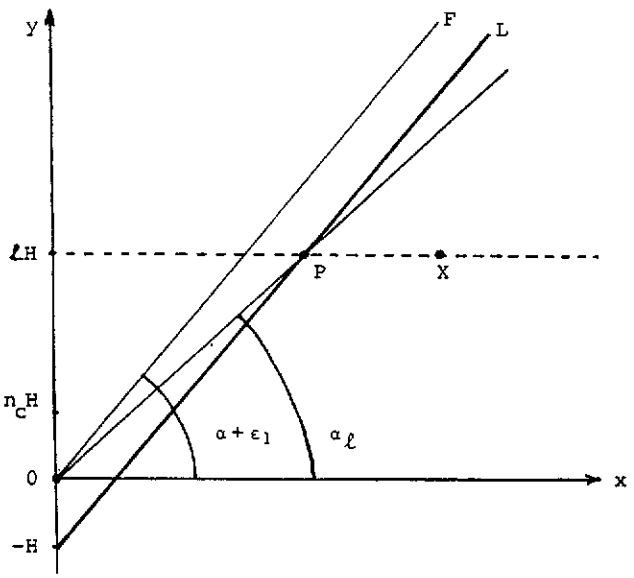


Figure 2.7 The determination of the slope violation line L on plane Q.

Now consider a vertex $X \in V(D)$ which is on plane Q and level ℓ. Let θ be the angle the line joining the origin to X makes with the x-axis (see Figure 2.7). Then the **slope violation criteria** reads as follows:

A block in the model with centre at X is said to violate the wall slope restriction at azimuth β if either

$$\ell < n_c \quad \text{and} \quad \theta \geq \alpha-\varepsilon_2$$

or

$$\ell \geq n_c \quad \text{and} \quad \theta > \alpha_\ell \, .$$

From equations (2.11) and (2.15) we note that

(i) As ε_1 and $\varepsilon_2 \to 0$, $n_c \to \infty$ (i.e. $n_c \to N$).

(ii) As $\ell \to \infty$ (and so $\ell \to N$), $\alpha_\ell \to \alpha+\varepsilon_1$.

(i) and (ii) are desirable practical requirements.

The rationale behind the construction of the slope violation model described above is that when considering a vertex (or block) close to the origin, we need to use the strictest angle $\alpha-\varepsilon_2$ as maximum wall slope allowable. This is because the wall angles are quite sensitive to block displacement at such position within the block model and using the strictest wall angle earlier (i.e. close to the origin) ensures that a large number of vertices further up are not required. The wall angles being less sensitive further away from the origin is reflected in the model, whereby, the critical angle α_ℓ approaches the largest allowable angle $\alpha+\varepsilon_1$.

We should note, in the above discussion, that the slope violation criteria are based on knowing what the wall angle restriction is at a particular azimuth. Since, in practical applications, the maximum wall angles are specified at a finite (≤ 8) set of azimuths, a linear spline through these values, is used in the MSP algorithm to determine the desired wall angles at azimuths not in this set.

2.4.3 Proof and Implementation of the MSP Algorithm

We now show that the set S generated by the MSP algorithm is a unique minimum search pattern.

Theorem 2.4 *The set S as constructed by the MSP algorithm is the minimum search pattern for the given wall slope restrictions.*

Proof. Suppose S is not minimum and let T be a minimum search pattern. Order the elements in S and T in increasing order of the

ℓ-component. Now it is obvious that (1,0,0) must be a member of both S and T. Suppose (ℓ, i, j) is the first element such that $(\ell, i, j) \in S$ but $(\ell, i, j) \notin T$. From the algorithm (ℓ, i, j) cannot be obtained by repeatedly applying the search pattern formed by the elements before (ℓ, i, j) and hence, by the elements in T. Since the vertex with coordinates (ℓ, i, j) violates the wall slope restriction and is not represented in T, T cannot be a minimum search pattern. This contradicts the assumption that S is not minimum. Hence, S is the minimum search pattern. □

The MSP algorithm has been coded in FORTRAN-77 and implemented on the VAX-785 computer. In this implementation a two-dimensional integer array A is used to represent the 'plan view' of the conical-shaped pit with vertex at the origin formed by applying S repeatedly to the required level N. The value $A(j, i)$ represents the level at which a block with easting iB and northing jW is removed. A value of zero signifies that block (i, j) is not removed at any level. Obviously, if a block is removed at a particular level, then all blocks above it are also removed.

The output given by the software consists of:

(i) The minimum search pattern S in the form of a set of coordinates (ℓ, i, j) of blocks; and

(ii) The array A. This array identifies the blocks which must be removed to expose the block with centre at the origin.

For example, the knight's move search pattern for the cubic block model is the set S given by:

$$S = \{(1,0,0),\ (1,1,0),\ (1,0,1),\ (1,-1,0),\ (1,0,-1),$$
$$(2,-2,-1),\ (2,-2,1,)\ (2,-1,-2),\ (2,-1,2),\ (2,1,-2),$$
$$(2,1,2),\ (2,2,-1),\ (2,2,1)\}.$$

The array A formed by repeated application of S to level 15 is shown in Figure 2.8.

Array A

```
 0  0  0 21 20 20 19 18 18 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 18 18 19 20 20 21  0  0  0
 0  0 21 20 20 19 18 18 17 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 17 18 18 19 20 20 21  0  0
 0 21 20 20 19 18 18 17 16 16 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 16 16 17 18 18 19 20 20 21  0
21 20 20 19 18 18 17 16 16 15 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 15 16 16 17 18 18 19 20 20 21
20 20 19 18 18 17 16 16 15 14 14 13 13 13 13 13 13 13 13 13 13 13 13 13 13 14 14 15 16 16 17 18 18 19 20 20
20 19 18 18 17 16 16 15 14 14 13 12 12 12 12 12 12 12 12 12 12 12 12 12 12 13 14 14 15 16 16 17 18 18 19 20
19 18 18 17 16 16 15 14 14 13 12 12 11 11 11 11 11 11 11 11 11 11 11 11 11 12 12 13 14 14 15 16 16 17 18 18 19
18 18 17 16 16 15 14 14 13 12 12 11 10 10 10 10 10 10 10 10 10 10 10 10 10 11 12 12 13 14 14 15 16 16 17 18 18
18 17 16 16 15 14 14 13 12 12 11 10 10  9  9  9  9  9  9  9  9  9  9 10 10 11 12 12 13 14 14 15 16 16 17 18
17 16 16 15 14 14 13 12 12 11 10 10  9  8  8  8  8  8  8  8  8  8  8  9 10 10 11 12 12 13 14 14 15 16 16 17
17 16 15 14 14 13 12 12 11 10 10  9  8  8  7  7  7  7  7  7  7  8  8  9 10 10 11 12 12 13 14 14 15 16 17
17 16 15 14 13 12 12 11 10 10  9  8  8  7  6  6  6  6  6  6  6  7  8  8  9 10 10 11 12 12 13 14 15 16 17
17 16 15 14 13 12 11 10 10  9  8  8  7  6  6  5  5  5  5  5  6  6  7  8  8  9 10 10 11 12 13 14 15 16 17
17 16 15 14 13 12 11 10  9  8  8  7  6  6  5  4  4  4  4  4  5  6  6  7  8  8  9 10 11 12 13 14 15 16 17
17 16 15 14 13 12 11 10  9  8  7  6  6  5  4  4  3  3  3  4  4  5  6  6  7  8  9 10 11 12 13 14 15 16 17
17 16 15 14 13 12 11 10  9  8  7  6  5  4  4  3  2  2  2  3  4  4  5  6  7  8  9 10 11 12 13 14 15 16 17
17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  2  1  2  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  1  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  2  1  2  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
17 16 15 14 13 12 11 10  9  8  7  6  5  4  4  3  2  2  2  3  4  4  5  6  7  8  9 10 11 12 13 14 15 16 17
17 16 15 14 13 12 11 10  9  8  7  6  6  5  4  4  3  3  3  4  4  5  6  6  7  8  9 10 11 12 13 14 15 16 17
17 16 15 14 13 12 11 10  9  8  8  7  6  6  5  4  4  4  4  4  5  6  6  7  8  8  9 10 11 12 13 14 15 16 17
17 16 15 14 13 12 11 10 10  9  8  8  7  6  6  5  5  5  5  5  6  6  7  8  8  9 10 10 11 12 13 14 15 16 17
17 16 15 14 13 12 12 11 10 10  9  8  8  7  6  6  6  6  6  6  7  8  8  9 10 10 11 12 12 13 14 15 16 17
17 16 15 14 14 13 12 12 11 10 10  9  8  8  7  7  7  7  7  7  7  8  8  9 10 10 11 12 12 13 14 14 15 16 17
17 16 16 15 14 14 13 12 12 11 10 10  9  8  8  8  8  8  8  8  8  8  8  9 10 10 11 12 12 13 14 14 15 16 16 17
18 17 16 16 15 14 14 13 12 12 11 10 10  9  9  9  9  9  9  9  9  9  9 10 10 11 12 12 13 14 14 15 16 16 17 18
18 18 17 16 16 15 14 14 13 12 12 11 10 10 10 10 10 10 10 10 10 10 10 10 10 11 12 12 13 14 14 15 16 16 17 18 18
19 18 18 17 16 16 15 14 14 13 12 12 11 11 11 11 11 11 11 11 11 11 11 11 11 12 12 13 14 14 15 16 16 17 18 18 19
20 19 18 18 17 16 16 15 14 14 13 12 12 12 12 12 12 12 12 12 12 12 12 12 12 13 14 14 15 16 16 17 18 18 19 20
20 20 19 18 18 17 16 16 15 14 14 13 13 13 13 13 13 13 13 13 13 13 13 13 13 14 14 15 16 16 17 18 18 19 20 20
21 20 20 19 18 18 17 16 16 15 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 15 16 16 17 18 18 19 20 20 21
 0 21 20 20 19 18 18 17 16 16 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 16 16 17 18 18 19 20 20 21  0
 0  0 21 20 20 19 18 18 17 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 17 18 18 19 20 20 21  0  0
 0  0  0 21 20 20 19 18 18 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 18 18 19 20 20 21  0  0  0
```

```
M S P
1  0  0
1 -1  0
1  0 -1
1  0  1
1  1  0
2 -2 -1
2 -2  1
2 -1 -2
2 -1  2
2  1 -2
2  1  2
2  2 -1
2  2  1
```

Figure 2.8 Software output associated with the knight's move pattern.

2.4.4 Test Cases and Computational Results

The following are the results obtained from the application of the MSP software to various cases of interest using the cubic block model and the block model with block dimensions $H \times B \times W$ in the ratio 1:2:2. In all cases we determine the MSP to 20 levels using $\varepsilon_1 = \varepsilon_2 = \varepsilon$ for various values of ε. Particular note should be given to the influence of the tolerance parameters on the reduction in the size of, and in the maximum level in, the MSP. Note also that 'stable patterns' (that is no changes in the MSP over a certain range of tolerance values) are produced in Cases 1 and 2 for both models considered.

The Cubic Block Model

Case 1. 45° wall angle restriction throughout:

Tolerance parameter ε (degrees)	No. of elements (arcs) in MSP	Maximum level in MSP
0.5	101	19
1	73	13
2	29	6
3.5	13*	2
4	13*	2
4.5	13*	2

*Knight's move pattern (output given in Figure 2.8)

Case 2. 40° wall angle throughout:

Tolerance parameter ε (degrees)	No. of elements (arcs) in MSP	Maximum level in MSP
1	169	18
2	65	7
3	57	5
3.5	49	4
4	41	4
4.5	41	4

Case 3. 50° wall angle throughout:

Tolerance parameter ε (degrees)	No. of elements (arcs) in MSP	Maximum level in MSP
0.5	181	20
1	169	14
2	109	9
2.5	89	7
3	81	7
3.5	69	6
4	61	6
4.5	49	5

Case 4. Azimuth-dip pairs (0,45), (45,40), (90,30), (135,40), (180,45), (270,45):

Tolerance parameter ε (degrees)	No. of elements (arcs) in MSP	Maximum level in MSP
1	145	14
1.5	126	11
2	32	7
3	35	5
3.5*	22	4
4	20	4
4.5	18	3
5	12	2

*Output for this case is given in Figure 2.9.

Array A

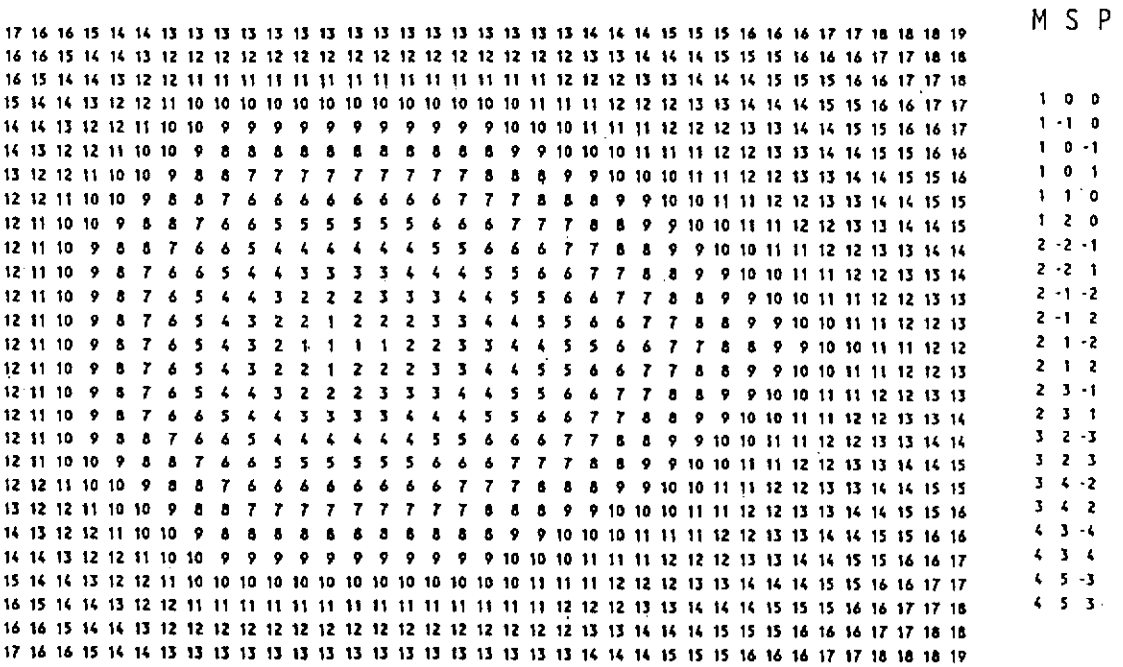


Figure 2.9 Software output (reduced for convenience) for Case 4 of the cubic block model.

Model with Block Dimensions 1:2:2

Case 1. 30° wall angle restriction throughout:

Tolerance parameter ε (degrees)	No. of elements (arcs) in MSP	Maximum level in MSP
0	205	20
1	129	11
2	121	9
3	97	7
3.5	9	3
4	9	3
5	9	3

Case 2. 45° wall angle restriction throughout:

Tolerance parameter ε (degrees)	No. of elements (arcs) in MSP	Maximum level in MSP
0	37	19
1	37	13
2	25	7
3	17	9
4	9	3
4.5	9	3
5	9	3

We point out here that the MSP with 9 elements in each of the cases 1 and 2 are dissimilar.

Case 3. Azimuth-dip pairs (0,45), (45,45), (90,40), (180,40), (225,45), (270,45), (315,45):

Tolerance parameter ε (degrees)	No. of elements (arcs) in MSP	Maximum level in MSP
0	71	19
1	74	18
2	33	7
3	25	9
3.5	22	6
4	18	6
4.5	16	5

The azimuth-dip pairs considered here are identical to those considered by Chen (1976). We note that:

(i) For 0° tolerance the number of arcs obtained in our MSP is comparable to that given by Chen (73 arcs) who does not address the question of optimality.

(ii) For a 2° tolerance the number of arcs in the MSP more than halved with the maximum level dropping from 19 to 7. Moreover, the range of angles within the pit for the 0° case is not much better than that for the 2° case. Further savings in the computation is achieved with a

4.5° tolerance where there are only 16 arcs in the MSP as compared to 71 in the 0° case; the maximum level dropping from 19 to 5.

The test results above, show that the MSP algorithm is extremely effective and that much larger search patterns unnecessarily result if the tolerance parameters ε_1 and ε_2 are not incorporated in the slope violation model described in Section 2.4.2. Thus, considerable savings in computation may be achieved in the application of graph theoretic techniques to the open pit limit problem by selecting suitable tolerance parameters in the MSP algorithm. We note that tolerance parameters of approximately 10% of the required wall slope angles yield extremely practical and efficient results.

2.5 Implementation of The Lerchs-Grossman Algorithm

Since the publication of the Lerchs-Grossman algorithm, a number of authors (Chen 1976; Lipkewich and Borgman 1969; Robinson 1975; Rychkun and Chen 1979) have considered the problems associated with its implementation. Space and time complexities (discussed in detail in Chapter 3) are the causes of such problems. Ore-bodies, for which optimum pit designs are sought, may be as small as 5,000 blocks and as large as half a million blocks or more. Efficient storage and manipulation of the large graphs, resulting from the modelling of such ore-bodies, are therefore, essential. Here, we present some strategies of implementing the Lerchs-Grossman algorithm, in graph-theoretic sense, which have been adopted by Lipkewich and Borgman (1969) and Chen (1976), and discuss extensions and modifications we propose for these methods.

It is quite obvious that the optimum pit outline for an

ore-body mined to a specific level is necessarily contained within the optimum pit outline if the ore-body is mined to a lower level. Lipkewich and Borgman (1969) make use of this fact to save computer storage. Their method is to introduce the vertices of the graph of the ore-body (referred to as the parent graph) one level at a time, starting from the top level. Initially, the normalized tree consists of the vertices of the top level connected to the root vertex v_o. The strong branches, that is the positive-valued vertices (and incident arcs), are removed. The vertices which are removed are, necessarily, in the optimum closure. Each time the vertices of a level are introduced, they are connected by p-arcs to v_o, the Lerchs-Grossman algorithm is applied to the new tree and the resulting strong vertices (and incident arcs) removed. This procedure continues until the final level is processed. After processing each level, the data is compacted to allow for the removal of the strong vertices and incident arcs. At that time Lipkewich and Borgman (1969) did not have the capability of handling variable wall slope restrictions; they applied their method to the simple 45° wall slope restriction in a three-dimensional cubic block model.

Chen (1976) improved this technique by adopting a search pattern idea to account for general wall slope restrictions and a method (referred to as 'prepass' in his paper) of removing 'unwanted' vertices from the input parent graph before processing. The idea of the prepass was to identify all the (profitable) vertices with positive weights in a given level and all the vertices obtained by applying the relevant search pattern to each of them. Thus a prepass, throughout the ore-body, would exclude all the

vertices (unprofitable or outside of the pit wall constraints) before the pit optimization technique was applied.

We extend these ideas further by applying the following concepts and procedures:

(i) The minimum search pattern (MSP) which, basically, removes any redundant arcs from the parent graph.

(ii) The bounding algorithm which is a far more sophisticated prepass concept. This algorithm, presented and discussed in Chapter 4, produces a much tighter bound for the optimum pit outline and hence further reduces the size of the parent graph.

(iii) The use of the arc labelling technique, discussed in Section 2.2, for ease in manipulating the graph.

(iv) The efficiency of level by level optimization may be enhanced by storing the strong vertices (down to a particular level) in descending order of weight and building up the branch of the strongest vertex first until it becomes weak or is removed. Repeat with the next strongest, etc; the idea being that the strongest branch would more likely support a weak one than a less stronger one would, and hence, a better chance of resulting with a strong branch which is then removed from the normalized tree.

CHAPTER 3

LINEAR PROGRAMMING MODELS

As linear programming (LP) is one of the most used techniques by Operations Research analysts, it is natural to investigate whether it could be applied to the pit limit problem. In this chapter we look at an LP model of the pit limit problem and a simplified version proposed by Meyer (1969). We show that the Lerchs-Grossman (L-G) graph-theoretic algorithm is equivalent (step-by-step) to solving, using a modified dual-simplex technique, the dual of the LP formulation of the problem. With this equivalence, we provide an alternate non graph-theoretical method of solving the pit limit problem, and provide an estimate of the time complexity of the Lerchs-Grossman algorithm.

3.1 The LP Formulation of the Pit Limit Problem

Consider a block model with n blocks $b_1, b_2, \ldots, b_n$. Let D be the directed graph associated with the block model with vertices v_i, $i = 1$ to n, representing the blocks b_i, $i = 1$ to n, respectively, and arcs a_j, $j = 1$ to m, representing the mining restrictions. The **node-arc incidence matrix** of D is the $n \times m$ matrix $A = (a_{ij})$ defined by

$$a_{ij} = \begin{cases} 1, & \text{if arc } a_j \text{ is incident out of } v_i, \\ -1, & \text{if arc } a_j \text{ is incident into } v_i, \\ 0, & \text{otherwise.} \end{cases}$$

Note that every column of A has exactly two non-zero entries 1 and -1. Let

$$x_i = \begin{cases} 1, & \text{if block } b_i \text{ is mined} \\ 0, & \text{otherwise.} \end{cases}$$

If the mining of block b_i requires the removal of block b_j (i.e. D contains the arc (v_i,v_j), then we may express this restriction by the inequality

$$x_i - x_j \leq 0 \ .$$

Thus, the mining restrictions may be expressed as a set of linear constraints. In fact, the pit limit problem may be stated as follows:

$$\text{maximize} \qquad f = \sum_{i=1}^{n} m_i x_i$$

subject to

$$A^T x \leq 0$$

and

$$x_i = 0,1 \quad \text{for} \quad 1 \leq i \leq n \ ,$$

where m_i is the net profit for block b_i, A is the node-arc incidence matrix of D and x is the $n \times 1$ matrix with components x_i. Since the matrix A is totally unimodular (see Papadimitriou and Steiglitz (1982, p318), we may replace the restriction $x_i = 0,1$ by $0 \leq x_i \leq 1$ for each i. Hence the LP formulation of the pit limit problem now reads as follows.

Problem I

$$\text{Maximize} \qquad f = \sum_{i=1}^{n} m_i x_i$$

subject to

$$A^T x \leq 0 \ ,$$

$$x \leq 1$$

and

$$x \geq 0 \ ,$$

where 1 and 0 are considered as $n \times 1$ matrices with elements all

equal to *1* and all equal to *0*, respectively.

Despite the simplicity of the LP formulation, the solution of such a model is not computationally feasible for most ore-bodies because of the dimensions of the matrix *A* (both *m* and *n* may be of order 10^6 or more). Further, the use of sparse matrix techniques would not result in sufficient computational savings to make the method feasible. If one is to apply LP, in this form, then the problem size must be reduced. Myer (1969) attempts to simplify the model by modifying it as follows. Consider an ore-body model having *m* sections each consisting of *n* columns with a square base of unit area. Let x_{ij} denote the depth to which column j, in section i, is mined. Then, assuming a constant wall slope angle restriction of α, the mining restrictions may be expressed, linearly, as

$$|x_{ij} - x_{i,j-1}| \leq \tan \alpha \ ,$$

$$|x_{ij} - x_{i-1,j}| \leq \tan \alpha \ ,$$

$$|x_{ij} - x_{i-1,j-1}| \leq \sqrt{2} \tan \alpha \ ,$$

and

$$|x_{ij} - x_{i+1,j-1}| \leq \sqrt{2} \tan \alpha \ ,$$

with the i's and j's suitably restricted. The last two restrictions above did not appear in Meyer's formulation, but are necessary to satisfy the wall slope restrictions diagonally. We note that in this model there are considerably less variables and constraints than in the block model. Letting $p_{ij}(x_{ij})$ denote the net profit resulting from mining column (i,j) to depth x_{ij}, the objective function to be maximized becomes

$$\sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}(x_{ij}) \ .$$

This function is generally non-linear in x_{ij}. Making a number of

assumptions and using the ideas of separable programming, Meyer linearizes the objective function. Unfortunately, his assumptions imply that costs, revenue and geology do not vary with depth, thus severely limiting the practical use of his model. Additionally, there is a problem when the ore-vein does not extend to the surface.

The effect of the uniformity assumption of Meyer's may be lessened by using the available block values to fit a piecewise-linear function to approximate the column profit functions p_{ij}. However, we will not pursue this line of thought, rather, we consider the dual of Problem I and show that the Lerchs-Grossman algorithm is nothing but a more systematic linear programming technique of solving it than the standard dual-simplex method. Treatment of the dual-simplex method may be found in various Linear Programming texts, such as, Phillips, Ravindran and Solberg (1976), Murty (1983) and Calvert and Voxman (1989).

3.2 The Dual of Problem I

Introducing slack variables, Problem I may be written as follows.

$$\text{Maximize} \qquad f = \sum_{i=1}^{n} m_i x_i$$

subject to

$$A^T x + u = 0 \,, \tag{3.1}$$

$$x + \tilde{u} = 1 \,, \tag{3.2}$$

and

$$x, u, \tilde{u} \geq 0 \,,$$

where $u = (u_i)$ and $\tilde{u} = (\tilde{u}_i)$ are matrices of order $m \times 1$ and $n \times 1$, respectively, consisting of the slack variables required for the equality constraints. We write the dual of the above problem by

defining the variables

$$\pi = (\pi_1, \pi_2, \ldots, \pi_m)^T$$

and

$$y = (y_1, y_2, \ldots, y_n)^T$$

associated with the constraints (3.1) and (3.2), respectively, as follows.

$$\text{Minimize} \qquad g = \sum_{i=1}^{n} y_i$$

subject to

$$A\pi + y \geq M$$

and

$$\pi, y \geq 0 \ ,$$

where $M = (m_i)$ is the $n \times 1$ matrix consisting of the block profit values m_i. Introducing slack variables, the dual of Problem I may be written as follows.

Problem II

$$\text{Minimize} \qquad g = \sum_{i=1}^{n} y_i$$

subject to

$$A\pi + y - s = M$$

and

$$\pi, y, s \geq 0 \ ,$$

where $s = (s_i)$ is the $n \times 1$ matrix consisting of the slack variables s_i required for the equality constraints.

The complementary slackness property states that, at the optimum, we have

$$u_i \pi_i = 0 , \qquad \text{for} \quad 1 \le i \le m ,$$

$$\tilde{u}_i y_i = 0 , \qquad \text{for} \quad 1 \le i \le n , \tag{3.3}$$

and $$x_i s_i = 0 , \qquad \text{for} \quad 1 \le i \le n .$$

At this point, it is interesting to make the following observations:

(i) Suppose $y^0 = 0$ is a feasible solution of Problem II, giving

$$g_{opt} = 0 = f_{opt} .$$

Since y_i is the 'shadow price' on resource constraint i in Problem I, $f_{opt} = \sum_{i=1}^{n} y_i^0$ will not change for a change in the level of resource i from its value of 1. Thus, if this value is made less than 1 for each i, the optimum solution for Problem I is given by

$$x_i = 0 , \qquad \text{for all } i , \quad 1 \le i \le n ,$$

with $f_{opt} = 0$. So if $y = 0$ is a feasible solution for the dual problem, it is also optimum with the value of the objective function being zero. This dual solution, which corresponds to not mining any blocks ($x_i = 0$, for all i), is the trivial case; it suggests that there is no viable pit outline for the model.

(ii) If the optimum value of the objective function for the dual is non-zero, then the solution must have $y_j \neq 0$, for some j. By (3.3), $\tilde{u}_j = 0$; which means that $x_j = 1$ (the j-th constraint is a strict equality in Problem I) in the primal solution; in turn, meaning block b_j is mined and hence, an open pit is viable, with non-zero optimum profit.

3.3 The Modified Dual-Simplex Algorithm

The dual problem (Problem II) may best be solved by the dual-simplex method. The conventional way to achieve the optimum via this method is to always maintain the optimality condition and move towards a dual-feasible solution by removing the negative-valued dual variables from the basis. Since the objective function

$$g = \sum_{i=1}^{n} y_i ,$$

with $y_i \geq 0$ $(i=1,2,\ldots n)$, has an unconstrained minimum value of zero (achievable if and only if $y_i = 0$ for all i), an alternative way to reach optimality is to make the positive-valued dual variables, y_i $(i=1,2,\ldots n)$ move towards zero and remove the negative-valued dual variables π_j. We shall refer to this proposed method as the **Modified Dual-Simplex** (MDS) algorithm. We shall see later that, only the dual variables y_i $(i=1,2,\ldots n)$ and π_j $(j=1,2,\ldots n)$ need be considered as basic variables, and that the algorithm stops even when some of the basic variables y_i are negative; this is because (as we shall see in Theorem 3.5) if we remove these using the standard dual-simplex method, it will have no effect on the remaining positive basic variables y_i. In Section 3.4 we show that MDS and Lerchs-Grossman algorithms are stepwise equivalent.

We will adopt the tableau structure shown in Figure 3.1 for the description of the MDS algorithm, and use the following additional notation to that given for the dual problem:

c_i : the cost of resource i, that is, the coefficient of

y_i , if $1 \leq i \leq n$,

π_{i-n} , if $n < i \leq m+n$,

s_{i-m-n} , if $m+n < i \leq m+2n$,

in the objective function g.

$\bar{c}_i$: the relative cost for resource i.

$c, \bar{c}$: matrices of order $1\times(m+2n)$ consisting of costs c_i and $\bar{c}_i$, respectively.

$Y=(y_{ij})$: the $n\times n$ matrix of coefficients in the current tableau associated with the y_i variables.

$\bar{A}=(\bar{a}_{ij})$: The $n\times m$ matrix of coefficients in the current tableau associated with the π_i variables.

$\bar{M}=(\bar{m}_i)$: the $n\times 1$ matrix consisting of the right-hand side constants of the constraints.

B : the $n\times n$ basis matrix whose columns are the columns in the initial tableau corresponding to the current basic variables.

$S=(s_{ij})$: the $n\times n$ matrix of coefficients, in the current tableau, associated with the s_i variables.

c_B : the $1\times n$ matrix containing the coefficients of the basic variables in the objective function.

$\bar{T}$: the $n\times(2n+m+2)$ matrix $[c_B^T|Y|\bar{A}|S|\bar{M}]$ representing the current tableau (excluding $\bar{c}$).

$A_i, \bar{A}_i$: the i-th column in A and $\bar{A}$, respectively.

Y_i : the i-th column in Y.

We note that the relative cost matrix $\bar{c}$ is given by the equation

$$\bar{c} = c - c_B[Y|\bar{A}|S] \tag{3.4}$$

and that the *condition for optimality* is

$$\bar{c} \geq 0 \ . \tag{3.5}$$

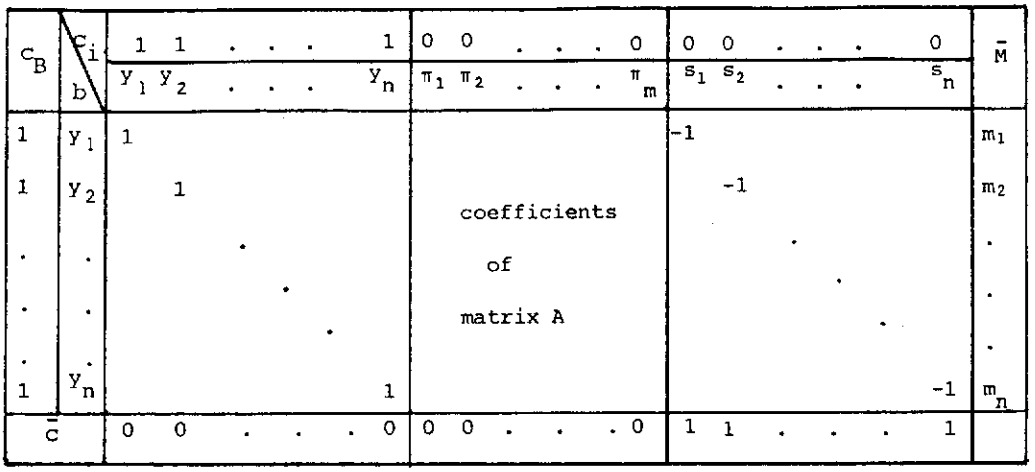


Figure 3.1 Tableau for the MDS algorithm (initial tableau shown); $b \equiv$ basis variables.

Now, in the initial tableau, we have

$$\begin{aligned} Y &= I \quad , \qquad \text{the } n\times n \text{ identity matrix,} \\ \bar{A} &= A \quad , \\ \bar{M} &= M \quad , \\ B &= I \quad , \\ S &= -I \quad , \\ c_B &= (1,1,\ldots,1) \end{aligned} \tag{3.6}$$

and so, initially,

$$\bar{c}_i = \begin{cases} 0 \ , & \text{for} \quad 1 \le i \le n+m \ , \\ 1 \ , & \text{otherwise} \ , \end{cases}$$

satisfying the optimality condition (3.5). Since any tableau in LP is obtained by pre-multiplying the initial tableau by the inverse of the current basis matrix B, by (3.6) we have that, at any stage,

$$\begin{aligned} B^{-1} &= Y \\ \text{and} \qquad & \qquad (3.7) \\ S &= -Y \ . \end{aligned}$$

Because of (3.7), we suppress the matrix S and associated headings from the tableau $\bar{T}$ in future illustrations.

The following is a description of the MDS algorithm. Note that the steps have been titled as in the Lerchs-Grossman algorithm; this is because the steps in both algorithms will be shown (in Section 3.4) to be equivalent.

The Modified Dual-Simplex Algorithm

Step 1 (Initialization)

Take the initial basis to be $\{y_1, y_2, \ldots, y_n\}$ with the values for the variables given by

$$y_i = m_i \ , \qquad i=1,2,\ldots,n$$

and all other variables zero, as the initial solution.

Step 2 (Optimality test)

Find basic variables y_p and y_q in rows p_r and q_r of the tableau, respectively, with the following properties:

$$y_p > 0 \ ,$$
$$y_q \leq 0 \ ,$$
$$\bar{a}_{p_r, j} = -\bar{a}_{q_r, j} = 1 \ , \qquad \text{for some } 1 \leq j \leq n$$

and proceed to Step 3. If no such variables are found, set each coefficient in c_B corresponding to a negative y_i in the basis to 0 and call this new vector c'_B. The solution to the primal problem (Problem I) is then given by

$$x^T = c'_B \, B^{-1} = c'_B Y \ ,$$

with optimal value $c'_B \, \bar{M}$. Stop.

Step 3 (Update)

y_p is the leaving basic variable and π_j is the entering basic variable. Use elementary row operations to eliminate the non-zero elements in column $\bar{A}_j$ using $\bar{a}_{p_r, j}$ as pivot. Update c_B. The values of the updated basis variables are then given by $\bar{M}$.

Step 4 (Normalization)

Find a basic variable π_l in the tableau with the following properties:

$$\pi_l = \bar{m}_s < 0 \ ,$$

and $$y_{st} = -1^* \qquad \text{for some } t \ , \quad 1 \le t \le n \ ,$$

where π_l appears in row s of the tableau. If not found, go to Step 2, otherwise y_t is the entering basic variable with value $-\bar{m}_s$ and π_l is the leaving basic variable. Use elementary row operations to eliminate the elements in column Y_t using y_{st} as pivot and multiply row s by -1. Repeat Step 4. □

Example 3.1 illustrates the implementation of the Modified Dual-Simplex algorithm. As mentioned earlier, matrix S is omitted from the tableaus in Figure 3.3(a).

Example 3.1 We wish to find the maximum closure of the graph D shown in Figure 3.2; the vertex weights are circled and the labels are shown next to their respective vertices and arcs (these will be used later). The solution to the problem is given in Figure 3.3(a).

*Note If t is not unique, a selection can be made (see Theorem 3.4) which corresponds to that in the L-G algorithm.

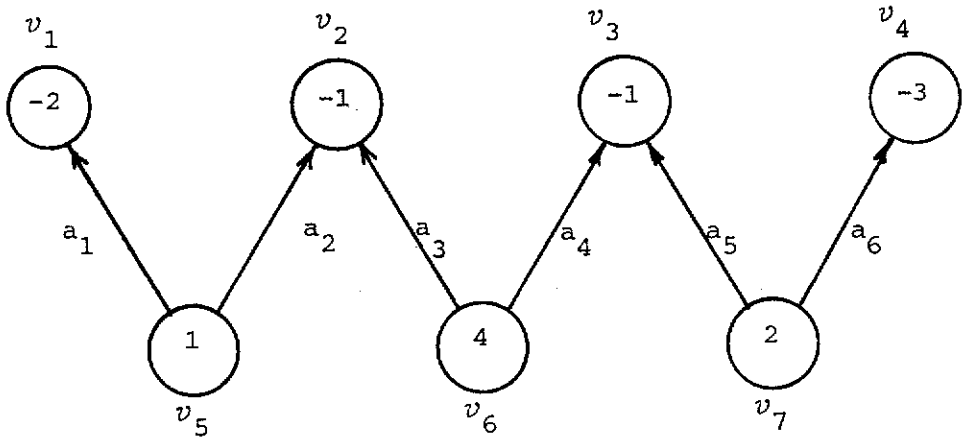


Figure 3.2 The graph in Example 3.1.

Note that a pivot cell in a tableau is circled and relevant rows are indicated by arrows. Tableau $\overline{T}_3$ shows that normalization is required since $\pi_2=-3$. In tableau $\overline{T}_5$, the optimality test succeeds, so by Step 2, we have

$$c'_B = (0,0,1,0,0,0,0) ,$$

$$x^T = c'_B Y = (0,1,1,0,0,1,0) \quad \text{and} \quad c'_B \overline{M} = 2 .$$

Thus, the solution is given by

$$x_2 = x_3 = x_6 = 1 \quad , \quad x_1 = x_4 = x_5 = x_7 = 0 ,$$

with the optimum value for the objective function equal to 2.

In the following section we show that the Modified Dual-Simplex (MDS) algorithm is equivalent to the Lerchs-Grossman algorithm, whereby each step in one corresponds to a step in the other.

$\bar{T}^0$

c_B	c_i	1	1	1	1	1	1	1	0	0	0	0	0	0	
	b	y_1	y_2	y_3	y_4	y_5	y_6	y_7	π_1	π_2	π_3	π_4	π_5	π_6	$\bar{M}$
1	y_1	1							-1						-2
1	y_2		1							-1	-1				-1
1	y_3			1								-1	-1		-1
1	y_4				1									-1	-3
1	y_5					1			1	(1)					1
1	y_6						1				1	1			4
1	y_7							1					1	1	2
	$\bar{c}$	0	0	0	0	0	0	0	0	0	0	0	0	0	

$\bar{T}^1$

c_B	c_i	1	1	1	1	1	1	1	0	0	0	0	0	0	
	b	y_1	y_2	y_3	y_4	y_5	y_6	y_7	π_1	π_2	π_3	π_4	π_5	π_6	$\bar{M}$
1	y_1	1							-1						-2
1	y_2		1			1			1		-1				0
1	y_3			1								-1	-1		-1
1	y_4				1									-1	-3
0	π_2					1			1	1					1
1	y_6						1				(1)	1			4
1	y_7							1					1	1	2
	$\bar{c}$	0	0	0	0	0	0	0	0	0	0	0	0	0	

$\bar{T}^2$

c_B	c_i	1	1	1	1	1	1	1	0	0	0	0	0	0	
	b	y_1	y_2	y_3	y_4	y_5	y_6	y_7	π_1	π_2	π_3	π_4	π_5	π_6	$\bar{M}$
1	y_1	1							-1						-2
1	y_2		1			1	1		(1)			1			4
1	y_3			1								-1	-1		-1
1	y_4				1									-1	-3
0	π_2					1			1	1					1
0	π_3						1				1	1			4
1	y_7							1					1	1	2
	$\bar{c}$	0	0	0	0	0	0	0	0	0	0	0	0	0	

$\bar{T}^3$

c_B	c_i	1	1	1	1	1	1	1	0	0	0	0	0	0	
	b	y_1	y_2	y_3	y_4	y_5	y_6	y_7	π_1	π_2	π_3	π_4	π_5	π_6	$\bar{M}$
1	y_1	1	1			1	1					1			2
0	π_1		1			1	1		1			1			4
1	y_3			1								-1	-1		-1
1	y_4				1									-1	-3
0	π_2		(-1)			-1				1		-1			-3
0	π_3					1					1	1			4
1	y_7						1						1	1	2
	$\bar{c}$	0	0	0	0	0	0	0	0	0	0	0	0	0	

$\bar{T}^4$ (Normalization)

c_B	c_i	1	1	1	1	1	1	1	0	0	0	0	0	0	
	b	y_1	y_2	y_3	y_4	y_5	y_6	y_7	π_1	π_2	π_3	π_4	π_5	π_6	$\bar{M}$
1	y_1	1				1				1					-1
0	π_1					1			1	1					1
1	y_3			1								-1	-1		-1
1	y_4				1									-1	-3
1	y_2		1				1			-1		(1)			3
0	π_3						1				1	1			4
1	y_7							1					1	1	2
	$\bar{c}$	0	0	0	0	0	0	0	0	0	0	0	0	0	

$\bar{T}^5$

c_B	c_i	1	1	1	1	1	1	1	0	0	0	0	0	0	
	b	y_1	y_2	y_3	y_4	y_5	y_6	y_7	π_1	π_2	π_3	π_4	π_5	π_6	$\bar{M}$
1	y_1	1				1				1					-1
0	π_1					1			1	1					1
1	y_3		1	1			1			-1			-1		2
1	y_4				1									-1	-3
0	π_4		1				1			-1		1			3
0	π_3		-1							1	1				1
1	y_7							1					1	(1)	2
	$\bar{c}$	0	0	0	0	0	0	0	0	0	0	0	0	0	

$\bar{T}^6$ (Final Tableau)

c_B	c_i	1	1	1	1	1	1	1	0	0	0	0	0	0	
	b	y_1	y_2	y_3	y_4	y_5	y_6	y_7	π_1	π_2	π_3	π_4	π_5	π_6	$\bar{M}$
1	y_1	1				1				1					-1
0	π_1					1			1	1					1
1	y_3		1	1			1			-1			-1		2
1	y_4				1			1					1		-1
0	π_4		1				1			-1		1			3
0	π_3		-1							1	1				1
0	π_6							1					1	1	2
	$\bar{c}$	0	0	0	0	0	0	0	0	0	0	0	0	0	

$c' = (0,0,1,0,0,0,0)$

$c_B' Y = (0,1,1,0,0,1,0)$

i.e. $x_2 = x_3 = x_6 = 1$

optimal value $= c_B' \bar{M} = 2$

Figure 3.3(a) Solution of Example 3.1 using the MDS algorithm.

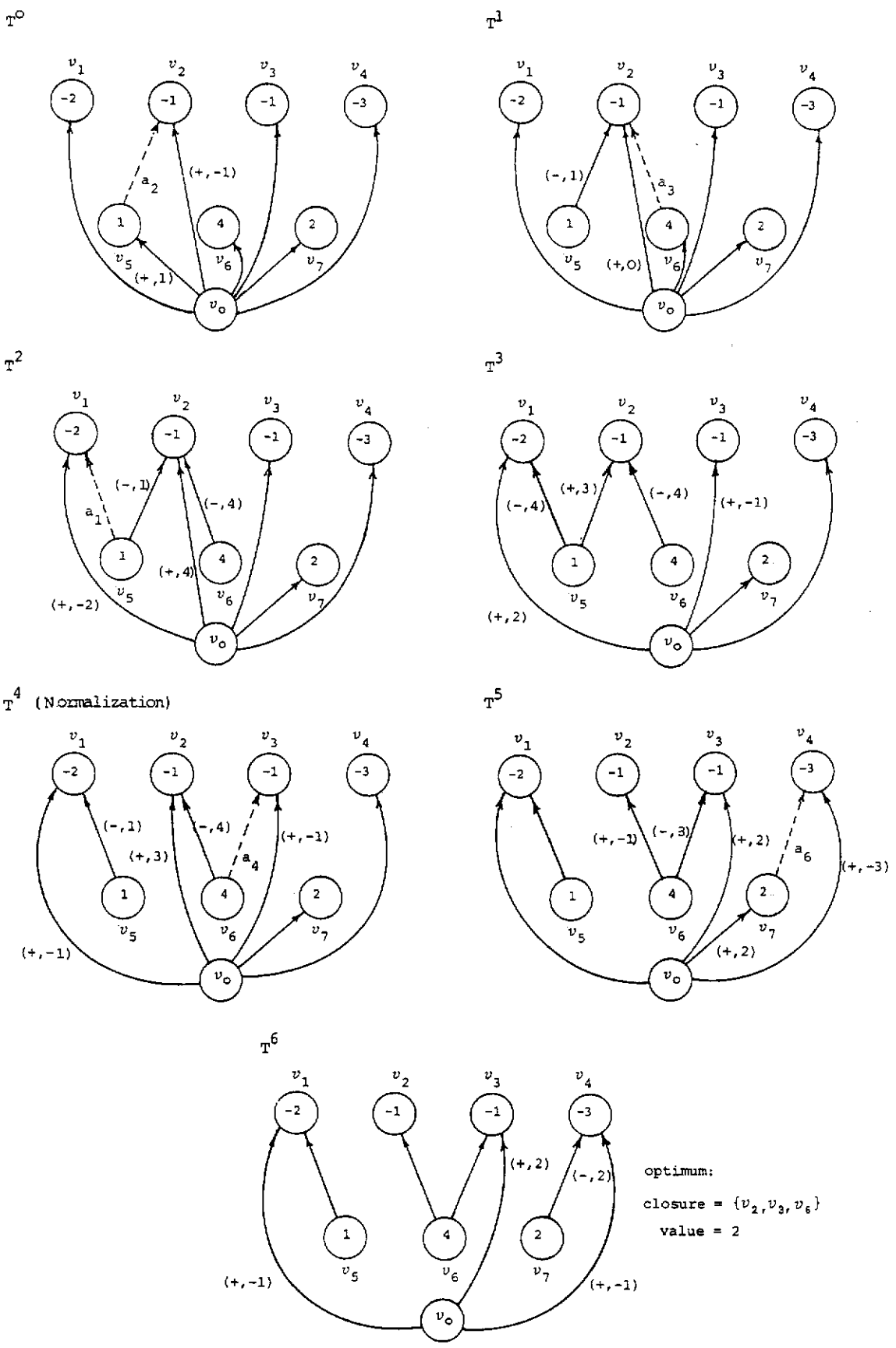


Figure 3.3(b) Solution of Example 3.1 using the L-G algorithm.

3.4 Equivalence of the MDS and L-G Algorithms

For convenience, we adopt the following notation and terminology. Suppose T is a tree produced by the L-G algorithm, as described in Chapter 2. We define an **internal** arc of T to be an arc in the set $A(D)$; that is, the root v_o of T is not one of its end vertices. We denote by $B(v_i)$ a branch in T with support arc (v_o,v_i), where $v_i \in V(D)$; and by $w(B(v_i))$ the weight of the branch. We denote by $B'_j(v_i)$ the branch rooted at $v_j \in B(v_i)$ $(j \neq i)$ in the updated tree T', obtained by removing arc (v_o,v_i) in T and supporting $B(v_i)$ with an internal arc a_k (not in T) incident at v_j. We denote "H is a subgraph of G" as $H \subset G$, as defined in Bondy and Murty (1976, p8), and the graph obtained by removing the vertices $V(H)$ and their incident arcs from G as $G \backslash H$.

Theorem 3.1. *Provided 'the same choice of pivot' is made, each iteration of the MDS algorithm resulting in tableau $\bar{T}$ corresponds, step by step, to an iteration of the L-G algorithm resulting in tree T, with the following correspondence of the variables. The dual variable y_i in $\bar{T}$ corresponds to the branch $B(v_i)$ in T, with value given by*

$$\begin{aligned} y_i &= w(B(v_i)) , && \text{for each } i \text{ such that } B(v_i) \text{ exists,} \\ &= 0 , && \text{otherwise.} \end{aligned}$$

The dual variable π_j in $\bar{T}$ corresponds to an internal arc a_j in T, with value given by

$$\begin{aligned} \pi_j &= w(a_j) , && \text{if } a_j \text{ is an m-arc ,} \\ &= -w(a_j) , && \text{if } a_j \text{ is a p-arc ,} \\ &= 0 , && \text{if } a_j \text{ is not in } T. \end{aligned}$$

Note. By 'the same choice of pivot' we mean that the same corresponding entering and leaving basic variables are chosen in the two algorithms; for example, if y_i is chosen as the leaving basic variable in the MDS algorithm, then the strong branch $B(v_i)$ is chosen in the update step in the L-G algorithm.

We devote the remainder of this section to provide a formal proof of Theorem 3.1. We achieve this through the following sequence of results.

Lemma 3.1 *The entries in the submatrix* $M = [Y|\bar{A}|S]$ *of any tableau* $\bar{T}$ *in the MDS algorithm are 0, +1 or -1.*

Proof. In the algorithm, the matrix B consists of n linearly independent column vectors selected from the columns of the matrix $[I|A]$. Thus B is totally unimodular (see Papadimitriou and Steiglitz 1982, p317), and since B is non-singluar, $det\ B = \pm 1$. Let P be any column vector representing a column of the submatrix $M_o = [I|A|-I]$ of the initial tableau. Then, either P contains one non-zero element ± 1 or two non-zero elements -1 and $+1$. Let $\bar{P} = (\bar{p}_1, \bar{p}_2, \ldots, \bar{p}_n)^T$ be the updated column vector P in any subsequent tableau. Then,

$$\bar{P} = B^{-1}P \ ,$$

i.e.
$$B\bar{P} = P \ . \qquad (3.8)$$

By Cramer's rule, the solution to (3.8) may be written

$$\bar{p}_i = \frac{det(B_i)}{det(B)}$$

$$= \pm\ det(B_i) \ ,$$

where B_i is the same matrix as B but with its i-th column replaced by P. Thus, B_i is totally unimodular. Therefore $\bar{p}_i = 0, -1$ or $+1$. □

We first show that Theorem 3.1 holds when the algorithms proceed without the normalization step being required.

Theorem 3.2 *Provided the normalization step is not required, each iteration of the L-G algorithm resulting in tree T, corresponds to an iteration of the MDS algorithm resulting in tableau $\bar{T}$, whereby a branch $B(v_i)$ in T, with $v_i \in V(D)$, corresponds to the dual variable y_i being basic in $\bar{T}$, with value $w(B(v_i))$; if no such branch exists, then $y_i = 0$. Moreover, if y_i appears in row r of $\bar{T}$, then the entries of row r in matrix Y are given by*

$$y_{rj} = 1 , \qquad \text{if} \quad v_j \in V(B(v_i)) ,$$
$$= 0 , \qquad \text{otherwise.}$$

Proof. Let the sequence of trees generated by the L-G algorithm be $T^z (z=0,1,2,\ldots)$ and the sequence of tableaus generated by the MDS algorithm be $\bar{T}^z (z=0,1,2,\ldots)$. Let Y^z denote the Y sub-matrix in tableau $\bar{T}^z$. We prove the theorem by induction on z.

Initially, each v_i in T^0, the initial tree in the L-G algorithm, is a branch $B(v_i)$ with $w(B(v_i)) = m_i$ $(i=1,2,\ldots,n)$. Since the basic variables in $\bar{T}^0$, the initial tableau in the MDS algorithm, are $y_i (i=1,2,\ldots,n)$, with values

$$y_i = m_i \qquad i=1,2,\ldots,n ,$$

and $Y^0 = I$, the statement of theorem is true for $z = 0$. Assume the statement is true for all $z \leq m$, and consider updating T^m and $\bar{T}^m$. Suppose y_p and y_q are two basic variables located in rows r and t of

$\bar{T}^m$, respectively, satisfying

$$y_p = w(B(v_p)) > 0$$

and (3.9)

$$y_q = w(B(v_q)) < 0 \ .$$

Also, suppose $a_j = (v_k, v_l)$ is an arc in D with $v_k \in V(B(v_p))$ and $v_l \in V(B(v_q))$, causing the update step in the L-G algorithm to proceed. Now the j-th column of matrix $\bar{A}$ in $\bar{T}^m$ is given by

$$\bar{A}_j = B^{-1}A_j \ , \tag{3.10}$$

and since $a_j = (v_k, v_l)$, the components of A_j (the j-th column of A) are given by

$$a_{kj} = -a_{lj} = 1$$

and (3.11)

$$a_{ij} = 0 \ , \quad i \neq l,k \ .$$

The r-th component of $\bar{A}_j$ in (3.10) is given by

$$\bar{a}_{rj} = \sum_{i=1}^{n} y^m_{ri} \, a_{ij} \ , \tag{3.12}$$

where y^m_{ri} are the elements of matrix Y^m in $\bar{T}^m$.

By hypothesis, $y^m_{rk} = 1$ and $y^m_{rl} = 0$, since $v_k \in V(B(v_p))$ and $v_l \notin V(B(v_p))$; and by (3.11), equation (3.12) becomes

$$\bar{a}_{rj} = 1 \ . \tag{3.13}$$

Similarly, since $y^m_{tk} = 0$ and $y^m_{tl} = 1$,

$$\bar{a}_{tj} = -1 \ . \tag{3.14}$$

Equations (3.9), (3.13) and (3.14) are the requirements for the update step in the MDS algorithm, with π_j the entering basic

variable for y_p. Introducing π_j in the basis in place of y_p (i.e. y_p is set to zero) results in the addition of row t to row r, through pivoting about $\bar{a}_{rj}$ given in (3.13). Now, by hypothesis, for each i such that $v_i \in V(B(v_p))$, corresponds to $y^m_{ti} = 1$ in Y^m; otherwise $y^m_{ti} = 0$. Similarly, $v_i \in V(B(v_q))$ corresponds to $y^m_{ri} = 1$ in Y^m; otherwise $y^m_{ri} = 0$. Since the branches are non-overlapping, pivoting results in

$$\begin{aligned} y^{m+1}_{ri} &= 1 \quad , \qquad \text{if } v_i \in V(B(v_p)) \cup V(B(v_q)) \ , \\ &= 0 \quad , \qquad \text{otherwise.} \end{aligned} \tag{3.15}$$

Thus the updated basic variable y_q, denoted y'_q, takes the value

$$y'_q = w(B(v_q)) + w(B(v_p)) \tag{3.16}$$

and the new branch $B(v_q)$ in T^{m+1}, denoted $B'(v_q)$, is such that

$$A(B'(v_q)) = A(B(v_p)) \cup A(B(v_q)) \cup \{a_j\}$$

and (3.17)

$$V(B'(v_q)) = V(B(v_p)) \cup V(B(v_q)) \ ,$$

with $w(B'(v_q))$ given by

$$w(B'(v_q)) = w(B(v_p)) + w(B(v_q)). \tag{3.18}$$

Equations (3.15) to (3.18) give

$$y'_q = w(B'(v_q)) \tag{3.19}$$

with

$$\begin{aligned} y^{m+1}_{ri} &= 1 \ , \qquad \text{if } v_i \in V(B'(v_q)) \ , \\ &= 0 \ , \qquad \text{otherwise.} \end{aligned} \tag{3.20}$$

It now remains to show that if a basic variable y_u ($u \neq p$ or q) appears in row s ($s \neq r, t$) of $\bar{T}^m$, then it will not be affected by

the update step; this corresponds to the fact that no other branches apart from $B(v_p)$ and $B(v_q)$ are affected in the update step in the L-G algorithm. We need to show that $\bar{a}_{sj} = 0$. By equation (3.12),

$$\bar{a}_{sj} = \sum_{i=1}^{n} y^m_{si}\, a_{ij}$$

$$= y^m_{sk} - y^m_{sl} \qquad \text{(from (3.11))}.$$

Now, since v_k and v_l are not in $B(v_u)$, by hypothesis, $y^m_{sk} = y^m_{sl} = 0$ and so, $\bar{a}_{sj} = 0$. The above results and, in particular, equations (3.19) and (3.20) show the statement of the theorem is true for tree T^{m+1} and tableau $\bar{T}^{m+1}$. Thus by induction, the statement is true for all cases. □

Corollary. *An entering arc a_j in the L-G algorithm, corresponds to entering the dual variable π_j in the basis in the MDS algorithm, with value given by*

$$\pi_j = w(a_j).$$

Moreover, if a_j supports branch B^j, then the coefficients y_{ri} $(i=1,2,\ldots,n)$ in row r, in which π_j appears in the tableau, are given by

$$y_{ri} = 1 , \qquad \text{if} \quad v_i \in V(B^j) ,$$

$$= 0 , \qquad \text{otherwise} .$$

Proof. In the proof of Theorem 3.2 we have seen that an entering arc a_j in the L-G algorithm corresponds to entering the basic variable π_j to replace y_p in the MDS algorithm. Updating the tableau by pivoting with the pivot in (3.13) results in π_j appearing in row r of the updated tableau, with value given by

$$\pi_j = \bar{m}_r$$
$$= w(B(v_p)) \ . \qquad (3.21)$$

Since branch $B(v_p)$ becomes branch $B^j = B'_k(v_p)$, supported by arc a_j in the updated tree, equation (3.21) gives

$$\pi_j = w(a_j) \ .$$

Moreover, since π_j takes over the coefficients y_{ri} associated with the leaving basic variable y_p in the tableau, and B^j has the same set of vertices as $B(v_p)$, Theorem 3.2 implies the values for y_{ri} as stated in the corollary. □

The following theorem gives a characterization for an *m*-arc and a *p*-arc in the MDS algorithm.

Theorem 3.3 *Provided the L-G algorithm proceeds without normalization being required, an internal arc a_j in tree T corresponds to a dual variable π_j being basic in a corresponding tableau $\bar{T}$ of the MDS algorithm, with value given by*

$$\pi_j = w(a_j) \quad , \quad \text{if } a_j \text{ is an } m\text{-arc},$$
$$= -w(a_j) \quad , \quad \text{if } a_j \text{ is a } p\text{-arc},$$
$$= 0 \quad , \quad \text{if } a_j \text{ is not in } T.$$

Moreover, if a_j supports branch B^j in T, then the coefficients y_{ri} $(i=1,2,\ldots,n)$ in row r in which π_j appears in $\bar{T}$, are given as follows. For an m-arc a_j,

$$y_{ri} = 1 \quad , \quad \text{if} \quad v_i \in V(B^j) \ ,$$
$$= 0 \quad , \quad \text{otherwise} \ ,$$

and for a p-arc a_j,

$$y_{ri} = -1 \quad , \quad \text{if} \quad v_i \in V(B^j) \ ,$$
$$= 0 \quad , \quad \text{otherwise} \ .$$

Proof. We adopt the same methodology in proving the theorem as for Theorem 3.2. Thus, we use the sequences T^z $(z=0,1,\ldots)$ and $\bar{T}^z$ $(z=0,1,\ldots)$ and induction on z. Since there are no internal arcs in T^0, we consider the case $z=1$. Suppose $a_j = (v_k, v_l)$ is an arc in D, which is entered by the L-G algorithm to update T^0 to T^1. By equation (2.3), a_j is an m-arc and by Theorem 3.2 there corresponds a dual basic variable π_j in $\bar{T}^1$ with $\pi_j = w(a_j)$ and the coefficients y_{ri} satisfying $y_{ri} = 1$, if v_i is in the branch supported by a_j and $y_{ri} = 0$, otherwise. Since a_j is the only internal arc in T^1, the statement of the theorem is true for the case $z=1$. Assume the statement of the theorem is true for all $z \leq m$, and consider updating T^m and $\bar{T}^m$. Proceeding as in the proof of Theorem 3.2, we have two basic variables y_p and y_q located in rows r and t of $\bar{T}^m$, respectively, and corresponding branches $B(v_p)$ and $B(v_q)$ in T^m satisfying

$$y_p = w(B(v_p)) > 0$$

and (3.22)

$$y_q = w(B(v_q)) < 0 \ .$$

Also, suppose $a_s = (v_k, v_l)$ is an arc in D with $v_k \in V(B(v_p))$ and $v_l \in V(B(v_q))$, causing the update step in the L-G algorithm to proceed. By the same argument as in the proof of Theorem 3.2, the coefficients of the matrix $\bar{A}$ in $\bar{T}^m$ satisfy

$$\bar{a}_{rs} = 1 \tag{3.23}$$

and

$$\bar{a}_{ts} = -1 \ . \tag{3.24}$$

Equations (3.22) to (3.24) cause the update step in the MDS algorithm to proceed; the entering basic variable π_s for y_p corresponds to entering arc a_s in T^m and the removal of arc (v_o, v_p). π_s takes the value

$$\pi_s = w(B(v_p))$$
$$= w(a_s) \qquad (3.25)$$

and by the corollary of Theorem 3.2, since a_s supports branch $B(v_p)$ in T^{m+1} (denoted $B'_k(v_p)$), we have

$$y^m_{rj} = 1 \quad , \qquad \text{if } v_j \in V(B'_k(v_p)) \ ,$$
$$= 0 \quad , \qquad \text{otherwise} \ , \qquad (3.26)$$

where y^m_{rj} are the elements of matrix Y^m in $\bar{T}^m$. Equations (3.25) and (3.26) are the conditions for an m-arc, as required. We now need to show that any dual variable π_j in $\bar{T}^m$, updated during the update step, maintains its correspondence with the arc it is associated with in T^m. Let a_j be an internal m-arc in branch $B(v_p)$ of T^m, π_j the corresponding dual variable in $\bar{T}^m$ and $B^j \subset B(v_p)$, the branch supported by a_j. Column s of matrix $\bar{A}$ in $\bar{T}^m$ is given by

$$\bar{A}_s = B^{-1}A_s \ ,$$

where A_s is column s in matrix A and B the basis matrix. If π_j appears in row u in $\bar{T}^m$, then

$$\bar{a}_{us} = u\text{-th component of } \bar{A}_s$$
$$= \sum_{i=1}^{n} y^m_{ui}\, a_{is} \ . \qquad (3.27)$$

But since $a_s = (v_k, v_l)$, we have

$$a_{ks} = -a_{ls} = 1$$

and $\qquad (3.28)$

$$a_{is} = 0 \quad , \quad i \neq k, l \ .$$

Equations (3.27) and (3.28) give

$$\bar{a}_{us} = y^m_{uk} - y^m_{ul} \ . \qquad (3.29)$$

Since $v_l \notin V(B(v_p))$, $y^m_{ul} = 0$ by Theorem 3.2, and so (3.29) becomes

$$\bar{a}_{us} = y^{m}_{uk} \ . \tag{3.30}$$

By hypothesis, since a_j is an m-arc, (3.30) becomes

$$\begin{aligned} \bar{a}_{us} &= 1 \ , && \text{if } v_k \in V(B^j) \ , \\ &= 0 \ , && \text{otherwise} \ . \end{aligned} \tag{3.31}$$

Hence, if a_j is not on the path $P(v_p, v_k)$ joining v_p to v_k, that is a_j does not support a branch containing v_k, then by (3.31), $\bar{a}_{us} = 0$ and pivoting does not update π_j. On the other hand, if a_j is on $P(v_p, v_k)$, $\bar{a}_{us} = 1$ and pivoting about (3.23) results in subtracting row r from row u in $\bar{T}^m$. Since $B^j \subset B(v_p)$, and by (3.26), the coefficients y^{m+1}_{ui} in $\bar{T}^{m+1}$ satisfy

$$\begin{aligned} y^{m+1}_{ui} &= -1 \ , && \text{if } v_i \in V(B'_k(v_p)) \backslash V(B^j) \ , \\ &= 0 \ , && \text{otherwise} \ . \end{aligned} \tag{3.32}$$

The updated value for π_j, denoted π'_j, is given by

$$\begin{aligned} \pi'_j &= \pi_j - w(B(v_p)) \\ &= w(a_j) - w(B(v_p)) \\ &= -[w(B(v_p)) - w(a_j)] \ . \end{aligned} \tag{3.33}$$

But by equation (2.4), the updated weight for a_j, $w'(a_j)$, in the L-G algorithm, is given by

$$w'(a_j) = w(B(v_p)) - w(a_j) \ ,$$

so (3.33) becomes

$$\pi'_j = -w'(a_j) \ . \tag{3.34}$$

Equations (3.32) and (3.34) correspond with the fact that a_j changes status to a p-arc, supporting the branch $B'_k(v_p) \backslash B^j$ in T^{m+1}, as required.

If a_j is an m-arc in branch $B(v_q)$ of T^m, supporting branch B^j (see Figure 3.4), then from (3.29),

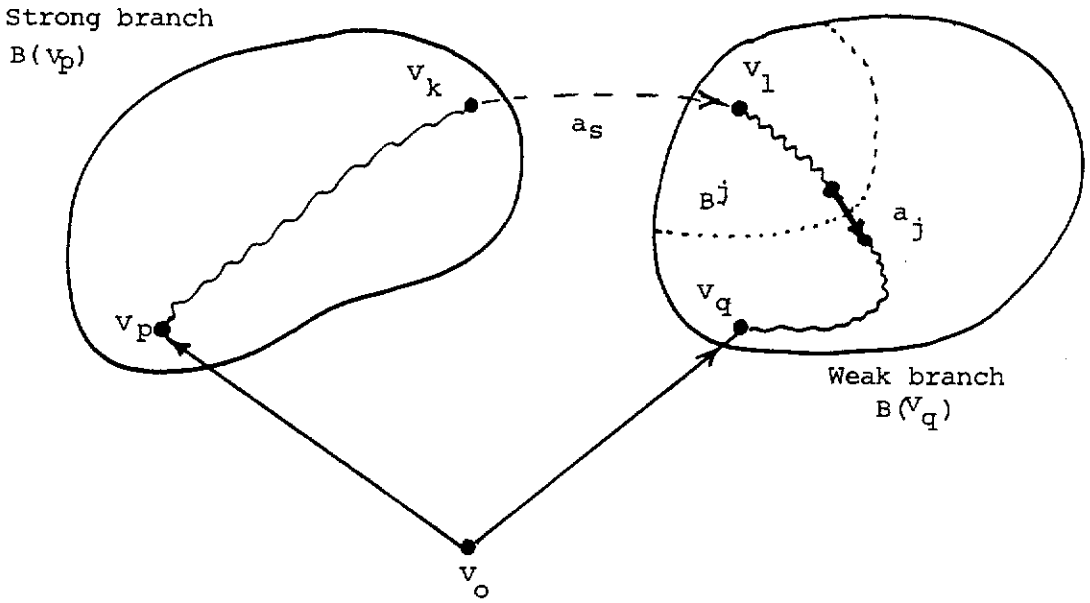


Figure 3.4 m-arc a_j in tree T^m.

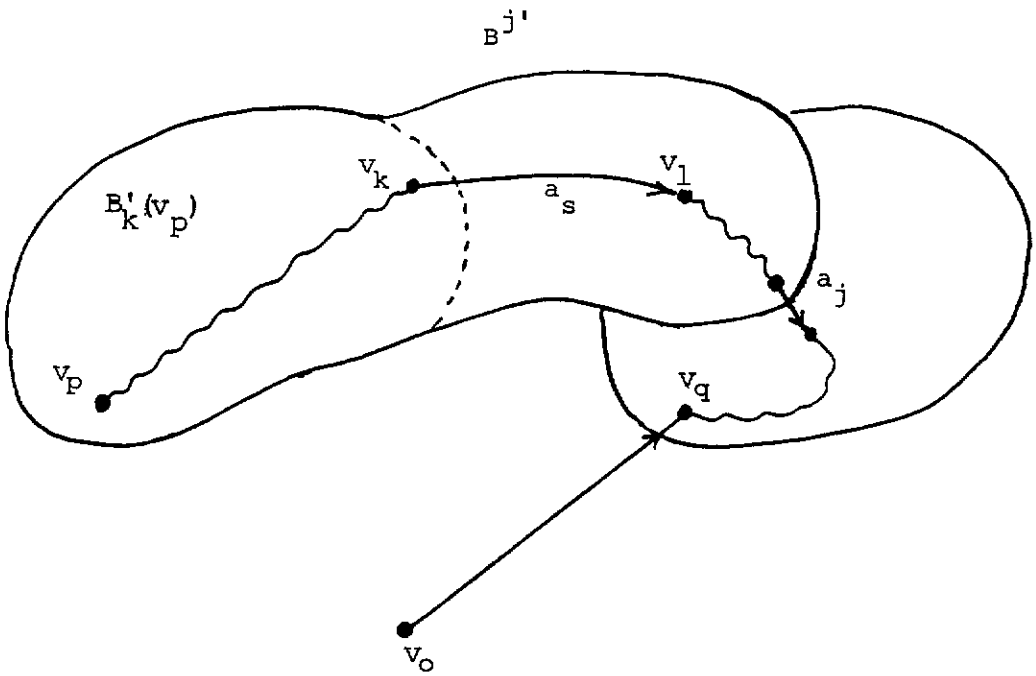


Figure 3.5 m-arc a_j in tree T^{m+1} (after update).

$$\bar{a}_{us} = -y^{m}_{ul} ,$$

since $v_k \notin V(B(v_q))$. But, by hypothesis, $y^{m}_{ul} = 1$, if $v_l \in V(B^j)$ and $y^{m}_{ul} = 0$, otherwise. Thus ,

$$\begin{aligned} \bar{a}_{us} &= -1 , && \text{if } v_l \in V(B^j) , \\ &= 0 , && \text{otherwise .} \end{aligned}$$

$\bar{a}_{us} = 0$ gives the same result as before, that is, no update if a_j is off the path $P(v_1, v_q)$. When a_j is on $P(v_1, v_q)$, π_j is updated and pivoting about (3.23) results in row r being added to row u. Thus,

$$\begin{aligned} \pi'_j &= \pi_j + w(B(v_p)) \\ &= w(a_j) + w(B(v_p)) \end{aligned} \qquad (3.35)$$

and by (3.26), coefficients y^{m+1}_{ui} are given by

$$\begin{aligned} y^{m+1}_{ui} &= 1 , \quad \text{for} \quad v_i \in V(B'_k(v_p)) \cup V(B^j) , \\ &= 0 , \quad \text{otherwise .} \end{aligned} \qquad (3.36)$$

But by equation (2.5), the updated weight for a_j in the L-G algorithm is

$$w'(a_j) = w(a_j) + w(B(v_p)) ,$$

so (3.35) becomes

$$\pi'_j = w'(a_j) . \qquad (3.37)$$

Equations (3.36) and (3.37) correspond with the fact that a_j does not change status and supports the branch $B^{j'}$ in T^{m+1}, formed by connecting the branches $B'_k(v_p)$ and B^j with arc a_s, as required (see Figure 3.5).

Consider, now, an internal p-arc a_j in branch $B(v_p)$ of T^m, π_j its corresponding dual variable appearing in row u of $\bar{T}^m$ and $B^j \subset B(v_p)$, the branch it supports. By hypothesis,

$$y_{ui}^{m} = -1 \quad , \qquad \text{if } v_i \in B^j \ ,$$
$$= 0 \quad , \qquad \text{otherwise} \tag{3.38}$$

and

$$\pi_j = -w(a_j) \ . \tag{3.39}$$

Since $v_1 \notin B(v_p)$ and $B^j \subset B(v_p)$, equations (3.38) and (3.29) imply

$$\bar{a}_{us} = -1 \quad , \qquad \text{if } v_k \in B^j \ ,$$
$$= 0 \quad , \qquad \text{otherwise.}$$

Thus, if a_j is on the path $P(v_p, v_k)$, then $v_k \in B^j$ and $\bar{a}_{us} = -1$, and pivoting about (3.23) results in the addition of row r to row u in $\bar{T}^m$. By (3.26) and (3.38),

$$y_{ui}^{m+1} = 1 \quad , \qquad \text{if } v_i \in V(B_k'(v_p)) \backslash V(B^j) \ ,$$
$$= 0 \quad , \qquad \text{otherwise} \tag{3.40}$$

and by (3.39)

$$\pi_j' = \pi_j + w(B(v_p))$$
$$= -w(a_j) + w(B(v_p)) \ . \tag{3.41}$$

But, by equation (2.4),

$$w'(a_j) = w(B(v_p)) - w(a_j) \ ,$$

so equation (3.41) becomes

$$\pi_j' = w'(a_j) \ . \tag{3.42}$$

Equations (3.40) and (3.42) correspond with the fact that a_j changes status to an m-arc, supporting branch $B_k'(v_p) \backslash B^j$ in T^{m+1}, as required.

Similarly, if a_j is a p-arc in $B(v_q)$, we obtain

$$y_{ui}^{m+1} = -1 \ , \qquad \text{if } v_i \in V(B_k'(v_p)) \cup V(B^j) \ ,$$
$$= 0 \quad , \qquad \text{otherwise} \tag{3.43}$$

and

$$\pi'_j = -w'(a_j) \quad , \tag{3.44}$$

where $w'(a_j) = w(a_j) + w(B(v_p))$.

Again, equations (3.43) and (3.44) corespond with the fact that a_j remains a *p-arc*, supporting branch $B^{j'}$ in T^{m+1}, as required. Thus, the statement of the theorem is true for $i = m+1$. By induction, the theorem is proved. □

It is clear from the argument in the proof of Theorem 3.3 that we have also shown the following corollary.

Corollary. *The update steps in the Lerchs-Grossman algorithm are equivalent to the update steps in the Modified Dual-Simplex algorithm.* □

The following theorem shows that the normalization steps in the MDS and L-G algorithms are equivalent when a certain pivot is chosen in the MDS algorithm. It will also be evident from the argument in the proof of the theorem that normalization does not alter the validity of Theorems 3.2 and 3.3.

Theorem 3.4 *The normalization steps in the L-G and MDS algorithms are equivalent. In particular, if $a_s = (v_k, v_q)$ is the strong arc which is removed from the non-normalized tree in the L-G algorithm, then the negative-valued variable π_s is replaced from the basis of the corresponding tableau in the MDS algorithm by the positive-valued variable y_q.*

Proof. Suppose $a_s = (v_k, v_q)$ is a strong internal arc in branch $B(v_p)$ of tree T, obtained immediately after the update step in the L-G algorithm. Let B_s be the branch supported by a_s. By Theorem 3.3, corresponding to a_s, is the basic variable π_s appearing in (say) row r of the tableau $\bar{T}$ in the MDS algorithm, corresponding to T. By Lemma 2.3, a_s is a p-arc and by Theorem 3.3,

$$\pi_s = -w(a_s) \tag{3.45}$$

and the coefficients y_{ri} in $\bar{T}$ are given by

$$\begin{aligned} y_{ri} &= -1 \quad , \qquad \text{if} \quad v_i \in V(B_s) \ , \\ &= 0 \quad , \qquad \text{otherwise} \ . \end{aligned} \tag{3.46}$$

Since $w(a_s) > 0$, equation (3.45) implies $\pi_s < 0$. Choices for an entering basic variable to replace π_s are all those with corresponding negative coefficients in row r in $\bar{T}$. Candidates for this are any of the y_i variables such that $y_{ri} = -1$ (since corresponding relative costs are zero). y_q satisfies this condition since $v_q \in V(B_s)$ and, by (3.46),

$$y_{rq} = -1 \ . \tag{3.47}$$

Pivoting, about (3.47), we obtain

$$\begin{aligned} y_q &= -\pi_s \\ &= w(a_s) \qquad \text{(from (3.45))} \\ &= w(B(v_q)) \ ; \end{aligned} \tag{3.48}$$

the last equality coming from the fact that arc a_j is removed from T and v_o is joined to the root v_q of B_s with arc (v_o, v_q), giving rise to branch $B(v_q)$ in the normalized tree T'. Denoting all updated variables from tableau $\bar{T}$ with a dash, π_s becomes

$$\pi'_s = 0 \ , \tag{3.49}$$

as required, to correspond to the removal of a_s from T. We now show

that all arcs and branches updated in obtaining tree T' during normalization in the L-G algorithm, is equivalent to updating the corresponding variables of $\bar{T}$ in the MDS algorithm.

Consider arc a_j $(j \neq s)$ on the unique path $P(v_p, v_q)$ from v_p to v_q in $B(v_p)$ and let B_j be the branch it supports in T (see Figure 3.6). By Theorem 3.3, π_j is a basic variable in $\bar{T}$; suppose it appears in row t. If a_j is an m-arc, then (again by Theorem 3.3) the coefficients y_{ti} in $\bar{T}$ satisfy

$$\begin{aligned} y_{ti} &= 1 \quad , && \text{if} \quad v_i \in V(B_j) \ , \\ &= 0 \quad , && \text{otherwise} \ . \end{aligned} \tag{3.50}$$

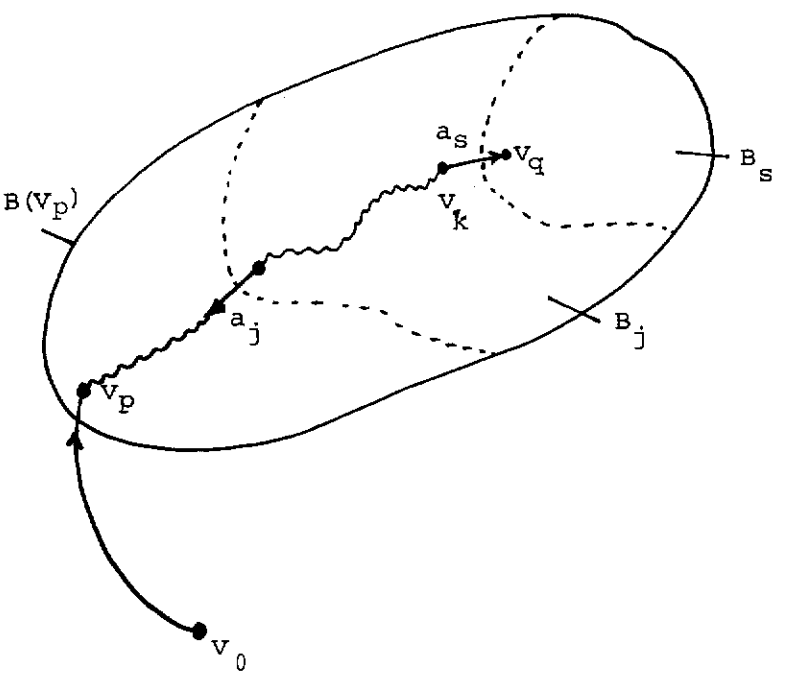


Figure 3.6

Since $v_q \in V(B_s)$ and $B_s \subset B_j \subset B(v_p)$, (3.50) gives

$$y_{tq} = 1 \ , \tag{3.51}$$

and hence pivoting about (3.47) results in the addition of row r to

row t in $\bar{T}$. By Theorem 3.3 and equation (3.45), the updated variable π_j is given by

$$\begin{aligned} \pi'_j &= \pi_j + \pi_s \\ &= w(a_j) - w(a_s) \end{aligned} \tag{3.52}$$

and by equations (3.46) and (3.50), and the fact that $B_s \subset B_j$, the updated coefficients y_{ti} become

$$\begin{aligned} y'_{ti} &= 1 \quad , \qquad \text{if} \quad v_i \in V(B_j)\backslash V(B_s) \\ &= 0 \quad , \qquad \text{otherwise} . \end{aligned} \tag{3.53}$$

By (2.6) and Theorem 3.3, equations (3.52) and (3.53) correspond to the update of an m-arc a_j during normalization in the L-G algorithm, namely, that it remains an m-arc and its weight is reduced by $w(a_s)$ $(= w(B(v_q)))$.

Similarly, by Theorem 3.3, if a_j is a p-arc, equations (3.50) and (3.51) are replaced by

$$\begin{aligned} y_{ti} &= -1 \quad , \qquad \text{if} \quad v_i \in V(B_j) \ , \\ &= 0 \quad , \qquad \text{otherwise} \end{aligned} \tag{3.50'}$$

and

$$y_{tq} = 1 \ , \tag{3.51'}$$

respectively. Pivoting, as before, results in subtracting row r from row t; and so by Theorem 3.3 and equation (3.45), we have

$$\begin{aligned} \pi'_j &= \pi_j - \pi_s \\ &= -w(a_j) + w(a_s) \\ &= -[w(a_j) - w(a_s)] \end{aligned} \tag{3.52'}$$

and by equations (3.46) and (3.50') the coefficients y_{ti} become

$$\begin{aligned} y'_{ti} &= -1 \quad , \qquad \text{if } v_i \in V(B_j)\backslash V(B_s) \ , \\ &= 0 \quad , \qquad \text{otherwise} . \end{aligned} \tag{3.53'}$$

By (2.6) and Theorem 3.3, equations (3.52′) and (3.53′) correspond to the update of a p-arc a_j during normalization in the L-G algorithm, namely, that it remains a p-arc but with its weight reduced by $w(a_s)$.

If a_j is not on the path $P(v_p, v_q)$, then $V(B_j) \cap V(B_s) = \phi$, and so $v_q \notin V(B_j)$. Here, equations (3.51) and (3.51′) are replaced by

$$y_{tq} = 0$$

and hence, pivoting about (3.47) will not alter π_j; this corresponds to arc a_j not being affected during normalization in the L-G algorithm.

Finally, we consider the update of the appropriate branches during normalization. Since branch $B(v_p)$ is in T, by Theorem 3.2, variable y_p is basic in the corresponding tableau $\bar{T}$, having value $w(B(v_p))$ and (taking u to be the row in which it appears in $\bar{T}$) coefficients y_{ui} satisfying

$$\begin{aligned} y_{ui} &= 1 \quad , && \text{if} \quad v_i \in V(B(v_p)) \ , \\ &= 0 \quad , && \text{otherwise} \ . \end{aligned} \tag{3.54}$$

Since $v_q \in V(B(v_p))$,

$$y_{uq} = 1$$

and thus pivoting about (3.47) results in the addition of row r to row u. Equations (3.45), (3.46) and (3.54), and the fact that $B_s \subset B(v_p)$ give the updated variable y_p and coefficients y_{ui}, after pivoting, to have the values

$$\begin{aligned} y'_p &= y_p + \pi_s \\ &= w(B(v_p)) - w(a_s) \\ &= w(B(v_p) \backslash B_s) \end{aligned} \tag{3.55}$$

and

$$y'_{ui} = 1 , \qquad \text{if} \quad v_i \in V(B(v_p))\backslash V(B_s) ,$$
$$= 0 , \qquad \text{otherwise} . \qquad (3.56)$$

Pivoting to remove π_s from the basis and replacing it by y_q, updates the coefficients y_{ri} in (3.46) to

$$y'_{ri} = 1 , \qquad \text{if} \quad v_i \in V(B_s) ,$$
$$= 0 , \qquad \text{otherwise} . \qquad (3.57)$$

By Theorem 3.2 and the fact that the new branches $B(v_p)$ and $B(v_q)$ in T' are $B(v_p)\backslash B_s$ and B_s, respectively, equations (3.55) and (3.56), and (3.48) and (3.57) correspond with the branch update during normalization in the L-G algorithm. □

From the argument in the proof of Theorem 3.4, it is clear we have the following corollary.

Corollary. *Theorems 3.2 and 3.3 hold even when the normalization step is required in both algorithms.* □

Theorem 3.1 follows immediately from Theorems 3.2 to 3.4 and their corollaries.

Figure 3.3(b) gives the solution of the problem in Example 3.1 using the L-G algorithm. Note the correspondence of the trees with the tableaus in Figure 3.3(a).

We now show that the MDS algorithm, as described in Section 3.3, finds the optimum solution to the primal problem (Problem I) as a linear programming technique, in its own right.

Theorem 3.5 *The Modified Dual-Simplex algorithm converges to the optimum solution of Problem I in a finite number of steps.*

Proof. Since the y_i's and π_i's are the only variables entering and leaving the basis in the MDS algorithm, the relative costs $\bar{c}_i$ $(i=1,2,\ldots,n+m)$ associated with these variables, initially zero, will remain zero throughout the execution of the algorithm up to the stop instruction in Step 2. That is, no row operations will be performed on $\bar{c}$ when updating the tableaus. Thus, the optimality condition $\bar{c} \geq 0$, will hold up to that point.

It is now required to show that when the optimality test succeeds in Step 2, the usual dual-simplex method applied from then on, replaces the negative-valued basic variables y_i with slack/dual variables and, consequently, changes c_B to c'_B where the appropriate components of c_B are changed from 1 to 0, while keeping the positive-valued basic variables y_i unchanged. Now, the only negative basic variable occurring in the last tableau $\bar{T}_f$ before the stop instruction, can only be a y_i, since a negative π_i is removed in the normalization step. Suppose the basic variable y_p, with value $w(B(v_p)) < 0$, has an entry in row r of $\bar{T}_f$. By Theorem 3.2, $y_{rj} = 1$ (for some j) and since $S = -Y$, $s_{rj} = -1$. Thus the slack variable s_j is a candidate for the entering basic variable. Hence, there will always be a slack variable s_j or dual variable π_j to enter the basis to replace y_p. However, by Theorems 3.2 and 3.3, the latter is equivalent to introducing an internal arc a_j in the L-G algorithm, connecting the branch $B(v_p)$ to another branch $B(v_q)$; also, since $\bar{a}_{rj} = -1$ (in this case), the arc a_j is directed from $B(v_q)$ to $B(v_p)$, and since $w(B(v_p)) < 0$, and the optimality test holds, $w(B(v_q)) < 0$. Pivoting then results in a new branch $B'(v_p)$, with weight

$$w'(B(v_p)) = w(B(v_q)) + w(B(v_p))$$
$$< 0 ,$$

corresponding to the new basic variable y'_p, and hence not changing the optimal value of the solution (given by L-G) nor the strong vertices in the optimum closure. Thus, the negative-valued basic variables y_i must eventually be replaced by the slack variables s_j.

Now, since $y_{rj} = -s_{rj} = 1$, if another basic variable y_k $(k \neq p)$ is in row t in $\overline{T}_f$, then by Theorem 3.2, $y_{tj} = 0$ (since $v_j \in B(v_p)$ and so $v_j \notin B(v_k)$). Hence, $s_{tj} = 0$. Thus pivoting about s_{rj} will not affect the entries in row t. Hence all positive-valued basic variables y_k in $\overline{T}_f$ will not be affected by pivoting to remove the negative-valued y_i variables from the basis. Thus, setting the r-th component of c_B to zero and repeating for each negative-valued variable y_p in the basis, produces the same effect as the standard dual-simplex method in replacing these variables with slack variables. Since by Theorem 2.3 the solution to Problem II is reached by the L-G algorithm in a finite number of steps, by Theorem 3.1 and the above argument, so will the MDS algorithm.

Finally, since Problem II is the dual of Problem I, the simplex multipliers obtained in Step 2 are, in fact, a solution to Problem I. □

3.5 The MDS Algorithm in 'Revised' Form

For purposes of efficiency in programming the MDS algorithm, in particular, for large sparse systems, we present the MDS algorithm in a form following along the lines of the Revised Simplex Method (see Calvert and Voxman 1989, p234). The following is additional notation (to that given in Sections 3.1-3.3) required for this revised form of the MDS algorithm.

e_j : the j-th column of I ($n \times n$ identity matrix).

s : a counter for the number of pivoting steps.

arclabel : $n \times 1$ matrix used to indicate the row in the tableau in which a π_i basic variable appears; thus,

$arclabel(r) = k$, if π_k appears in row r,

$= 0$, otherwise.

P_S : $n \times n$ elementary (or pivoting) matrix (see below).

$L = (\ell_i)$: $n \times 1$ matrix containing elements of P_S (see below).

$c = (c_i)$: same as c_B.

We define the **Revised MDS Tableau** as the tableau R given by the equation

$$R = [c | B^{-1} | \bar{M}] \ .$$

The information in R is all that is required to execute the MDS algorithm. The inverse of the basis matrix, B^{-1}, is kept in the product form

$$B^{-1} = P_S \, P_{S-1} \, \ldots \, P_1 \ ,$$

where each pivoting matrix P_j differs from the identity matrix in just one column, the elements of which are stored in L (see Calvert and Voxman 1989, p239); the actual values of these elements are given in the algorithm.

Algorithm

/****** Initialization *****/

$\bar{M}$=M
c=(1,1,...,1)
s:=0
finished:=false

while (not finished) do
    found:=false
    k:=0

/****** Optimality Test ****/

    while (k<m and not found) do
        k=k+1
    (i) $\bar{A}_k = A_k$, if s=0
        $= P_s P_{s-1} \cdots P_1 A_k$, otherwise.
    (ii) Search c, $\bar{A}_k$ amd $\bar{M}$ for rows r and t containing
            $c_r=1$, $\bar{a}_{rk}=1$ and $\bar{m}_r>0$,
        and $c_t=1$, $\bar{a}_{tk}=-1$ and $\bar{m}_t<0$.
        if (search succeeds) then found:=true
    end-while/*(k<m and not found)*/
    if (not found) then finished:=true
    if (not finished) then
      s:=s+1
      Construct new P_s by replacing column r of I by
      column $L=(\ell_i)$ given by

$$\ell_i = \begin{cases} -\bar{a}_{ik} & , \text{ if } i \neq r \\ 1 & , \text{ if } i = r \end{cases}$$

/***** Update *****/

      $\bar{M}:=P_s\bar{M}$
      $c_r=0$
      arclabel(r):=k
      normalized:=false
      r:=0

/***** Normalization **/

      while (not normalized) do
         found:=false
         while (not found and r<n) do
              r:=r+1
              if ($c_r=0$ and $\bar{m}_r<0$) then found:=true
         end while
         if (not found and r=n) then normalized:=true
         if (found) then
           p:=arclabel(r)
   (iii) $\bar{Y}_p = P_s P_{s-1} \cdots P_1 e_p$

(iv) $s:=s+1$

Construct new P_s by replacing column r of I by $L=(\ell_i)$ where ℓ_i is given by

$$\ell_i=\begin{cases} \bar{y}_{ip} & , \text{ if } i\neq r \\ -1 & , \text{ if } i=r \end{cases}$$

(v) $\bar{M}:=P_s\bar{M}$

$c_r:=1$

end if

end while /*not norm*/

end if /*not finished*/

end while /*not finished*/

/***** Finalization ****/

For each $1\leq i\leq n$ such that $\bar{m}_i<0$, set $c_i:=0$.

For each $1\leq j\leq n$,

$$x_j:=cP_sP_{s-1}\cdots P_1e_j$$

Optimal value:=$c\bar{M}$

Remark 3.1 A *cyclic* method of selecting the entering basic variable may be used as in the revised simplex method. After the search succeeds in step (ii) with $k = k_o$ in the algorithm, the next search begins with $k = k_o+1$ and exhausting the search when $k = k_o-1$, after wrapping around to $k = 1$ when $k = m$ is reached and dispensed with.

Remark 3.2 Note that if the test for normalization

$$c_r = 0 \quad \text{and} \quad \bar{m}_r < 0$$

succeeds with $r = r_o$, the search for the same condition resumes with $r = r_o+1$; that is normalization will not ever be required again (within the current normalization step) for $r\leq r_o$. The reason for this is that in the L-G algorithm, if an arc is weak, it cannot become strong after normalization of the tree within which it resides. A proof of this follows.

Since *p*-arcs can only be strong (Lemma 2.3) and arcs do not change status during normalization (see (2.6)), we need only

consider a weak p-arc a_t. Let a_s be a strong arc, then

$$w(a_s) > 0 .$$

If a_t is on the path P whose arcs may be updated during normalization, then by (2.6) either a_t is unchanged or its weight, $w(a_t)$, is reduced by $w(a_s)$; that is,

$$\begin{aligned} w'(a_t) &= w(a_t) - w(a_s) \\ &< 0 \quad , \qquad \text{since } w(a_t) \leq 0. \end{aligned}$$

Thus, a_t remains weak after normalization.

3.6 Time Complexity of the MDS Algorithm

As the node-arc incidence matrix A is sparse (2 non-zero elements per column), only the non-zero elements are stored in a simple data list in the computer. Each column A_j in A, representing arc a_j in the directed graph D, may be represented by a data node consisting of three memory locations as shown in Figure 3.7.

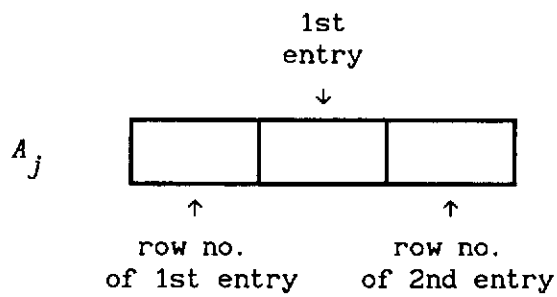


Figure 3.7

The value of the second entry in the column is not explicitly recorded as it is the negative of the first entry.

For a pivoting matrix P_s, all that is required to be stored is the column number which the non-unit column L is to occupy in I and the non-zero entries of L. The node structure for P_s is as shown in Figure 3.8.

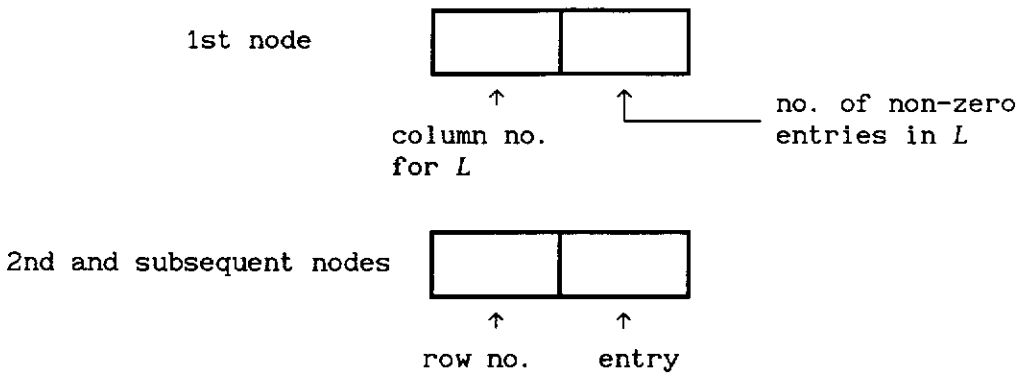


Figure 3.8

In the following complexity analysis, we will assume that the average density of a pivoting matrix P_s, stored as a data list described above, is the same as the density ρ of the matrix $[I|A]$ in the initial tableau; that is, of order $\frac{1}{n}$. Let q be the total number of pivots required by a problem solved by the MDS algorithm. We now analyze the time complexity of each step of the MDS algorithm as described in Section 3.5. Since the complexity increases as more pivots are made, we make the analysis for the $(s+1)$-th pivot, for any $s \geq 0$.

Step (i): Since multiplication of an $n \times 1$ matrix by a P_i requires at most n multiplications (see Calvert and Voxman 1989, p247), then $\bar{A}_k$ requires at most sn multiplications. But since the average density for P_i (stored as a data list of non-zero entries in L) is ρ, the number of multiplications is $s\rho n$.

Step (ii): $O(n)$ comparisons are required for this step.

Since k can be as large as m, the total number of operations (multiplications and comparisons) for Steps (i) and (ii) for the whole algorithm is $(s\rho n+O(n))m = s\rho nm+O(nm)$.

In the normalization procedure, a maximum of n comparisons are required to test for $c_r=0$ and $\bar{m}_r<0$. Each time the test succeeds, Step (iii) requires $s\rho n$ multiplications. Hence, we have a maximum total of $s\rho n^2+n$ operations each time normalization is required. The other operations, not mentioned above, are not significant in number as compared to the ones analysed.

The worst-case complexity of the MDS algorithm, without normalization ever being required, requires $n-1$ pivots. In this case, the initial basis $\{y_1, y_2, \ldots, y_n\}$ is reduced to a basis consisting of a variable y_p, for some p, and $n-1$ π_j variables. This corresponds to progressing from a tree with n branches to a tree with a single branch in the L-G algorithm. For this case, the time complexity T is given by

$$
\begin{aligned}
T &= O\left[\sum_{s=1}^{q} s\rho nm + qnm + qn\right] \\
&= O\left[\frac{q(q+1)}{2}\rho nm + qnm + qn\right] \\
&= O\left[q(q+1)m + qnm + qn\right], \qquad \text{since } \rho = \frac{2}{n}, \\
&= O\left[2(n-1)nm + n(n-1)\right], \qquad \text{since } q = n-1, \\
&= O\left[n^2m + n^2\right].
\end{aligned}
$$

In the case of an open pit mine model, $m = O(n)$ and so, $T = O(n^3)$.

Although there may be some time efficiency gain in implementing the L-G algorithm in a graph-theoretic way, there will, correspondingly, be a trade off in space complexity. However, this saving will not be of significant magnitude, and because of the equivalence of the two methods, we expect that the time complexity of the L-G algorithm is no better than $O(n^3)$ and most likely worse, when normalization is taken into account.

3.7 Space Complexity of the MDS Algorithm

Again we assume that no normalization is ever required in the execution of the MDS algorithm. The matrices c, $\bar{M}$ and arclabel require $3n$ storage locations. B^{-1}, stored in product form $P_1 P_2 \ldots P_q$, where q is the number of pivots, requires $\rho qn+q$ nodes to store the non-zero elements of the pivoting matrices in the product. Matrix A requires m nodes. Taking q to be $n-1$, and using the fact that each node has three storage locations for the elements of A and two storage locations for those of the pivoting matrices, we have that the storage requirement S, is given by

$$S = 2n + 2[\rho(n-1)n+n-1] + 3m \ .$$

If we take $m = dn$, where d is the size of the minimum search pattern for the block model, and $\rho = \frac{2}{n}$, the density of A, S becomes

$$S = 3n + 2[2(n-1)+n-1] + 3dn$$

$$\simeq (9+3d)n \ .$$

Again, we expect that the storage requirement for the L-G algorithm is of the same order of magnitude as S given above. Consider a simple ore-body model of a relatively small size of say, 50,000 cubic blocks and with a simple, 45° wall slope requirement throughout the pit. d is then 13 (the 'knight's move' search pattern) and S is approximately 2.5 million main storage locations. For a 500,000 block model, this translates to over 24 million storage locations! Note that d may be considerably larger for a variable wall slope requirement.

The implementation of the L-G or MDS algorithms is, with respect to the space complexity discussed above, a major problem. Although one can alleviate such a problem by optimizing level by level as discussed in Section 2.5, the amount of data reorganization and compaction may increase substantially the time complexity of the

algorithm. With this in mind, we have taken the view that it is better to reduce the amount of unnecessary data from the model, as much as possible, before applying these optimization techniques. This is the main consideration in Chapter 4 in which, we devise a fast and efficient technique of finding limits or a bound in the data within which, the optimum pit contour resides. All the blocks (and hence, data) outside this bound are discarded before the application of an optimization technique.

complexity problems associated with a true 3-D optimizer, such as, the Lerchs-Grossman graph theoretic method discussed in Chapter 2 and the Network Flow method discussed in Chapter 5.

In this chapter, we discuss the Johnson-Sharp method and present a modification to allow for some variability in wall slope restrictions not otherwise catered for. We also modify the method to properly handle 'air blocks'. The theory of bounding the optimum pit is developed and a fast, DP-based, bounding algorithm is presented. The practical implementation of the bounding algorithm is discussed and its complexity analyzed. Desirable practical features such as minimum pit bottom width are discussed and implemented.

4.1 The Modified Johnson-Sharp Method

Following along the same line of development of this method as in Caccetta and Giannini (1988a), we first assume that the maximum wall slope angle α (defined below) is constant throughout each vertical cross-section. Later, we show how to relax this assumption slightly. For each cross-section of the ore-body model, we adopt the following notation:

w : block width,

h : block height,

I : total number of rows (levels) of blocks,

J : total number of columns of blocks,

α : maximum wall slope angle; $\tan \alpha = h/w$,

n_1, n_J : the number of air blocks in columns 1 and J, respectively; we will assume that we do not have air trapped in solid, i.e. all air blocks are above the surface,

m_{ij} : net profit value of the block in row i and column j; a value of zero represents an air block and a negative value, a waste block; a block with a break-even profit value is given a small (in comparison to typical block values) negative value.

In our procedure, the pit wall will be allowed to intersect the sides of the cross-section in the case where the intersection is with air blocks (see Figure 4.1). This feature, considered by Caccetta and Giannini (1986a) correctly finds the optimum pit outline on the cross-section without wall slope restriction violation.

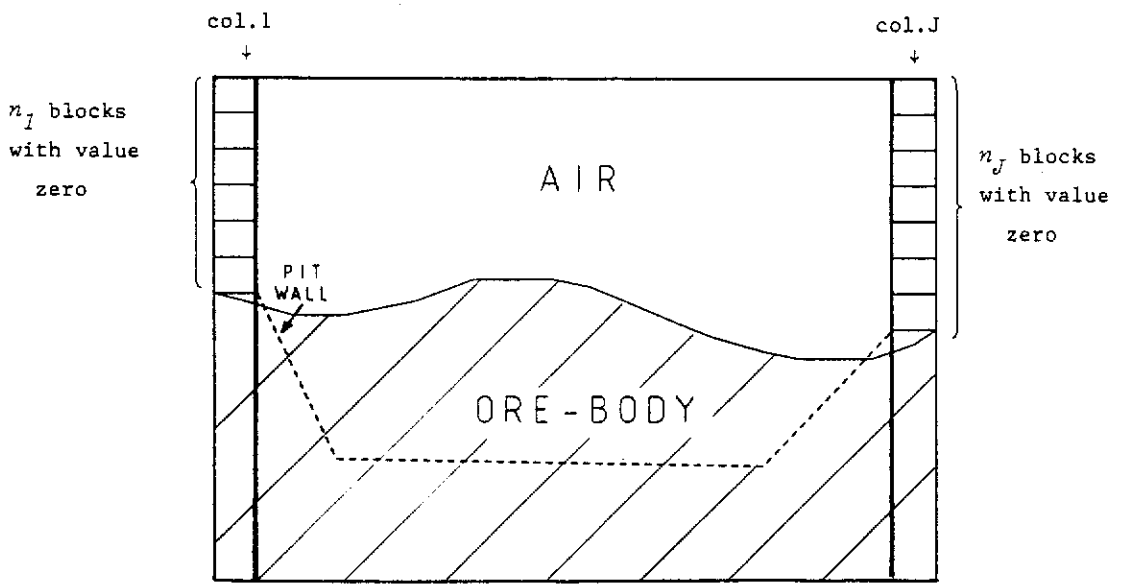


Figure 4.1 A typical cross-section model.

Let

$$M_{ij} = \sum_{k=1}^{i} m_{kj} , \qquad \text{for each } 1 \le i \le I \text{ and } 1 \le j \le J .$$

Note that M_{ij} represents the profit realized in extracting a single

column of blocks with block (i,j) as its base. In order to provide the necessary boundary conditions, it is convenient to extend our block model by adding two columns (columns 0 and $J+1$) and a row (row 0) to each cross-section, and define

$$M_{oj} = 0 , \qquad 1 \le j \le J ;$$

$$M_{io} = \begin{cases} 0 , & 0 \le i \le n_1 , \\ -\infty , & n_1 < i \le I ; \end{cases}$$

$$M_{i,J+1} = \begin{cases} 0 , & 0 \le i \le n_J , \\ -\infty , & n_J < i \le I . \end{cases}$$

Thus, M may be thought of a matrix of order $(I+1) \times (J+2)$. Also note that any feasible pit contour on a cross-section must contain at least one element of row zero or one element of column $J+1$ of M. We refer the reader to the standard texts of Bellman (1957) and Bellman and Dreyfus (1962) or Denardo (1982) for procedures of formulating problems as DP models.

Consider the problem of finding the contour of the optimum pit with bottom in level i_m in section k. Let P_{ij} be the maximum contribution (to net profit) of columns 0 to j to any feasible pit configuration that includes block (i,j) as a boundary element on section k, consisting only of i_m rows (including row 0). Let P be the matrix of order $i_m \times (J+2)$ consisting of elements P_{ij}. Let $P' = (P'_{ij})$ be the corresponding matrix if section k is extended to include level i_m. We now formulate the problem as a DP problem with the assumption that the maximum wall slope may be achieved by the removal of appropriate blocks within the section.

The connecting relationship between P' and P is as follows:

$$P_{oj} = M_{oj} , \qquad 0 \le j \le J+1 ,$$

$$P'_{io} = M_{io} , \qquad 1 \le i \le i_m$$

and for $j = 1,2,\ldots,J+1$,

$$P'_{i_m-t,j} = \begin{cases} M_{i_m,j} + \max\limits_{r=0,-1} \left\{P'_{i_m+r,j-1}\right\}, & t=0, \\ M_{i_m-t,j} + \max\limits_{r=-1,0,1} \left\{P'_{i_m-t+r,j-1}\right\}, & t=1,2,\ldots,T, \\ P_{i_m-t,j}, & t=T+1,\ldots,i_m, \end{cases} \qquad (4.1)$$

where T is the largest integer $(1 \le T \le i_m)$ whereby at block $(i_m-T+1, j-1)$ there is a change in either the profit value or decision variable r in updating P to P'. Note that in the above, the decision variable r (which provides the required wall slope restriction) is initially set to zero at each block in row zero, and can only take on the values 0 and $+1$ for the case $t=T=i_m$. In the case when the maximum in (4.1) is not unique, we break the tie by choosing the largest value of r. We thus have a standard DP formulation, with the columns in M corresponding to stages and the blocks within a column corresponding to the states for that stage. In what follows, we will take a change in P-value at a particular block, in updating P to P', to mean a change in the profit value or decision variable or both.

On the completion of the update of P, we find that one of the following two cases arises:

Case 1. Changes in P-values reach level 0 or column $J+1$ between level 1 and n_J. It can easily be shown then that the optimum contour, extending to level i_m, may be found by the following procedure:

If changes in P-values reach level 0, let K be the minimum integer such that $P'_{oK} \neq P_{oK}$. Find V and j_o such that

$$V = \max_{K \le j \le J+1} \left\{P'_{oj}\right\}$$

and j_o is the smallest integer satisfying $V = P'_{oj_o}$. Then the optimum pit contour is obtained by backtracking (i.e. following the successive decisions (values of r)) starting from block $(0, j_o)$; the optimum pit value being V.

If changes in P-values reach column $J+1$, let L be the minimum integer satisfying $1 \leq L \leq n_J$ and $P'_{L,J+1} \neq P_{L,J+1}$. Find V and i_o such that

$$V = \max_{L \leq i \leq n_J} \left\{ P'_{i,J+1} \right\}$$

and i_o is the smallest integer such that $V = P'_{i_o,J+1}$. Then the optimum pit contour is obtained by backtracking, starting from block $(i_o, J+1)$; the optimum pit value being V.

If changes in P-values reach both level 0 and column $J+1$, then the block giving maximum value for P' is searched along row 0 $(j \geq K)$ and along column $J+1$ (for $1 \leq i \leq n_J$).

Case 2. Changes in P-values do not reach level 0 or column $J+1$ between levels 1 and n_J.

In this situation, it is obvious that all pit contours traced from level 0 or column $J+1$ between levels 1 and n_J, will not pass level $i_m - 1$. We must now consider all pits having at least one block in level i_m on their contours and then select the optimum.

Let P^R_{ij} denote the maximum contribution (to net profit) of columns $J+1$ to j to any feasible pit configuration that includes the block (i,j) as boundary element. Then we have the backward recurrence relation

$$P^R_{ij} = M_{ij} + \max_r \left\{ P_{i+r,j+1} \right\}, \qquad i=0,1,\ldots,i_m \tag{4.2}$$

$$\text{with} \quad P^R_{i,J+1} = M_{i,J+1},$$

for $j = J, J-1, \ldots, 0$ and r (the decision variable) taking on the possible values -1 (except when $i=0$), 0, and $+1$ (except when $i=I$). Thus the value of the optimum pit having block (i_m, j) on its contour is given by

$$V_j = P'_{i_m, j} + P^R_{i_m, j} - M_{i_m, j} .$$

Hence the optimum pit having at least one block in level i_m on its contour has value $V = \max_{0 \le j \le J+1} \{V_j\}$ and contour obtained by backtracking, both to the left and to the right of block (i_m, j_o), where j_o is the minimum integer such that $V = V_{j_o}$.

We now adopt the notation S_{ik} to represent the optimum profit value of the pit contour in section k, mined to level i. Thus, the value V found above is, in fact, $S_{i_m, k}$; and by renaming P' as P and repeating the above procedure for $i_m = 1, 2, \ldots, I$ and $k = 1, 2, \ldots, K$, where K is the total number of sections in the block model, we find all S_{ik} values. Note that pit contours corresponding to the S_{ik} values need also be kept to find the final three-dimensional 'optimal' pit.

As for M, we treat S_{ik} as elements of the matrix S and add row 0 and columns 0 and $K+1$. Redefining P_{ik} as the maximum contribution (to net profit) of sections 0 to k to any feasible pit configuration that mines to level i in section k, we obtain the recurrence relation

$$P_{ik} = S_{ik} + \max_r \left\{ P_{i+r, j-1} \right\} , \quad 0 \le i \le I , \quad 1 \le k \le K+1$$

$$P_{io} = S_{io} , \quad 0 \le i \le I , \tag{4.3}$$

where the boundary conditions for S are similar to those for M,

defined earlier, and the decision variable r taking values as in (4.1).

Finally, we let m and l be the smallest integers such that

$$P_{om} = \max_{0 \le k \le K+1} \{P_{ok}\}$$

$$P_{l,K+1} = \begin{cases} \max_{1 \le i \le n_K} \{P_{i,K+1}\} , & n_K > 0 \\ 0 , & \text{otherwise} \end{cases}$$

and $$R = \max \{P_{om}, P_{l,K+1}\} ,$$

where n_K is the number of pit contours in air in section K. Then R gives the 'optimum' three-dimensional pit value and, starting from the 'block' giving the total contribution R, the successive decisions (values of r) addressing the various two-dimensional optimal cross-sectional pit outlines, form the outline of the overall pit.

Remark 4.1

In the above discussion, we have assumed that the wall slope angle α in the cross-section is related to the block dimensions as given by the equation $\tan \alpha = h/w$, and thus restricting the choices for the decision variable r in (4.1) and (4.2) to those of the set $\{-1,0,1\}$. More generally, we require several 'block jumps' to achieve the maximum wall slope allowable. In this situation we let N, the number of block jumps, be defined by the equation

$$N = \lfloor w(\tan \alpha)/H \rfloor$$

and restrict the choices for the decision variable r to those of the set $\{-N,\ldots,N\}$ in (4.1) and (4.2).

A similar set of choices for r may be found for (4.3) in case where the other dimension of the block is different to that of w.

4.2 The Two Wall Slope Per Section Problem

We may relax, slightly, the single wall slope restriction per section we imposed above, by introducing the situation where we have two different wall slope restrictions, one on each side of the cross-section.

Suppose the two side walls of the pit make maximum (acute) angles α_1 and α_2 with the horizontal and that

$$\alpha_2 > \alpha_1 ,$$

$$\tan \alpha_2 = 2 \tan \alpha_1 ,$$

and

$$\tan \alpha_2 = h/w ,$$

with α_1 on the right of the cross-section. The assumption $\tan \alpha_2 = 2 \tan \alpha_1$ makes the wall slope angle α_1 achievable by taking two blocks along and one up, repetitively, in moving to the right along the pit contour.

Let r_{ij} denote the value of the decision variable r at block (i,j) of a cross-section. Then, the two wall slope restrictions, as defined above, are achieved by modifying the choices for r in the recurrence relations (4.1) and (4.2), as follows:

Case 1 *(Committed to a single choice).* In (4.1), if $r_{i,j-1} = 1$ and $(i,j-1)$ is not an air block, then $r_{ij} = 0$. In (4.2), if $r_{i,j+1} = -1$ and $(i,j+1)$ is not an air block, then $r_{ij} = 0$.

Case 2 *(Restriction upwards or downwards).* In (4.1), if $r_{i+1,j-1} = 1$ and $(i+1,j-1)$ is not an air block, then $r_{ij} = 0$ or -1 (restriction downwards). In (4.2), if $r_{i-1,j+1} = -1$ and block $(i-1,j+1)$ is not an air block, then $r_{ij} = 0$ or 1 (restriction upwards).

If neither of the two cases above arise, r_{ij} is unrestricted in the choices for r.

Other pairs of wall slopes are possible by considering similar modifications to the ones considered above. Figure 4.2 illustrates the optimum pit contour (indicated by solid lines) obtained by the DP method, for the single cross-section shown with $n_1 = 3$ and $n_J = 6$. The block values (m_{ij}) are indicated within each block. The optimum pit value is 107 units. Note the intersection of the pit walls with the sides of the section; without this modification, the 'optimum' pit outline one obtains (shown in dotted lines), has a value of 5 units. Figure 4.3 shows the optimum pit contour for the data in Figure 4.2, using the two wall slope restrictions discussed above; the value of this pit is 103 units.

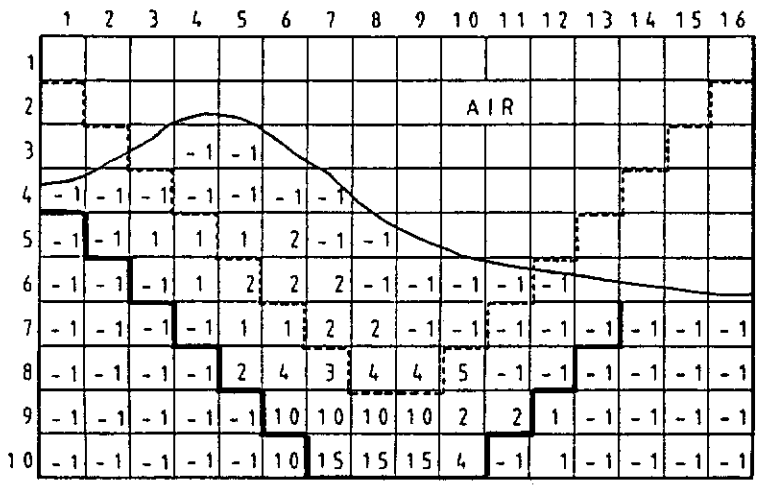


Figure 4.2 DP application with single wall slope restriction.

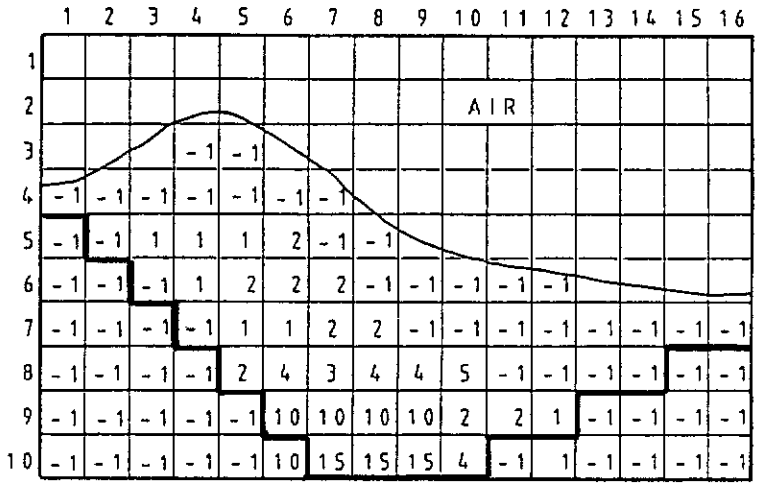


Figure 4.3 DP application with two wall slope restrictions.

Remark 4.2

The DP method, together with the modification and additional features discussed above, was written in FORTRAN-77, requiring 700 lines of code. The program was implemented on the Systems Research Institute of Australia's UNIVAC 1100/61 computer and later on the VAX-785 computer at Curtin University of Technology. The software was tested on data from a producing mine in the north of Western Australia, the results of which are discussed in Caccetta and Giannini (1986a). The testing of the software shows that it is an extremely fast method; this will be confirmed in Section 4.8 where time complexity of the method is discussed.

Two other features are sometimes desired by the mining industry in an open pit optimization technique, namely, a *minimum pit bottom width* requirement and *incremental pit generation*. These features, which we discuss below, may be easily incorporated in the DP software developed. A third feature, namely, *fixed total tonnage requirement* is discussed in Section 4.10.

4.3 Minimum Pit Bottom Width

In order to facilitate manouverability of equipment such as shovels and trucks and the construction of ramps, a minimum pit width may be specified for the bottom level in the pit. This constraint may be incorporated in the modified Johnson-Sharp DP method but has not found favour with the other techniques discussed in Chapters 2, 3 and 5.

Let the minimum width required in the pit be μ and the block width in a cross-section be w and in the longitudinal section (i.e. perpendicular to the cross-sections), b. Then, on each cross-section, the width of the pit contour on the bottom level must be at least $n_w = \lceil \frac{\mu}{w} \rceil$ blocks. We modify the algorithm presented in Section 4.1 as indicated in the following steps. Note that the matrix P^R is updated at each level using a similar procedure to that for updating P in (4.1).

For each cross-section k, $1 \le k \le K$, do;

For each level i, $1 \le i \le I$, do;

(i) Update P and P^R.

(ii) Find V_j defined by the equation

$$V_j = P_{ij} + P^R_{i,\,j+n_w-1} + \sum_{q=j+1}^{j+n_w-2} M_{iq} \,,$$

for $j = 1,2,\ldots,J-n_w+1$.

(iii) Find j_o, such that, j_o is the least integer satisfying

$$V_{j_o} = \max_j \{V_j\}.$$

(iv) Find the contour $C_k^{(i)}$ by tracing the successive decisions to the left of the block (i, j_o) in (4.1), and to the right of block (i, j_o+n_w-1) in (4.2).

next i

next k .

The contour $C_k^{(i)}$, consisting of the blocks on the pit walls, is optimum with bottom on level i, in section k, and having a width of at least n_w blocks. The value V_{j_o} of the pit with contour $C_k^{(i)}$ is denoted S_{ik}. The following steps are then performed with the profit matrices P and P^R being generated as before, but for a single, fictitious, longitudinal section consisting of values S_{ik} instead of M_{ij}, and n_w replaced by $n_b = \lceil \frac{\mu}{b} \rceil$.

For each level i, $1 \le i \le I$, do;

(i) Update P and P^R.

(ii) Find V_k defined by the equation

$$V_k = P_{ik} + P^R_{i,k+n_b-1} + \sum_{q=k+1}^{k+n_b-2} S_{iq} ,$$

with k in the range $[1, K-n_b+1]$.

(iii) Find k_o and $V^{(i)}$ satisfying

$$V_{k_o} = \max_k \{V_k\} ,$$

and $$V^{(i)} = V_{k_o} .$$

(iv) Find the contour $C^{(i)}$ by tracing the successive decisions to the left of (i, k_o) and to the right of (i, k_o+n_b-1).

next i.

The final three-dimensional 'optimum' pit contour C, with minimum pit width μ and corresponding pit value R, are then given by

$$R = \max_{1 \le i \le I} \{V^{(i)}\} ,$$

and

$$C = C^{(i_o)} ,$$

where i_o is such that $R = V^{(i_o)}$.

4.4 Incremental Pit Generation

We noted above, one situation when it is desirable to know the optimum pit to each level. Knowledge of such optimum pits also provides useful information for determining alternative cash flows over the life of the mine. This is very important as one seeks to maximize revenue in the early stages of the mine to help compensate the high initial capital investment.

Since the optimum pit contour to level $i-1$ must necessarily be completely contained within the one to level i, our implementation of the graph-theoretic algorithm described in Section 2.5, provides the incremental pit contours. In the DP application, the generation of the incremental pits is a byproduct of our modification for the pit width constraint problem discussed in Section 4.3; $C^{(i)}$ and $V^{(i)}$ $(i=1,2,\ldots,I)$ are the incremental pits and pit values, respectively.

4.5 Bounding Theory

As has already been mentioned in the introduction to this chapter, the dynamic programming method discussed above, produces an optimum three-dimensional open pit contour only in special circumstances. Consider, for instance, the possibility of obtaining two adjacent cross-sections having the two pit configurations within them without at least one column of blocks in common. Then, clearly, no pit is feasible under these conditions, let alone optimum.

The real power of the method however, is that it may be incorporated in a technique for providing a bound for the optimum pit. The idea behind the bounding algorithm is simply to speedily and efficiently remove, from the block model, blocks which cannot be within the optimum pit contour. The 'true optimizer', such as the

graph-theoretic method of Lerchs-Grossman (discussed in Chapter 2) or the Network Flow method (discussed in Chapter 5), is then applied on the reduced block model consisting of all remaining blocks. This results in easing the computational effort associated with the implementation of these optimizers which have much greater complexities than the bounding algorithm. These computational savings become even more important when the optimization method is repeatedly applied on a block model for various values of the parameters of the problem, such as, cost factors, and (as we shall see in Section 4.10) for other restrictions, such as, specific ore and waste tonnage requirements.

The idea of bounding the optimum pit was first proposed by Barnes and Johnson (1982) (see also Caccetta and Giannini 1985). However, their work does not include a thorough mathematical analysis of the problem, their algorithm lacks implementation detail, and is not supported by test applications. Moreover, they did not have the important tool in the form of the minimum search pattern (presented in Section 2.4) to efficiently apply this algorithm. In this section we develop the theory of bounding the optimum pit and in the subsequent sections show how this theory is effectively and efficiently implemented in an algorithm, through the use of the minimum search pattern. We begin the analysis of bounding the optimum pit by defining certain essential concepts.

Definition 4.1 The set $\mathcal{S} = \{(\alpha_i, \theta_i) : 1 \leq i \leq n\}$ where α_i is the maximum wall slope angle (in degrees) allowed at a specific orientation given by azimuth θ_i in an open pit model, is called a **slope definition.**

At this point we note that in practice, the wall slope restriction for a given pit, for an azimuth other than the ones specified, is taken as the linearly interpolated value of the two neighbouring slope restrictions given.

Definition 4.2 Given two wall slopes having angles α and β with the horizontal. If $\alpha > \beta$, the wall slope with angle α is said to be **less severe** than that with angle β.

Definition 4.3 Given two blocks A and B in a block model with B on a level higher than A, and a slope restriction α. If the angle which the line segment $\overline{AB}$ makes with the horizontal is greater than α, then block B is said to **overlie** block A.

Remark 4.3 If block B overlies block A and in turn, block C overlies block B, using the same wall slope restriction for the same orientation, then it follows that block C overlies block A.

Definition 4.4 The **neighbor blocks** (or **neighbors**) for a given block b is the set of blocks identified by applying the minimum search pattern to b.

Definition 4.5 Given two distinct slope definitions $\mathcal{S} = \{(\alpha_i,\theta_i) : 1 \le i \le n\}$ and $\mathcal{T} = \{(\beta_i,\theta_i) : 1 \le i \le n\}$ for an open pit model, defined on the same set of azimuths $\Theta = \{\theta_1,\theta_2,\ldots,\theta_n\}$. If at each azimuth θ_i, $\alpha_i \ge \beta_i$ for $1 \le i \le n$, then $\mathcal{S}$ is said to be **less severe** than $\mathcal{T}$.

We wish to explain the rationale behind the slope severity

definitions given in Definition 4.2 and 4.5. The reason why the steeper of two slopes is referred to as the less severe slope is because the open pit with a steeper slope is more profitable, i.e. less severe in waste removal costs.

For completeness we also include, here, the following definitions of terms (given in a more general sense in Section 1.2), which are more suited to a block model.

Definition 4.6 The surface separating a set of blocks from a block model the removal of which forms an **open pit**, and the remaining blocks in the model, is called the **open pit contour**, or simply, **contour**.

Definition 4.7 A contour $\mathcal{C}$ is a **feasible open pit contour** under slope definition $\mathcal{S}$, if the walls of the open pit, outlined by $\mathcal{C}$, conform with the wall slope restrictions specified in $\mathcal{S}$.

Definition 4.8 If the total value of the blocks removed from a block model, resulting in a feasible open pit contour $\mathcal{C}$, under a given slope definition, is maximum, then $\mathcal{C}$ is an **optimum open pit contour**.

Because of the non-uniqueness of optimum contours as defined in Definition 4.8, an extra condition may be imposed to guarantee uniqueness. The following theorem provides this condition.

Theorem 4.1 *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two feasible pit contours under the same slope definition $\mathcal{S}$. If $\mathcal{C}_1$ and $\mathcal{C}_2$ are optimum, then either they coincide or else the total value of the blocks contained within $\mathcal{C}_1$ and $\mathcal{C}_2$, but not in both, is zero.*

Proof. Let $\mathcal{C}_1 \cap \mathcal{C}_2$ be the set of blocks belonging to both of the pits with contours $\mathcal{C}_1$ and $\mathcal{C}_2$; $\mathcal{C}_1 \cup \mathcal{C}_2$ the set of blocks belonging to either of the pits with contours $\mathcal{C}_1$ and $\mathcal{C}_2$. Let B_1 be the set of blocks inside $\mathcal{C}_1$ but outside $\mathcal{C}_2$, and B_2 the set of blocks inside $\mathcal{C}_2$ but outside $\mathcal{C}_1$. Let $v(\mathcal{C}_1)$ and $v(\mathcal{C}_2)$ be the total net value of blocks contained within $\mathcal{C}_1$ and those contained within $\mathcal{C}_2$, respectively. Figure 4.4 shows the two pits with contours $\mathcal{C}_1$ and $\mathcal{C}_2$ and the sets B_1 and B_2. It is clear that both the contours for $\mathcal{C}_1 \cap \mathcal{C}_2$ and $\mathcal{C}_1 \cup \mathcal{C}_2$ are feasible pit contours under $\mathcal{S}$. If both $\mathcal{C}_1$ and $\mathcal{C}_2$ are optimum, then $v(\mathcal{C}_1) = v(\mathcal{C}_2)$. Suppose $v(\mathcal{C}_1 \cap \mathcal{C}_2) = z$. Then $v(\mathcal{C}_1) = v(\mathcal{C}_2) \geq z$, and so $v(B_1) \geq 0$. If $v(B_1) > 0$, then

$$\begin{aligned} v(\mathcal{C}_1 \cup \mathcal{C}_2) &= v(\mathcal{C}_2) + v(B_1) \\ &> v(\mathcal{C}_2) , \end{aligned}$$

contradicting the fact that $\mathcal{C}_2$ is optimum. Thus $v(B_1) = 0$. In a similar fashion, we can show that $v(B_2) = 0$. Hence $v(B_1 \cup B_2) = v(B_1) + v(B_2) = 0$. □

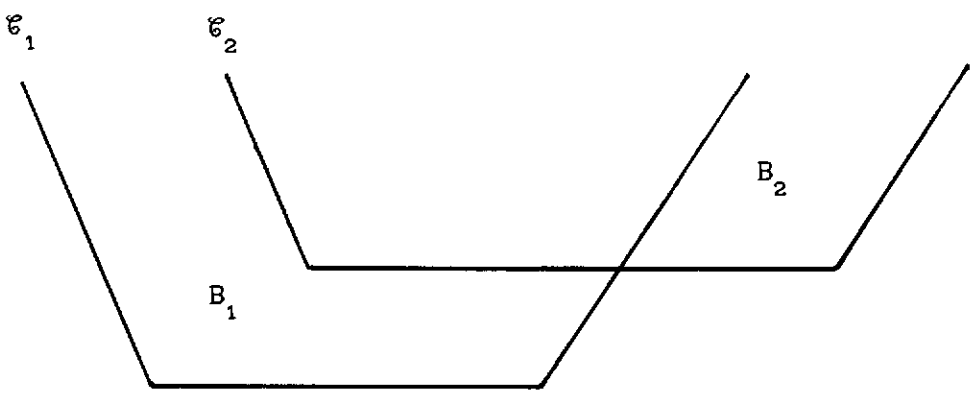


Figure 4.4

We may use Theorem 4.1 to show that the optimum pit contour containing the minimum number of blocks, referred to as the **revised optimum pit contour**, is unique.

Theorem 4.2 *The revised optimum pit contour, feasible under a given slope definition, is unique.*

Proof. We use the same notation here as in Theorem 4.1. It is clear that if the optimum pit value is zero, then the revised optimum pit contour contains an empty set of blocks and, hence, is unique. So, we may assume that the optimum pit value is positive. Suppose there are two distinct revised optimum contours $\mathcal{C}_1$ and $\mathcal{C}_2$, feasible under slope definition $\mathcal{S}$ with $v(\mathcal{C}_1) = v(\mathcal{C}_2) = v_{opt}(\neq 0)$, the optimum pit value, and each containing n blocks (the minimum number). By Theorem 4.1, the total value of the blocks in $\mathcal{C}_1 \cup \mathcal{C}_2$, but excluding those in $\mathcal{C}_1 \cap \mathcal{C}_2$, is zero. If $\mathcal{C}_1 \cap \mathcal{C}_2$ is empty, then $\mathcal{C}_1 \cup \mathcal{C}_2$ is a feasible pit contour under $\mathcal{S}$, with value $v(\mathcal{C}_1 \cup \mathcal{C}_2) = 2v_{opt} = 0$, contradicting the fact that $v_{opt} \neq 0$. If $\mathcal{C}_1 \cap \mathcal{C}_2$ is not empty, then (again by Theorem 4.1) it is an open pit with value $v(\mathcal{C}_1 \cap \mathcal{C}_2) = v_{opt}$, with a contour feasible under $\mathcal{S}$.

Since $\mathcal{C}_1$ and $\mathcal{C}_2$ are distinct, there is at least one block within $\mathcal{C}_2$ which is not within $\mathcal{C}_1$. Thus, the contour for $\mathcal{C}_1 \cap \mathcal{C}_2$ is optimum and contains less than n blocks. This contradicts the fact that $\mathcal{C}_2$ has the minimum number of blocks. Hence $\mathcal{C}_1 = \mathcal{C}_2$. □

For our purposes, we define a bound for the optimum open pit as follows.

Definition 4.9 A contour is a **bound** for the optimum open pit, with a given slope definition $\mathcal{S}$, if it contains all the blocks in the revised optimum pit contour feasible under $\mathcal{S}$.

The following theorem follows immediately from Theorems 4.1 and 4.2.

Theorem 4.3 *An optimum pit contour under a given slope definition, is a bound for the optimum open pit.*

In view of the above discussion, we will take **the optimum pit contour** to mean the revised optimum pit contour, in all the work that follows.

The following theorem is the basic result used to devise our bounding algorithm.

Theorem 4.4 *Given an economic block model and a pit contour $\mathcal{C}$ (not necessarily optimum), which is feasible under some pit slope definition $\mathcal{S}$. Consider the modified economic block model obtained by removing the blocks within $\mathcal{C}$. If the optimum pit contour under $\mathcal{S}$, for the modified block model, does not contain any blocks, then $\mathcal{C}$ is a bound for the optimum pit contour.*

Proof. The optimum pit contour $\mathcal{C}_o$ (of non zero net value) must contain blocks within $\mathcal{C}$ only, otherwise the portion of the contour of $\mathcal{C}_o$ outside $\mathcal{C}$ is a feasible pit contour for the modified block model, with non-positive net value and so (by Theorem 4.2) $\mathcal{C}_o$ cannot be the optimum. By Theorem 4.3, $\mathcal{C}_o$ is a bound and hence $\mathcal{C}$ is a bound. □

4.6 Bounding the Optimum Pit Using DP

The speed and efficiency of the dynamic programming method of attempting to find the optimum pit contour, was noted earlier and is discussed further in Sections 4.8 and 4.9. Here, we propose a method of finding a bound for the optimum pit contour using this dynamic programming technique.

Suppose that the required slope definition is $\mathcal{S} = \{(\alpha_i, \theta_i) : 1 \le i \le n\}$. Let $\mathcal{T} = \{(\beta_i, \theta_i) : 1 \le i \le n\}$ be the slope definition defined on the same set of azimuths as for $\mathcal{S}$, where β_i is given by

$$\beta_i = \beta = \max_j \{\alpha_j\} \quad , \qquad 1 \le i \le n.$$

It follows that $\mathcal{T}$ is less severe than $\mathcal{S}$. We now propose to apply the dynamic programming technique on the block model using slope definition $\mathcal{T}$ and follow the procedure outlined in Theorem 4.4. However, as the DP method does not produce a feasible contour $\mathcal{C}$ in general, we cannot (as yet) use the conclusion of the theorem. The problem is resolved in Theorem 4.5 which guarantees a bound from the following algorithm.

The DP-based Bounding Algorithm

Step 0 Call the given block model, the current block model and let Y be an empty set of blocks.

Step 1 Apply the DP pit optimization technique to the current block model using the least severe slope restriction β throughout i.e. within each cross-section and along the 'longitudinal cross-section'.

Step 2 Call the blocks within the contour obtained in Step 1, set Y_1, and remove these blocks from the current block model. Set $Y := Y \cup Y_1$.

Step 3 Using slope definition $\mathcal{S}$, for each block in Y_1 identify the overlying blocks that are still in the block model. Call this set Y_2. If Y_2 is empty, stop; the contour $\mathcal{C}$ formed by the blocks in Y

is a bound for the optimum pit contour. Otherwise, remove the blocks in Y_2 from the block model, set $Y:=Y \cup Y_2$ and go to Step 1. □

Theorem 4.5 *The DP-based algorithm converges in a finite number of iterations, to a bound $\mathcal{C}$ for the optimum pit with slope definition $\mathcal{S}$.*

Proof. It is clear from the algorithm that a sequence of pit contours, feasible under $\mathcal{S}$, is constructed with the property that each contour completely contains the blocks of the preceeding one in the sequence. This construction continues until no set of viable (i.e. profitable) blocks exist in the block model when DP is applied. This occurs when Y_2 is empty in Step 3, since no new set of viable blocks may be found in reapplying DP. Since there are only a finite number of such blocks, the algorithm converges in a finite number of steps.

Now it is clear that the end contour $\mathcal{C}$ is feasible under $\mathcal{S}$. Modify the original block model by removing from it the blocks in $\mathcal{C}$. Suppose $\mathcal{C}'$ is the optimum pit contour for the modified block model under $\mathcal{S}$. Further, suppose the set of blocks Y', contained within $\mathcal{C}'$, is non empty. It follows then, that $v(Y') \geq 0$. But as defined earlier, $\mathcal{C}'$ contains the minimum number of blocks, and by Theorem 4.2, is unique. So we must have $v(Y') > 0$. But since the blocks in Y' are viable under slope definition $\mathcal{S}$, which is more severe than the slope definition $\mathcal{T}$ in the DP application, then these blocks are also viable under DP. This is a contradiction since no viable blocks are found by DP outside $\mathcal{C}$. Hence Y' must be empty. By Theorem 4.4, $\mathcal{C}$ is a bound. □

The Best-Valued Cross-Section bounds outlined in Barnes and Johnson (1982) are expected to be slightly tighter than the one presented in this section but the time complexity, in the worst case, is significantly increased due to the double application of the DP technique. We will see later in our test cases that our bounding algorithm is extremely efficient and tight that this extra effort is not warranted.

4.7 Implementation of the DP-based Bounding Algorithm

Barnes and Johnson (1982) do not indicate how their bounding algorithm is implemented using DP. The main question to be answered is how Step 3, namely the identification of overlying blocks, is to be carried out. Without the minimum search pattern concept presented in Chapter 2, this implementation would be difficult and expensive. Further, the authors do not provide any test cases and computational results.

In this section we present implementation details and software description of the DP-based bounding algorithm. A proof that this software, called BOUND, produces a bound for the optimum pit contour is also given. BOUND is written in FORTRAN-77 and was developed and tested on the VAX-785 computer at Curtin University of Technology. The computational results from these tests are discussed in Section 4.9.

Input to BOUND

All integer values

NSECTS	:	Number of sections in the block model.
NROWS, NCOLS	:	Number of rows and columns of blocks per section, respectively.

H, B, W : Height, breadth and width of each block in the model, respectively.

BLVS : A three-dimensional array consisting of the net profit values (in dollars, rounded to the nearest dollar) of the blocks in the model; BLVS (I,J,K) represents the value of a block in row I, column J and section K.

ANGLE : The maximum angle in the required slope definition (in degrees); corresponds to angle β as defined in Section 4.6.

S : A two-dimensional array containing the minimum search pattern; S(*n,1*), S(*n,2*) and S(*n,3*) represent the *n*-th set of 'coordinates' (ℓ, i, j) in the MSP.

D : Size of set S.

Output from BOUND

All integer values

LBND : A two-dimensional array consisting of the positions of the leftmost blocks on the pit contour in each cross-section; LBND (I,J) represents the column number of the leftmost block on the pit contour in section I and row J.

RBND : As for LBND but consisting of the positions of the rightmost blocks on the pit contour.

Description of the Main Modules

PIT2D

This routine repeatedly performs the two-dimensional DP technique described in Section 4.1, on each cross-section of the block model. It obtains the optimum two-dimensional pit contours and their respective pit values for each level within each section. These are then passed on to module PIT3D for further processing.

PIT3D

This routine performs DP on a 'longitudinal section'; the 'blocks' in this section represent the values of the optimum cross-section pit contours down to each level (obtained from PIT2D). The output from this module consists of the arrays LBND and RBND containing the limits on the three-dimensional pit contour determined by the DP method.

BOUNDPIT

For each block within the limits on the contour produced by PIT3D, this routine basically identifies (under the required slope definition given by the MSP in array S) those overlying blocks not within these limits. The limits on the contour are subsequently updated to accommodate these overlying blocks. This is achieved through two routines: PROCESSLEVEL and PROCESSBLK.

PROCESSLEVEL

This routine identifies all the blocks on the contour defined by the limits in LBND and RBND for a specific level (or row) L of the block model. Call the blocks identified, set C_L.

PROCESSBLK

This routine identifies the neighbor blocks for a given block in the block model and updates the limits in LBND and RBND if the blocks are not within these limits. This is achieved as follows. Suppose a block b bas 'coordinates' (ℓ, i, j). The MSP is applied to b to obtain the neighbor blocks with coordinates (ℓ', i', j') given by the equations

$$\ell' = \ell - S(n,1) ,$$

$$i' = i + S(n,2) ,$$

and $$j' = j + S(n,3) ,$$

for $n=1,2,\ldots,D$.

Now, for each n $(1 \le n \le D)$,

if $i' <$ LBND(j', ℓ') or LBND(j', ℓ') = NULL (i.e. no limits exist),

then LBND$(j', 1') = i'$.

Similarly,

if $i' >$ RBND(j', ℓ') or RBND(j', ℓ') = NULL,

then RBND$(j', \ell') = i'$.

We now show that the steps in the DP-based algorithm given in the previous section may be achieved by the BOUND algorithm given below.

The BOUND Algorithm

Step 0 (Initialization)

(i) Find the range of the decision variable r in the DP application to each cross-section. This is obtained by finding the maximum number of block jumps N_1, within each

cross-section, to allow for the wall slope restriction specified by the parameter ANGLE. N_1 is given by

$$N_1 = \left\lceil \frac{B}{H} \tan(\pi . \text{ANGLE}/180) \right\rceil .$$

(ii) As in (i) but for the 'longitudinal section', the number of block jumps N_2 is given by

$$N_2 = \left\lceil \frac{W}{H} \tan(\pi . \text{ANGLE}/180) \right\rceil .$$

(iii) Initialize LBND and RBND arrays to indicate no limits exist.

Step 1

Call routines PIT2D and PIT3D to apply DP to the current block model with the decision variable r taking values within the ranges $[-N_1, N_1]$ and $[-N_2, N_2]$, respectively.

Step 2

If the limits of the pit contour, output by PIT3D in Step 1 are not within the limits given by LBND and RBND, update these arrays to include these blocks.

Step 3

(i) Find the lowest level L_m which the pit contour, obtained in Step 1, extends to.

(ii) For each level $L = L_m, L_{m-1}, \ldots, 1$ (in that order),

(a) Call PROCESSLEVEL to identify the blocks C_L on the contour on level L.

(b) For each block b in C_L,

call PROCESSBLK to identify the neighbors for b which are not within the limits given in LBND and RBND and update these arrays to include these neighbor blocks.

(iii) If no such neighbor blocks are identified in Step 3 (ii) then LBND and RBND give the limits of a bound for the optimum open pit; Stop.

Otherwise, in the initial block model, set all the block values within the limits in LBND and RBND to zero (equivalent to removing these blocks) and go to Step 1 with this modified block model. □

It remains to show that the BOUND algorithm converges in a finite number of iterations and that the resulting arrays LBND and RBND give the bound $\mathfrak{C}$ defined in Step 3 of the DP-based algorithm.

Theorem 4.6 *The BOUND algorithm converges, in a finite number of iterations, to the bound $\mathfrak{C}$ for the optimum pit contour, described by the DP-based bounding algorithm.*

Proof. We show that the steps in the BOUND and DP-based algorithms correspond, with the blocks contained within the limits in LBND and RBND being those of set Y.

It is clear that Steps 1 and 2 are equivalent in both algorithms, with the set Y_1 of blocks identified by DP in Step 1 being included within the limits LBND and RBND. We now show that Step 3 in BOUND identifies the blocks of Y_2. First note that identifying the neighbors for the blocks in C_L on level L and, in turn, identifying the neighbors of the neighbors on level $L-1$, etc., identifies all the overlying blocks for the blocks in C_L (see Remark 4.3).

We now show that for the blocks which are not within the current limits LBND and RBND, identifying those which overlie the blocks in C_L and updating the arrays LBND and RBND to accommodate

them, also identify all those which overlie the blocks within region R_L on level L, bounded by C_L. Suppose this is not the case and let X_1 be a block in $R_L \backslash C_L$, and X_2 a block outside the current limits, which overlies X_1 but not identified by PROCESSBLK for each block in C_K, $K=L, L-1, \ldots, 1$. Let P be the plane perpendicular to level L and passing through (the centre of) blocks X_1 and X_2. Let Z_1 and Z_2 be the two blocks in C_L and (with centres) on P, as shown in Figure 4.5.

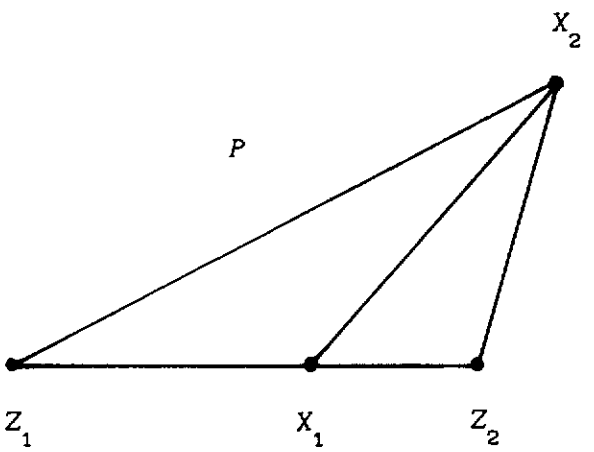


Figure 4.5

It is obvious from Figure 4.5 that X_2 will overlie Z_1 or Z_2 (X_2 overlies Z_2 is shown), since either the slope of $\overline{Z_1 X_2}$ or $\overline{Z_2 X_2}$ is greater than that for $\overline{X_1 X_2}$; hence, X_2 will be identified by PROCESSBLK. This is a contradiction to the assumption, and so all blocks outside the current limits in LBND and RBND, which overlie the blocks in R_L, are identified when processing the overlying blocks for C_L.

From Remark 4.3 and the fact that overlying blocks for each level are included within the limits in the higher levels (in Step 3 (ii)) the overlying blocks of the sets Y_2 of the DP-based algorithm are being included in set Y, identified by LBND and RBND. Note that

zeroizing the block values of the block model within LBND and RBND in Step 3(iii) of BOUND, is equivalent to removing the blocks of Y from the model in Step 3 of the DP-based algorithm.

Since the steps in the algorithm correspond with LBND and RBND identifying the set of blocks Y, the contour identified by these arrays is, in fact, the required bound $\mathcal{C}$, and by Theorem 4.5, is obtainable in a finite number of iterations. □

4.8 Complexity Analysis for BOUND

For convenience, we adopt the following notation:

p : number of rows of blocks per section,

q : number of columns of blocks per section,

s : number of sections,

n : total number of blocks in the model.

As pointed out in the DP formulation for the optimum pit problem in Section 4.1, within each cross-section of the block model the columns correspond to stages and the rows correspond to states. Thus, for the two-dimensional application of DP on the cross-sections, there are q stages and p states per stage, while for the longitudinal section, there are s stages and p states per stage. Let d_1 and d_2 be the maximum number of possible decisions at each state within each cross-section and within each longitudinal section, respectively.

For the worst-case complexity analysis we estimate the number of operations in the DP application as follows. For a single cross-section, we assume that DP is performed from level 1 to level i for $i=1,2,\ldots,p$. Applying DP twice using forward and backward recursion, within a cross-section with i levels, requires $O(2d_1 iq)$ operations. Summing over all levels, we obtain $O(2\frac{1}{2}d_1 p(p+1)q)$

operations for the DP applications on a section. Locating the optimum pit contour to each level, requires two operations and one comparison per block on each row, amounting to an equivalent of $3pq$ operations. For s cross-sections, the total number of operations to obtain the two-dimensional pit contours for the model is $O((d_1(p+1)+3)pqs)$. The application of DP to the 'longitudinal section' requires $O(d_2sp)$ operations. Now, $n = pqs$ and in practice, d_1 and d_2 are less than 10, p is not more than 50 or 60 for a medium to large ore-body model and d_2 is less than p. So, the three-dimensional contour, resulting from the DP method, requires $O((d_1(p+1)+3)pqs + d_2sp) = O((d_1p+d_1+3)n + \frac{d_2}{q} n) = O(n)$.

The foregoing analysis gives, essentially, the time complexity for Steps 1 and 2 of the BOUND algorithm. The bulk of the work in Step 3 is in the processing of each block on each of the level contours C_L, $L = L_m, \ldots, 1$. Each contour C_L has a maximum of $2(s+q)$ blocks, and since $L_m \leq p$, there is a maximum of $2p(s+q)$ blocks on all such contours. Since each block on these contours requires the application of the MSP to locate its neighbors, the total number of operations required to identify overlying blocks is $O(2Dp(s+q))$, where D is the size of the MSP. In practice, D is $O(1)$ and so Step 3 in BOUND has time complexity $O(n)$. Combining this with the time complexity for Steps 1 and 2, gives the time complexity for one iteration of BOUND as $O(n)$. Certainly the total number of iterations cannot be more than the number of positive-valued blocks in the model, thus giving a worst-case time complexity of BOUND as $O(n^2)$. However, in practice, 2 or 3 iterations are usually sufficient to produce the required bound, giving an actual run-time for BOUND more like $O(n)$.

We estimate the computer memory requirement for BOUND as

follows. For the dynamic programming application, the arrays P, P^R, M (the array containing the block net profit values m_{ij}), and two arrays which are used to store the decision variable r_{ij} for the forward and backward recursion, are each of dimension p by q. The arrays LBND and RBND in the bounding algorithm are each of dimension s by p. This gives a total of $(2s+6q)p$ memory locations. In practical applications, it is quite reasonable to assume $s \simeq q$, and so, the total number of memory locations is not much more than about $8qp$. Since $n = spq$ and $s \gg 8$ in the design of large pits, the total memory requirement for BOUND is, consequently, *very much less than* n. Note also, that it is independent of D. This last fact tells us that no matter how intricate the wall slope requirement is, it does not change the memory requirement for BOUND.

4.9 Computational Performance of BOUND

The BOUND algorithm was extensively tested on a number of pit design situations, the results of which are given in Chapter 5 where its use in connection with the network flow technique is analyzed. This latter technique was applied to the data before and after the application of BOUND, resulting in the same optimum pit contour, and consequently, verifying the validity of the bound outlines obtained from it. The results also show that BOUND is extremely effective in reducing the size of the data in the problem, typically, by as much as 85% (96% in one application), taking only seconds of CPU time. This observation and the fact that the memory requirement for BOUND is very much less than n and independent of D, makes it extremely valuable in designing pits as much as seven times larger and at a fraction of the run time required by an optimizer on a given computer without bounding.

4.10 Application Strategy for the BOUND Algorithm

The net profit value m_i of a particular block b_i in the block model, is ascertained using economic parameters such as extraction and transportation costs, price of metal, etc. These parameters will vary in time and, consequently, change the ultimate shape of the optimum pit. However, using past trends and forecasts, it is possible to give a range for each of the economic parameters used in estimating m_i consistent with the time of actually mining block b_i. Thus, each block value m_i, $1 \le i \le n$, may be more confidently known to be within some range $[m_i^-, m_i^+]$, $1 \le i \le n$.

It is clear that using the values m_i^+, $1 \le i \le n$, as the net profit values for the blocks in the model, results in an optimum pit contour $\mathcal{C}^+$ which, necessarily, contains all the blocks within any optimum pit contour resulting from a model, with block values m_i satisfying

$$m_i < m_i^+ , \qquad 1 \le i \le n .$$

Hence, the bound for $\mathcal{C}^+$ will also be a bound for any optimum pit contour resulting when using block values within the ranges $[m_i^-, m_i^+]$, $1 \le i \le n$.

In view of the above discussion, the best strategy in determining the optimum pit contour is to perform the following steps:

(i) Apply the BOUND algorithm on the block model using the block values m_i^+, $1 \le i \le n$, and obtain a bound $\mathcal{C}$.

(ii) Modify the block model to one containing only the blocks within $\mathcal{C}$.

(iii) Apply an optimization method such as Network Flow (discussed in Chapter 5) on the modified block model, using different choices for the block values $m_i \in [\bar{m_i}, \overset{+}{m_i}]$, $1 \le i \le n$, to obtain a set of possible pit outlines. The decision for the ultimate pit contour selected from this set would then depend on engineering, political and possibly, environmental viewpoints.

4.11 Fixed Total Tonnage Requirement

Another important use of the bound $\mathcal{C}$, described above, is seen in the following problem commonly found in pit design.

> *Find an optimum pit contour which satisfies the geometric wall slope constraints and which yields a specified total tonnage.*

The problem is closely related to production scheduling in which, the tonnage specification may be stipulated by government, customer demand or economics. In formulating this problem, we make use of the following notation.

n : number of blocks in the model,

b_i : block i $(i=1,2,\ldots,n)$,

w_i : weight of b_i,

m_i : net value of b_i,

T : desired total tonnage

x_i : variable with value 1 if b_i is mined or value 0, otherwise,

γ : $\{b_i \mid x_i=1,\ 1 \le i \le n\}$,

$\mathcal{C}_\gamma$: Pit contour obtained in removing set γ from the block model.

Mathematically, the problem may be written

$$\text{maximize} \quad f = \sum_{i=1}^{n} m_i x_i$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i = T,$$

$$\text{and} \quad \mathcal{C}_\gamma \text{ is a feasible pit contour.}$$

This problem is equivalent (Everett 1963) to the problem:

$$\text{maximize} \quad g = \sum_{i=1}^{n} m_i x_i - \lambda \sum_{i=1}^{n} w_i x_i$$

$$\text{subject to} \quad \mathcal{C}_\gamma \text{ being feasible,}$$

where $\lambda(\geq 0)$ is the lagrange multiplier. So, for a fixed λ, we have the problem

$$\begin{aligned} &\text{maximize} \quad g = \sum_{i=1}^{n} (m_i - \lambda w_i) x_i \\ &\text{subject to} \quad \mathcal{C}_\gamma \text{ being feasible.} \end{aligned} \tag{4.4}$$

Thus, the fixed tonnage problem reduces to the standard pit limit problem (4.4), with block values $m_i - \lambda w_i$.

In real applications, the tonnage obtained from the optimum pit when $\lambda=0$ i.e. the unconstrained problem, exceeds T. As λ increases the block values $m_i - \lambda w_i$ decrease, giving rise to an optimum pit contained within the one corresponding to a smaller value of λ. The problem is thus solved by systematically incrementing λ until the tonnage obtained is as close to T as practically possible. This is efficiently achieved, in practice, by starting with the bound $\mathcal{C}$ discussed in Section 4.9, altering the block values within $\mathcal{C}$ by incrementing λ as discussed, and applying a pit optimization algorithm within $\mathcal{C}$.

Since each value of λ gives rise to a solution to a different fixed tonnage problem, it is possible to obtain a series of pits (each contained within the next of higher tonnage) of different tonnages. A sequence of such pits could be used to produce a possible schedule for the mining of the whole ore-body to its optimum limits.

The solution of the fixed tonnage problem thus requires a number of applications of an appropriate pit limit optimization algorithm. This highlights further the importance of a fast and efficient algorithm for solving the pit limit problem. We present such a method in Chapter 5.

CHAPTER 5

THE NETWORK MODEL

The solution of the pit limit problem, by modelling it as a maximum flow problem in a network, was proposed by Picard (1976). A network N is obtained from the digraph D representing the ore-body by adding two vertices s (the source) and t (the sink). The vertex s is joined to every vertex of D with positive weight, by an arc with capacity equal to the weight of the vertex. The vertices of D with non-positive weight, are each joined to t by an arc with a capacity equal to the negative of the weight of the vertex. The arcs of D are all given a capacity of ∞. The maximum closure of D (and hence the optimum pit contour) is shown to correspond to a minimum cut of N.

In this chapter we present an outline of the theory of the maximum flow - maximum closure equivalence. We discuss the selection of the Dinic algorithm to solve this particular maximum flow problem and present modifications and specific data structures for this algorithm, to efficiently cater for the special structure of the network N. A strategy for the solution of the pit limit problem, using this algorithm, is presented. Computational results from test cases, demonstrating the strength of the method, are also given in this chapter. Pseudo code of the relevant software is given in the Appendix.

5.1 Formulation of the Problem

As in Section 3.1, we let D be the digraph representing the ore-body block model with the set of vertices $V = \{v_1, v_2, \ldots, v_n\}$

representing the blocks and the set of arcs A, representing the mining restrictions; m_i, the weight of each vertex v_i of D, is the value of the block represented by v_i.

The optimum pit limit problem consists of finding $Y \subset V$ such that $\sum_{v_i \in Y} m_i$ is maximum, subject to Y being a closure of D, i.e. $v_i \in Y$ and $(v_i, v_j) \in A \Rightarrow v_i \in Y$. This problem may also be expressed as a *0-1* programming problem (similar to that given in Section 3.1) as follows. Let

$$x_i = 1 , \qquad \text{if } v_i \in Y ,$$
$$= 0 , \qquad \text{otherwise .}$$

Then, it is easily seen that the problem is equivalent to the problem

$$\begin{aligned} &\text{Max} \quad z = \sum_{i=1}^{n} m_i x_i \\ &x_i \leq x_j , \qquad \text{for } (v_i, v_j) \in A , \\ &x_i = 0, 1, \qquad i = 1, 2, \ldots, n . \end{aligned} \tag{5.1}$$

The condition $x_i \leq x_j$ for $(v_i, v_j) \in A$ is equivalent to the restriction

$$a_{ij} x_i (-1 + x_j) = 0 ,$$

where a_{ij} is the (i, j) element of the **adjacency matrix** of D, i.e.

$$a_{ij} = 1 , \qquad \text{if } (v_i, v_j) \in A ,$$
$$= 0 , \qquad \text{otherwise.}$$

Problem (5.1) can then be written as

$$\text{Max} \quad z = \sum_{i=1}^{n} m_i x_i$$

$$\text{subject to} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i (-1 + x_j) = 0 , \tag{5.2}$$

$$x_i = 0, 1, \qquad i = 1, 2, \ldots, n.$$

Because $a_{ij}x_i(-1+x_j) \leq 0$, for all $x_i = 0,1$ and $x_j = 0,1$, we may use the penalty function concept to express (5.2) as

$$\text{Max} \quad z = \sum_{i=1}^{n} m_i x_i + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i(-1+x_j) \tag{5.3}$$

$$x_i = 0,1, \qquad i=1,2,\ldots,n \ ,$$

where λ is a positive number large enough to ensure that an optimal solution of (5.3) satisfies $\sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i(-1+x_j) = 0$, i.e. that we have a closure of D.

Finally, we may write (5.3) as a minimization problem and obtain the following problem, equivalent to (5.1).

$$\text{Min } g(x) = \sum_{i=1}^{n} -m_i x_i + \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda a_{ij} x_i(-1+x_j) \tag{5.4}$$

$$x_i = 0,1, \qquad i=1,2,\ldots,n \ ,$$

where x is the vector with components x_i, $i=1,2,\ldots,n$.

In the following section we show that problem (5.4) is equivalent to solving a maximum flow problem in a network.

5.2 An Equivalent Maximum Flow Problem

A general treatment of the maximum flow problem may be found in many standard texts such as Bondy and Murty (1976), Even (1979) and Tarjan (1983). However, for the sake of completeness, some of the basics associated with this work is included here.

A **network** is a directed graph $N = (V,A)$ with two distinguished vertices, a **source** s and **sink** t, and with a non-negative function c defined on A. If $a \in A$, $c(a)$ is called the **capacity** of a. For convenience we take $c(a) = 0$ if $a \notin A$. A **flow** f on N is a non-

negative function on A, such that,

(i) $0 \le f(a) \le c(a), \qquad \forall\, a \in A$.

(ii) Let $\alpha(v)$ and $\beta(v)$ be the sets of arcs incoming to vertex v and outgoing from v, respectively. For every $v \in V-\{s,t\}$,

$$\sum_{a\in\alpha(V)} f(a) - \sum_{a\in\beta(v)} f(a) = 0 .$$

The **total flow** F of f is defined by

$$F = \sum_{a\in\alpha(t)} f(a) - \sum_{a\in\beta(t)} f(a).$$

The **maximum flow problem** is to find an f for which F is maximum.

Given a node partition $S,\overline{S}$ of V where $s \in S$, $t \in \overline{S}$, $S\cup\overline{S} = V$ and $S\cap\overline{S} = \phi$. A **cut** $(S,\overline{S})$ of the network N is defined to be the set of arcs of A whose start-vertex is in S and end-vertex is in $\overline{S}$. The **capacity of the cut** $(S,\overline{S})$, denoted $c(S,\overline{S})$, is defined as

$$c(S,\overline{S}) = \sum_{a\in(S,\overline{S})} c(a) .$$

Suppose $V = \{v_0, v_1, \ldots, v_{n+1}\}$, where v_0 is the source and v_{n+1} is the sink for N. For convenience, we express the capacity of the cut $(S,\overline{S})$ as follows:

$$c(S,\overline{S}) = \sum_{i\in I} \sum_{j\in\overline{I}} c_{ij}$$

where $I = \{i \,|\, v_i \in S\}$, $\overline{I} = \{j \,|\, v_j \in \overline{S}\}$ and c_{ij} is the capacity of the arc (v_i, v_j). A **minimum cut** is defined to be a cut with minimum capacity.

Now, any cut of N may be represented by a vector $(x_0, x_1, x_2, \ldots, x_{n+1})$, where $x_j = 0$ or 1 for $j=1,2,\ldots,n$, $x_0 = 1$ and

$x_{n+1} = 0$, and by defining $S = \{v_i | x_i=1\}$ and $\bar{S} = \{v_i | x_i=0\}$, every vector of this form represents some cut $(S,\bar{S})$ of N. Also, the capacity of the cut corresponding to the vector $x = (x_1, x_2, \ldots, x_n)$ in (5.4) and $x_0 = 1$ and $x_{n+1} = 0$, may be represented as

$$c(x) = \sum_{i=0}^{n+1} \sum_{j=0}^{n+1} c_{ij} x_i (1-x_j) \ . \qquad (5.5)$$

Substituting for x_0 and x_{n+1} in (5.5) yields

$$c(x) = \sum_{j=1}^{n+1} c_{oj} + \sum_{i=1}^{n} (c_{i,n+1} - c_{oi}) x_i + \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_i (1-x_j) \ , \qquad (5.6)$$

which has the same form as the expression of $g(x)$ in (5.4); the two expressions differing only by a constant term. Hence, solving problem (5.4) is equivalent to finding a minimum cut in a related network which may be constructed as follows.

Let $I^+ = \{i | m_i>0;\ i=1,2,\ldots,n\}$ and $I^- = \{i | m_i<0;\ i=1,2,\ldots,n\}$. To the graph D, add a source v_o and a sink v_{n+1}. For $i \in I^+$, add an arc (v_o, v_i) with capacity $c_{oi} = +m_i$. For $i \in I^-$, add an arc (v_i, v_{n+1}) with capacity $c_{i,n+1} = -m_i$. To each arc in the original graph D, associate a capacity of value λ. Then the minimum cut formulation for the network N constructed, is exactly (5.4).

Because λ is any large number, we may put λ equal to infinity. By max-flow min-cut theorem (Bondy and Murty 1977, p196), a minimum cut for N (and hence a maximum closure of D) may be found by any maximum flow algorithm.

The remainder of this chapter is devoted to the implementation

of an efficient maximum flow algorithm designed specifically for the characteristic network N which results in the pit limit problem, as formulated above.

5.3 Solution of the Maximum Flow Problem

There are several algorithms available for solving the maximum flow problem; a list of these together with their time complexities, is given in Tarjan (1983, p98). Time complexity, simplicity in its design and development , and required data structures were considered in the choice of one of these algorithms to adapt it specifically for the network arising in the pit limit problem. Dinic's algorithm has time complexity $O(n^2m)$ (where $n = |V|$ and $m = |A|$) and although there are other algorithms with a slightly better time complexity, its efficiency and simplicity of implementation is superior (Cheung 1980; Imai 1983). Before proceeding with the adaption of Dinic's algorithm to solve the pit limit problem, we outline some associated results and steps in the general algorithm (Even 1979).

Let E be the edge set of the underlying graph of N i.e. the graph on the same vertex set V as N and such that to each arc in A there is an edge in E with the same ends. Take the flow and capacity functions f and c on E to be precisely those defined on A, respectively. If an edge $e = uv \in E$ has flow $f(e)$, then we say that e is **useful from** u **to** v if one of the following two conditions holds:

(i) $(u,v) \in A$ and $f(e) < c(e)$.

(ii) $(v,u) \in A$ and $f(e) > 0$.

The **layered network** of N with flow f is defined by the following algorithm:

Step 1 $V_0 := \{s\}$, $i := 0$.

Step 2 Construct $T := \{v \mid v \notin V_j$, for $j \leq i$ and there is a useful edge from a vertex in V_i to $v\}$.

Step 3 If T is empty, the present total flow F is maximum, stop.

Step 4 If T contains t then $\ell := i+1$, set $V_\ell := \{t\}$ and stop.

Step 5 $V_{i+1} := T$, increment i and return to Step 2. □

For each $1 \leq i \leq \ell$, let E_i be the set of edges useful from a vertex in V_{i-1} to a vertex in V_i. The sets V_i are called **layers**.

Consider the layered network $\tilde{N}$ with arcs formed from the edges of E_j directed from V_{j-1} to V_j for $1 \leq j \leq \ell$, and with capacities defined by the function $\tilde{c}$ as follows:

For every edge $e = uv$ in E_j with $u \in V_{j-1}$ and $v \in V_j$,

(i) if $(u,v) \in A$, then $\tilde{c}(e) = c(e) - f(e)$,

(ii) if $(v,u) \in A$, then $\tilde{c}(e) = f(e)$.

We refer to an arc (u,v) of $\tilde{N}$ as a **forward arc** if $(u,v) \in A$, and as a **reverse arc** if $(v,u) \in A$. We define a **maximal flow** $\tilde{f}$ in $\tilde{N}$ as a flow which satisfies the condition that for every path $se_1v_1e_2v_2 \ldots v_{\ell-1}e_\ell t$, where $v_j \in V_j$ and $e_j \in E_j$, there is at least one edge e_j such that $\tilde{f}(e_j) = \tilde{c}(e_j)$. A maximal flow is not necessarily maximum (Even 1979, p98). This maximal flow is essential (as we shall see) in establishing a maximum flow in N.

Dinic's method of constructing a maximal flow in $\tilde{N}$ is as follows: Assume $\tilde{c}(e) > 0$ for each edge e in $\tilde{N}$.

Dinic's Algorithm

Step 1 For each e in $\tilde{N}$, mark e 'unblocked' and set $\tilde{f}(e):=0$.

Step 2 Set $v:=s$ and empty the stack S.

Step 3 If there is no unblocked edge vu, with u in the next layer, then (v is a **dead-end** and) perform the following operations:

Step 3.1 If $v=s$, stop; the present $\tilde{f}$ is maximal.

Step 3.2 Delete the top-most edge $e=uv$ from S.

Step 3.3 Mark e 'blocked' and set $v:=u$.

Step 3.4 Repeat Step 3.

Step 4 Choose an unblocked edge $e=vu$, with u in the next layer. Put e in S and set $v:=u$. If $v\neq t$, then go to Step 3.

Step 5 The edges on S form an **augmenting path** $se_1v_1e_2v_2\ldots v_{\ell-1}e_\ell t$, i.e. a path from s to t through which the present flow may be increased. Perform the following operations:

Step 5.1 $\Delta:= \underset{1\leq i\leq \ell}{\text{Min}} (\tilde{c}(e_i)-\tilde{f}(e_i))$.

Step 5.2 For each $1\leq i\leq \ell$, $\tilde{f}(e_i):=\tilde{f}(e_i) + \Delta$ and if $\tilde{f}(e_i) = \tilde{c}(e_i)$, then mark e_i 'blocked'.

Step 5.3 Go to Step 2. □

Once the maximal flow $\tilde{f}$ with total value $\tilde{F}$ is found for $\tilde{N}$, the flow f in N is updated as follows: For each $e = uv$ in $\tilde{N}$,

(i) if (u,v) is a forward arc, then $f(e):=f(e)+\tilde{f}(e)$,

(ii) if (u,v) is a reverse arc, then $f(e):=f(e)-\tilde{f}(e)$.

The total flow F becomes $F:=F+\tilde{F}$. The process repeats by finding a layered network of N with the new flow f, etc. When the process stops in Step 3, in finding the layered network of N with flow f, the minimum cut is given by $(S,\bar{S})$ where $S = V_0 \cup V_1 \cup \ldots \cup V_i$ (Even 1979, p98).

5.4 Implementation of Dinic's Algorithm

Before implementing Dinic's algorithm, we modify our block model by removing all blocks from the original ore-body model which are not within the bound for the optimum pit, output by BOUND (see Chapter 4). Thus, we redefine D to be the digraph (on V, the set of vertices within the bound) representing the mining restrictions for the modified block model. The network N (as constructed in Section 5.2) for this modified block model, has special structure, that is

(i) All **internal arcs**, that is those arcs of digraph D, have infinite capacities.

(ii) The end-vertices of all the arcs with the source s as start-vertex are the positive-valued vertices of D.

(iii) The end-vertex of all the arcs with the negative-valued vertices of D as start-vertices is the sink t.

(iv) The internal arcs of N are generated by applying the minimum search pattern to each vertex representing a block (see Remark 2.1).

We will refer to those arcs in N which have s or t as an end vertex, as **external arcs**; vertices of D, i.e. those other than s or t, will be referred to as **internal vertices**.

We adopt the following data structures to accommodate the special structure of N. Let

$p, q, p_1 p_2, \ldots$: represent certain vertices of D,

n : the total number of internal vertices,

c_{pq} : the capacity for arc (p,q) in N; take $c_{pq} = 0$ to mean arc (p,q) is not in N,

d : the size of the minimum search pattern.

Figure 5.1 shows the data structures used in the implementation of Dinic's maximum flow method on N. The external arcs (with their capacities) are represented by the arrays *sarc(2,pos)* and *tarc(n)* connected to the source and sink, respectively. The internal arcs of N and $\tilde{N}$ are represented by the arrays *farc(n,d)* (forward arcs) and *rarc(n,d,2)* (reverse arcs). The successor vertices in *farc* and *rarc* are stored such that those associated with useful edges are stored first; a count for the number of these useful edges is stored in arrays *nusef* and *nuser*, respectively. Obviously, infinite capacities are not stored for the forward arcs and the capacities for the reverse arcs are precisely the flows on the corresponding forward arcs in N. Each arc in N has a corresponding reverse arc represented in *rarc* and thus, initially, (when the flow is zero) all reverse arc capacities are set to zero (indicating the absence of such arcs in $\tilde{N}$).

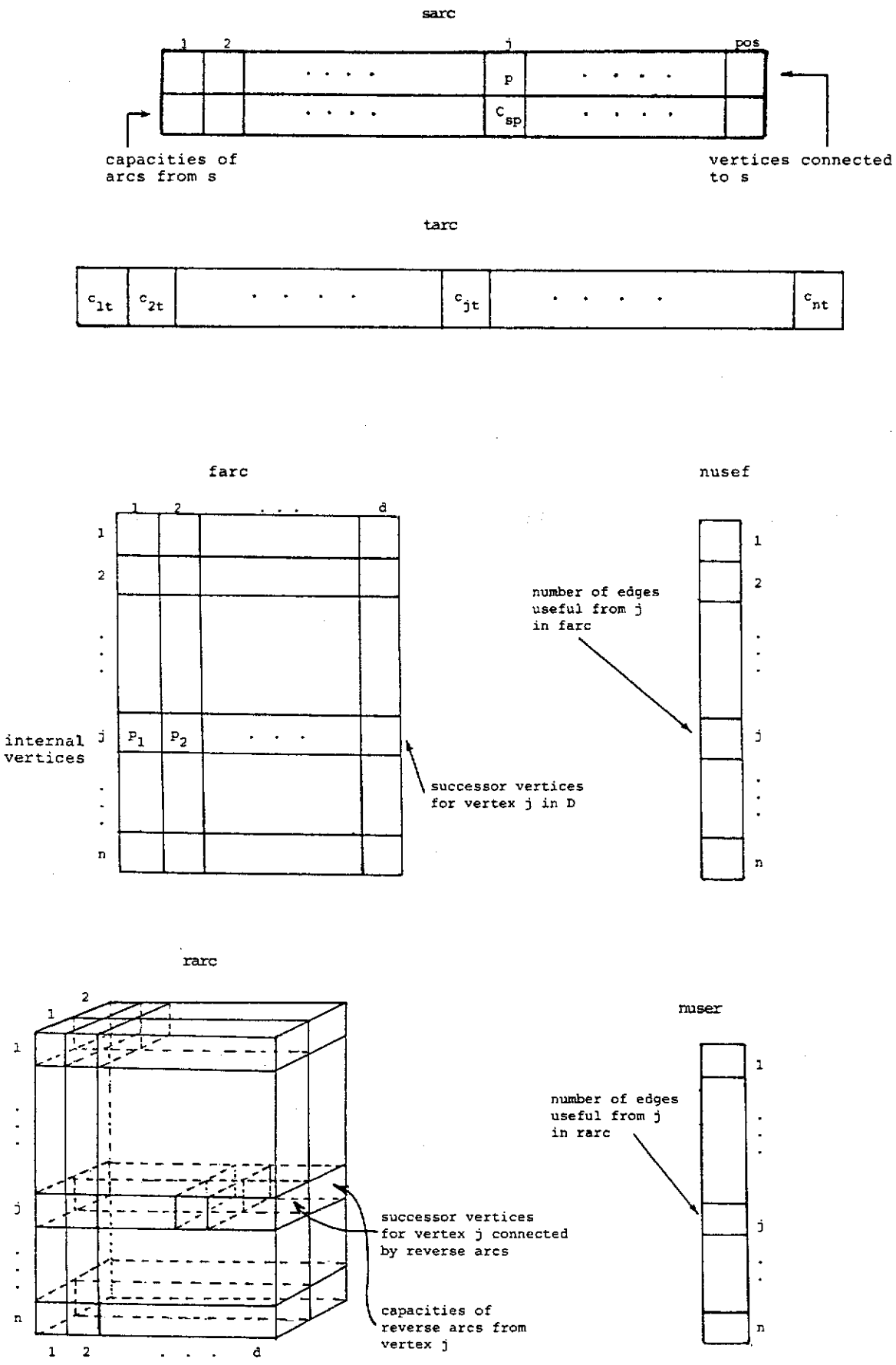


Figure 5.1 Data structures for Dinic's algorithm.

The procedure for implementing Dinic's algorithm on N consists of four main steps as indicated by the flowchart in Figure 5.2. Firstly, the **layer data structure** (not related to layers in the network) is created. This data structure (see Figure 5.3(b)) is used to convert each block 'address' (level, column, section) within the bound, to a vertex label in N; the labels for the internal vertices ranging from 1 to n in the order shown in Figure 5.3(a). For convenience, the sink is labelled 0 and source $n+1$. Next, the arcs of the network are generated and stored (with their corresponding capacities) in arrays *sarc*, *tarc*, *farc* and *rarc*, as indicated in Figure 5.1. This is accomplished using a procedure called **arcgen**, the pseudo code for which is given in the Appendix. Thirdly, the procedure **dinic** is used to apply Dinic's maximum flow algorithm to find the minimum cut for N. The cut $(S,\bar{S})$ is represented by a one-dimensional array *cut*, such that $cut(i) = 1$ or 0, depending on whether vertex v_i is in S or not, respectively. Finally, the blocks within the optimum pit outline are identified using the procedure **pitout** which converts vertex labels (given by array *cut*) back to the block addresses (level, column, section) within the original block model. Pseudo code for both **dinic** and **pitout** are also found in the Appendix.

The four steps, described above, are incorporated in a software system called PITOPTIM. The steps may be executed consecutively and output files created following each step for examination purposes, if desired. Alternatively, the process may be executed in one step. The system also contains various other service routines (pseudo codes for some of these are included in the Appendix) such as input and output modules. Output modules are available to output the

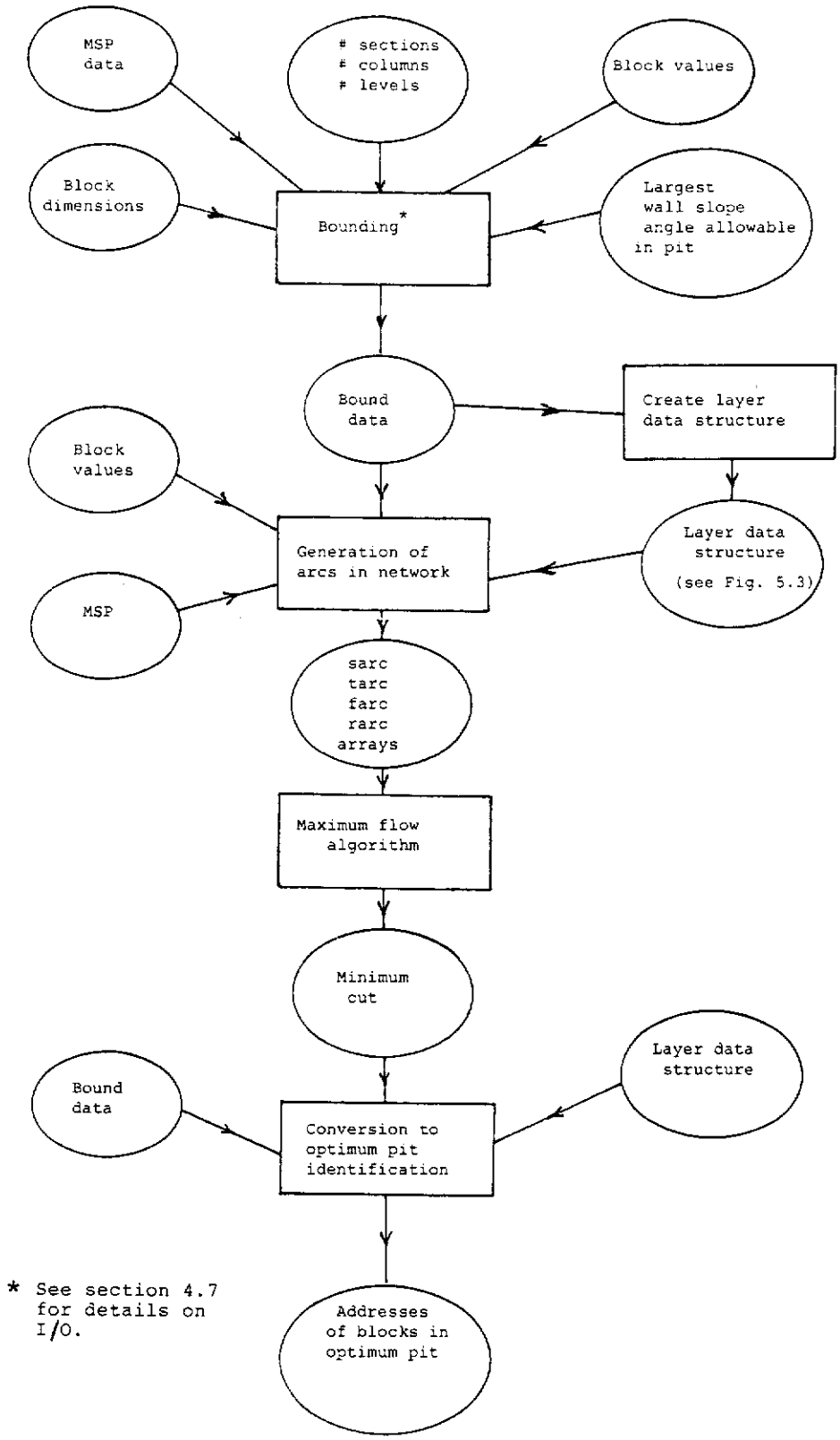


Figure 5.2 Execution steps in the design of an open pit.

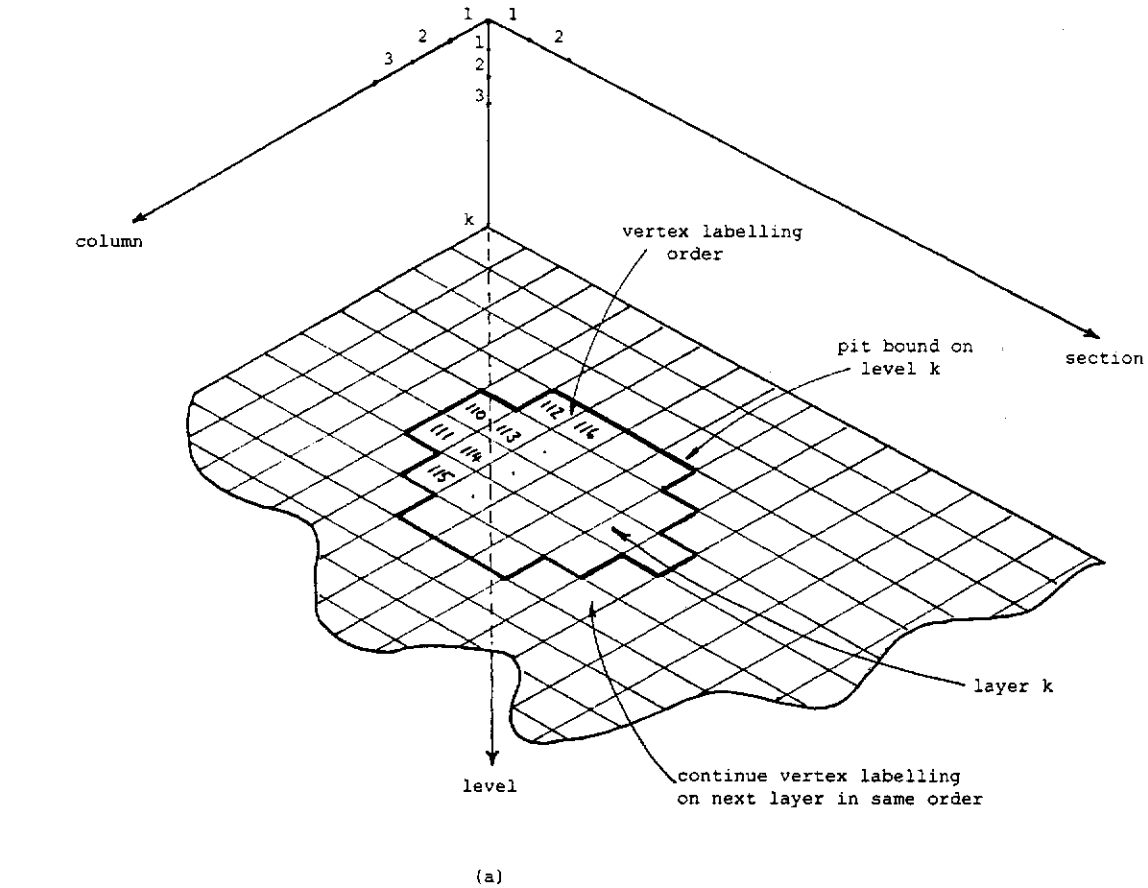


Figure 5.3 **(a)** Block coordinate system and layer definition.
(b) The layer data structure.

layer data structure, the arcs produced during arc generation and the final pit outline. The routines **boundoutline** and **pitoutline** may be used to get a display of the bound and the optimum pit on the line printer for inspection.

The programs in PITOPTIM are written in the C language and were developed on the VAX 785 computer with a UNIX operating system.

5.5 Complexity Analysis

As discussed in Section 5.3, the worst-case time complexity of Dinic's maximum flow algorithm is $O(n^2m)$ where n is the number of vertices and m the number of arcs in the network. The network N associated with the pit limit problem has $m = O(n)$, since each vertex has a maximum out-degree of d. Consequently, the run-time for PITOPTIM is expected to be, at worst $O(n^3)$, comparable to the MDS/Lerchs-Grossman algorithm but without taking normalization into account. However, the run-times shown for the results in Section 5.6 suggest that the time complexity for PITOPTIM is more like $O(n^{1.27})$. The reason for this is evidently due to the special structure of N.

We estimate the space complexity of PITOPTIM as follows. The array *sarc* requires $2\times pos$ locations; *tarc*, *nusef*, *nuser*, *cut* and a working array *aux*, each requires n locations; *farc* requires $n\times d$ and *rarc*, $n\times d\times 2$ locations. Since $pos\leq n$, this amounts to $(7+3d)n$ memory locations (not including the layer data structure which is comparatively negligible). This compares with $(9+3d)n$ memory locations for the MDS/Lerchs-Grossman algorithm assuming absence of normalization and sparsity of the associated arrays (see Section 3.5).

5.6 Computational Results

The bounding and pit optimization software BOUND and PITOPTIM, respectively, were thoroughly tested on data from two gold producing mines in Western Australia. For reasons of confidentiality, we shall refer to the two ore-bodies as ore-body A and ore-body B. Ore-body A was used extensively to obtain technical data on the performance of both BOUND and PITOPTIM. Ore-body B was treated purely as a typical open pit design, as required by the 'client' mining company.

Ore-body A, 185 metres by 260 metres by 45 metres deep, was block modelled using four different block sizes from 5 metre cube to 1.5 metre cube. This led to models of increasingly large number of blocks, ranging from 17,316 to 629,520, covering the broad spectrum of pit sizes from small to large. The geology required a slope definition given by the set R of azimuth and dip pairs, where

$$R = \{(0^\circ,55^\circ),(90^\circ,60^\circ),(180^\circ,55^\circ),(270^\circ,50^\circ)\} .$$

The corresponding MSP, output by the MSP algorithm with tolerance $\varepsilon_1=\varepsilon_2=5^\circ$, is given by the set S, where

$$\begin{aligned} S = \{&(1,0,0),(1,-1,0),(2,-1,-1),(2,-1,1),(2,0,-1),\\ &(2,0,1),(2,1,-1),(2,1,0),(2,1,1),(3,-1,-2),\\ &(3,-1,2),(3,0,-2),(3,0,2)\}. \end{aligned}$$

The programs were tested on a relatively unloaded 4 MIPS machine with 16 MByte of RAM and 220 MByte of virtual memory (approximately equivalent to a SUN 3) using a UNIX operating system. The UNIX time command gives real times, total elapsed time for the process, and "user" times, which reflect the time required to execute the algorithm. These times and the memory requirements for

BOUND and PITOPTIM (with and without BOUND), for each of the block models, along with details of block dimensions, number of blocks within the model, the bound and the optimum pit, are included in Table 5.1.

BLOCK MODEL	Cube size (metre)	5	3	2	1.5
	n	17,316	78,690	263,120	629,520
WITH BOUND	n_b	2,649	13,146	43,196	107,325
	t_b (user)	0.2	1.2	4.8	14
	t_b (real)	0.5	1.7	6.0	17
	t_o (user)	0.5	4.1	24	73
	t_o (real)	0.5	6.3	97	2,240
	memory (Mbytes)	0.7	3.5	12	29
WITHOUT BOUND	t_o (user)	1.0	5.6	36	-
	t_o (real)	1.3	11	1,620	-
	memory (Mbytes)	4.3	20	65	156
	n_o	2,465	11,460	35,962	91,246

n : number of blocks in block model

n_b : number of blocks within the bound

n_o : number of blocks within the optimum pit

t_b (user) : "user" time for BOUND in minutes

t_b (real) : "real" time for BOUND in minutes

t_o (user) : "user" time for PITOPTIM in minutes

t_o (real) : "real" time for PITOPTIM in minutes

Table 5.1 Computing times for BOUND and PITOPTIM.

It can readily be seen that the application of BOUND drastically reduces the total elapsed (real) time and memory

requirements for the large block models. For the 263,120 block model, the total elapsed time for the pit optimization decreased from 27 hours (1620 minutes) to 1.7 hours (97 + 6 = 103 minutes) when used in conjunction with BOUND. The memory requirements were also decreased from 65 Mbytes to 12 Mbytes. The memory and time requirements for the 629,520 block model were such that the pit optimization could only be achieved on this machine with the use of BOUND. It should be noted that for the larger pits, where the memory requirements for PITOPTIM exceed the 16 Mbyte available, the time taken is greatly increased by the requirement of virtual memory.

An examination of the processing (user) time for PITOPTIM revealed that it is $O(n^{1.27})$, where n is the number of blocks. As this processing time for PITOPTIM includes the execution of modules other than the network flow algorithm, the latter is therefore better than $O(n^{1.27})$. This is a lot more favourable than the theoretical prognosis of $O(n^3)$.

Theoretically, the performance of BOUND may be examined by the use of two parameters r (percentage reduction in the number of blocks) and τ (tightness of the bound) as defined by the equations

$$r = \frac{n - n_b}{n} \times 100$$

and

$$\tau = \frac{n_b - n_o}{n - n_o} \times 100 \ ,$$

where n, n_b and n_o are defined as follows:

n : number of blocks within the block model,

n_b : number of blocks within the bound,

n_o : number of blocks within the optimum pit.

Note that the tighter the bound, the closer τ is to zero and when no bound exists, $\tau = 100$. These two parameters, r and τ, are displayed for the different block models in Table 5.2. Note that r varies from 83 to 85 percent and τ from 1.2 to 3.0 percent, indicating the proficiency of the data reducing algorithm BOUND. Note also that the running time for BOUND, even for the largest block model of 629,520 blocks, was no more than 17 minutes.

cube size (metre)	5	3	2	1.5
n	17,316	78,690	263,120	629,520
n_b	2,649	13,146	43,196	107,325
n_o	2,465	11,460	35,962	91,246
r	84.7	83.3	83.6	83.0
τ	1.2	2.5	3.2	3.0

n : number of blocks in block model

n_b : number of blocks within the bound

n_o : number of blocks within the optimum pit

r : percentage reduction in the number of blocks

τ : tightness of the bound

Table 5.2 Performance of BOUND.

Ore-body B, approximately 250 metres by 569 metres by 66 metres deep, had a 40° maximum wall slope requirement throughout the pit due to its unstable geology. A block height of 3 metres was also stipulated for its block model. As the other two dimensions of the block were not of importance in the operations associated with the mining of the ore-body, they were chosen to economize on the size of

the MSP associated with the slope requirement of the optimum pit walls. The dimensions of the base of the block were made 3.58 metres by 3.58 metres. These dimensions give the diagonal of each vertical face of the block an angle, of approximately 40°, consistent with the wall slope requirement of the pit. This block shape best suits the design of the pit walls through block removals. The MSP, with a tolerance of 4°, resulting from this block shape, is the expected 'knight's move' pattern (see Figure 2.8). The total number of blocks resulting in the block model was 244,860.

Application of BOUND to the block model of ore-body B, resulted in only 9,364 blocks to be considered (a reduction of 96%), of which, 5,529 were in the optimum pit contour output by PITOPTIM. The total (user) time required to design the pit amounted to 7.8 minutes, and required only 3.3 MBytes of memory. Application of PITOPTIM, without BOUND, resulted in the same pit contour in 34.3 minutes and requiring 61 MBytes of memory!

5.7 Conclusion

The concepts presented in this thesis, namely, the minimum search pattern, the bounding technique and the network flow application, combined, form a powerful software system for the efficient optimum open pit design. This is readily evident from the results given in the previous section.

It was not the aim of this thesis to give comparative performances of the various pit optimization techniques available, but to successfully address the computational problems associated with the implementation of these techniques, particularly when applied to large ore-bodies. However, through the concepts

presented here, we are now capable of designing optimum pits, as much as, seven times larger on a given computer than the current state-of-the-art software and at a small fraction of the currently required computer time . As an example, Alford and Whittle (1986) reports a pit designed, on a small 30,000 block model, in 3 hours real time, using the Lerchs-Grossman algorithm. Using BOUND and PITOPTIM, we show we can design two pits of over one-quarter of a million (263,120) blocks within this time using only 12 MBytes of memory (see Table 5.1). Even if BOUND is not used, PITOPTIM, which uses a network flow technique, required only 11 minutes total elapsed time to design a pit in a 78,690 block model. This suggests, as shown theoretically, that the Lerchs-Grossman algorithm is not as efficient as our implementation of Dinic's network flow optimization technique.

APPENDIX

PSEUDO CODE FOR THE MAJOR MODULES

IN

PITOPTIM

Global Variables :

integer :

```
        source          *source = n+1
        sink            *sink = 0
        n               *Number of internal vertices in the
                        *graph (ie. not the source or the sink)

        d               *Number of vertices in the minimum
                        *search pattern

        pos             *Number of positive-valued vertices in
                        *the graph

        sarc(2,pos)     *sarc(1,pos) contains the vertices of
                        *the graph which have positive value (ie
                        *are connected to the source).
                        *sarc(2,pos) contain the capacities of
                        *arcs from the source to the vertices

        tarc(n)         *Contains the capacities of the arcs of
                        *the graph connected to the sink.  If a
                        *vertex p is not connected to the sink
                        *then its associated arc capacity,
                        *tarc(p) = 0

        farc(n,d)       *Two dimensional array of forward arcs.
                        *For a vertex p, farc(p,d) contains all
                        *the vertices to which p connects with
                        *forward arcs. The capacities of these
                        *arcs are infinite.

        rarc(n,d,2)     *Three dimensional array of reverse arcs.
                        *For a vertex p, rarc(p,d,1) contains all
                        *the vertices to which p connects with
                        *reverse arcs.  rarc(p,d,2) contains the
                        *respective capacities.

        nuses           *Number of edges useful from the source.
        nusef(n)        *Nusef(p) is the number of forward arcs
                        *useful from vertex p.

        nuser(n)        *Nuser(p) is the number of reverse arcs
                        *useful from a vertex p.
```

```
cut(n)          *During the labelling and scanning in
                *create_layered_network() this contains
                *the labels of the vertices.  When the
                *labelling and scanning has finished
                *cut(p) contains the number of useful
                *edges from vertex p.  On completion
                *of Dinic's algorithm it contains the
                *"cut" : cut(p)=1 if p ∈ S ie. p is in
                *the optimum pit, otherwise cut(p)=0.

aux(n)          *During create_layered_network() aux(n)
                *contains the vertices in the order in
                *which they are scanned.  In construct_
                *path it contains the path from the
                *source to the sink.

nmin            *Large negative integer used to denote
                *unlabelled vertices = -n-1.

lblsnk          *The layer in which the sink is
                *labelled.

floval          *Total flow through the network.
levext          *Maximum value of level in the block
                *model

sectext         *Maximum value of section in the block
                *model

colext          *Maximum value of column in the block
                *model

blkval(lev,col,sect)
                *Bblock value at (lev,col,sect)
bottom          *Maximum level within the bound
leftbnd(sect,lev)
                *Left bound
rightbnd(sect,lev)
                *Right bound
layer[][]       *Data structure containing the
                *information required to relate block
                *coordinates with the labels of the
                *vertices of the graph
```

```
Module main()
*********************************************************************
*                                                                   *
*  Mainline for the modified Dinic's algorithm                      *
*                                                                   *
*********************************************************************

local variables:
logical : finished

finished := false
do while (not finished)
  finished := create_layered_network()
  * create_layered_network returns TRUE if the next layered
  * network could not be created
  if (not finished)
    * construct path and update capacities in the original network
    construct_path_and_update_capacities()
  else           *finished
    * get cut
    for i from 1 to n do
      cut(i) := minimum(1,cut(i)-nmin)          *=i for i ∈ S
                                                *=0 otherwise
    end for
    output cut()
  end if
end do
end module main()


Module create_layered_network()
*********************************************************************
*                                                                   *
*  This module creates the next layered network.  As the            *
*  vertices are labelled, cut(p) = label and aux(wr) contains       *
*  the vertices in the order of labelling.  aux(1) is the first     *
*  vertex labelled from the source.  Following the scanning of      *
*  a vertex cut(p) = number of useful edges from p.  When the       *
*  sink is reached the layered network has been created.  If it     *
*  is impossible to get to the sink then TRUE is returned.          *
*                                                                   *
*********************************************************************

local variables:
integer : rd          *Number of vertices read from aux(n).
          wr          *Number of vertices scanned and written to
                      *aux(n).

          p, q        *Vertices.
          label       *Label of vertex.
          m           *Number of useful edges from a vertex.
          i           *Index.
          cap         *Capacity of an arc.
```

```
* scanning and labelling
lblsnk := n
* set cut(i) to unlabelled
for i from 1 to n do
  cut(i) := nmin
end for
* initialise nuses, nusef, nuser
nuses := 0
for i from 1 to n do
  nusef(i) := 0
  nuser(i) := 0
end for
rd := 0
wr := 0

* start with source
p := source
label := -1
do while (cut(p) + lblsnk ≠ 0)
  * scan external arcs; p is never the sink
  if (p = source) then
    m := pos
    i := 1
    do while (i ≤ m)          *none of these q's are 0 (undefined)
      q := sarc(1,i)
      cap := sarc(2,i)
      if (cap = 0) then
        * swap
        sarc(1,i) := sarc(1,m)
        sarc(2,i) := sarc(2,m)
        sarc(1,m) := q
        sarc(2,m) := cap
        m := m-1
      else
        case (cut(q) - label)
          <0  cut(q) := label          *unlabelled
              wr := wr+1
              aux(wr) = q
              i := i+1
          =0  i := i+1                 *already labelled in this
                                       *layer
          >0  *swap                    *labelled in previous layer
              sarc(1,i) := sarc(1,m)
              sarc(2,i) := sarc(2,m)
              sarc(1,m) := q
              sarc(2,m) := cap
              m := m-1
        end case
      end if
    end do
    nuses := m
  else                                  *p ≠ source
    * look for the sink first
    cap := tarc(p)
    * the arc to the sink is only useful if cap > 0
```

```
  if (cap > 0) then
    q := sink
    lblsnk := -label
    if (cut(q) - label < 0) then        *if the sink hasn't been
      cut(q) := label                   *labelled then label it
      wr := wr+1
      aux(wr) := q
    end if
  end if
end if
* scan internal vertices only if p is not in the 2nd top layer
if (p ≠ source and cut(p)-1+lblsnk ≠ 0) then
  * forward arcs (capacitites = ∞)
  m := d
  i := 1
  do while (i ≤ m)
    q := farc(p,i)
    if (q = 0) then                     *q nonexistent vertex -
                                        *if there are less than d
                                        *neighbours farc(p,i) will
                                        *contain some zeros
      *swap
      farc(p,i) := farc(p,m)
      farc(p,m) := q
      m := m-1
    else
      case  (cut(q)-label)
      <0  cut(q) := label               *unlabelled
          wr := wr+1
          aux(wr) := q
          i := i+1
      =0  i := i+1                      *already labelled in this
                                        *layer
      >0  *swap                         *labelled in previous layer
          farc(p,i) := farc(p,m)
          farc(p,m) := q
          m := m-1
      end case
    end if
  end do
  nusef(p) := m
  *reverse arcs
  m := d
  i := 1
  do while (i ≤ m)
    q := rarc(p,i,1)
    cap := rarc(p,i,2)
    if (q = 0) then                     *q nonexistent vertex
      *swap
      rarc(p,i,1) := rarc(p,m,1)
      rarc(p,i,2) := rarc(p,m,2)
      rarc(p,m,1) := q
      rarc(p,m,2) := cap
      m := m-1
    else
      if (cap = 0) then
        *swap
        rarc(p,i,1) := rarc(p,m,1)
        rarc(p,i,2) := rarc(p,m,2)
        rarc(p,m,1) := q
```

```
            rarc(p,m,2) := cap
            m := m-1
          else
            case  (cut(q)-label)
            <0  cut(q) := label          *unlabelled
                wr := wr+1
                aux(wr) := q
                i := i+1
            =0  i := i+1                 *already labelled in this
                                         *layer
            >0  *swap                    *labelled in previous layer
                rarc(p,i,1) := rarc(p,m,1)
                rarc(p,i,2) := rarc(p,m,2)
                rarc(p,m,1) := q
                rarc(p,m,2) := cap
                m := m-1
            end case
          end if
        endif
      end do
      nuser(p) := m
    end if

    * p has been scanned so set cut(p) to the # usable edges
    if (p = source) then
      cut(p) := nuses
    else
      if (tarc(p) > 0) then
        cut(p) := nusef(p)+nuser(p)+1
      else
        cut(p) := nusef(p)+nuser(p)
      end if
    end if

    * get next p to be scanned
    rd := rd+1
    if (rd > wr)
      return TRUE
    else
      p := aux(rd)
      label := cut(p)-1
    end if
  end do while (cut(p) + lblsnk ≠ 0)
  end module create_layered_network()
```

```
Module construct_path_and_update_capacities()
*****************************************************************
*                                                               *
*  This module contructs all possible paths in the layered      *
*  network created by create_layered_network(). It augments     *
*  the flow through the path and updates the capacities along   *
*  the path using update_path_cap().                            *
*                                                               *
*****************************************************************

local variables:
logical : maximal     *TRUE if the flow through the layered network
                      *is maximal

          path        *TRUE if a path through the layered network
                      *has been constructed.

integer : p, q        *Vertices.
          k           *Index of aux(n) which contains the path.
          k0          *First blocked edge along the path.
          m           *Number of edges useful from a vertex.
          delta       *Amount by which the flow along a particular
                      *path can be incremented.


maximal := false
path := false
* start with source
q := source
k := 0
do while (not maximal)
  do while (not maximal and not path)
    * construct path from q to sink
    k = k+1
    aux(k) := q
    if (k > lblsnk) then              *sink will be at lblsnk+1
      if (q ≠ sink) then
        k := k-1                      *backtrack
        p := aux(k)
        cut(p) := cut(p)-1
        * if q is in the same layer as the sink then p must be
        * an internal arc, # layers always ≥ 3 ie. p ≠ source
        if (p < q) then
          nuser(p) := nuser(p)-1
        else
          nusef(p) := nusef(p)-1
        end if
        m := cut(p)
      else    *q = sink
        path := true
      end if
```

```
    else
      p := aux(k)
      m := cut(p)
    end if

    do while (m=0 and not maximal and not path)      *backtrack
      k := k-1
      if (k = 0) then
        maximal := true
      else
        p := aux(k)
        cut(p) := cut(p)-1
        if (p = source) then
          nuses := nuses-1
        else
          if (p < q) then
            nuser(p) := nuser(p)-1
          else if (p > q and q ≠ sink)
            nusef(p) := nusef(p)-1
          end if
        end if
        m := cut(p)
        if (m = 0) then        *need new q
          q := p
        end if
      end if
    end do while (m=0 and not maximal and not path)

    * get q
    if (not maximal and not path) then       *m cannot be 0
      if (p = source) then
        m := nuses
        q := sarc(1,m)
      else     *p ≠ source, q could be the sink or an
               *internal vertex
        * check to see if the sink is available
        if (tarc(p) > 0) then
          q := sink
        else
          m := nusef(p)
          if (m ≠ 0) then
            q := farc(p,m)
          else
            m := nuser(p)
            q := rarc(p,m,1)
          end if
        end if
      end if
    end if
  end do

  * have path;  k = lblsnk+1, aux(lblsnk+1) = sink,
  * augment flow through path
  if (path) then
    * get minimum capacity along the path
    delta := capacity(aux(1),aux(2))
    for i from 2 to k-1
      if (aux(i+1) = sink or aux(i) < aux(i+1)) then
                                              *excludes forward arcs
        delta := minimum(delta,capacity(aux(i),aux(i+1))
```

```
      end if
    end for

    * update flow and capacities
    k0 := update_path_cap(k,delta)

    floval := floval + delta
    * get new path from first blocked edge
    * ie start with p:=aux(k0)
    path := false
    k := k0-1
    q := aux(k0)
  end if (path)
end do while (not maximal)
end module construct_path_and_update_capacities()

Module update_path_cap(k,delta)
*************************************************************************
*                                                                       *
*  This module  updates  capacities along a  path  through the          *
*  layered network.                                                     *
*                                                                       *
*************************************************************************

local variables :
integer : k          *Index of aux(n), the path though the layered
                     *network.

          delta      *Amount by which the flow though a path can
                     *be incremented.

          p, q       *Vertices.
          c          *Capacity.
          j, i       *Indices.

* update flow and capacities
do while (k > 1)
      k := k-1
      p := aux(k)
      q := aux(k+1)
      *if capacity(p,q)-delta = 0 then k0=k
      *and cut(p) := cut(p)-1
      if (p=source or q=sink or p<q) then     *bypass subtracting
                                              *delta from infinite
                                              *capacitites
        * update capacities
        if (p = source) then
          j := nuses
          sarc(2,j) = capacity(p,q)-delta
        else if (q = sink) then
          tarc(p) := capacity(p,q)-delta
```

```
        else    *p<q and so (p,q) is a reverse arc
          j=nuser(p)
          rarc(p,j,2) := capacity(p,q)-delta
        end if

        if (capacity(p,q)-delta ≤ 0) then
          cut(p) := cut(p)-1
          if (p = source) then
            nuses := nuses-1
          else if (p ≠ source and q ≠ sink) then
            nuser(p) := nuser(p)-1
          end if
          k0 := k
        end if

      else    *p≠source and q≠sink and p>q
        *(p,q) is a forward arc and so update corresponding
        *reverse arc (q,p) capactity
        j := 0
        for i from 1 to d do          *linear search
          if (rarc(q,i,1) = p) j=i
        end for
        *update capacity
        rarc(q,j,2) := rarc(q,j,2)+delta
      end if
end do
end module update_path_cap()

Module capacity(p,q)
**********************************************************************
*                                                                    *
*  Finds the capacity of an arc from p to q.                         *
*                                                                    *
**********************************************************************
local variables :
integer : p, q       *Vertices.
          cap        *Capacity.

if (p = source) then
  cap := sarc(2,nuses)
else                  *p ≠ source
  if (q = sink) then
    cap := tarc(p)
  else                *internal arc
    if (p < q) then   *reverse arc
      cap := rarc(p,nuser(p),2)
    end if
  end if
end if
return cap
end module capacity()
```

```
Module minimum(x,y)
*****************************************************************
*                                                               *
*  Returns the minimum of x and y.                              *
*                                                               *
*****************************************************************
local variables :
integer : x, y

if (x ≤ y) then
  return x
else
  return y
end if
end module minimum()


Module arcgen()
*****************************************************************
*                                                               *
*  arcgen() creates the data structures sarc, tarc, farc and    *
*  rarc and returns the number of blocks within the bound with  *
*  positive value.  It uses the module f(lev,col,sect) which    *
*  returns the label of the vertex corresponding to the block   *
*  at (lev,col,sect).                                           *
*                                                               *
*****************************************************************

local variables:
integer :
          i, j, k          *Indices
          sarc_index       *Number of vertices with positive value
          col              *Column
          sect             *Section
          lev              *Level
          firstsect        *First section within bound
          lastsect         *Last section within bound
          label            *Label of vertex
          weight           *Value of block
          newcol           *New column
          newsect          *New section
          newlev           *New level
          newweight        *New block value
          neighbour        *Index for vertex in minimum search
                           *pattern

          newlabel         *New label of vertex
          countforward     *Number of forward arcs which point
                           *outside total mining volume.
```

```
* allocate space for sarc, tarc, farc and rarc
allocate sarc(2,n)    *reallocate to sarc(2,pos) at end of
                      *algorithm
allocate tarc(n)
allocate farc(n,d)
allocate rarc(n,d,2)

* initialize tarc(), farc(), rarc()
for i from 1 to n do
  tarc(i) := 0
  for i from 1 to d do
    farc(i,j) := 0
    for k from 1 to 2 do
      rarc(i,j,k) := 0
    end do
  end do
end do

countforward := 0
sarc_index := 0

* put values in sarc, tarc, farc and rarc
* bottom is the highest value of level
for lev from 1 to bottom do
  * Pit is connected.  Uneven surface is allowed for by setting
  * block value = 0 (air blocks).
  sect := 1
  do while (leftbnd(sect,lev) = -1 and sect ≤ sectext)
    sect := sect+1
  end do
  firstsect := sect
  do while (leftbnd(sect,lev) ≠ -1 and sect ≤ sectext)
    sect  := sect+1
  end do
  lastsect := sect-1

  for sect from firstsect to lastsect do
    for col from leftbnd(sect,lev) to rightbnd(sect,lev) do
      * get weight of block
      weight := blkval(lev,col,sect)
      if (weight ≠ 0) then
        * get label of block
        label := f(lev,col,sect)
        * put block in an external arc array sarc(2,n) or
        * tarc(n)
        if (weight < 0) then        *vertex connects to sink
          tarc(label) := -weight
        else                        *vertex connects to source
          sarc_index := sarc_index+1
          sarc(1,sarc_index) := label
          sarc(2,sarc_index) := weight
        end if
        * put block in farc and rarc
        * locate internal forward and reverse arcs connected to
        * block, put relevant info in farc() and rarc()
        * apply msp to lev,col,sect to find forward arcs
        * need to check if within bound for forward arcs, bounding
        * algorithm may give bound which intersects with edge,
        * ie. not a feasible pit outline.
        neighbour := 0
```

```
for each (l,i,j) in msp
  neighbour := neighbour+1
  newlev := lev-l
  if (newlev > 0) then
    newsect := sect+j
    * check newsect within bounds
    if (newsect >0 and newsect ≤ sectext)
      * newsect within bound
      newcol := col+i
      * check newcol within bounds
      if (newcol ≥ leftbnd(newsect,newlev) and
        newcol ≤ rightbnd(newsect,newlev) and newcol ≠ -1)
        * newcol within bounds
        newweight := blkval(newlev,newcol,newsect)
        if (newweight ≠ 0) then
          newlabel := f(newlev,newcol,newsect)
          farc(label,neighbour) := newlabel
        end if
      else
        * newcol outside bound
        if ((newcol ≥ 1 and newcol < leftbnd(newsect,newlev))
          or (newcol > rightbnd(newsect,newlev) and
               newcol ≤ colext))
          * newcol outside bound but within total volume
          output ("forward arc newcol is out of bound but
                  within total volume, SOMETHING IS VERY
                  WRONG-should be fatal-check input of msp
                  data")
        else
          * newcol outside bound and outside total volume
          countforward := countforward+1
        end if
      end if *newcol
    else
      * newsect outside bound
      countforward := countforward + 1
    end if *newsect
  end if *newlev
end for each

* apply msp in reverse to lev,col,sect to find reverse
* arcs.  Need to check if within bound for reverse arcs
* as a lot of them will be out of bound, this is OK.
neighbour := 0
for each (l,i,j) in msp
  neighbour := neighbour+1
  newlev := lev+l
  if (newlev ≤ bottom) then
    newsect := sect-j
    * check sect within bounds
    if (newsect > 0 and newsect ≤ sectext)
      newcol := col-i
      * check newcol within bounds
      if (newcol ≥ leftbnd(newsect,newlev) and
            newcol≤rightbnd(newsect,newlev) and newcol≠ -1)
        newweight := blkval(newlev,newcol,newsect)
        if (newweight ≠ 0) then
          newlabel := f(newlev,newcol,newsect)
          rarc(label,neighbour,1) := newlabel
        end if
```

```
            end if *newcol
          end if *newsect
        end if *newlev
      end for each
    end if *(weight ≠ 0)
  end for *col
 end for *sect
end for *lev from 1 to bottom do

* output error message if forward arcs were found which went out
* of total volume
if (countforward > 0)
  output ( countforward "forward arcs went out of total volume.
           Not enough padding on data.
           Bound from bounding algorithm is not a feasible pit
           outline.  So the optimum pit output by Dinic's
           algorithm may not be the one that should be mined.  Go
           back to the mining engineers and get more padding on
           the data.  This SHOULD have been done during the
           bounding.")

* reallocate memory for sarc to reduce its size
reallocate sarc(2,n) to sarc(2,sarc_index)

return (sarc_index)
end module arcgen()

Module createlayer()
*************************************************************************
*                                                                       *
*  This module uses leftbnd() and rightbnd() from the bounding          *
*  algorithm to set up the data structure layer.  Layer has a           *
*  pointer for each level to a block of data  which contains            *
*  cumulative sums of the number of blocks in each section of           *
*  that level within the bound.                                         *
*  ie  layer[0][0] = bottom                                             *
*      for lev = 1 to bottom                                            *
*      layer[lev][0] = first section within bound in this level         *
*      layer[lev][1] = last section within bound in this level          *
*      layer[lev][2] = total # of blocks in previous levels             *
*      layer[lev][3] = # blocks in 1st section within bound             *
*      layer[lev][4] = # blocks in 1st+2nd section within bound         *
*          ..                                                           *
*      layer[lev][m+2] = # blocks in 1st+2nd+ ... +mth section          *
*          ..                                                           *
*          ..                                                           *
*************************************************************************
```

```
local variables:
integer:

lev                 *Level
sect                *Section
total_blocks        *Cumulative number of blocks within bound
blocks_this_level   *Cumulative number of blocks in current level
firstsect           *First section within bound in current level
lastsect            *Last section within bound in current level
number_sects        *Number of sections within bound in current
                    *level

sect_count          *Index for sections within bound in current
                    *level

number_cols         *Number of columns within bound in this
                    *section in this level


* initialize total_blocks and blocks_this_level
total_blocks := 0
blocks_this_level := 0

* allocate space for layer (as per figure 4)
allocate layer as a pointer to (bottom+1) integer pointers
*put bottom in layer[0][0]
allocate layer[0] to point at an integer
layer[0][0] := bottom

*put relevant values in layer[lev]
for lev from 1 to bottom do
  total_blocks := total_blocks + blocks_this_level
  sect := 1
  do while (leftbnd(sect,lev)) = -1 and sect ≤ sectext)
      sect := sect + 1
  end do
  firstsect := sect

  sect := sect - 1
  do while (sect < sectext)
      sect := sect + 1
      if (leftbnd(sect,lev) ≠ -1) lastsect := sect
  end do

  number_sects := lastsect - firstsect + 1    *number of sections
                                              *within bound in
                                              *this level
  * allocate space for information about this level
  allocate layer[lev] to point to number_sects+3 integers
  layer[lev][0] := firstsect
  layer[lev][1] := lastsect
  layer[lev][2] := total_blocks
  blocks_this_level := 0
  sect_count := 0

  for sect from firstsect to lastsect do
    sect_count := sect_count + 1
    if (leftbnd(sect,lev) ≠ -1)
      number_cols := rightbnd(sect,lev) - leftbnd(sect,lev) + 1
```

```
    else
      number_cols := 0
    endif
    blocks_this_level := blocks_this_level + number_cols
    layer[lev][sect_count+2] := blocks_this_level
  end for
end for

* return total number of blocks within bound
return (total_blocks + blocks_this_level)
end module createlayer()



Module f(lev,col,sect)

*********************************************************************
*                                                                   *
*  This module assigns a label to each of the blocks within the *
*  bound which are used to identify the vertices of the graph.  *
*                                                                   *
*********************************************************************

local variables:
integer:

lev         *Level
col         *Column
sect        *Section
m           *Index for section within bound
label       *The integer value given to the vertex


if ((leftbnd(sect,lev) ≤ col) and (col ≤ rightbnd(sect,lev))
  m := sect - layer[lev][0] + 1
  if (m > 1)
    label := layer[lev][2] +          * # blocks in previous layers
             layer[lev][m+1] +        * # blocks in sections up to
                                      * this one
             col - leftbnd(sect,lev) + 1
                                      * # blocks in this section up
                                      * to and including col
  else
    label := layer[lev][2] +          * #blocks in previous layers
             col - leftbnd(sect,lev) + 1
                                      * # blocks in this section up
                                      * to and including col
  endif
  return (label)
else
  output("***error: block", lev, col, sect, "is not within bound")
  return (NULL)
endif

end module f()
```

```
Module pitout()

*********************************************************************
*                                                                   *
*  This algorithm creates the "cut" in terms of the coordinates     *
*  of the block model                                               *
*  ie   cut[label] → cut[level][column][section].                   *
*                                                                   *
*********************************************************************

local variables:
integer:

i ,j, k          *Indices
bottom           *Maximum level within the bound
totalblocks      *Number of blocks within the bound
l                *Level
s                *Section
levlimit         *Number of blocks in all levels up to and
                 *including the current level

lastlevlimit     *Levlimit for previous level
sectlimit        *Number of blocks in all sections in all levels
                 *up to and including the current section

label            *Vertex label
columns          *Number of columns from left bound
m                *Index for section
lev              *Level
col              *Column
sect             *Section

* NB: arrays are indexed from zero as in the 'c' programs


* allocate space for cutarray[][][] which has the same dimensions
* as blkval[level][column][section]
allocate cutarray[levext][colext][sectext]

* initialize cutarray[][][]
for k = 0 to sectext-1 do
  for i = 0 to levext-1 do
    for j = 0 to colext-1 do
      cutarray[i][j][k] := 0
    end for
  end for
end for

bottom := layer[0][0]
m := layer[bottom][1]-layer[bottom][0]+1
totalblocks := layer[bottom][2]+layer[bottom][m+2]

l := 1                *start at level 1
s := 1                *start at first section within bound
levlimit := layer[l+1][2]          *number of blocks in level 1
lastlevlimit := 0
sectlimit := layer[l][s+2]         *number of blocks in sections
                                   *up to and including this one
for label = 1 to totalblocks do
  if (label > levlimit) then
```

```
    lastlevlimit := levlimit
    * increase levlimit
    l := l+1
    if (l < bottom) then
      levlimit := layer[l+1][2]
    else
      levlimit := totalblocks
    end if
    * reset section limit
    s := 1
    sectlimit := lastlevlimit + layer[l][s+2]
  end if
  lev := l                              *got level

  do while (label > sectlimit)
    * increase sectlimit
    s := s+1
    sectlimit := lastlevlimit + layer[l][s+2]
  end do
  sect := s + layer[l][0] - 1           *got section

  * get column
  if (s > 1) then
    columns := label - (layer[lev][2] + layer[lev][s+1])
  else
    columns := label - layer[lev][2]
  end if

  col := leftbnd[sect-1][lev-1] + columns - 1
  * put value in cutarray
  cutarray[lev-1][col-1][sect-1] := cut[label]
end for

end module pitout()
```

REFERENCES

Albach, H. (1967). Long range planning in open-pit mining. *Management Science*, 13 ppB549-B568.

Alford, C.G. and Whittle, J. (1986). Application of Lerchs-Grossman pit optimization to the design of open pit mines, in Davidson, J.R. (ed), *Australasian Institute of Mining and Metallurgy Symposium 48*, pp201-207. AusIMM, Parkville, Vic.

Barnes, R.J. and Johnson, T.B. (1982). Bounding techniques for the ultimate pit limit problem. *Proceedings 17th APCOM Symposium of the Society of Mining Engineers*, pp263-273. AIME, New York.

Bellman, R.E. (1957). *Dynamic Programming*. Princeton University Press, Princeton, N.J.

Bellman, R.E. and Dreyfus, S.E. (1962). *Applied Dynamic Programming*. Princeton University Press, Princeton, N.J.

Bondy, J.A. and Murty, U.S.R. (1977). *Graph theory with applications*. Macmillan, London.

Caccetta, L. and Giannini, L.M. (1990). Application of operations research techniques in open pit mining, in Byong-Hun Ahn (ed), *Asian-Pacific Operations Research: APORS '88*, pp707-724. Elsevier Science Publishers B.V., North-Holland.

Caccetta, L. and Giannini, L.M. (1988a). An application of discrete mathematics in the design of an open pit mine. *Discrete Applied Mathematics*, 21 pp1-19.

Caccetta, L. and Giannini, L.M. (1988b). The generation of minimum search patterns in the optimum design of open pit mines. *Proceedings of the Australasian Institute of Mining and Metallurgy*, 293(5) pp57-61.

Caccetta, L. and Giannini, L.M. (1986a). Optimization techniques for the open pit limit problem. *Proceedings of the Australasian Institute of Mining and Metallurgy*, 291(8) pp57-63.

Caccetta, L., Giannini, L.M. and Carras, S. (1986b). The optimum design of large open pit mines, in Davidson, J.R. (ed), *Australasian Institute of Mining and Metallurgy Symposium 48*, pp190-195. AusIMM, Parkville, Vic.

Caccetta, L. and Giannini, L.M. (1985). On bounding techniques for the optimum pit limit problem. *Proceedings of the Australasian Institute of Mining and Metallurgy*, 290(4) pp87-89.

Calvert, J.E. and Voxman, W.L. (1989). *Linear Programming*. Harcourt Brace Jovanovich, Orlando, Florida.

Chen, T. (1976). 3D pit design with variable wall slope capabilities. *Proceedings 14th APCOM Symposium of the Society of Mining Engineers*, pp615-625. AIME, New York.

Cheung, To-Yat (1980). Computational comparison of eight methods for the maximum network flow problem. *ACM Transactions on Mathematical Software*, 6(1) pp1-16.

Couzens, T.R. (1979). Aspects of production planning: operating layout and phase plans, in Crawford, J.T. and Hustruld, W.A. (eds), *Open pit mine planning and design*, pp218-231. AIME, New York.

Dagdalen, K. and Francois-Bangarcon, D. (1982). Towards the complete double parameterization of recovered reserves in open pit mining. *Proceedings 17th APCOM Symposium of the Society of Mining Engineers*, pp288-296. AIME, New York.

Dagdalen, K. and Johnson, T.B. (1986). Open pit mine production scheduling by lagrangian parameterization. *Proceedings 19th APCOM Symposium of the Society of Mining Engineers*, pp127-142. AIME, Co.

Denardo, E.V. (1982). *Dynamic Programming: models and applications*. Prentice-Hall, Englewood Cliffs, N.J.

Even, S. (1979). *Graph Algorithms*. Computer Science Press, Potomac, Maryland.

Everett, H. (1963). Generalized lagrange multiplier method for solving problems of optimum allocation of resources. *Journal of Operations Research*, 11 pp399-417.

Francois-Bongarcon, D. and Guibal, D. (1982). Algorithms for parameterizing reserves under different geometrical constraints. *Proceedings 17th APCOM Symposium of the Society of Mining Engineers*, pp297-309. AIME, New York.

Francois-Bongarcon, D. and Marechal, A. (1976). A new method for open pit design: parameterization of the final pit contour. *Proceedings 14th APCOM Symposium of the Society of Mining Engineers*, pp573-583. AIME, Penn.

Fraser, S. (1971). Computer-aided scheduling of open-pit mining operations. *Canadian Institute of Mining*, 12 pp379-383.

Fytas, K. and Calder, P. (1986). A computerized model of open pit short and long-range production scheduling. *Proceedings 19th APCOM Symposium of the Society of Mining Engineers*, pp109-119. AIME, Co.

Gershon, M. (1982). A linear programming approach to mine scheduling optimization. *Proceedings 17th APCOM Symposium of Mining Engineers*, pp483-493. AIME, New York.

Gignac, L. (1975). Student award paper: Computerized ore evaluation and open pit design. *Proceedings 36th Annual Mining Symposium of the Society of Mining Engineers*, pp46-53. AIME, New York.

Imai, H. (1983). On the practical efficiency of various maximum flow algorithms. *Journal of the Operations Research Society of Japan*, 26(1) pp61-82.

Janssen, A.T. (1969). Long-range production planning of direct-shipping ore from several deposits, in Weiss, A. (ed), *A decade of digital computing in the mineral industry*, pp459-481. AIME, New York.

Johnson, T.B. (1969). Optimum open-pit mine production scheduling, in Weiss, A. (ed), *A decade of digital computing in the mineral industry*, pp539-562. AIME, New York.

Johnson, T.B. and Mickle, D.G. (1971). Optimum design of an open pit - an applicaiton in uranium, in Decision Making in the Mineral Industry, *Canadian Institute of Mining*, 12 pp331-338.

Johnson, T.B. and Sharp, W.R. (1971). A three-dimensional dynamic programming method for optimal ultimate open pit design. *U.S. Bureau of Mines Report of Investigations*, 7553.

Kahle, M.B. and Scheaffer, F.J. (1979). Production schedules, in Crawford, J.T. and Hustruld, W.A. (eds), *Open pit mine planning and design*, pp236-242. AIME, New York.

Kim, Y.C. (1979a). Open-pit limits analysis: technical overview, in Weiss, A. (ed), *Computer methods for the 80's in the mineral industry*, pp297-303. AIME, New York.

Kim, Y.C. (1979b). Production scheduling: technical overview, in Weiss, A. (ed), *Computer methods for the 80's in the mineral industry*, pp610-614. AIME, New York.

Koenigsberg, E. (1982). The optimum contours of an open pit mine: an application of dynamic programming. *Proceedings 17th APCOM Symposium of the Society of Mining Engineers*, pp274-287. AIME, New York.

Koskiniemi, B.C. (1979). Hand methods, in Crawford, J.T. and Hustruld, W.A. (eds), *Open pit mine planning and design*, pp187-194. AIME, New York.

Lerchs, H. and Grossman, I.F. (1965). Optimum design of open pit mines. *Canadian Institute of Mining Bulletin*, 58 pp47-54.

Lipkewich, M.P. and Borgman, L. (1969). Two- and three-dimensional pit design optimization techniques, in Weiss, A. (ed), *A decade of digital computing in the mineral industry*, pp505-523. AIME, New York.

Meyer, M. (1969). Applying linear programming to the design of ultimate pit limits. *Management Science*, 16 ppB121-B135.

Murty, Katta (1983). *Linear Programming*. Wiley, New York.

Papadimitriou, C.H. and Steiglitz, K. (1982). *Combinatorial Optimization: algorithms and complexity.* Prentice Hall, New Jersey.

Phillips, D.T., Ravindran, A. and Solberg, J. (1976). *Operations Research: principles and practice.* Wiley, New York.

Picard, J.C. (1976). Maximum closure of a graph and applications to combinatorial problems. *Management Science*, 22 pp1268-1272.

Robinson, R.H. (1975). Programming the Lerchs-Grossman algorithm for open pit design. *Proceedings 13th APCOM Symposium of the Society of Mining Engineers*, ppBIV 1 - BIV 17. Clausthal, W. Germany.

Roman, R.J. (1973). The use of dynamic programming for determining mine-mill production schedules. *Proceedings 10th APCOM Symposium of the Society of Mining Engineers*, pp165-169. The South African Institute of Mining and Metallurgy, Johannesburg.

Rychkun, E.A. and Chen, T. (1979). Open-pit mine feasibility method and application at Placer Development, in Weiss, A. (ed), *Computer methods for the 80's in the mineral industry*, pp304-309. AIME, New York.

Tarjan, R.E. (1983). *Data structures and network algorithms.* SIAM/CMBS, Philadelphia, Penn.

Weiss, A. (1979). *Computer methods for the 80's in the mineral industry.* AIME, New York.

Wilke, F.L., Mueller, K. and Wright, E. (1984). Ultimate pit and production scheduling optimization. *Proceedings 18th APCOM Symposium of the Society of Mining Engineers*, pp29-38. AIME, New York.

Wright, E.A. (1987). The use of dynamic programming for open pit mine design: some practical implications. *Mining Science and Technology*, 4 pp97-104.