

T-3412

MODIFIED TREE GRAPH ALGORITHM FOR
ULTIMATE PIT LIMIT ANALYSIS

ARTHUR LAKES LIBRARY
COLORADO SCHOOL of MINES
GOLDEN, COLORADO 80401

by

Panlop Huttagosol

T-3412

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


Golden, Colorado

Date April 28, 1988

Signed:

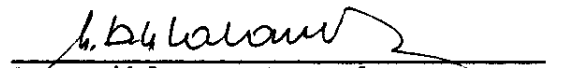

Panlop Huttagosol

Approved:


Dr. Robert E. Cameron
Thesis Advisor

Golden, Colorado

Date April 28, 1988


Dr. Miklos D.G. Salamon
Professor and Head,
Mining Engineering Department

ABSTRACT

The use of computer models to design the ultimate pit limits during feasibility study and long range mine planning is becoming a significant practice in open pit mining. One of the limitations of determining optimal ultimate pit limits using true optimizing techniques has been a great deal of computer resources required. The maximum closure of a graph model, which is a true optimizing technique developed by Lerchs and Grossmann, is extended for solving an ultimate pit limit problem. The tree configuration of the graph theory can be reduced to only connections between positive vertices. The extended method is shown to require less computer memory and processing time while yielding an optimal solution.

TABLE OF CONTENTS

	<u>Page</u>
ABSTRACT.....	iii
LIST OF FIGURES.....	vi
LIST OF TABLES.....	ix
ACKNOWLEDGEMENTS.....	x
Chapter 1. INTRODUCTION.....	1
1.1 Introduction to the Problem.....	1
1.2 Definition of Ultimate pit limit Problem... 3	3
1.3 Scope of the Study.....	4
Chapter 2. COMPUTERIZED ULTIMATE PIT LIMIT MODEL.....	6
2.1 Background.....	6
2.2 Model Development.....	9
2.3 Assumptions of the Model.....	11
Chapter 3. ANALYSIS OF TRUE OPTIMIZING TECHNIQUES.....	13
3.1 Introduction.....	13
3.2 Dynamic Programming Methods.....	13
3.3 Linear Programming Method.....	19
3.4 Network Flow Programming Method.....	21
3.4.1 Steps of Algorithm.....	22
3.4.2 An Example.....	24
3.5 Maximum Closure of A Graph Method.....	31
3.5.1 Steps of Algorithm.....	31
3.5.2 An Example.....	39
3.6 Justification for True Optimizing Purpose.....	49

TABLE OF CONTENTS
(continued)

	<u>Page</u>
Chapter 4. THE PROPOSED METHOD OF MODIFIED-TREE GRAPH ALGORITHM.....	51
4.1 Introduction.....	51
4.2 An Approach of Linear Programming.....	52
4.3 The Modified-Tree Graph Algorithm.....	55
4.4 An Example.....	71
4.5 Some Considerations for the Proposed Algorithm.....	82
4.6 Conclusion.....	90
Chapter 5. IMPLEMENTATIONS ON A CASE STUDY.....	92
5.1 Introduction.....	92
5.2 The Physical Model.....	93
5.3 The Computer Program.....	95
5.4 Implementations and Results.....	98
5.5 Analysis of Results.....	101
5.6 Experience on a Microcomputer.....	103
Chapter 6. CONCLUSION AND RECOMMENDATION.....	106
6.1 Conclusion.....	106
6.2 Recommendation.....	107
REFERENCES CITED.....	108
APPENDIX A: Computer Code.....	112
APPENDIX B: Program Description.....	184
APPENDIX C: Block Grade Distribution by Level.....	193
APPENDIX D: Summation of ore reserves.....	196

LIST OF FIGURES

	<u>Page</u>
Figure 1. Three-dimensional block model represents ore deposit.....	10
Figure 2. A given block model for dynamic programming method.....	15
Figure 3. The column sum value M_{ij}	15
Figure 4. Block cross section shows P_{ij} values and the optimal two-dimensional pit.....	17
Figure 5. A given two-dimensional economic block model...	25
Figure 6. A network flow configuration represents an ultimate pit limit problem.....	27
Figure 7. An example network with initial flow.....	28
Figure 8. An example network after breakthrough.....	28
Figure 9. An example network with maximum flow.....	30
Figure 10. A Directed graph represents an ultimate pit limit problem and slope constraint.....	34
Figure 11. Tree configuration at initialization step.....	40
Figure 12. Tree configuration after first connection.....	41
Figure 13. Tree configuration after second connection.....	41
Figure 14. Tree configuration after third connection.....	44
Figure 15. Tree configuration after connection of the second positive vertex.....	44
Figure 16. Tree configuration after normalization.....	46

LIST OF FIGURES
(continued)

	<u>Page</u>
Figure 17. Tree configuration after connection of the second positive vertex to its next overlying negative vertex.....	46
Figure 18. Tree configuration showing the maximum closure of a graph.....	48
Figure 19. A given example of two-dimensional block model.....	57
Figure 20. The maximum closure of a graph for the given block model.....	58
Figure 21. The modified-tree configuration showing the maximum closure of a graph.....	59
Figure 22. A given example of block model.....	72
Figure 23. Modified-tree configuration at initialization.	72
Figure 24. Modified-tree configuration after the first connection.....	74
Figure 25. Modified-tree configuration after the second connection.....	74
Figure 26. Modified-tree configuration after the third connection.....	76
Figure 27. Modified-tree configuration after connection of the second positive vertex.....	76
Figure 28. Modified-tree configuration after normalization.....	78
Figure 29. Modified-tree configuration after connection of the second positive vertex to its next overlying negative node.....	78

LIST OF FIGURES
(continued)

	<u>Page</u>
Figure 30. Modified-tree configuration showing the maximum closure of a graph.....	80
Figure 31. A given problem of two-dimensional block model for some consideration.....	83
Figure 32. The original tree configuration showing the final graph of a given problem.....	83
Figure 33. The tree configuration after initialization...	87
Figure 34. Tree configuration after connection of the first positive vertex.....	87
Figure 35. Tree configuration after connection of the second positive vertex.....	89
Figure 36. The final tree configuration after connection of the last positive vertex.....	89
Figure 37. Flowchart of the maximum network flow algorithm.....	96
Figure 38. Flowchart of the graph and the modified-tree graph algorithm.....	97
Figure 39. Plot of CPU time required by each method.....	100

LIST OF TABLES

	<u>Page</u>
Table 1. Number of blocks for each case study.....	99
Table 2. CPU time required by each method.....	101
Table 3. Processing time on a microcomputer.....	104

ACKNOWLEDGEMENTS

I am indebted to many person for their help in completion of this thesis.

First of all, I would like to thank Dr. Miklos D.G. Salamon, Dr. Matthew J. Hrebar, Dr. Kadri Dagdelan, and Dr. Robert E. Cameron for serving on my committee and giving invaluable suggestions from time to time. Preferably, Dr. Robert Cameron, my advisor, deserves special thanks for his advice and assistance which helped this paper come about.

I deeply appreciate the support and typing assistance from my wife, Yaowanuj.

Finally, I would like to acknowledge the people of Thailand for providing the scholarship and financial support during my study at Colorado School of Mines.

Chapter 1

INTRODUCTION

1.1 INTRODUCTION TO THE PROBLEM

Open pit mine planning generally involves decisions on many principle problems, such as mining method, equipment selection, mine plant size, and long range production scheduling. The implementation of these plans normally extends over many years and incorporates large amounts of capital investment and financial risk. Thus, before initial engineering work for open pit mine can begin, an ultimate pit limit of the mineral deposit delineating the recoverable ore reserve, the size and the shape of the potential pit must be determined.

The use of computerized ultimate pit limit models for mine planning has become more practical in recent years due to the advances in computer technology and the development of various pit optimizing techniques. Heuristic optimizing techniques, such as the moving cone method, may be one of the most widely accepted techniques in the mining industry today because of its rapid execution speed and easy conceptualization. However, heuristic methods many times fail to generate true optimal designs.

In contrast, a true optimizing technique, such as the maximum network flow or the graph method, does generate an optimal solution. They normally require large computer memory capacity, significant computational effort, and may employ somewhat hard to understand mathematical techniques. The use of true optimal pit designs becomes necessary though when exploiting an ore body of decreasing grade and increasing financial risk.

In recent years, continuing improvements in memory storage capacities and processing speeds of a computer have fostered a rapid increase in the opportunity to apply large computer programs, such as the ultimate pit limit algorithms. In addition, many researchers have worked towards making the algorithms usable within the computer technology. Rychkum and Chen (1979) proposed the technique of Placer's bound for reducing the computer's memory storage required by the graph theory algorithm. The bounding concepts presented by Barnes (1982) can be used to significantly improve the computational effort required by an ultimate pit limit algorithm. These techniques accompanying the advances in computer technology may provide a potential in developing an inexpensive tool for designing the true ultimate pit limits.

In this thesis the graph algorithm, which is one of the true optimizing techniques, is modified in such a way that it requires less computer memory and processing time and still yields true optimal results. The algorithm developed is not an application of a new theory since it still utilizes the fundamental concept of the existing algorithm given by Lerchs and Grossmann(1965). However, it differs from the original graph algorithm in the way the tree configurations is recognized when searching for the maximum closure of a graph. The study is not an effort to introduce a new bounding algorithm, but it is an attempt to directly improve a true optimizing technique.

1.2 DEFINITION OF ULTIMATE PIT LIMIT PROBLEM

The ultimate pit limit problem can be defined as the determination of the size and shape of an open pit mine at the end of its life such that the maximum value of a mineral deposit is realized and the pit slope requirements are satisfied. Typically, the problem is that of maximizing the difference between the total mine value of ore material to be extracted and the total costs of removing that ore material and its overlying waste material. This type of ultimate pit limit analysis may be viewed as the determination of a single period open pit production schedule.

In modern open pit mine planning, the ultimate pit limit problem is merely a part of multi-period production scheduling problem (Johnson, 1968; Benham, 1978; and Dagdelen, 1985). When the economic objective is to maximize the total discounted values (NPV's) of the mine, the ultimate pit limit cannot, by itself, be used to obtain optimal mine schedules since its purpose is to define a three-dimensional pit such that the pit value is maximized within any single time period. The open pit production scheduling model will determine a series of ore and waste extraction schedules which maximize net present value of the mine. In addition, the production scheduling model also takes into account other constraints such as ore grade distribution and mine capacity which are not directly considered by the ultimate pit limit model. However, the design of the ultimate pit limit within any given single period is one of the important steps towards producing an optimal multi-period production schedule.

1.3 SCOPE OF THE STUDY

The objective of this thesis is separated into two parts. The first part is to review the ultimate pit optimizing techniques that are capable of providing true optimal solutions. The second objective is to develop an

extension of a graph method given by Lerchs and Grossmann(1965) that utilizes less computer memory and processing time. The first part provides the fundamental concepts for understanding the second parts.

The computer methods for designing the ultimate pit limits and the block model requirements will be briefly reviewed in Chapter 2. Also, the general assumptions on using the computer models will be given.

In Chapter 3, the true optimizing techniques which utilize mathematical model will be discussed in more details. The true optimizing algorithms that are practical and are able to yield true optimal solution and the three-dimensional pit geometry will be coded and executed on a mainframe computer.

The proposed method of an extended graph algorithm is given in Chapter 4. It will be illustrated how the method is able to yield the true optimal solution and potentially requires less computer resources. The implementation of the developed algorithm is presented in Chapter 5. The existing true optimizing algorithms are also applied in order to compare the results. The conclusions and recommendations from the study are given in Chapter 6.

Chapter 2

COMPUTERIZED ULTIMATE PIT LIMIT MODELS

2.1 BACKGROUND

Prior to the wide spread use of computer techniques, the ultimate pit limit design was carried out by using classical manual methods. Manual methods usually begin with vertical sections which consist of cross, longitudinal, and radial sections. The ultimate pit limit is defined in each section by expanding the design until the break-even stripping ratio is reached along some specified pit slope (Soderberg and Rausch, 1968). Then, the pit limits for all vertical cross sections are integrated together in order to generate the final pit geometry.

In the process of smoothing and shaping the pit boundaries in order to avoid pit slope violations for the three-dimensional geometry, the basic problem in the manual method arises because the total value of the mine may greatly change. Furthermore, the process of determining the pit limit on each vertical section is a trial and error procedure and usually requires a number of approximations and repetitive work. Therefore, hand methods were replaced by computer methods.

Programming logic included in early computer methods were patterned after the conventional manual methods. They utilized a break-even stripping ratio as a basic design criterion. The main difference from manual techniques is that they use block and multiple cone concepts instead of vertical sections to generate the final pit geometry.

Computerized ultimate pit limit analysis has received a lot of attention since its first introduction. The list of significant computer techniques that have been developed and used in the mining industry can be given as follows :

- multiple moving cone methods,
- dynamic programming methods,
- linear programming method,
- maximum closure of a graph method, and
- maximum network flow programming method.

Kim(1978) classified these methods into two categories; heuristic optimization algorithms and rigorous optimizing algorithms. Moving cone methods fall under the heuristic classification because they seem to work in nearly most cases, but lack rigorous mathematical proof that they actually obtain the ultimate optimal pit limit. The last four methods are categorized as the rigorous optimizing techniques since they can be mathematically shown that they produce an optimal final pit design.

Most of the computer methods in use by mining companies utilize the concept of multiple cones for removal increments. A pit is generated and analyzed by constructing an inverted cone and moving its vertex from one ore block to another. The cone shape is defined such that it conforms with pit slope constraints. The computer is employed in the calculation of net value of the cone by summing values of all ore and waste material enclosed within the cone. The mathematical models applied for determining the cumulative value may be different depending on the actual technique. Finally, the pit expansion is obtained by removing the frustrums of all cones with net positive value.

From the point of view of finding the mutual effect between the overlapping cones, the moving cone heuristics algorithms are significantly restrictive due to the large amount of computational effort required (Barnes, 1982). For the above reason, moving cone algorithms usually terminate after cones having their vertices located on all positive ore blocks have been evaluated. The determination of a mutual support between the overlapping cones is not usually required by most moving cone algorithms. This means that moving cone methods can not be practically used to generate the true ultimate pit limits. However, the methods are widely used in the mining industry because they are very

In attempts to consider the mutual support between the overlapping cones, the rigorous optimizing techniques were developed applying more sophisticated mathematical concepts, such as the dynamic programming, linear programming, maximum network flow programming, and the graph theory. Eventhough, these techniques can produce the true ultimate pit limits, they still have some limitations in real practice as will be discussed later in Chapter 3.

2.2 MODEL DEVELOPMENT

A key point in the design and operation of recent ultimate pit algorithms is the utilization of a three-dimensional model to represent geologic and economic information of the ore body. An ore body and surrounding waste material is normally modelled on a computer by subdividing it into small rectangular blocks in sufficient numbers so that the flow and form of material in the mining and processing procedure can be described in reality (Johnson, 1968; and Stanley, 1979). For convenience, the three-dimensional fixed block model, as illustrated in Figure 1, in which all blocks are of the same size, has been widely employed for pit limit analysis (Kim, 1978).

Once subdivided into small blocks, various grade extension techniques, such as polygonal method, inverse distance methods, and the kriging method, are used to extend

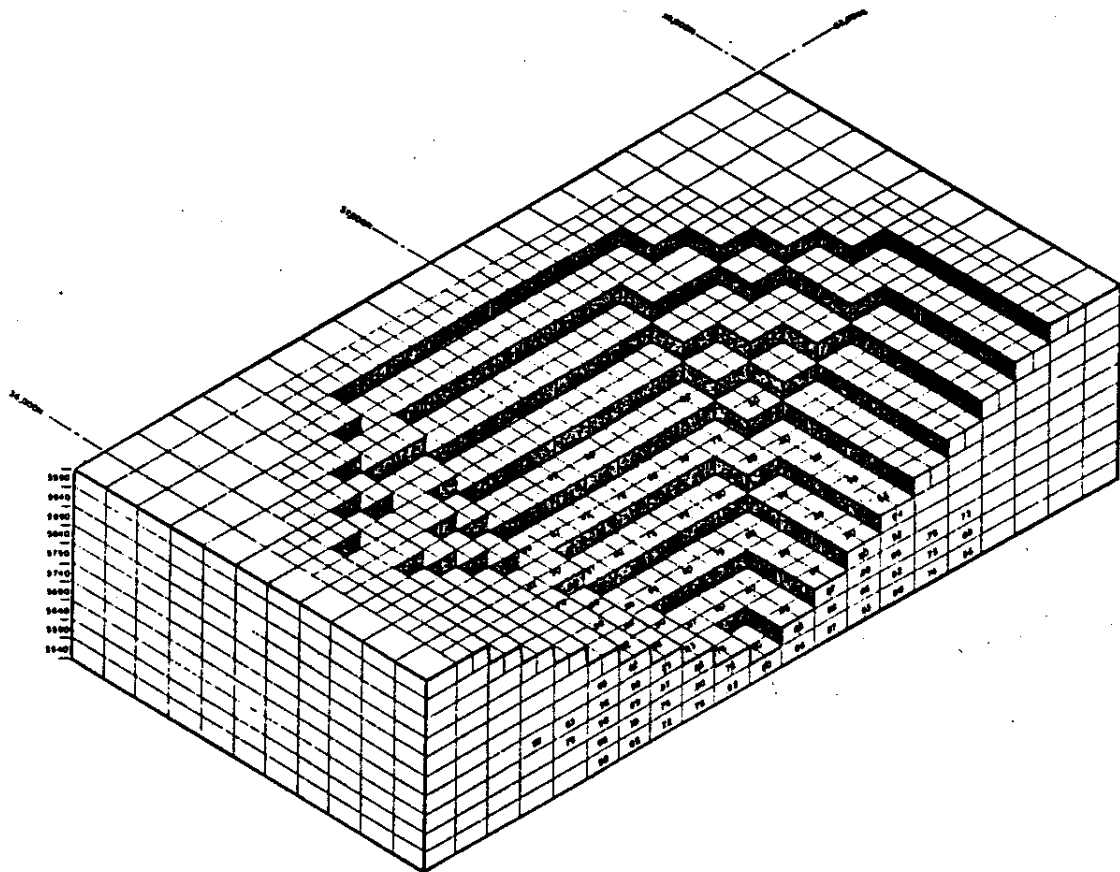


Figure 1. Three-Dimensional Block Model Represents Ore Deposit (from Pana, 1965).

distance methods, and the kriging method, are used to extend drill hole assays to each block. Most importantly, the information assigned to a given block in the model is the quality and characteristics of different material existing in the deposit.

Since the basic input into most ultimate pit algorithms is the economic block inventory, the final step in the model development is to convert the geologic block inventory into financial information. At this step, a block is specified to be either ore or waste according to its net value obtained from mining. The economic block valuation is made by weighing the recoverable ore values against the total cost of mining, milling, smelting, and refinery, but excluding the stripping cost related to the block. If the resultant net value is positive, or negative but still greater than the negative value of the cost of removal, the block is classified as possible ore, otherwise the block is waste and is assigned a negative value equal to the cost of removing it to the waste dump.

2.3 ASSUMPTIONS OF MODEL

Due to the fact that the problem of defining ultimate pit contours by using the computer techniques usually utilizes the block and multiple cone concepts as discussed in the previous sections, the following assumptions are

generally made in order to clarify the model optimization :

1. The ore body and surrounding material can be divided into a finite number of mineable units or blocks. Actual open pit mining is accomplished in terms of blocks and the geological and financial values of these blocks can be determined.

2. At any given pit slope constraints, it must be possible to define a set of overlying blocks which must be removed in order to mine a particular block. The block dimensions can be designed such that the relationship between the vertical and horizontal dimension represents the slope angle. Usually, the dimensions are not far from the real mining practice, for example, the block height may be assigned equal to the bench height which may be influenced by equipment capabilities.

3. Minor factors that may affect the pit geometry other than the geological conditions, such as the layout of haul roads and in-pit equipments, are not very important to the solution and therefore ignored.

Chapter 3

ANALYSIS OF TRUE OPTIMIZING TECHNIQUES

3.1 INTRODUCTION

The application of mathematical models in solving for the ultimate pit limits has received more attention since a paper written by Lerchs and Grossmann (1965) was presented to the mining industry. From then until now, there have been four rigorous optimizing models developed and used for solving ultimate pit limit problem :

1. dynamic programming model,
2. linear programming model,
3. maximum network flow programming model, and
4. maximum closure of a graph model.

To understand how these rigorous methods are mathematically proven and how they consider the mutual support between the overlapping pits, it is necessary to review and discuss the four methods in more details.

3.2 DYNAMIC PROGRAMMING METHODS

An original dynamic programming algorithm was demonstrated for determining the optimal configuration of blocks to be removed in two dimensional cross sections by Lerchs and Grossmann (1965). The algorithm simply employs the dynamic programming model which provides the advantage

of fast computational speed. However, this two dimensional dynamic programming approach becomes impractical in defining the three-dimensional pit geometry because it is able to generate the optimal contour only in a given cross section. The final pit geometry of all cross sections, when they are assembled together, do not yield the true ultimate pit limit and may violate the maximum allowable pit slopes.

In order to demonstrate how the dynamic programming method generates an optimum pit limit, the steps of algorithm for solving the two-dimensional pit will be given. To obtain an optimal cross-sectional pit with 45-degree maximum allowable slope (or an up one over one pattern) from a two-dimensional block model as shown in Figure 2, the algorithm proceeds as follows :

Step 1 For a cross section i , let m_{jk} represents the dollar value of block in column j of level k (Figure 2).

Step 2 For all j ($j = 1$ to 8) and all k ($k = 1$ to 3), calculate;

$$M_{jk} = \sum_{z=1}^k m_{jz} \quad (3.1)$$

The new tableau of M_{jk} is constructed as shown on Figure 3. M_{jk} represents a cumulative value realized in extracting a single column j from the top level down to level k .

	1	2	3	4	5	6	7	8
1	-1	-1	-1	-1	-1	-1	-1	-1
2	-2	-2	+1	-2	+2	+1	-2	-2
3	-3	-3	+3	+4	-1	+4	-3	-3

Figure 2. A Given Two-Dimensional Cross Section of an Example Block Model.

	1	2	3	4	5	6	7	8
1	-1	-1	-1	-1	-1	-1	-1	-1
2	-3	-3	0	-3	+1	0	-3	-3
3	-6	-6	+3	+1	0	+4	-6	-6

Figure 3. A Column Sum Values M_{jk} Calculated from Figure 2.

Step 3 Add an artificial column zero, $j=0$, and add an artificial level, $k=0$. Let $P_{j0} = 0$ for all j ($j = 0$ to 8). Then, column by column starting with column $j=1$ to $j=8$, compute value of P_{jk} for all k ($k=1$ to 3) as :

$$P_{jk} = M_{jk} + \text{Max}[P_{j-1,k-1}, P_{j-1,k}, P_{j-1,k+1}] \quad (3.2)$$

Indicate a block that yields the maximum value for computing P_{jk} in eq. (3.2) by an arrow going from block (j,k) to that block in column $j-1$ (see Figure 4). P_{jk} is the maximum possible contribution of column 1 to j to any feasible two-dimensional pit that contains block (j,k) on its contour.

Step 4 Find a value P_{max} which represents the maximum value of the potential two-dimensional pit as :

$$P_{\text{max}} = \text{MAX } P_{j1}, \forall j$$

For this case, P_{max} is obtained from block $(8,1)$, then the ultimate pit limit is determined by tracing back the arrows from block $(8,1)$ until block $(0,0)$ is reached (Figure 4).

In the past several years, the Lerchs and Grossmann two-dimensional dynamic programming algorithm has been extended by many attempts for solving the three-dimensional ultimate pit limit problems. The algorithms developed by Johnson and Mickle (1970) and Johnson and Sharp (1971) are still close to the original two-dimensional algorithm except that they also consider the block value information from

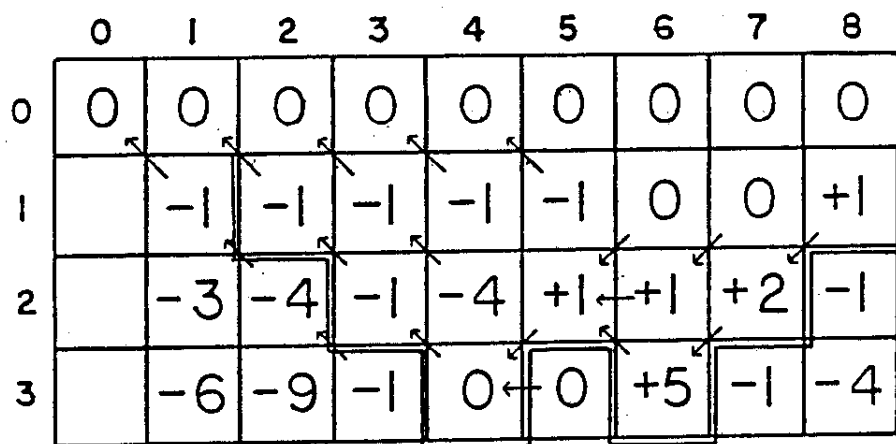


Figure 4. Block Cross Section Shows P_{jk} Values and an Optimal Two-Dimensional Pit.

cross sections perpendicular to a longitudinal section of interest. The improvement of this method is presented in Barnes (1982) and it is entitled the Best-Valued Cross Section Algorithm. Although, these extended algorithms of the two-dimensional dynamic programming consider the information of block values in perpendicular cross sections, the configuration of generated pit still fails to follow three-dimensional pit slope constraints since the process of pit reaming and smoothing is required in the final stage. However, the technique is computationally fast and can be used as a bounding algorithm in generating an outer bound for the true optimizing algorithms (Barnes, 1982).

In Koenigsberg (1982), a three-dimensional dynamic programming is proposed for solving a three dimensional case of ultimate pit limit problem. The algorithm does not search for only the block value information in its particular cross section, but also investigates relationship between levels within a column, between columns within a section, and between sections. However, a generalized pit slope constraints is still infeasible with the three-dimensional dynamic programming algorithm because the method is recursive and requires that the pit slope definition must be consistent throughout the model. The technique is also highly tied to the geometry of the blocks.

It can be concluded that the dynamic programming methods are still impractical for determining the true ultimate pit limits with three-dimensional geometry and generalized pit slope requirements.

3.3 LINEAR PROGRAMMING METHOD

The linear programming model is the common method among the operation research techniques applied in optimal pit limit analysis. It has been used by many researchers for representation of the ultimate limit problems (Johnson, 1968; Koenigsberg, 1982,; and Dagdelen, 1985). The ultimate pit limit problem can be formulated as a linear programming problem with the following structure :

$$\text{MAX } z = \sum_{n=1}^N c_n x_n$$

Subject to :

$$\begin{aligned} -x_t + x_n &\leq 0, \quad \forall n \\ x_m &\geq 0 \end{aligned} \quad (3.3)$$

where :

- n = block number in a more natural three dimensional notation,
- N = total number of blocks for the problem (N=NXxNYxNZ where NX is section dimension, NY is column dimension, and NZ is level dimension),
- t = every overlying block which must be removed before mining block n,
- x_m = binary integer fraction of material mined from m

block m ($x_m = 1$ if block m is mined, otherwise
 $x_m = 0$).

In the application of linear programming technique, the number of constraints must not be too large because the required computer time for solutions is proportional to the third power of the number of constraints (Fytas and Calder, 1986). The total number of sequencing constraints, that a particular block cannot be mined unless all the overlying blocks have been removed, is equal to the total number of overlying blocks. This means that the linear programming method can be applied to only a small model.

Moreover, the solution to the linear programming problem is really a 0-1 integer programming problem which requires a significant computational effort that tends to increase exponentially with the number of decisional variables or number of blocks (Fytas and Calder, 1986). It can be concluded that the linear programming model cannot practically solve a large problem, such as the ultimate pit limit problem. However, this large binary linear programming problem may be transformed into an equivalent bipartite network flow programming problem which is a special structure found in certain linear programming problems and can be exploited in the design of highly efficient network solution techniques.

3.4 NETWORK FLOW PROGRAMMING ALGORITHM

The network flow algorithm for determining ultimate pit limits, which is based on the theory of maximum flow and minimum cut, was originally developed by Johnson(1968). The problem is formulated such that the nodes in the network flow are equated to the mining blocks assigned the financial values and the arcs indicate pit slope constraints for feasible mining sequences. The problem is transformed from a binary integer linear programming model as described in the previous section, that is, each node is defined as an indicator variable (X_n) and each arc connecting a positive-valued node to an overlying negative-valued node is defined in such a way that it can represent the precedence constraint.

To solve for a maximal flow of the defined network will be equivalent to solving the ultimate pit limit problem if :

1. the arc capacity from the imaginary source node to any positive node is set equal to the block value of that positive node,
2. the arc capacity from any negative node to the imaginary sink node is set equal to the absolute block value of that negative node, and
3. an infinity capacity is given to any arc connecting from the positive node to the overlying negative node.

This ultimate pit limit problem is shown to be bipartite network, that is, the only considered arcs are those from the source to positive nodes, from each positive node to its overlying negative nodes, and from negative nodes to the sink node. The well-known labeling technique can be effectively solve this bipartite network (Johnson, 1968; and Barnes, 1982).

3.4.1 Steps of Algorithm.

The maximal flow labeling algorithm which is applied to the ultimate pit limit problem can be given in the following steps :

Step 1 Each block is categorized into either positive node or negative node items depending on its economic block value.

Step 2 The source node S and sink node T is created in the network so that the feasible flow can be assigned starting at the source and ending at the sink.

Step 3 Place all positive nodes in the network next to the source. Construct arcs from the source S to each positive node and assign an upper bound capacity to each arc equal to the corresponding positive block value.

Step 4 Place all negative nodes in between the positive nodes and the sink. Connect each positive node to each of its respective overlying negative nodes with an arc having

infinite flow capacity. These negative nodes must be removed before the positive node can be mined.

Step 5 Connect each negative node to the sink T with an arc having a flow capacity equal to the absolute value of corresponding negative block value.

Step 6 Solve this network flow problem for the maximum flow or minimum cut by using the labeling algorithm as given later in this section.

Step 7 All nodes or blocks connected by the arcs that belong to the maximum flow or minimum cut must be included in the ultimate pit limit.

Let define variable F_{xy} as the flow from node x to node y and variable C_{xy} as the upper bound capacity of arc(x,y), and assign some arbitrary flow from the source to the sink, the labeling algorithm can be described as follows :

Routine A : Labeling process

1) Label the source node S as $[-, \infty]$. Node S is now labeled but has not been scanned.

2) Select any labeled and unscanned node x :

a. for each unlabeled node y that has $F_{xy} < C_{xy}$, label node y as $[x, +E(y)]$, where :

$$E(y) = \text{MIN} \{E(x), C_{xy} - F_{xy}\}$$

b. for each node y that has $F_{xy} > 0$, label node y as $[x, -E(y)]$, where :

$$E(y) = \min \{E(x), F_{xy}\}$$

Node x is now scanned and node y is labeled.

3) If the sink node can be labeled, or breakthrough, go to Routine B. Otherwise, stop. The maximum flow is obtained. All labeled nodes indicate the blocks to be mined, except for the source node.

Routine B : Flow augmentation

- 1) Set node z equal to the sink node T.
- 2) If node z is labeled $[n, +E(z)]$, increase F_{nz} by $E(z)$. If node z is labeled $[n, -E(z)]$, decrease F_{nz} by $E(z)$.
- 3) If node n is node S, delete all labels and go to Step 1. of Routine A. Otherwise, define z to be n and go to Step 2. of this routine.

The labeling algorithm is demonstrated in the given example as presented in the next section.

3.4.2. An Example.

A small example problem will be used to demonstrate the network flow algorithm. Consider a two dimensional block model with a given cross sectional configuration and slope of 1:1 as shown in Figure 5. The algorithm can proceed as follows:

Step 1. Block 5 and 6 are positive blocks while block 1 to 4 are categorized as negative ones depending on given block values.

1 -2	2 -2	3 -2	4 -3
	5 +5	6 +5	

Figure 5. A Given Two-Dimensional Economic Block Model.

Step 2. The imaginary source node S and sink node T, are created in the network (Figure 6).

Step 3. Construct arcs from the source S to node 5 and 6. Assign an upper bound capacity of five units to the arc connecting to node 5 equal to the corresponding positive block value, and so does the arc connecting to node 6 (Figure 6).

Step 4. Connect the positive node 5 to each of its respective overlying negative node, which are node 1, 2 and 3, with an arc having an infinite flow capacity (Figure 6). Also connect the positive node 6 to its overlying negative nodes, node 2, 3, and 4, in the same manner.

Step 5. Connect each negative node, node 1 to 4, to the sink node T with an arc having upper bound flow capacity equal to 2, 2, 2, and 3 units respectively (Figure 6).

Step 6. Solve this network problem in Figure 6. for the maximum flow (or minimum cut) by using the labeling algorithm. At first, the network is arbitrarily assigned an initial flow as shown in Figure 7. Node S is labeled as $[-, \infty]$, but is still unscanned. Now select node y which is unlabeled and $F_{xy} < C_{xy}$. For example, node 6 is such a node and it can be labeled as $[S, +5]$. By repeating this labeling routine, the node T can be labeled as $[3, +1]$.

A breakthrough is obtained since the node T is labeled.

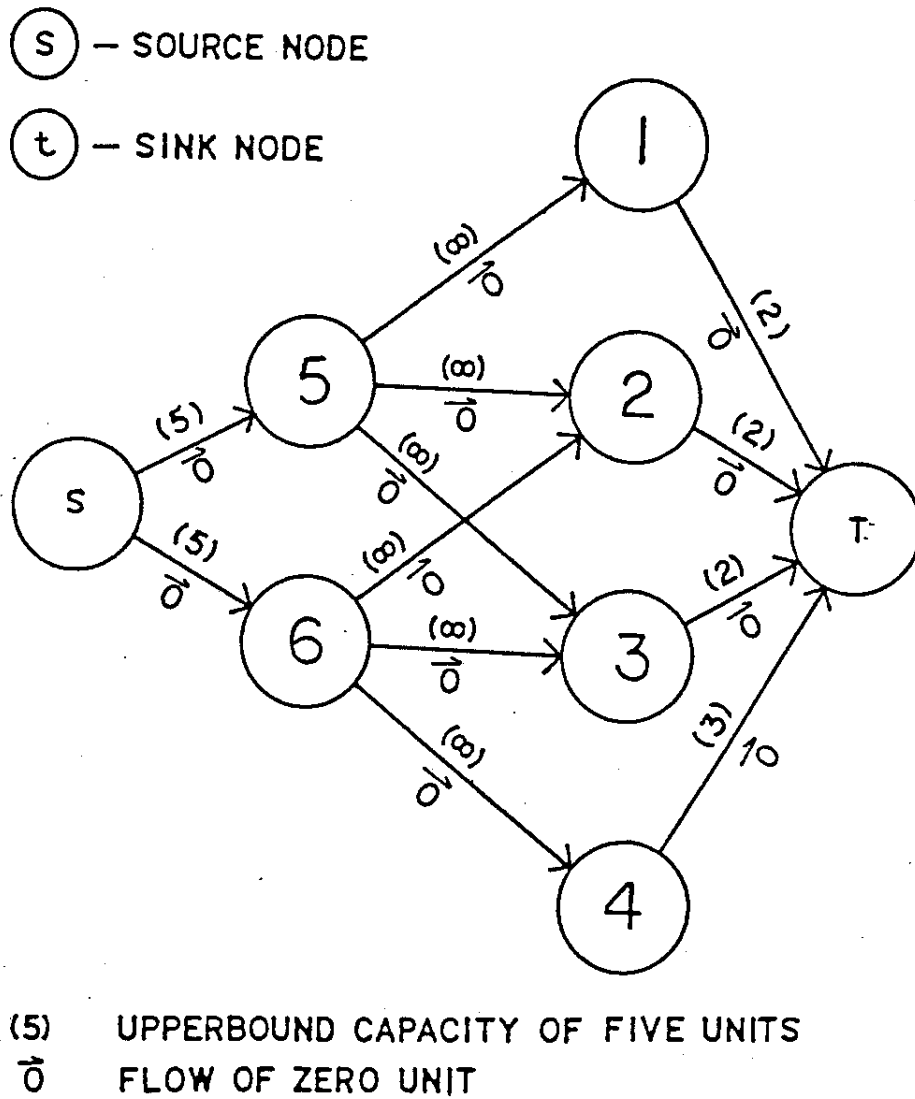


Figure 6. A Network Flow Configuration Represents an Ultimate Pit Limit Problem.

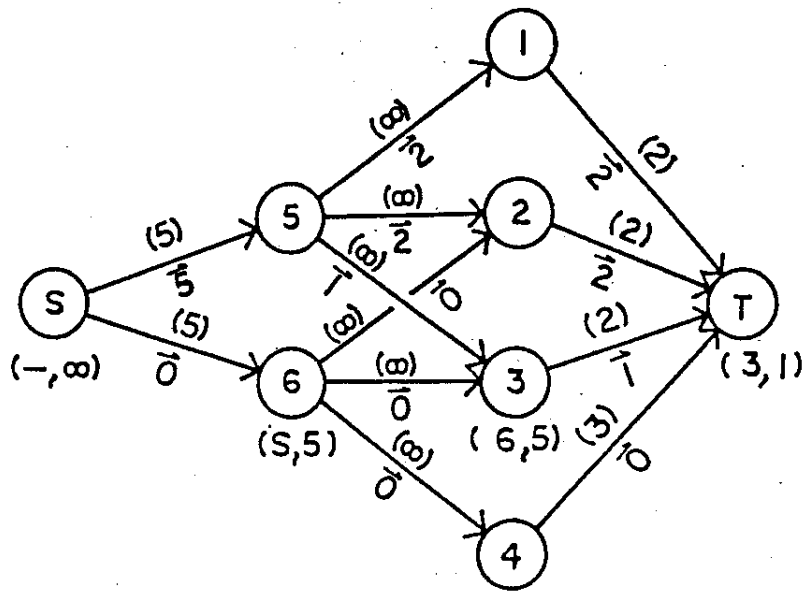


Figure 7. An Example Network with Initial Flow.

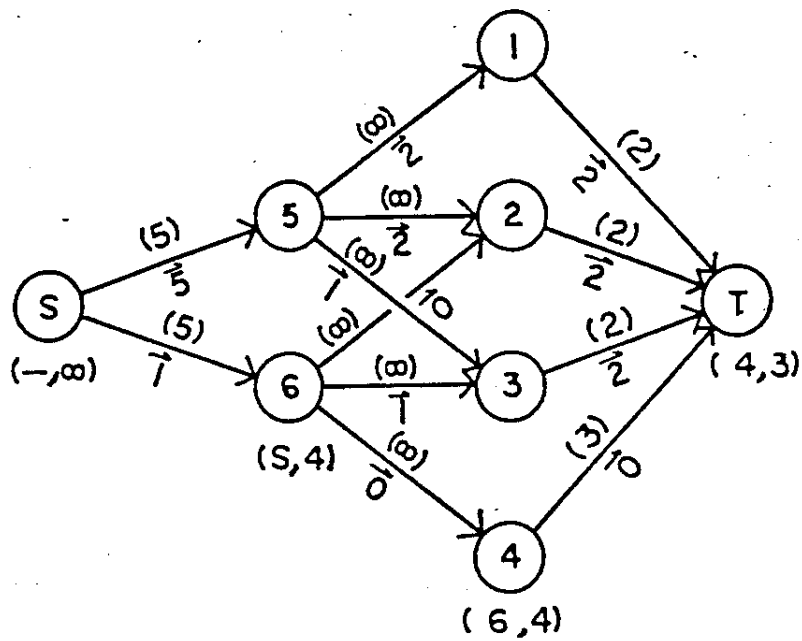


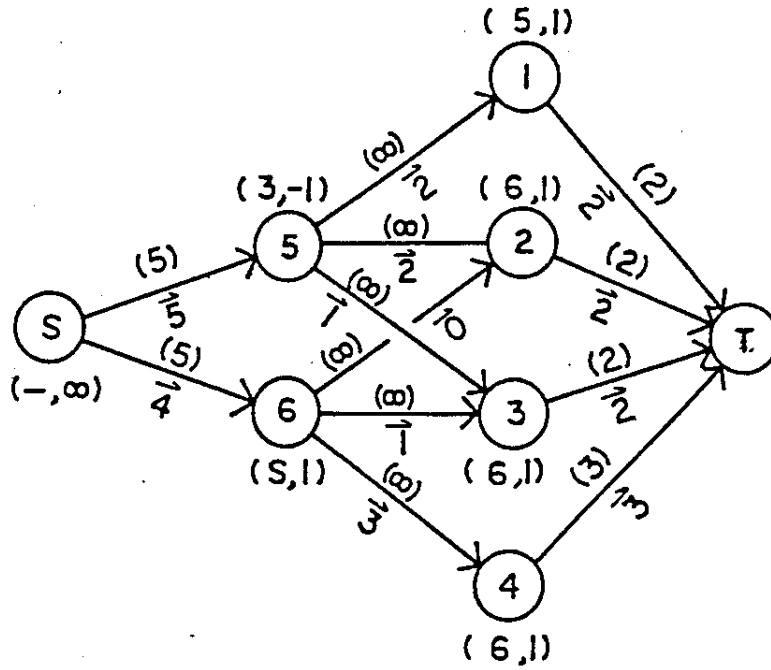
Figure 8. An Example Network after Breakthrough.

Assign one unit of flow through arc (S,6), arc (6,3), and arc (3,T). Next, delete all labels and, by applying the labeling routine again, a breakthrough occurs (Figure 8). Again, three units of flow through new path along arc (S,6), arc (6,4), and arc (4,T) is assigned as shown in Figure 9. The labeling algorithm is repeated but a non-breakthrough is obtained. The total flow from the source S to the sink T is the optimal amount.

Step 7. The optimal solution to the ultimate pit limit problem is obtained by removing all labeled nodes, except the source node S. The blocks that are to be mined are blocks 1, 2, 3, 4, 5 and 6.

Due to the structure of the network, the positive blocks can only contribute their dollar values to pay for removing themselves and their overlying waste blocks. The mutual support between positive blocks to the overlying negative block exists due to the fact that the arcs connecting from those positive nodes to the common negative node allows the flow between those positive nodes to be possible.

The network flow method of determining the ultimate pit limit can be generalized to variable pit slope constraints and three dimensional pit without additional difficulty.



MINE LABELED NODES 1, 2, 3, 4, 5, AND 6

Figure 9. An Example Network with Maximum Flow.

3.5 MAXIMUM CLOSURE OF A GRAPH ALGORITHM

The application of a graph theory for designing the ultimate pit limits was originally presented by Lerchs and Grossmann(1965). The algorithm formulates the block model into a network tree, that is, the vertices are equated to mining blocks and the imposed directed arcs represent pit slope constraints. These directed arcs indicate the relationship between waste blocks that must be removed before mining a particular ore block. Since any feasible contour of a pit is obtained by a closure of a graph, then, the problem of defining an ultimate pit limit becomes the determination of a closure of a graph with maximum mass.

The directed graph problem can be transformed into a network flow problem due to the fact that it is a special structure of the network flow programming model (Lerchs and Grossmann, 1965). In Barnes (1982), the graph concept is mathematically proven to be equivalent to the network flow programming concept when it is applied for solving the ultimate pit limit.

3.5.1 Steps of Algorithm

The essential elements of the graph algorithm, which is different from those of the network flow algorithm, can be given as follows.

Let y be a block node or vertex, either positive or negative vertex.

x be a positive vertex.

z be a negative vertex.

D a constant, dummy root node.

$P(y)$ is a predecessor or root node of y .

$S(y)$ is a direct successor or branch node of y .

$Q(y)$ be another successor node of node $P(y)$, but is recognized in terms of y .

$BV(y)$ is an original economic block value of y .

$CV(y)$ is the value supported by an arc that has node y as a branch vertex, or the cumulative value of node y and all its successors, i.e.,

$$CV(y) = BV(y) + \sum_i CV(i) \quad (3.4)$$

where i is all successors of node y and $P(i) = y$.

Initialization Step Let a graph $G = [Y, A]$ represents the whole block model with a set of vertices Y and a set of directed arcs A . Each vertex or node y represents each mining block y and contains a mass equal to the block value. The algorithm classifies each vertex into either positive or negative node category corresponding to its block value.

Each directed arc represents the slope constraint such

that it points from a positive node to its overlying negative node (Figure 10). For any two vertices of a directed arc or an edge, one vertex located at the root end of the edge is defined as the predecessor vertex, the other one on the branch end is called a successor. Any one vertex may have more than one successor but it must have only one predecessor since there may be many edges connecting from one vertex to its branch nodes while there is only one edge linking the vertex to its predecessor.

A dummy node D is added to the tree and will be used only as a reference vertex. The graph G is initially normalized by building an arbitrary tree T in the graph G . This initial T is a graph $[Y, A_0]$ where A_0 is a set of all dummy arcs connecting the dummy D to every node y .

The algorithm assigns value to a term $BV(y)$ equal to a given block value of block y and initializes the following variables for every node y as :

$$\begin{aligned} P(y) &= D \\ S(y) &= D \\ Q(y) &= D \end{aligned} \quad (3.5)$$

In general, the algorithm starts constructing the tree T from the blocks in the uppermost level of the block model and proceeds down level by level. The tree T will be transformed into a successive tree in the following steps.

1	2	3	4
-2	-2	-2	-3
	5	6	
	+5	+5	

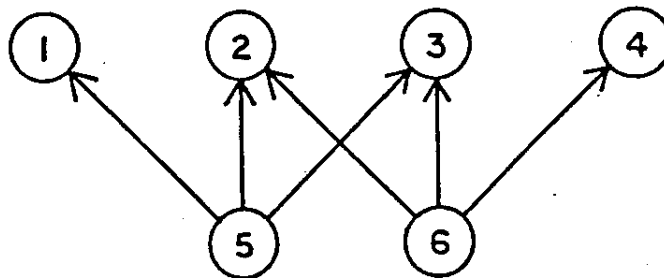


Figure 10. A Directed Graph Represents an Ultimate Pit Limit Problem and Slope Constraint.

Search Step For a strong positive vertex x , the algorithm searches for its weak overlying negative vertex or node z of which :

$$P(z) = D, \text{ or}$$

$$P(z) = w \quad (3.6)$$

where w in eq. (3.6) is another weak positive vertex. Any vertex can be defined as weak or strong if it is located on a weak or strong tree respectively. A tree is strong if its total mass is positive, otherwise, it is weak.

A tree may contain p -edges (plus-edge) and m -edges (minus-edge). An m -edge is an arc that points toward the root of the tree, while a p -edge points away from it. The total value supported by any edge is calculated by summing the mass of all the vertices further from the root. A p -edge is strong if it supports a positive value, otherwise, it is weak. An m -edge is strong if it support a zero or negative value, otherwise it is weak.

Transformation and Connection Step If a negative node z located on a weak tree is found overlying the positive node x located on the strong tree T , the tree T will be transformed into a successive tree T_{t+1} by removing the strong edge connecting from the dummy D to the tree T_t and establishing the directed arc from the strong vertex x to its the overlying negative vertex z . The tree T_{t+1} is

obviously the connection of the transformed tree containing node x to the tree containing node z .

For the tree T_t , it used to have the chain of vertices linking node x to the dummy root. But after transformation, those vertices on the chain will be linked to the dummy by the new reverse chain passing through node x , z and some vertices of the tree on which node z has been located before connection. This means that all the edges on the chain from node x to the node next to the dummy root of T_t have changed their status in such a way that a p -edge in T_t becomes an m -edge in T_{t+1} and vice versa. Tree T_t is transformed as :

$$\begin{aligned} S'(x) &= P(x) \\ P'(y) &= u \\ S'(y) &= P(y) \end{aligned} \quad (3.7)$$

where y is every node on the chain from node x to the dummy root of T_t , excluding node x and the dummy root, and a symbol $'$ denotes properties of the new tree T_{t+1} . Node u in eq. (3.7) is the successor of node y located on the transformation chain.

The positive node x is connected to the negative node z as :

$$\begin{aligned} P'(x) &= z \\ S'(z) &= x \end{aligned} \quad (3.8)$$

If the original $S(z) \neq D$, it means that there have been the

other positive node, v , connected to node z before connection, or $S(z) = v$. Node v must be recognized as another successor of node z in terms of $Q(x)$ as follow :

$$Q'(x) = v \quad (3.9)$$

$Q'(x)$ in eq. (3.9) is not the successor node of node x , but the successor of node z since $P(Q'(x)) = P(v) = z$. The purpose of using Q in terms of x is to reduce the number of variables used rather than defining Q in terms of z .

If z has many successors, they can be recognized as :

$$Q(x_1) = x_1, Q(x_2) = x_2, Q(x_3) = x_3, \dots$$

where $x_1, x_2, x_3, x_4, \dots$ are successors of node z of which

$$S(z) = x_1, \text{ and}$$

$$P(x_1) = z, P(x_2) = z, P(x_3) = z, P(x_4) = z, \dots$$

The term Q can also be used to keep track of successors of a positive vertex in the same manner if it has more than one negative successors.

Evaluation and Normalization Step After transformation and connection, the term $CV(y)$ must be reevaluated for every node y in the tree T_{t+1} (see eq. [3.4]). The total mass of the whole tree can be obtained from the value of $CV(k)$, where k is the extreme root node of the tree directly connected to the dummy, or $P(k) = D$.

The resultant tree T_{t+1} is normalized if the dummy root

is not common to any strong edge of the tree. The tree T is changed into tree T^{t+2} and T^{t+3} in such a way that each strong edge of T^{t+1} has the dummy root D as its root vertex. The strong edge that is not connected to the dummy root will be erased and a new arc from the dummy D is created as :

$$\begin{aligned} P'(u) &= D \\ Q'(u) &= D \\ S'(y) &= Q(u) \end{aligned} \quad (3.10)$$

where a strong edge occurs between node u and node y , and $P(u) = y$. The edge is a strong m -edge if y is a negative node, u is a positive node, and $CV(u)$ is less than or equal to zero. The edge is a strong p -edge if y is a positive node, u is a negative node, and $CV(u)$ is greater than zero. The evaluation and normalization process continues until all strong edges have the dummy D as their extreme root.

The repetitive process of the search step through the evaluation and normalization step proceeds until no further transformation is possible. The algorithm terminates in a finite number of iterations. The basic concept justifying the finite number of iterations is given by Lerchs and Grossmann (1965). The ultimate pit limit is a maximum closure of a graph that contains all vertices located on strong edges having the dummy D as their extreme root.

3.5.2 An Example.

The same simple problem with slope constraint of 1:1 as given in Section 3.4.2 is used again (Figure 10) in order to demonstrate the true optimality of the graph algorithm. By using the steps of algorithm as given in the previous section, the ultimate pit limit problem can be solved as follows.

Initialization

Assign value to $BV(y)$ equal to an economic block value of block y ($y=1$ to 6) as :

$$BV(1) = -2, BV(2) = -2, BV(3) = -2,$$

$$BV(4) = -3, BV(5) = +5, BV(6) = +5$$

Next, each node is connected to the dummy root D (Figure 11) and node properties are initialized as (see eq. [3.5]).

$$P(y) = D ,$$

$$S(y) = D , \text{ and}$$

$$Q(y) = D , \text{ where } y = 1 \text{ to } 6.$$

Node 5 and 6 are classified as strong positive vertices, while node 1 to 4 are weak negative vertices.

Iteration No.1

Start with positive node 5, node 1 is found overlying node 5. Since node 5 is already connected to the dummy, $P(5) = D$, no transformation process is needed. The

S — STRONG BRANCH

W — WEAK BRANCH

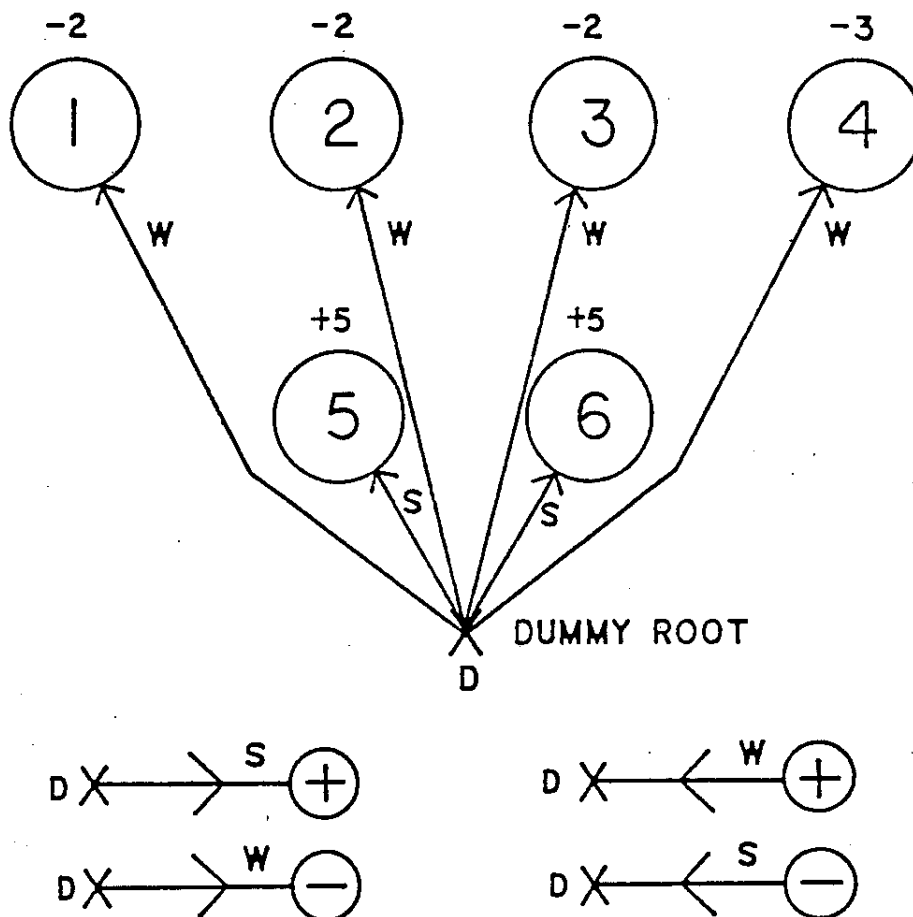


Figure 11. Tree Configuration at Initialization Step.

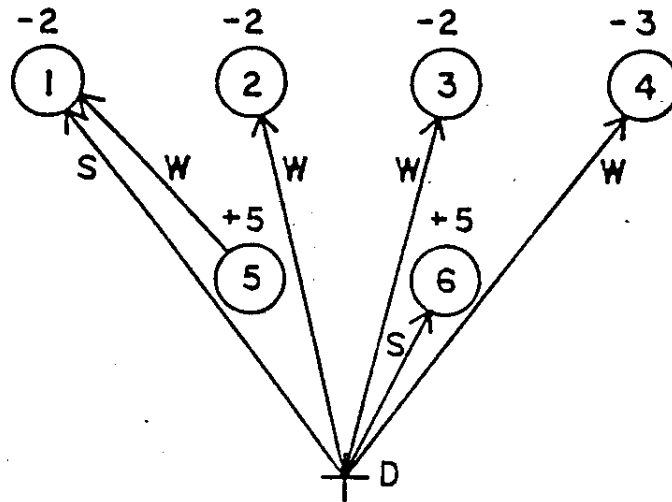


Figure 12. Tree Configuration after the First Connection.

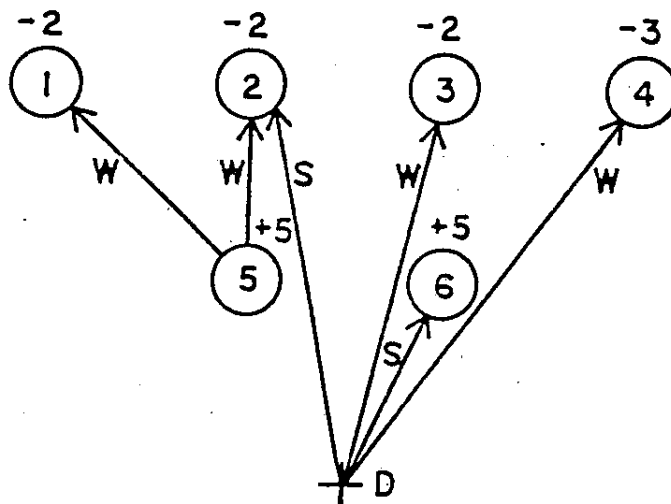


Figure 13. Tree Configuration after the Second Connection.

connection proceeds as (see eq. [3.8]) :

$$P(5) = 1 , \text{ and}$$

$$S(1) = 5$$

The tree configuration after connection is shown in Figure 12. Then, all edges of the tree are evaluated using eq. (3.4) as :

$$CV(5) = BV(5) = +5 , \text{ and}$$

$$CV(1) = BV(1)+CV(5) = -2+5 = +3$$

No normalization process is needed since the strong edge is already connected to the dummy. The tree is strong due to the positive total mass, and so do node 1 and 5. The algorithm continues searching for the next overlying negative node of node 5, which is node 2. Since $P(5) = 1$, the algorithm proceeds as :

transformation (eq. [3.7]);

$$S(5) = 1 ,$$

$$P(1) = 5 , \text{ and}$$

$$S(1) = D$$

connection;

$$P(5) = 2 , \text{ and}$$

$$S(2) = 5$$

evaluation;

$$CV(1) = BV(1) = -2 ,$$

$$CV(5) = BV(5)+CV(1) = 5-2 = +3 , \text{ and}$$

$$CV(2) = BV(2)+CV(5) = -2+3 = +1$$

The tree (Figure 13) is still strong, so the algorithm

searches for the next overlying weak negative node. Node 3 is such a qualified node (see eq. [3.6]). The iteration process continues as :

transformation; $S(5) = 2$,

$$P(2) = 5$$
 ,

$$S(2) = D$$
 , and

$$Q(2) = 1$$

connection; $P(5) = 3$, and

$$S(3) = 5$$

evaluation; $CV(2) = BV(2) = -2$,

$$CV(5) = BV(5) + CV(1) + CV(2) = 5 - 2 - 2 = +1$$
 ,

$$CV(3) = BV(3) + CV(5) = -2 + 1 = -1$$

The tree (Figure 14) becomes weak due to the negative total mass, and so does the positive node 5. Then, the algorithm retrieves the next available strong positive node from the list of all positive nodes. Node 6 is such a node. The first weak node found overlying node 6 is node 2 (see eq. [3.6]). Since the root of node 6 is the dummy, $P(6) = D$, the algorithm continues the iteration process as :

connection; $P(6) = 2$, and

$$S(2) = 6$$

evaluation; $CV(6) = BV(6) = 5$,

$$CV(2) = BV(2) + CV(6) = -2 + 5 = 3$$

This new tree (Figure 15) has a strong p-edge occurs

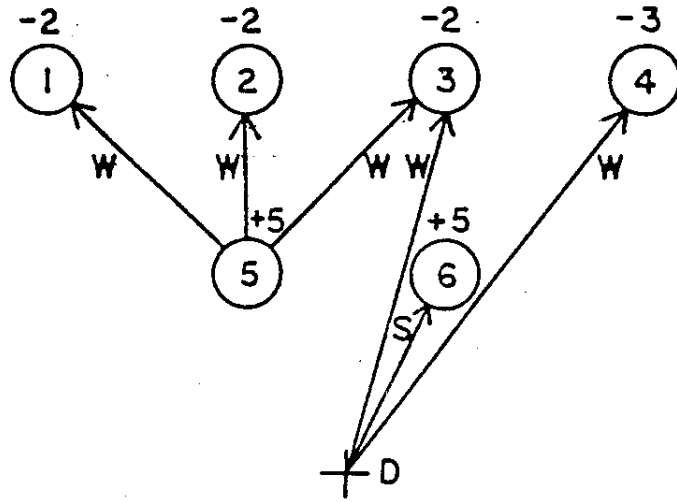


Figure 14. Tree Configuration after the Third Connection.

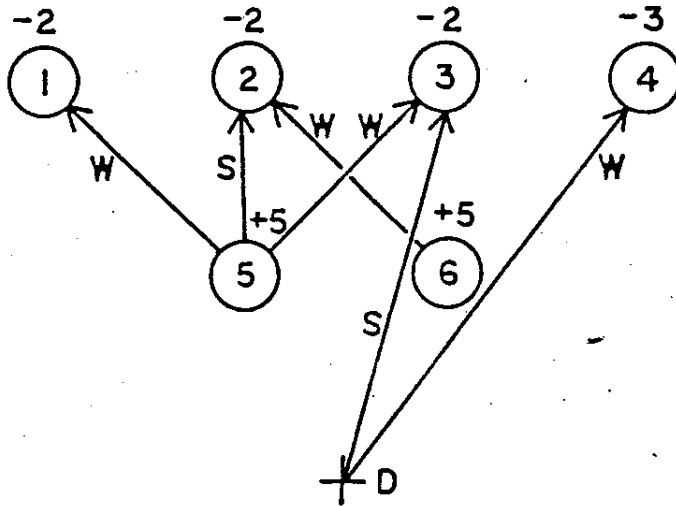


Figure 15. Tree Configuration after Connection of the Second Positive Vertex.

between node 5 and node 2, and this edge is not common to the dummy root. The normalization step proceeds such that the strong edge connecting node 5 to node 2 is deleted and node 2 is reconnected to the dummy root as (see eq. [3.10]):

$$P(2) = D ,$$

$$Q(2) = D , \text{ and}$$

$$S(5) = 1$$

The successive tree (Figure 16) containing node 2 and node 6 is automatically strong because of its positive total mass. However, the algorithm still evaluates the tree containing node 1, 3, and 5 (Figure 16) as :

evaluation; $CV(5) = BV(5) + CV(1) = 5 - 2 = +3$, and

$$CV(3) = BV(3) + CV(5) = -2 + 3 = +1$$

So, both successive trees are strong, but the algorithm still searches for the next overlying weak node for node 6. Node 4 is found to be a qualified node. Then,

transformation; $S(6) = 2$,

$$P(2) = 6 , \text{ and}$$

$$S(2) = D$$

connection; $P(6) = 4$, and

$$S(4) = 6$$

evaluation; $CV(2) = BV(2) = -2$,

$$CV(6) = BV(6) + CV(2) = 5 - 2 = +3 , \text{ and}$$

$$CV(4) = BV(4) + CV(6) = -3 + 3 = 0$$

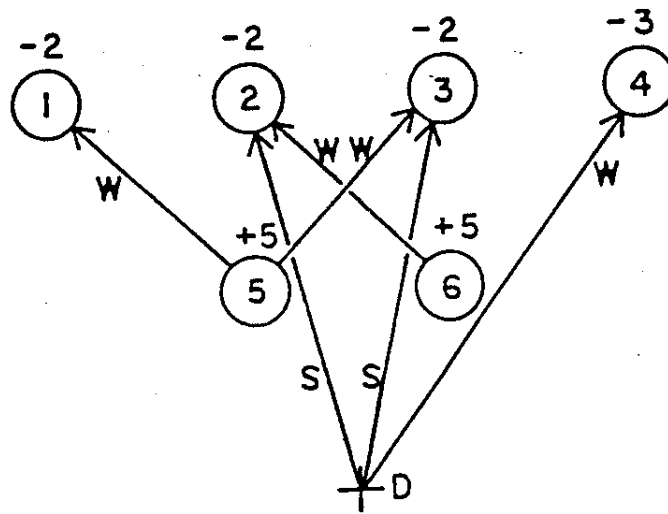


Figure 16. Tree Configuration after Normalization.

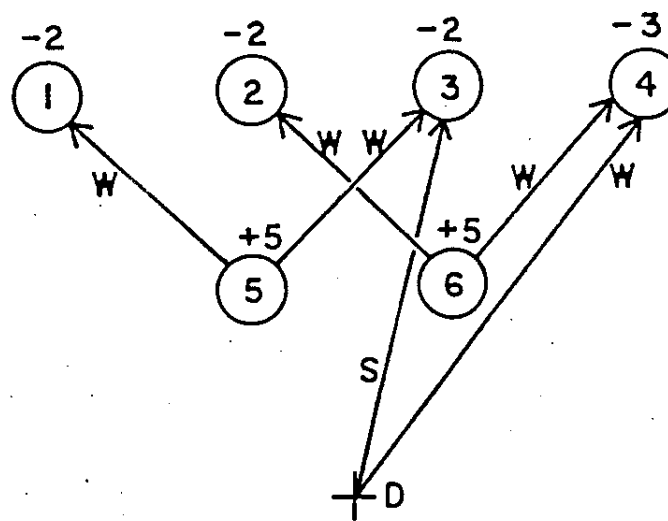


Figure 17. Tree Configuration after Connection of the Second Positive Vertex to Its Last Overlying Negative Vertex.

The tree containing node 6 (Figure 17) becomes weak, and so does the positive node 6.

Iteration No.2

The algorithm repeats the list of positive nodes again starting with node 5. Node 5 is strong, then, the search step looks for the overlying weak negative node. Node 2 is found, so the algorithm continues as :

transformation; $S(5) = 3$,

$P(3) = 5$,

$S(3) = D$, and

$Q(3) = 1$

connection; $P(5) = 2$, and

$S(2) = 5$

evaluation; $CV(3) = BV(3) = -2$,

$CV(5) = BV(5)+CV(1)+CV(3) = 5-2-2 = +1$,

$CV(2) = BV(2)+CV(5) = -2+1 = -1$,

$CV(6) = BV(6)+CV(2) = 5-1 = 4$, and

$CV(4) = BV(4)+CV(6) = -3+4 = +1$

No strong edge that is not common to the dummy is found. The final tree configuration (Figure 18) is found to be the maximum closure of a graph and all vertices are strong. The ultimate pit limit includes blocks 1 to 6.

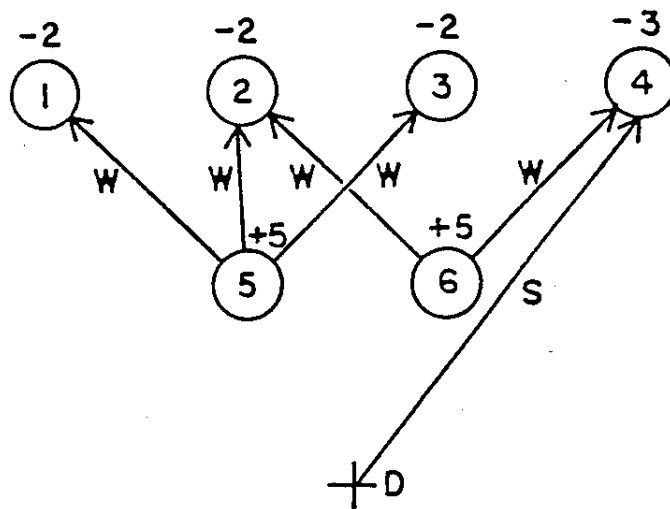


Figure 18. The final tree configuration showing the maximum closure of a graph

3.6 JUSTIFICATION FOR TRUE OPTIMIZING PURPOSE

Among these rigorous optimizing techniques, only the dynamic programming algorithm cannot provide a three-dimensional pit configuration unless the additional reaming subroutine is employed. Even with the use of the three-dimensional dynamic programming algorithm, the pit generated may not completely represent three-dimensional pit slope constraints (Seymour, 1985). One of the main reasons that severely restricts the applicability of the dynamic programming algorithm is that a generalized pit slope requirement is not possible with the algorithm since the pit slope must be consistent throughout the block model (Barnes, 1982). In contrast to the other techniques, the dynamic programming algorithm does not utilize a cone generating routine for obtaining a set of overlying blocks. Thus, the method may yield final pit limits which are far from optimal and will be eliminated from further consideration in this study.

The other three algorithms, the linear programming, the network flow programming, and the maximum closure of a graph, are the true optimizing techniques that can provide true three-dimensional ultimate pit limits. In Chen (1976), the graph algorithm has been shown to be a practical method for determining the ultimate pit limit with variable pit

slopes. It is obvious that the cone pattern for defining a set of overlying blocks, which must be removed before a given block is mined, can be applied to the other true optimizing algorithms without any difficulty.

Since the linear programming algorithm is impractical in real application and it is equivalent to the maximum network flow programming, it will also be eliminated from implementation on the case study in Chapter 5.

Chapter 4

THE PROPOSED METHOD OF MODIFIED-TREE
GRAPH ALGORITHM

4.1 INTRODUCTION

The major disadvantage of true optimizing methods is the tremendous requirement of computer memory and processing time. According to Barnes (1982), it was not uncommon in the past to exploit more than tens of CPU hours in computing a true ultimate pit limit. The requirement of large computer memory may limit the application of true optimizing methods to only on a high cost mainframe computer.

In the recent years, the improvement on the problem has mostly concentrated on reduction of number of blocks to be input into the pit limit algorithms. Many bounding techniques, such as in Rychkum and Chen (1979), and Barnes (1982), have been introduced as a prepass routine in an attempt to eliminate unnecessary blocks from the model without reducing the optimality of the ultimate pit solution.

In this chapter, a technique that directly reduces the computer resources required, both CPU time and memory, by an introduction of an extended Lerchs-Grossmann graphical algorithm is presented. The new algorithm recognizes that

certain modifications can be made to the tree configuration reducing its size, thus reducing storage requirements and processing time.

4.2 AN APPROACH OF LINEAR PROGRAMMING

The ultimate pit limit problem discussed under linear programming techniques (see eq. [3.3]) and illustrated in Figure 5 was formulated as follows :

$$\begin{aligned} \text{MAX } z &= c_1 x_1 + c_2 x_2 + c_3 x_3 + c_4 x_4 + c_5 x_5 + c_6 x_6 \\ \text{Subject to :} & \\ & -x_1 + x_5 \leq 0 \\ & -x_2 + x_5 \leq 0 \\ & -x_3 + x_5 \leq 0 \\ & -x_2 + x_6 \leq 0 \\ & -x_3 + x_6 \leq 0 \\ & -x_4 + x_6 \leq 0 \\ & x_n = 0 \text{ or } 1, \forall n \end{aligned}$$

Close consideration of the problem reveals that the objective function of this linear programming problem can be rearranged into the following three possible combinations :

1) If the positive block 5 must be mined, but not the positive block 6, then

$$\text{MAX } z = (c_1 x_1 + c_2 x_2 + c_3 x_3 + c_5 x_5) + (c_4 x_4 + c_6 x_6)$$

According to the slope constraints, the following relationships must be obtained; $x_1 = x_2 = x_3 = x_5 = 1$, and $x_4 = x_6 = 0$. So, the objective function can be transformed to :

$$\text{MAX } z = (c_1 + c_2 + c_3 + c_5) x_5 + (c_4 + c_6) x_6, \text{ or}$$

$$\text{MAX } z = s_5 x_5 + s_6 x_6$$

Where: $s_5 = c_1 + c_2 + c_3 + c_5$, and
 $s_6 = c_4 + c_6$

2) If the positive block 6 must be mined, but not the positive block 5, the objective function can be obtained as

$$\text{MAX } z = s_5 x_5 + s_6 x_6$$

Where: $s_5 = c_1 + c_5$, and
 $s_6 = c_2 + c_3 + c_4 + c_6$

3) If both of the positive block 5 and 6 must be mined, or neither is to be mined, the objective function can be formed as :

$$\text{MAX } z = s_5 x_5 + s_6 x_6$$

Where: $s_5 = c_1 + c_5$, or $c_1 + c_2 + c_3 + c_5$, or $c_1 + c_2 + c_3 + c_5$, and
 $s_6 = c_2 + c_3 + c_4 + c_6$, or $c_3 + c_4 + c_6$, or $c_4 + c_6$ respectively.

It can be concluded that for all of the above conditions, the objective function of the ultimate pit limit problem may be modified such that the number of variables is

reduced to only the total number of positive ore blocks. At the same time, it is shown that the problem has already considered and included the slope constraints into its objective function.

The main problem, then, is to determine the right combination for the value of all s 's. Fortunately, by the nature of the Lerchs-Grossmann graphical method, the maximum closure of a graph or an optimum tree is obtained through a repetitive process of computing the cumulative value supported by each edge connecting to any positive vertex of the successive tree which is equivalent to the calculation of the s values. This means that the solution of the linear programming problem, as constructed for ultimate pit limit analysis, is equivalent to solution of the maximum closure of a graph.

The graph method can, therefore, be used to solve the modified linear programming problem and can be used as the basis to reduce the graphical tree configuration. The Lerchs-Grossmann graphical technique which assures a true optimal solution can be modified such that the paths connecting a positive ore node to some of its overlying waste nodes are neglected. The sum value of the positive node and some of the overlying negative nodes is temporarily used as a new value of the positive node before applying the

graph algorithm. This new value of the positive node is recalculated through a repetitive process, and after a definite number of iterations, the true optimal solution can be obtained.

The new tree configuration will consist of the paths that directly connect one positive node to another positive node. All negative nodes can be eliminated from the modified tree and the relationship between a positive node and its overlying blocks can be recognized in terms of new modified values. If some additional properties of positive nodes are included, the computer algorithm can then be designed such that less computer memory and less processing time are required.

4.3 THE MODIFIED-TREE GRAPH ALGORITHM

One of the main problems generally encountered with the application of the original graph algorithm on a computer is the limitation of RAM capacity. For each iteration of the algorithm, the maximum number of nodes, including both positive and negative nodes, is normally limited to some maximum number if only RAM memory is used to store node properties. However, node properties can be stored and retrieved to and from auxiliary storage, e.g., a hard disk, reducing the a restriction on the maximum number of nodes.

As a result, this technique increases processing time significantly, especially for medium or large block model, since writing and reading data to and from the hard disk requires larger execution times. For these reasons, the graph algorithm must be extended in such a way that it minimizes the amount of data written and retrieved to and from the hard disk while increasing the maximum number of nodes to be executed at each iteration.

The proposed method for solving the ultimate pit limit problem on a computer employs almost the same steps of algorithm as the original graph method (see Section 3.5.1). The difference is that the tree configuration will be reduced to only connections between positive vertices. For example, if an ultimate pit limit problem as given in Figure 19. is solved by the original graph algorithm, the maximum closure of a graph or the final tree configuration is obtained as shown in Figure 20. If the new algorithm is applied, the tree configuration as recognized by the algorithm will be reduced to the one illustrated in Figure 21.

The modified algorithm still separates block nodes into positive node item and negative node item, but instead of connecting all nodes, including positive and negative nodes, in order to construct a network tree, only positive nodes

	1	2	3	4	5	6	7	8
1	-1	-1	-1	-1	-1	-1	-1	-1
2		-2	-2	-2	-2	-2	-2	
3			-3	+6	+6	-3		
4				+8	+8			

BLOCK 35 REPRESENTS MINING UNIT ON LEVEL 3 OF
COLUMN 5 WITH A VALUE OF +6

Figure 19. A Given Two-Dimensional Block Model with
Slope 1:1.

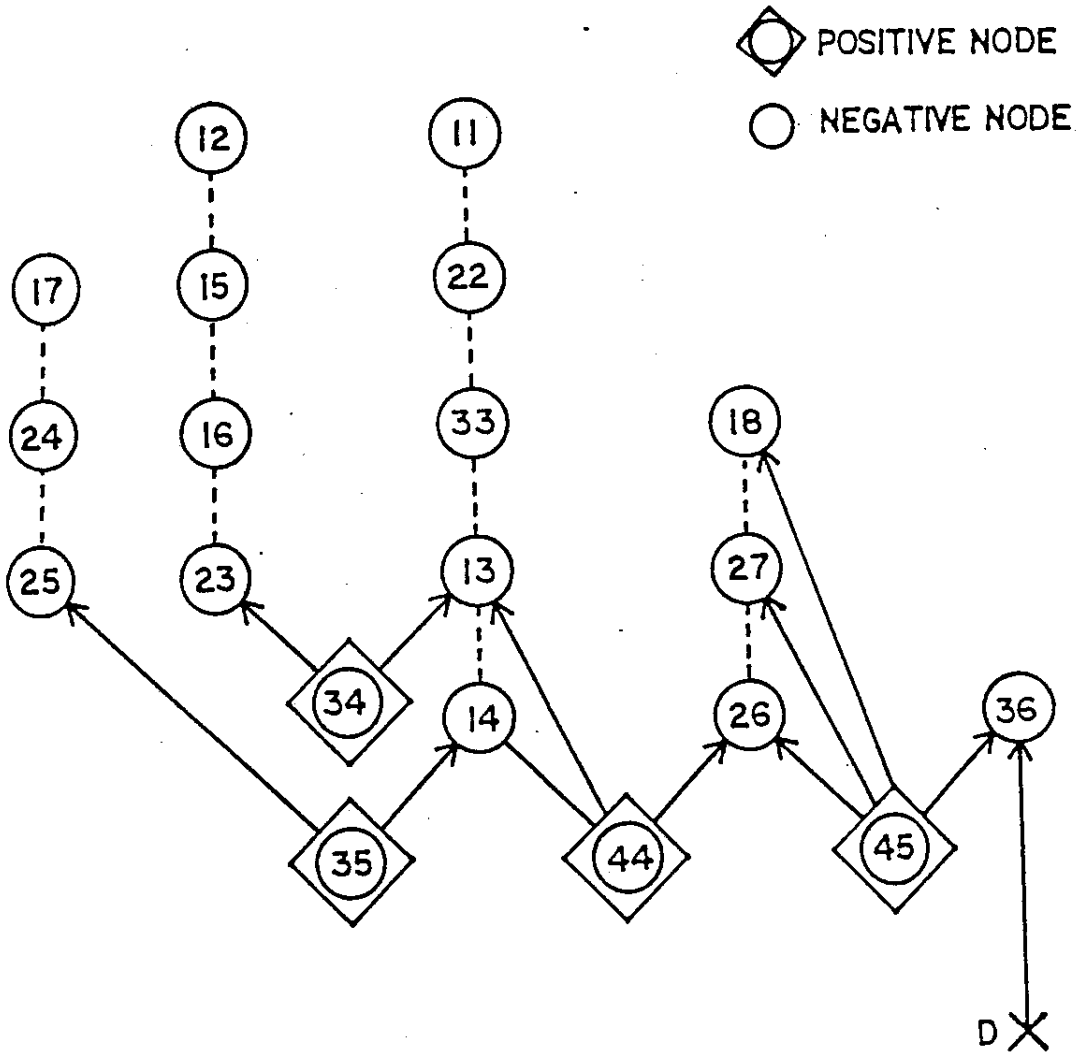


Figure 20. The Maximum Closure of a Graph for the Given Block Model in Figure 19.

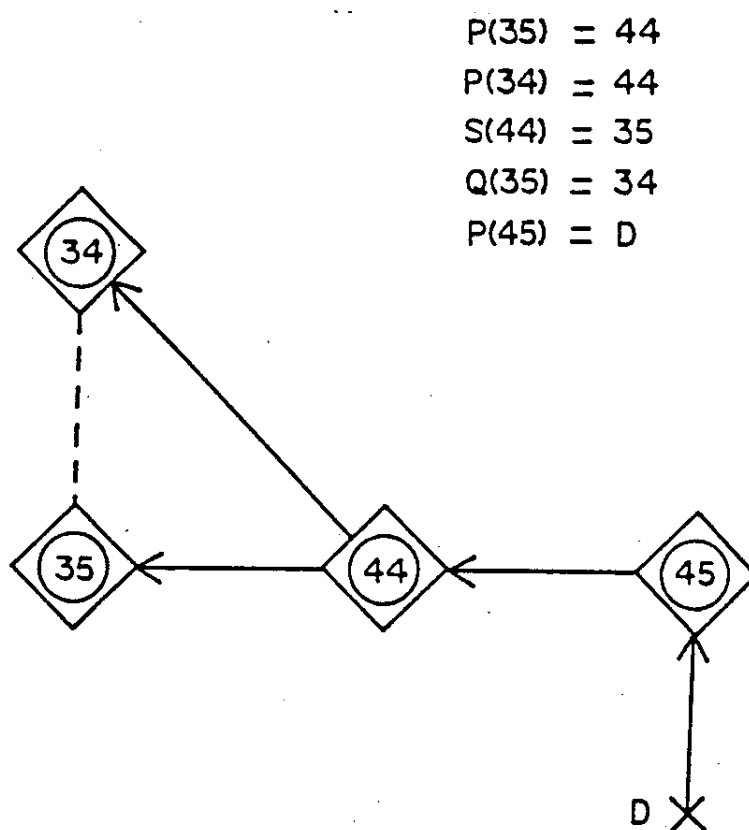


Figure 21. The Modified-Tree Configuration Showing the Maximum Closure of a Graph.

are directly connected together in the modified algorithm. The negative nodes can be left out from the tree while only their block values are used to calculate the total value of the tree, and an independent storage is created for keeping track of their positive root nodes. This reduced-tree graph algorithm is accomplished due to the following original graph algorithm's properties :

Property I

A particular negative node may be found overlying more than one positive node, but on the conventional tree configuration, there is only one positive node that is the predecessor node or root node of that negative node (the positive node may be the dummy root). This positive predecessor will pay for the costs of removing those negative successors.

Property II

When the mutual support between any two positive nodes must be considered in removing an overlying negative node, the tree must exist in such a way that the direct chain linking these two positive vertices together must pass through only one vertex which is that overlying negative node.

Property III

The transformation and normalization process of the original graph algorithm occurs only along the certain chain from a particular positive node down to the dummy root. This chain is the direct route to the dummy and it will pass through only some positive nodes and some linking negative nodes that connect those positive nodes together (including the one that is directly connected to the dummy root). The negative node that is not the linking node between any two of those positive nodes on the transformation or normalization path will never be considered in the transformation or normalization process of the graph algorithm.

By using the above properties, the modified algorithm can be completely given and the true optimality can be verified. Let :

- x,w denotes a positive vertex or node
- z denotes a negative block node
- D the terminal vertex, dummy root node, constant.
- P(x) be a positive node that is a predecessor or root node of node x
- S(x) be a positive node that is a direct successor or branch node of node x

Q(x) be a positive node that is another successor or branch node of node P(x) but is recognized in terms of node x in order to minimize a number of variables used.

N(x) an overlying negative block that needs a mutual support between node x and node P(x), this node links node x and P(x) together in the conventional tree.

R(z) be a positive node that pays for the costs of removing an overlying negative block z, or a predecessor root node of a negative node z in the conventional tree.

BV(y) be an original economic block value of all node y, where y is either positive or negative node.

SV(x) be a modified value of a positive node x, it is the sum of the positive block value and some of its overlying negative nodes, i.e.,

$$SV(x) = BV(x) + \sum_z BV(z) \quad (4.1)$$

where z is every negative node with R(z) = x and z ≠ N(x).

CV(x) be the total value supported by an edge that has node x as its branch node, or the cumulative value of all nodes on a subtree that has node x as its extreme root node, i.e.,

$$CV(x) = SV(x) + \sum_i CV(i) \quad (4.2)$$

where i is every successor node of node x and $P(i) = x$.

Let a graph $G = [X, A]$ represents a set of vertices x and directed arcs A where each vertex represents ore block or positive node containing a mass value equal to sum of the positive block value itself and some of its overlying waste block values, i.e., $SV(x)$. The steps of algorithm can be given as follows :

Initialization Step Classify every node y to be either positive or negative node depending on its economic block value $BV(y)$. For each positive node x , assign value to a term $SV(x)$ equal to $BV(x)$. Then, add a dummy vertex D and initialize the graph G as $[X, A_0]$ where A_0 is a set of all directed arcs connecting the dummy root D to every positive node x . The properties of each positive node x are initialized as :

$$\begin{aligned} P(x) &= D \\ S(x) &= D \\ Q(x) &= D \\ N(x) &= D \\ SV(x) &= BV(x) \end{aligned} \quad (4.3)$$

Since the modified-tree algorithm works in the same manner as the original graph algorithm of which each

negative node z must be connected to the dummy root, the term $R(z)$, for every negative node z , is used to recognize this concept as :

$$R(z) = D \quad (4.4)$$

Each arc connecting from the dummy to each positive vertex x is classified as a strong edge.

Search Step For a strong positive vertex v , search for an overlying negative node z of which $R(z)$ is weak or $R(z) = D$. If such a negative node is found, the algorithm goes to the transformation step. If not, the next positive node will be retrieved and the search step continues.

A positive node is defined as weak if it belongs to a tree that has a zero or negative mass value or cumulative value of all nodes on the tree, otherwise, it is strong. This cumulative value is not the sum of original block values, $BV(x)$, where x are all positive vertices on the tree, instead, it is calculated from the sum of modified values $SV(x)$, which are the summation of original economic value of block x itself and values of some of its overlying negative nodes (see eq. [4.1]). The calculation of this cumulative value is given in more detail in the transformation and evaluation step.

Transformation Step The tree T containing the strong positive vertex v is transformed into a successive tree

T_{t+1} . The change occurs along the direct chain linking node v to the dummy root of T_t . The tree T_t is transformed in such a way that :

$$\begin{aligned}
 S'(v) &= P(v) \\
 P'(x) &= u \\
 S'(x) &= P(x) \\
 N'(x) &= N(x) \\
 N'(P(x)) &= N(x) \\
 SV'(v) &= SV(v) + BV(N(v)) \\
 SV'(x) &= SV(x) - BV(N(x)) + BV(N'(x)) \quad (4.5)
 \end{aligned}$$

where x is every node on the direct chain linking node v to the dummy root of T_t , excluding node v and the dummy, i.e., node $P(v)$, $P(P(v))$, $P(P(P(v)))$, ..., etc. A symbol ' in eq. (4.5) denotes variables for the transformed tree T_{t+1} . And node u in eq. (4.5) is a successor of node x located on the transformation chain.

The change conforms with the original algorithm, that is, all the edges on the to be transformed chain of T_t have changed their status such that a p-edge in T_t becomes an m-edge in T_{t+1} and vice versa. But for the new tree configuration, which consists of only p-edges or the arcs that point away from the root, the transformation process just exchanges the positions between two vertices of any edge on the transformation chain.

Connection Step After the strong edge connecting from the dummy to the tree T containing node v has been removed in the transformation process, the algorithm establishes the directed arc from vertex $R(z)$ to vertex v to make a completion of the tree T . The connection is no longer the creation of the directed arc from node v to node z as in the original algorithm since the graph is reduced to only a set of positive vertices. The connection step proceeds as :

$$\begin{aligned} P'(v) &= R(z) \\ S'(R(z)) &= v \\ N'(v) &= z \end{aligned} \quad (4.6)$$

If $R(z) = D$, it will change to :

$$R'(z) = v \quad (4.7)$$

otherwise, it will not change the value.

If the vertex $S(R(z)) \neq D$, this vertex will be recognized in terms of v as :

$$Q'(v) = S(R(z)) \quad (4.8)$$

where the predecessor of $Q'(v)$ is still the vertex $R(z)$.

The term $Q(x)$ is introduced for the purpose of reducing a number of parameter used in the algorithm since any positive vertex in the tree may have more than one branch nodes. For example, if a positive vertex $R(z)$ has positive vertices $x_1, x_2, x_3, x_4, \dots$, as its successors, those nodes can be stored as :

$$Q(x_1) = x_2, Q(x_2) = x_3, Q(x_3) = x_4, \dots$$

where; $S(R(z)) = x_1$, and

$$P(x_1) = P(x_2) = P(x_3) = P(x_4) = \dots = R(z)$$

Evaluation and Normalization Step For each vertex x in the tree T_{t+1} , the value $CV(x)$ is calculated as given in eq. (4.2). This value represents the sum value of $SV(k)$ for every positive node k , including node x , of the subtree that has vertex x as its extreme root.

An edge connecting node x and node $P(x)$ is found strong if the value of $CV(x)$ is less than or equal to zero, or $CV(x)$ is positive and the sum of $CV(x)$ and $BV(N(x))$ is greater than zero. The first case is equivalent to when a strong m -edge is found in the original graph algorithm while the latter is equivalent to the case when a strong p -edge occurs. The normalization process must eliminate this strong edge and establish an edge from the dummy to node x .

If $CV(x)$ is negative, the tree is normalized as :

$$P'(x) = D$$

$$N'(x) = D$$

$$S'(P(x)) = D \quad (4.9)$$

The normalized tree containing node x becomes a weak tree, and so do all nodes in the tree, because the cumulative value of this successive tree is negative.

If $CV(x)$ is positive and the value of $CV(x)+BV(N(x))$ is also positive, the tree is normalized as :

$$P'(x) = D$$

$$S'(P(x)) = D$$

$$SV'(P(x)) = SV(P(x)) - BV(N(x)) \quad (4.10)$$

For this case, the tree containing node x becomes strong, since its resulting value is greater than zero.

At each iteration, the algorithm repeats from the search step through the evaluation and normalization step until no more qualified negative node or no more strong positive node is found. Since the modified algorithm employs the same fundamental concept of Lerchs and Grossmann theory as the original graph algorithm, the modified algorithm terminates after a definite number of iterations in the same manner. A maximum closure of a graph or an ultimate pit limit is obtained from a set of strong vertices and all negative blocks overlying these strong vertices.

It must be cited here that despite the tree is composed of connections between positive vertices, the relationship of any positive node and its overlying negative nodes still exists in the terms of $SV(x)$, $N(x)$, and $R(z)$. Since the term $R(z)$ can be independent from the modified-tree configuration and its properties (in terms of positive node), it is able to reduce the tree configuration to only

connections of positive vertices. To prove that the modified-tree graph algorithm is able to provide true optimal solution for all cases in the same manner as the original algorithm, the following theorem must be given in addition to the theorem presented in Lerchs and Grossmann(1965).

Theorem I

For a given tree, the transformation and normalization procedures of both graph algorithms occur on the chain containing the same elements.

Proof :

Let $[x_1, z_1, x_2, z_2, x_3, z_3, \dots, D]$ be the transformation chain or normalization chain of the original graph linking vertex x_1 to the dummy D , and let x_i be a positive vertex and z_i be a negative vertex. By using Property II and III of the original graph algorithm and by introducing a term $N(x_i) = z_i$, the given chain can be rearranged as $[x_1, N(x_1), x_2, N(x_2), x_3, N(x_3), \dots, D]$. Since N is a function of x_i and there must be only one element $N(x_i) = z_i$, then the chain can be reduced to $[x_1, x_2, x_3, \dots, D]$ in the modified-tree. In following the given steps of the modified-tree graph algorithm, it is obvious that the transformation and normalization processes are able to recognize and evaluate the same negative vertices on a given

chain of original tree.

Theorem II

For a given ultimate pit limit problem, a maximum closure of graph $G_x = (X, A_x)$ obtained from the modified-tree graph algorithm must contain the same mass or total value as a maximum closure of graph $G_y = (Y, A_y)$ obtained from original graph algorithm.

Proof :

By the definition of the problem, if a given original graph G_y of which each vertex y_i is assigned a numerical value $BV(y_i)$ called the mass of y_i , a closure of graph G_y with maximum mass must be determined. A set of elements that is a subset of Y must be found such that $\sum BV(y_i)$ is maximized. By using Property I of the original graph, the new algorithm neglects all the arcs connecting a positive vertex x_j to each of its negative successors z_k such that the new modified mass $SV(x_j)$, as given in eq. (4.1), is assigned to each positive vertex x_j , where X is a set of all positive vertices and $X \subseteq Y$. The modified algorithm attempts to find a closure of graph G_x such that $\sum SV(x_j)$ is maximized. Since the a given tree T_0 must be transformed into successive trees $T_1, T_2, T_3, \dots, T_n$ following given rules until no further transformation is possible. Theorem I guarantees that a transformed tree must contain the same

elements for both graph algorithms. The search step of the algorithms ensures that there will be no unsupported negative vertex when the final successive tree is reached. Then, for a graph G , it can be concluded that a set of elements which is a subset of X must be determined such that $\sum_j SV(x_j) = \sum_i BV(y_i)$ is maximized.

The advantage of the modified algorithm is that the number of variables used is drastically reduced because of the modified tree configuration. Moreover, the algorithm may reduce some steps of tracing back and forth when searching for some particular vertices of a tree in the evaluation process.

4.4 AN EXAMPLE

For ease of conceptualization, a simple two-dimensional example is presented here to demonstrate and clarify the algorithm as given in the previous section. The algorithm will solve for the ultimate pit limit from a given block model with a slope constraint 1:1 as shown in Figure 22.

Initialization

Assign value to $BV(y)$ equal to economic block value of each block y ($y = 1$ to 6). Classify block 1 to 4 as negative blocks while block 5 and 6 are positive blocks. Connect arc from the dummy root D to a positive block x (see Figure 23). Initialize the following terms of x ($x = 5, 6$)

1	2	3	4
-2	-2	-2	-3
	5	6	
	+5	+5	

Figure 22. A Given Two-Dimensional Block Model.

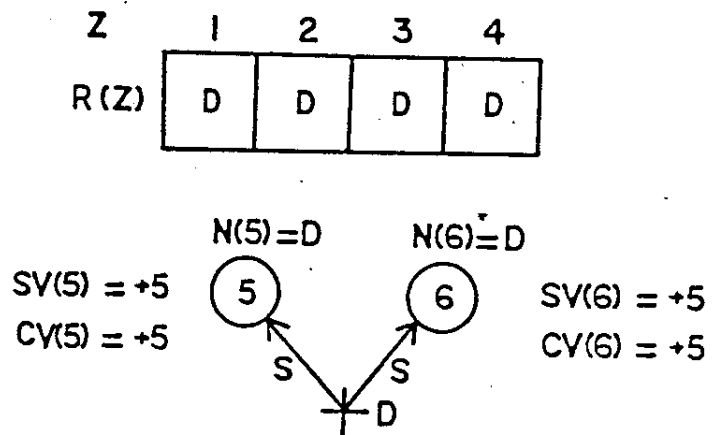


Figure 23. Modified-Tree Configuration at the Initialization Step

using eq. (4.3) as :

$$P(x) = D$$

$$S(x) = D$$

$$Q(x) = D$$

$$N(x) = D , \text{ and}$$

$$SV(x) = BV(x)$$

For each negative node z , for $z = 1$ to 4 , only a term $R(z)$ is initialized as (see eq. [4.4]) :

$$R(z) = D$$

Iteration No. 1

Start with positive node 5, the algorithm proceeds from the search step to the evaluation step (see Section 5.3). Node 1 is found to be the first overlying node of node 5. The transformation process does not change the tree configuration since the root node of node 5 is the dummy, or $P(5) = D$. The connection step (eq. [4.6] and [4.7]) proceeds as :

$$P(5) = R(1) = D ,$$

$$N(5) = 1 , \text{ and}$$

$$R(1) = 5$$

Since the first tree (Figure 24) contain only node 5, the evaluation step (eq. [4.2]) proceeds as :

$$CV(5) = SV(5) = +5 , \text{ and}$$

$$CV(5) + BV(1) = 5 - 2 = +3$$

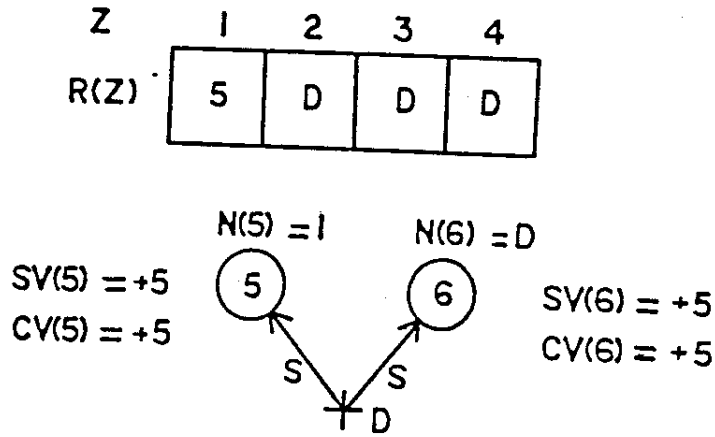


Figure 24. Modified-Tree Configuration after the First Connection.

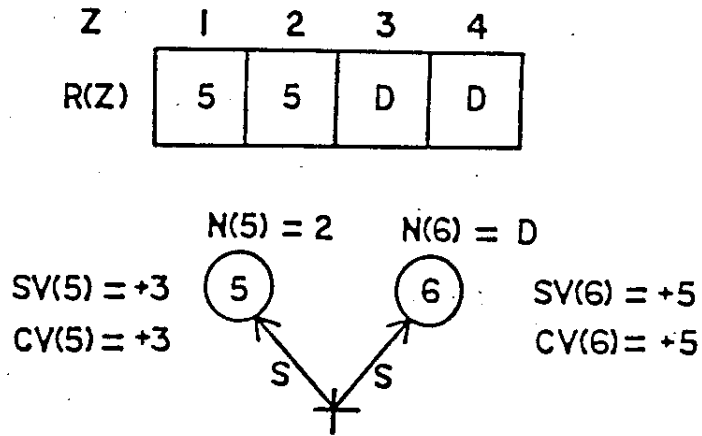


Figure 25. Modified-Tree Configuration after Connection of the Second Overlying Vertex.

This tree is strong because of the resultant positive value. No normalization process is needed since node 5 is already connected to the dummy. Then, the algorithm searches for the next overlying node which is node 2. The algorithm repeats as follows :

transformation(eq. [4.5]);

$$S(5) = D ,$$

$$SV(5) = SV(5)+BV(1) = 5-2 = +3$$

connection;

$$P(5) = R(2) = D ,$$

$$N(5) = 2 ,$$

$$R(2) = 5$$

evaluation;

$$CV(5) = SV(5) = +3 ,$$

$$CV(5)+BV(2) = 3-2 = +1$$

The successive tree (Figure 25) is still strong so that the algorithm continue searching for another overlying node of node 5. Node 3 is such a node, so :

transformation;

$$S(5) = D ,$$

$$SV(5) = SV(5)+BV(2) = 3-2 = +1$$

connection;

$$P(5) = R(3) = D ,$$

$$N(5) = 3 ,$$

$$R(3) = 5$$

evaluation;

$$CV(5) = SV(5) = +1 ,$$

$$CV(5)+BV(3) = 1-2 = -1$$

Since the overall value of the tree (Figure 26) becomes

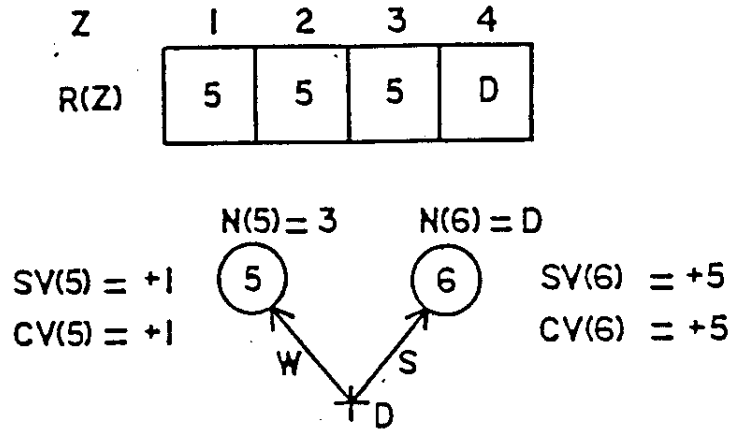


Figure 26. Modified-Tree Configuration after the Third Connection.

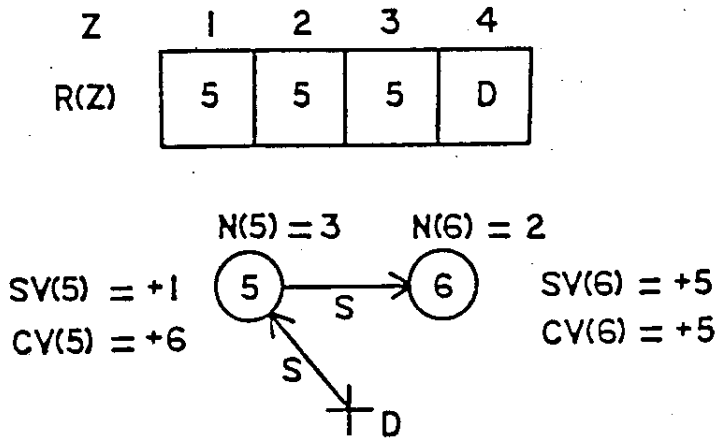


Figure 27. Modified-Tree Configuration after Connection of the Second Positive Vertex.

negative, $CV(5)+BV(3) < 0$, the tree is classified as weak, and so does node 5. Then, the algorithm retrieves the next available positive node from the list which is node 6. The search step finds that node 2 is the first node overlying node 6. Since node 2 has been supported by positive node 5, or $R(2) = 5$, node 6 will share a mutual support of block 2 with node 5. The algorithm proceeds as :

transformation; $S(6) = D$,
 $SV(6) = +5$
 connection; $P(6) = R(2) = 5$,
 $S(5) = 6$,
 $N(6) = 2$ (see eq. [4.6])
 evaluation; $CV(6) = SV(6) = +5$,
 $CV(6)+BV(2) = 5-2 = +3$

Now, the sum value of $CV(6)+BV(2)$ is greater than zero and node 6 is not common to the dummy (see Figure 27), or $P(6) = 5$, so the strong edge occurs and the normalization step (eq. [4.10]) is needed as :

normalization; $P(6) = D$,
 $S(5) = D$, and
 $SV(5) = SV(5)-BV(2) = 1+2 = +3$

After the normalization step, the tree configuration is illustrated in Figure 28. The tree containing node 6 is strong because of its positive total value. The algorithm

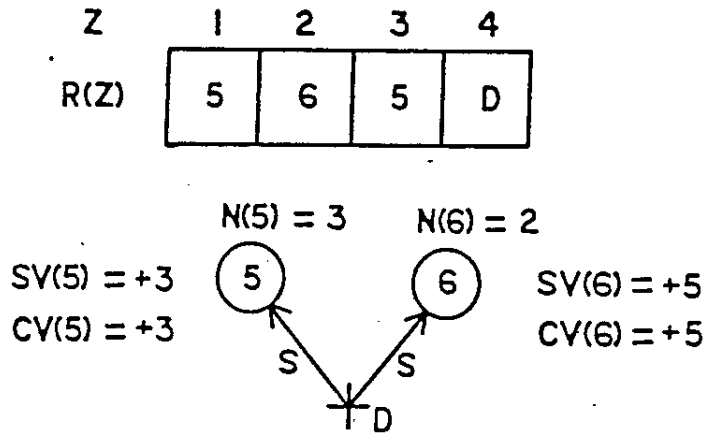


Figure 28. Modified-Tree Configuration after Normalization.

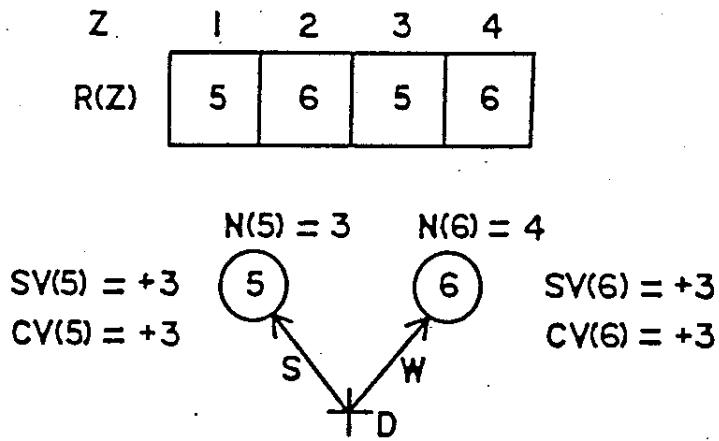


Figure 29. Modified-Tree Configuration after Connection of the Second Positive Vertex to Its Next Overlying Negative Node.

continues evaluating the tree containing node 5 (Figure 28), as :

$$CV(5) = SV(5) = +3 ,$$

$$CV(5)+BV(3) = 3-2 = +1$$

Also, the tree containing node 5 is strong because its total value becomes positive.

The search step continues looking for the next overlying negative node of node 6. Node 3 is such a node. However, node 3 has been supported by node 5, or $R(3) = 5$, and node 5 is strong, the search process will bypass node 3. The next negative node that is qualified is node 4, $R(4)=D$, then :

transformation;	$S(6) = D ,$
	$SV(6) = SV(6)+BV(2) = 5-2 = +3$
connection;	$P(6) = R(4) = D ,$
	$N(6) = 4 ,$
	$R(4) = 6 ,$ and
evaluation;	$CV(6) = SV(6) = +3 ,$
	$CV(6)+BV(4) = 3-3 = 0$

The tree containing node 6 (Figure 29) becomes weak because its total value is zero. Since the algorithm has used up all the positive nodes in the list, it will start over the positive node list again in the next iteration.

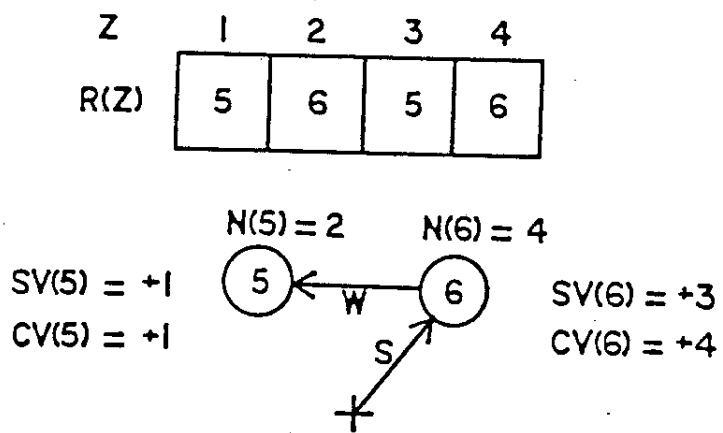


Figure 30. Modified-Tree Configuration Showing the Maximum Closure of a Graph, All Positive Vertices are Mined and So Do All Their Overlying Negative Blocks.

Iteration No. 2

The algorithm begins with node 5 which is the first node in the list and it is strong. The search step will bypass the overlying node 1 since the node 1 has already supported by node 5, $R(1) = 5$. The next qualified node is node 2 of which $R(2) = 6$, and node 6 is weak, then :

transformation; $S(5) = D$,
 $SV(5) = SV(5)+BV(3) = 3-2 = +1$
 connection; $P(5) = 6$,
 $S(6) = 5$,
 $N(5) = 2$
 evaluation; $CV(5) = SV(5) = +1$,
 $CV(5)+BV(2) = 1-2 = -1$,
 $CV(6) = SV(6)+CV(5) = 3+1 = +4$,
 $CV(6)+BV(4) = 4-3 = +1$

The sum value of $CV(6)+BV(4)$ is the cumulative value of the whole tree (Figure 30). Since the tree is strong and there is no more unsupported overlying negative node, or $R(z) \neq D$ and $R(z)$ is not weak ($z = 1$ to 4), the ultimate pit limit is found including strong node 5, 6, and all of their overlying negative nodes which are node 1, 2, 3, and node 4.

This example demonstrates that the new algorithm is capable of considering the mutual support between any two

positive nodes and yielding the true optimal solution as well as the original graph algorithm.

The modified-tree graph algorithm can be extended to solving the three-dimensional problem without any difficulty. The main problem concerning the three-dimensional case is the search for an overlying negative block. The algorithm can easily employ a cone generator which is able to create a three-dimensional inverted cone corresponding to given allowable pit slope requirements. After placing the cone vertex on a positive block, the blocks enclosed within the cone can be specified as the overlying blocks by the search step of the algorithm.

4.5 SOME CONSIDERATIONS FOR THE PROPOSED ALGORITHM

There are some considerations when the modified algorithm is applied to the problem of more than two positive nodes must create the specific mutual support of one common negative node together in order to remove all mining blocks.

For example, positive blocks 2, 3, and 4 of the problem as given in Figure 31 must create the specific mutual support of a negative block 1 together so that all blocks can be mined. For the original Lerchs and Grossmann algorithm, the maximum closure of a graph and the tree configuration can be found as shown in Figure 32. But the

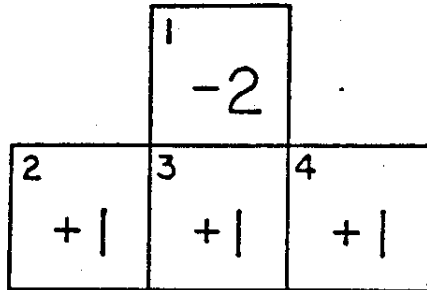


Figure 31. A Given Two-Dimensional Block Model for Some Consideration of Modified-Tree.

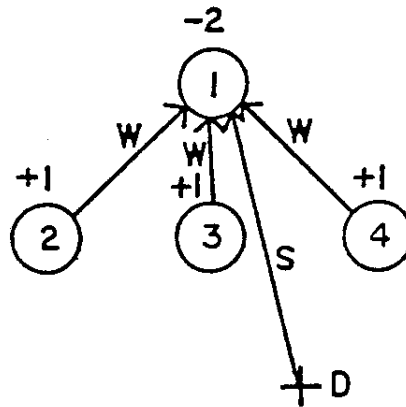


Figure 32. The Original Tree Configuration Showing the Final Graph of the Given Problem.

new tree configuration of the modified algorithm does not contain any negative node except in the imaginary connection which takes into account of only a negative node linking between any two positive nodes. This means that an additional parameter in terms of positive vertex x must be introduced to the algorithm so that the special problem can be solved.

Let a variable $M(x)$ represents a positive node that creates a specific mutual support of the same overlying negative block z with a positive node x (for the conventional tree, they are connected to the same negative node as shown in Figure 32). The above definition is true if and only if :

1. Node x and node $M(x)$ mutually support the specific overlying negative node z as :

$$N(x) = N(M(x)) = z \quad (4.11)$$

2. In addition to 1., if $R(z) = x$, then :

$$P(x) = D$$

$$P(M(x)) = x \quad (4.12)$$

and $M(x)$ must be the successive node of x and $CV(M(x))$ must be used in the calculation of $CV(x)$ using eq. (4.2).

3. If $R(z) = w$, where $w \neq D$, $w \neq x$, and $w \neq M(x)$, and $N(w) \neq z$, then :

$$P(x) = w, \text{ and}$$

$$P(M(x)) = w \quad (4.13)$$

and both x and $M(x)$ must be the successive nodes of w and $CV(x)$ and $CV(M(x))$ must be used in the calculation of $CV(w)$. For this case vertices w , x and $M(x)$ create a specific mutual support of a negative node z together.

4. It is possible that there will be more than two positive nodes create the specific mutual support of the same negative node as :

$$N(x_1) = N(x_2) = N(x_3) = N(x_4) = \dots = z$$

The algorithm memorizes the tree connection as :

$$M(x_1) = x_2, M(x_2) = x_3, M(x_3) = x_4, \dots$$

where $x_1, x_2, x_3, x_4, \dots$ are those positive nodes that create a specific mutual support of the same negative node in the graph algorithm.

In the same manner as the previous properties in 2. and 3., if $R(z) = x_1$ and $P(x_1) = D$, then (eq. [4.12]) :

$$P(x_2) = P(x_3) = P(x_4) = \dots = x_1$$

And if $R(z) = w$, where $w \neq D$, $w \neq x_1$, $w \neq x_2, \dots$, and $N(w) \neq z$, then (eq. [4.13]) :

$$P(x_1) = P(x_2) = P(x_3) = P(x_4) = \dots = w$$

After the additional parameter and the above properties are established, the special problem in Figure 31 can be solved using the modified-tree graph algorithm as follows.

Initialization

Nodes 2, 3, and 4 are positive nodes while node 1 is only one negative node. The graph is initialized as shown in Figure 33. and the following variables are initialized as :

$$P(x) = D ,$$

$$S(x) = D ,$$

$$Q(x) = D ,$$

$$M(x) = D ,$$

$$N(x) = D ,$$

$$SV(x) = BV(x) , \text{ and}$$

$$R(z) = D ,$$

where $x = 2, 3, \text{ and } 4$; and $z = 1$.

Iteration No.1

Node 2 is the first strong positive node retrieved from the queue. Since node 1 is the only one overlying negative node, the algorithm proceeds as :

transformation; $S(2) = D ,$

$$SV(2) = +1$$

connection; $P(2) = R(1) = D ,$

$$N(2) = 1 ,$$

$$R(1) = 2$$

evaluation; $CV(2) = SV(2) = +1 ,$

$$CV(2)+BV(1) = 1-2 = -1$$

The tree (Figure 34) becomes weak, so the next

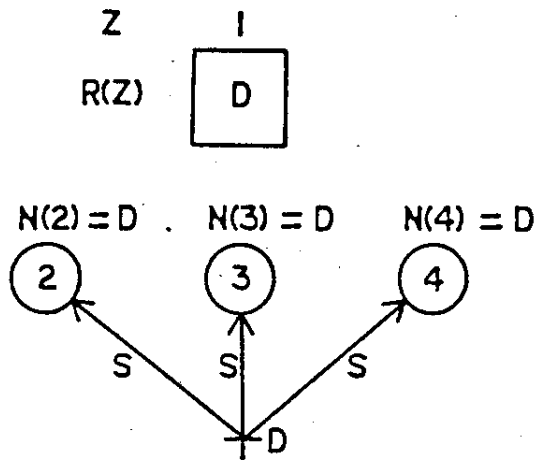


Figure 33. The Tree Configuration after Initialization.

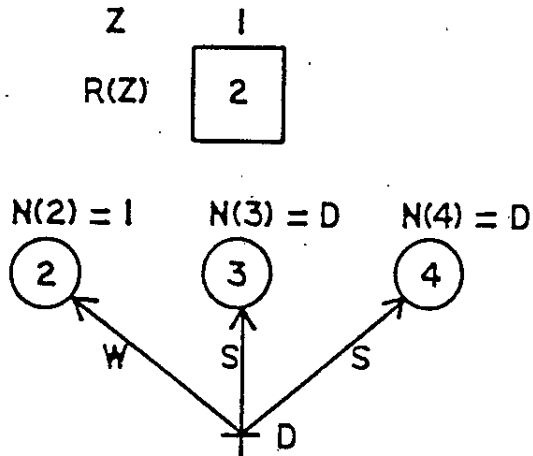


Figure 34. Tree Configuration after Connection of the First Positive Vertex.

available strong positive node which is node 3 will be put into the process. Since node 1 has been supported by node 2, or $R(1) = 2$, then node 3 share the mutual support of node 1 with node 2 as :

transformation; $S(3) = D$,

$$SV(3) = +1$$

connection; $N(3) = 1$,

that is $N(2) = N(3)$ as stated in eq. (4.11), so the connection step continues as (eq. [4.12]) :

$$P(3) = R(1) = 2$$

$$M(2) = 3 , \text{ (not } S(2) = 3)$$

evaluation; $CV(3) = SV(3) = +1$,

$$CV(2) = SV(2) + CV(3) = 1 + 1 = +2,$$

$$CV(2) + BV(1) = 2 - 2 = 0$$

The tree (Figure 35) becomes weak because of the total mass of zero. Then, positive node 4 is put into the next step as :

transformation; $P(4) = D$,

$$SV(4) = +1$$

connection; $N(4) = 1$,

that is $N(2) = N(3) = N(4)$, and the algorithm continues the connection step as :

$$P(4) = R(1) = 2 ,$$

$$M(3) = 4 ,$$

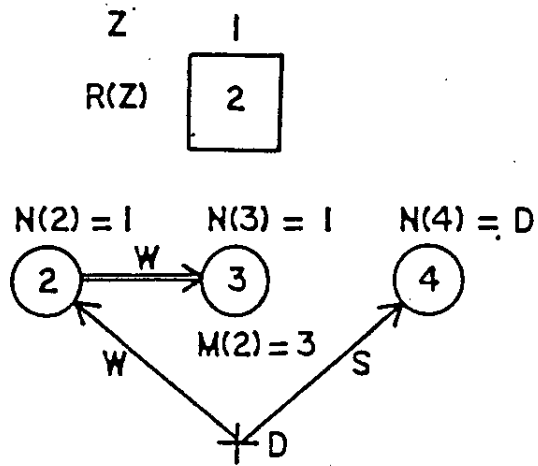


Figure 35. Tree Configuration after Connection of the Second Positive Vertex.

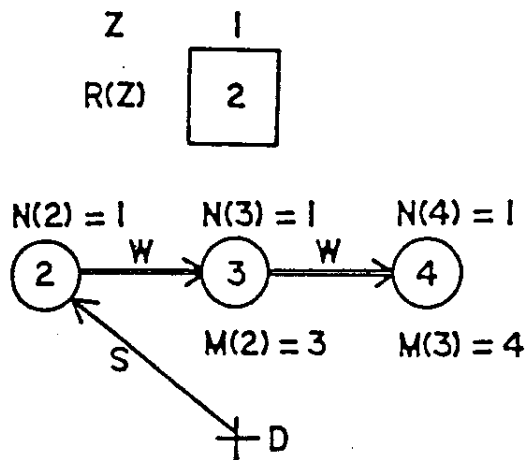


Figure 36. The Final Tree Configuration after Connection of the Last Positive Vertex.

evaluation;

$$CV(4) = SV(4) = +1 ,$$

$$CV(3) = SV(3) = +1 ,$$

$$CV(2) = SV(2) + CV(3) + CV(4) = 1 + 1 + 1 = +3 ,$$

$$CV(2) + BV(1) = 3 - 2 = +1$$

The final tree (Figure 36) becomes strong and the maximum closure of a graph is obtained. All blocks are included in the ultimate pit limit.

4.6 CONCLUSION

For the fundamental concept, the modified-tree graph algorithm is not different from the original graph algorithm. The proposed method applies the same concepts of tree transformation and normalization as outlined by Lerchs and Grossmann (1965). The main change is the tree configuration and the way the algorithm recognizes the connection between the positive vertices themselves and their overlying negative vertices.

For the modified-tree graph algorithm, the relationship of positive nodes and their overlying negative nodes can be recognized by creating some additional parameters in terms of positive nodes. Most of the arrays in the algorithm are dimensioned equals to the total number of positive blocks. Due to the facts that the number of positive ore blocks in a given model may be approximately 10% to 20% of the total blocks, the total number of variables required by the

algorithm is greatly reduced.

As illustrated in Section 4.4 and 4.5 using simple numerical examples, it can be concluded that the modified-tree graph algorithm is able to solve for an optimal ultimate pit limit. The algorithm can employ a cone pattern representing three-dimensional slope requirement in order to generate the three-dimensional pit geometry without any difficulty.

Chapter 5

IMPLEMENTATIONS ON A CASE STUDY

5.1 INTRODUCTION

An operational computer program was developed in order to carry out a comparative performance between the two existing true optimization algorithms (the network flow and the original graph methods) and the proposed algorithm of modified-tree graph method. The computer program SAMPIT.FOR is a pit optimizing routine utilizing the Johnson's Network Flow Algorithm, the original Lerchs and Grossmann's Graph Algorithm, and the modified-tree graph algorithm. The computer code is included in Appendix A.

The study attempts to solve a true ultimate pit limit for the given economic block model obtained from a porphyry copper deposit. The block model is discussed in more details in Section 5.2.

The description of the computer program is given in Section 5.3 and also in Appendix B. And the results from implementing the true optimizing algorithms is presented in Section 5.4. The analysis of the results obtained from each method in some orientations, such as, the computer memory required, and the processing time, is discussed in Section 5.5.

Since the modified-tree graph algorithm has the tendency of utilizing less computer memory, the study on the execution of the new developed algorithm for determining ultimate pit limits on a microcomputer is carried out in this chapter. The limitations of applying the algorithm on a microcomputer is given in Section 5.6.

5.2 THE GIVEN PHYSICAL MODEL

The block model of a given porphyry copper deposit, which is utilized in the study, consists of 225,000 blocks. The block dimensions were 80 feet in length and width and 40 feet in height, and the model volume was 6,000 feet by 6,000 feet by 1,600 feet. The maximum allowable pit slope angle in all directions was assumed to be 45 degrees.

The block model on which the program was tested was oriented so that the first two horizontal dimensions, along X-axis and Y-axis, correspond to the Easting and the Northing coordinate respectively, the third dimension in vertical direction or Z-axis corresponds to the elevation.

The block grade was obtained by extending the drilling assays to each block using the inverse distance square method. The block grade distribution by level is given in Appendix C. The economic assumptions made in the development of the economic block model are as follows:

Price of copper = \$0.95 per lb

Mining cost = \$0.95 per ton ore
Milling cost = \$2.75 per ton ore
Smelting & Marketing = \$0.40 per lb Cu
Administrative cost = \$0.25 per ton ore
Mill recovery = 85%
Ore concentrate = 25%
Smelting loss = 11.0 lb per ton concentrate
Refinery loss = 5.5 lb per ton blister
Stripping cost = \$0.80 per ton material
(Projected rate of change in stripping cost is
\$0.00025 per ft of increased lift)

It is assumed that there is no uncertainty associated with the ore body, mining and processing methods, and the market conditions, so that all geological and financial information are known for the given model.

The block values were written on a direct-access data file. The internal computer storage of about 1760 disk blocks (450 Kbytes) was required to store this economic block model. The original surface of the ore deposit is also required by the optimizing program. The values indicated surface elements were written on the data file in a sequential-access manner. The description of these input data files and some interactively obtained data is given in the computer program's description in Appendix B.

5.3 THE COMPUTER PROGRAM

The computer program SAMPIT.FOR was written in FORTRAN 77 and executed on a DEC VAX8600 running a VMS4.2 operating system. Three optimizing algorithms, which are maximal network flow, maximum closure of a graph, and modified-tree graph algorithm, were coded after the flow charts of algorithms as given in Figure 37 and 38. Each algorithm was set up as a separate subroutine under the same main driving routine. The choice of the algorithm can be interactively selected by the user. The computer code was designed in a modular form with a number of comments included as much as possible.

The input subroutines reads the economic block model, model and block dimensions, the surface data file, and pit slope angles, which are common to and required by each algorithm's optimizing routine.

True optimizing algorithms usually require a significant computational effort and may require a large number of blocks to be processed in each iteration. To avoid exceeding the limit imposed by computer's memory, each algorithm utilizes a bounding routine to reduce the number of blocks in the calculation process. A simple bounding technique, which generates an outer bound that includes all positive blocks and their overlying negative blocks, is

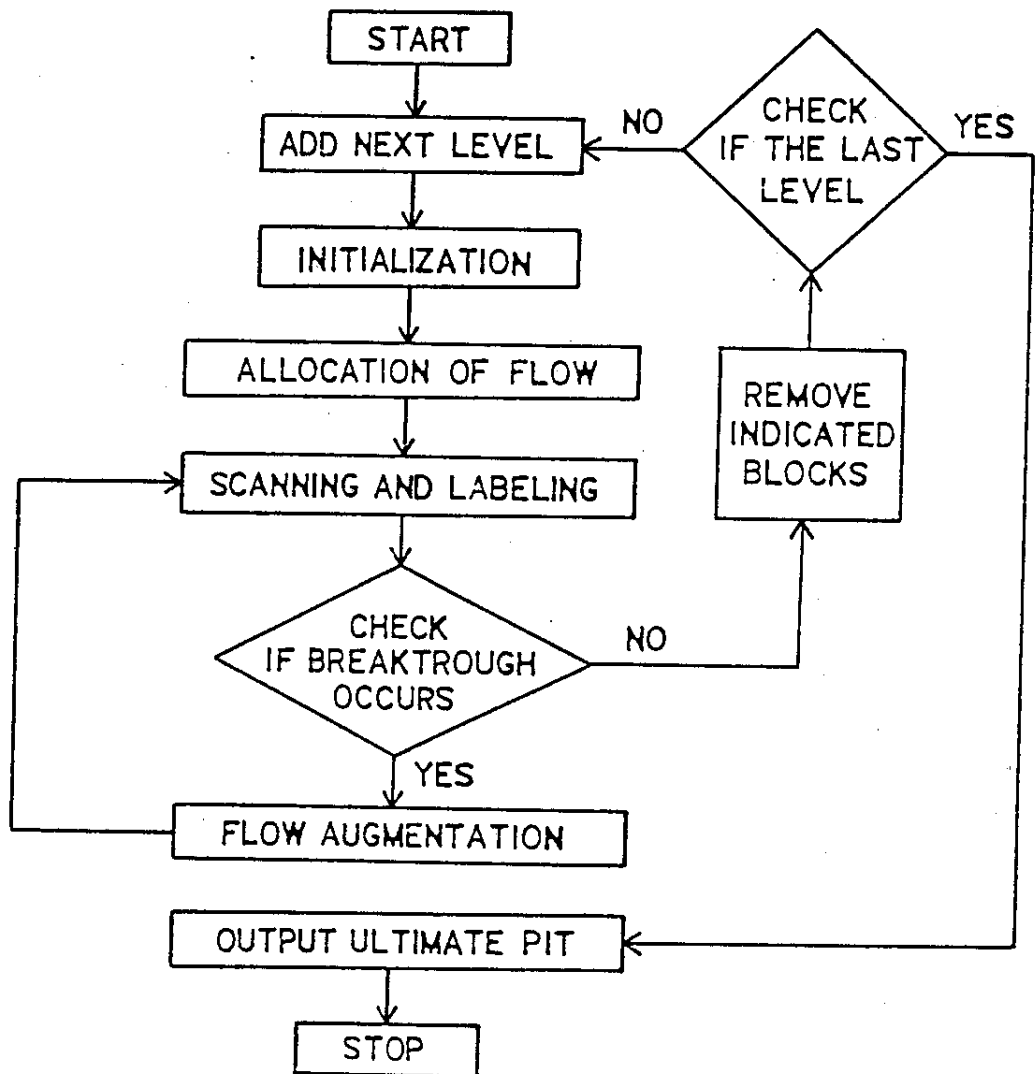


Figure 37. Flowchart of the Maximum Network Flow Algorithm.

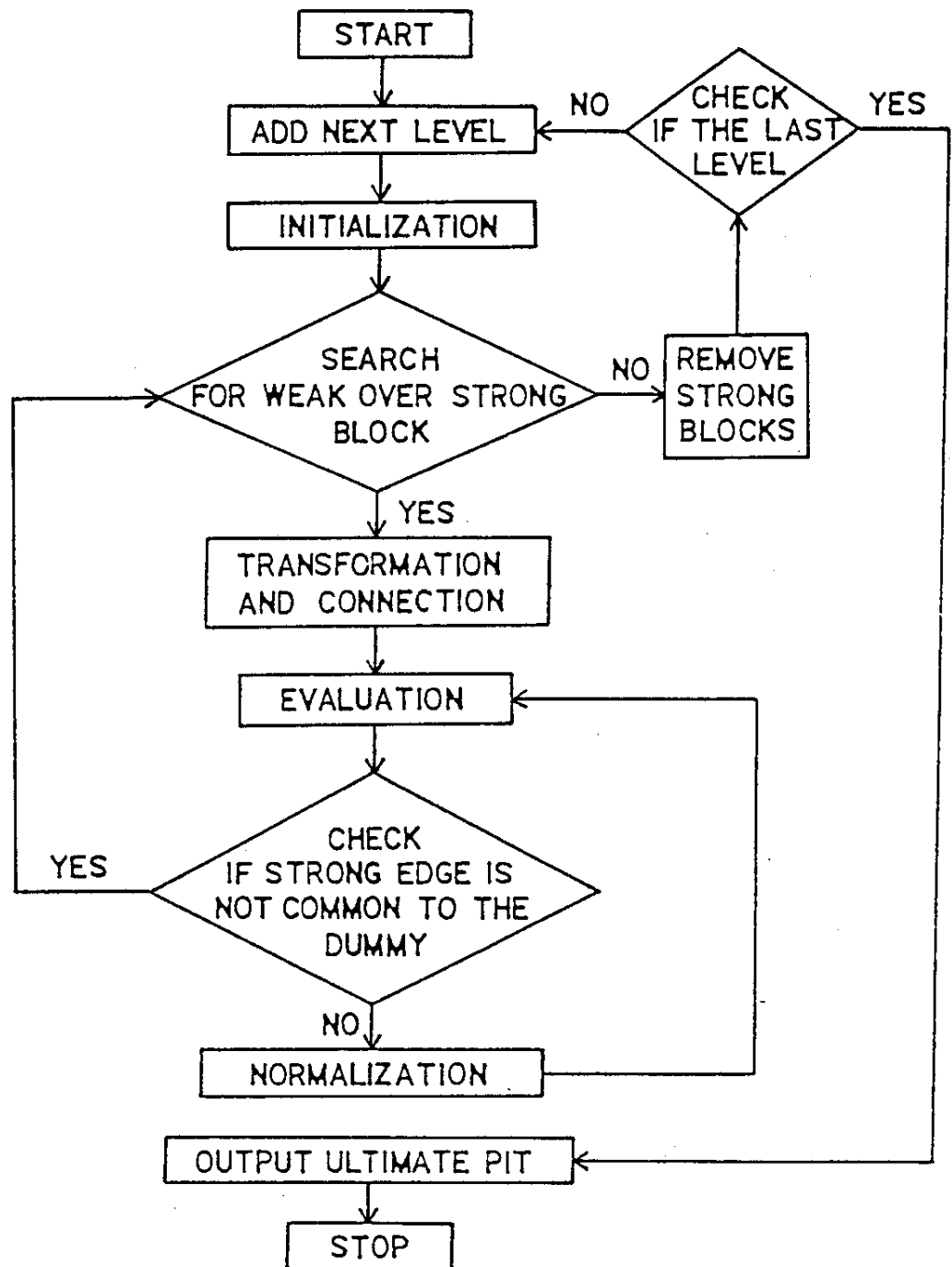


Figure 38. Flowchart of the Graph and the Modified-Tree Graph Algorithms.

employed in the program in order to eliminate unnecessary blocks and reduce the amount of computer memory required.

Each true optimizing subroutine operates on one level of the block model at a time and incrementally moves down from the top level to the bottom level. After finishing each level, a set of blocks that yields an incremental pit is removed from the algorithm so that the required computer memory space is reduced.

The purpose of the codes used in this study is to make a comparative study between these true optimizing algorithms.

5.4 IMPLEMENTATIONS AND RESULTS

In order to compare the results from the three methods for different cases, each method is executed on the same block model but the limitation on the maximum level of excavation is imposed as :

Case 1 The excavation is limited to the first 10 levels and total number of blocks is 56,250.

Case 2 The excavation is limited to the first 20 levels and the total number of blocks is 112,500.

Case 3 The excavation is limited to the first 30 levels and the total number of blocks is 168,750.

Case 4 No limitation, the full model is tested and the total number of blocks is 225,000.

The output obtained from Case I is the original surface file input for Case II. Likewise, the output of Case II is used for Case III, and the output of Case III for Case IV. The purpose of running four cases is to analyze the effects of depth on the algorithms.

The total number of blocks for each case before solving for the optimum pit limit and number of blocks enclosed within the optimum pit limit is given in Table 1. The summation of ore reserves enclosed within ultimate pit limits is given in Appendix D.

Table 1. Number of Blocks for Each Case Study.

<u>Item</u>	<u>Case I</u>	<u>Case II</u>	<u>Case III</u>	<u>Case IV</u>
1.Total blocks (including air blocks)	56,250	112,500	168,750	225,000
2.Blocks added * to the network				
- ore blocks	2,138	7,581	15,781	14,313
- waste blocks	4,317	22,001	41,090	50,535
3.Blocks removed by algorithms				
- ore blocks	884	1,988	9,557	9,357
- waste blocks	480	2,174	14,135	15,672

* The number of blocks added to the network is the total blocks within the original surface and the outer bound.

The processing time required by each algorithm for designing the final pit limits of these four cases is shown in Table 2. below.

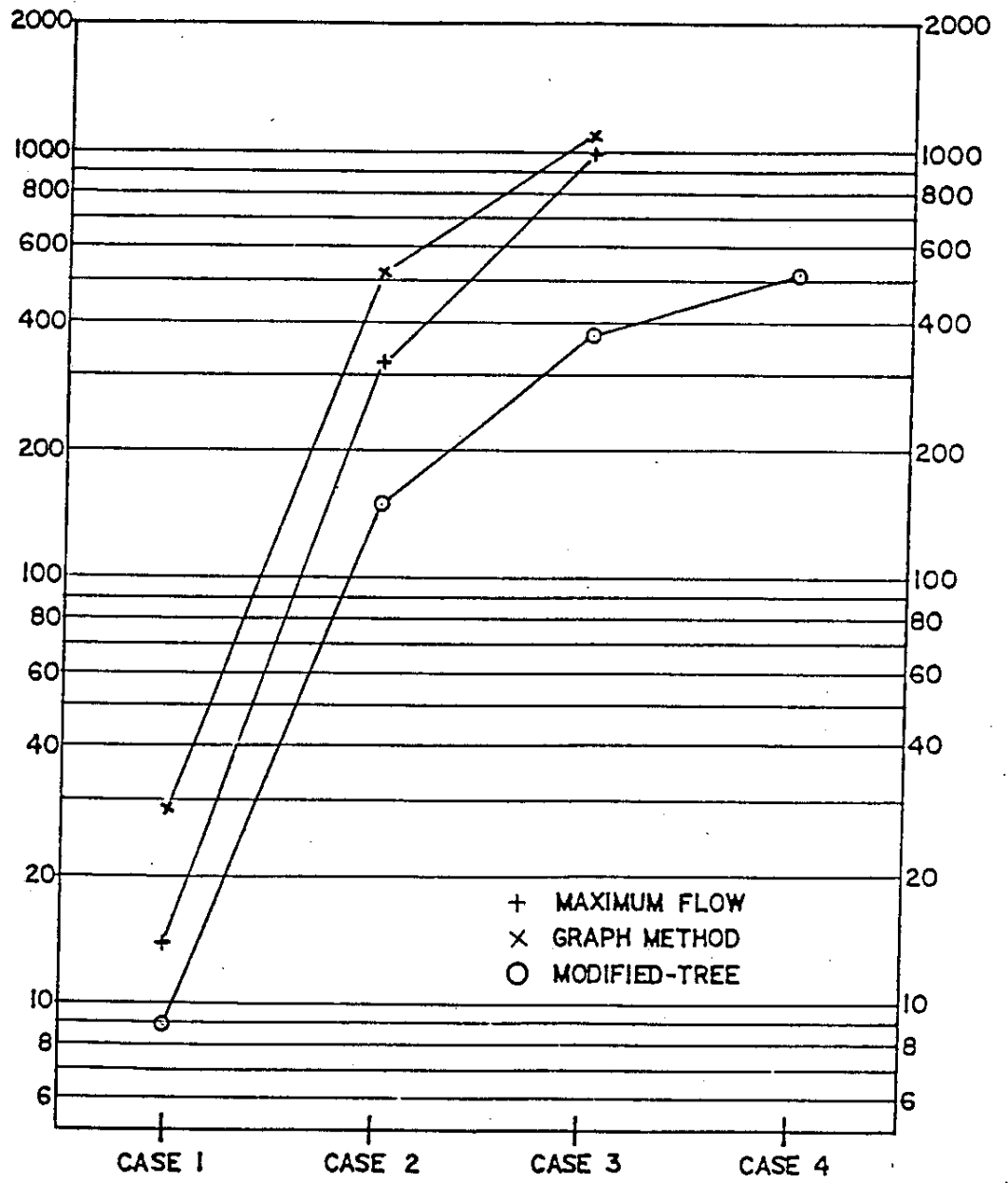


Figure 39. Plot of Execution Time on VAX8600 in Minutes

Table 2. CPU Time Required by Each Method (minutes).

<u>Methods</u>	<u>Case I</u>	<u>Case II</u>	<u>Case III</u>	<u>Case IV</u>
1. Maximum network flow method.	13.90	322.89	991.09	*
2. Original graph method.	28.31	524.76	1102.59	*
3. Modified-tree graph method.	8.96	150.21	375.53	514.87

* The computer processing time was too great to be determined within processing limits superimposed on the study.

The processing time for each method and each case is plotted and shown in Figure 39.

5.5 ANALYSIS OF THE RESULTS

The highlight of the results is that the modified-tree algorithm is the least time consuming method among the three optimizing algorithms applied to the given block model. As a result, solution time for the modified-tree graph algorithm is respectively less than the time required for the original graph algorithm by a factor varying between three to four on this case study. The main reason for the improvement is the modified-tree graph method recognizes a smaller tree configuration and utilizes less variables, therefore, the numerical operations are reduced. The graph algorithm requires the process of tracing back and forth in searching

for the vertices on the transformation chain and in its evaluation phase. After the tree is reduced to only connections between positive vertices, the mathematical operation correspondingly decreases.

The maximum network flow algorithm is faster than the original graph algorithm because it has the arrays to represent the relationship between positive nodes and overlying negative nodes. The graph algorithm generates a cone and iterates through the search process every time the positive node becomes strong and retrieved to the iterative process of the algorithm. However, as far as the algorithms are applied to the block model at depth, the processing time required by the network flow method tends to increase at a faster rate than the graph algorithms. Main reason for this behavior is that at each iteration of the labeling network flow algorithm, if any node can be labeled or has changed its status, all of the nodes that are connected to that node must be rescanned and relabeled. In contrast to the network flow algorithm, the graph method does not connect positive nodes to all of its overlying negative nodes since some of these overlying nodes may be supported by other positive nodes. When one node on the tree has changed its status, it does not require that all overlying nodes be changed or their status reevaluated.

The maximum network flow technique requires more computer memory to store the properties of its arcs. The number of arcs in the network tends to increase for a deposit at depth, therefore, a rapid increase in memory is found with depth. The modified-tree algorithm, since it eliminates the concept of physical arcs, requires the least computer memory. Most of the variable arrays used by the modified-tree algorithm are dimensioned equal to the number of positive ore blocks.

5.6 EXPERIENCE ON A MICROCOMPUTER

The modified-tree graph algorithm utilizes less computer memory than the other two true optimizing techniques. Therefore, the new algorithm was run on a microcomputer.

The modified-tree algorithm as coded in Appendix A can be run on a standard PC type microcomputer using a standard FORTRAN 77 compiler without any modifications. The same case study as given in Section 5.4 was used for the microcomputer test. The program was run on an IBM PC/AT microcomputer with RAM memory of 640K and clock speed of 6 MHz.

Computational results obtained are summarized in Table 3. as shown below. Case III and Case IV were not completed due to the long execution times.

Table 3. Processing Time Required by Each Case on a 6 MHz
IBM AT/PC.

<u>CASE</u>	<u>TIME (Min)</u>
I	189
II	2911
III	*
IV	*

* The problems are not completely solved due to very large processing time.

In general, the ultimate pit limit analysis has been limited to the use of medium or large size digital computers because the nature of the algorithms requires large memory capacities and rapid processing speed. However, this study has shown that the modified-tree graph algorithm can be executed on a microcomputer with some limitations. A microcomputer with 640K RAM memory can accept up to approximately 7,000 positive nodes (ore blocks) during each iteration and a total number of blocks between the original surface and the outer bound of about 75,000 blocks. Two major problems with using the modified-tree algorithm on a microcomputer are : long execution times it and can only handle small block models.

In the future, the continuing improvement in processing speeds of microcomputers, improving compilers taking

advantage of several megabyte of memory, and the decreasing cost of acquiring microcomputers may foster an increase in the opportunity to apply the modified-tree graph algorithm on a microcomputer.

Chapter 6

CONCLUSIONS AND RECOMMENDATIONS

6.1 CONCLUSIONS

The developed technique is not the discovery of the new theory in solving for the ultimate pit limit. The work in this thesis is just one step advance in making the true optimizing technique more practical. The modified-tree graph method is the extension of the original graph method and still applies the basic concepts of the Lerchs and Grossmann graph theory. However, the study does illustrate that there is still a potential way to improve true optimizing techniques.

The given example presented in Chapter 4 establishes that the modified-tree graph algorithm is able to generate the true ultimate pit limits. Moreover, the computer experience of executing the proposed algorithm and the existing true optimization algorithms shows that the new method utilized less computer resources than those of the existing methods.

Since the modified-tree graph algorithm requires less computer resources, it may be useful in applying the algorithm on medium size computers. With the gradually advances in high speed and more memory capacity of the low

cost microcomputer, it may be possible to operate the true optimizing algorithm at lower operating and capital costs.

6.2 RECOMMENDATIONS

Originally, the graph algorithm was developed from the construction of the tree configuration and is used for the explanation of the work of the Lerchs-Grossmann graph theory. However, the steps of both graph algorithm as given in this thesis have illustrated that they are purely mathematical operation. This means that there is still a potential extension of the algorithm mathematically in order to relax the processing time and number of variables required by the process.

It is obvious that there exists the connection between the linear programming, the network flow programming, and the graph theory, which are common methods among the operation reseach techniques. One model may be a special structure of the other. The method selected may depend on a particular problem that requires highly efficient solution. For the ultimate pit limit analysis, some methods may have an advantage over the others under given conditions. To select an appropriate method for a given ore body and a given tool may need some additional criteria and methodologies.

REFERENCES CITED

- Barnes, R.J., 1982, "Optimizing the Ultimate Pit", Master of Science Thesis, Colorado School of Mines, Golden, Colorado.
- Baumgartner, W., 1986, "Use Computer to Evaluate Quarry", Rock Products, May 1986, pp. 44-62.
- Benham, D., 1978, "Open Pit Production Scheduling", Master of Science Thesis, Colorado School of Mines, Golden, Colorado.
- Brackebush, F.W., 1971, "To Design Lakeshore Open Pit Hecla Thinks Big with Small Computer", Engineering and Mining Journal, July 1971, pp. 72-76.
- Carlson, T.R., et al., 1966, "Computer Techniques in Mine Planning", Mining Engineering, Vol.18, No. 5, pp.53-56.
- Chen, T., 1976, "3-D Pit Design with Variable Wall Slope Capabilities", In 14th APCOM, pp. 615-625, Pennsylvania State University, State College, Pennsylvania.
- Crawford, J.T., 1979, "Open Pit Limit Analysis - Some Observations on Its Use", In 16th APCOM, pp. 625-634, University of Arizona, Tucson, Arizona.
- Crawford, J.T., and R.K. Davey, 1979, "Case Study in Open Pit Limit Analysis", Computer Methods for the 80's in the Mineral Industry, ed. Alfred Weiss, pp. 310-318, SME-AIME, New York.
- Dagdelen, K., 1985, "Optimum Multi Period Open Pit Mine Production Scheduling", Ph.D. Dissertation, Colorado School of Mines, Golden, Colorado.
- Dantzig, G.B., and P. Wolfe, 1960, "Decomposition Principle for Linear Programs", Operation Research, Vol. 8, No.1, pp. 101-111.
- Davey, R.K., 1979, "Mineral Block Evaluation Criteria", Open Pit Mine Planning and Design, ed. J.T. Crawford and W.A. Hustrulid, pp. 85-96, SME-AIME, New York.

- Erickson, J.D., 1968, "Long Range Open Pit Mine Planning", Mining Engineering, Vol. 20, No. 4, pp. 75-78.
- Fairfield, J.D., and R.W. Leigh, 1968, "A Computer Program for the Design of Open Pits", In 7th APCOM, Colorado School of Mines Quaterly, Vol. 64, No. 3, pp. 329-339, Golden, Colorado .
- Fytas, K., and P.N. Calder, 1986, "A Computerized model of Open Pit Short and Long Range Production Scheduling", In 19th APCOM, pp. 109-119, SME-AIME, New York.
- Gangwar, A., 1982, "Using Geostatistical Ore Block Variances in Production Planning By Integer Programming", In 17th APCOM, ed. T.B. Johnson and R.J. Barnes, pp. 443-460, SME-AIME, New York.
- Gauthier, F.J., and K.G. Gray, 1971, "Pit Design by Computer at Gaspe Copper Mines Limited", CIM Bulletin, November, pp. 95-102.
- Hartman, R.J., and G.C. Varma, 1966, "3-D OPT - A Three Dimensional Optimum Pit Program and a Basis for Mining Engineering System", In 6th APCOM, Pennsylvania State University, State College, Pennsylvania.
- Iles, C.D., and G.A. Perry, 1983, "Sierrita Incremental Pit Design System", Mining Engineering, February 1983, pp. 152-155.
- Johnson, T.B., 1968, "Optimum Open Pit Mine Production Scheduling", Ph.D. Dissertation, University of California Berkeley, Operations Research Department.
- Johnson, T.B., and D.G. Mickle, 1970, "Optimum Design of an Open Pit - An Application in Uranium", In 9th APCOM, pp. 331-337, CIM Special Vol.12, Montreal, Canada.
- Johnson, T.B., and W.R. Sharp, 1971, "A Three-Dimensional Dynamic Programming Method for Optimal Open Pit Design", USBM RI 7553.
- Journal, A.G., and C.J. Huijbregts, 1978, Mining Geostatistics, Academic Press, New York.

- Kim, Y.C., 1978, "Ultimate Pit Limit Design Methodologies Using Computer Models - the State of the Art", Mining Engineering, Vol. 30, pp. 1454-1459.
- Koenigsberg, E., 1982, "The Optimum Contours of an Open Pit Mine: An Application of Dynamic Programming", In 17th APCOM, pp.274-287, Golden, Colorado.
- Koskiniemi, B.C., 1979, "Hand Methods", Open Pit Mine Planning and Design, ed. J.T. Crawford and W.A. Hustrulid, pp. 187-195, SME-AIME, New York.
- Lemieux, M., 1976, "A Different Method of Modeling a Mineral Deposit for a Three-Dimensional Open Pit Computer Design Application", In 14th APCOM, pp. 557-571, State College, Pennsylvania.
- Lemieux, M., 1979, "Moving Cone Optimization Algorithm", Computer Methods for the 80's in the Mineral Industry, ed. Alfred Weiss, pp. 329-345, SME-AIME, New York.
- Lerchs, H., and I.F. Grossmann, 1965, "Optimum Design of Open Pit Mines", CIM Bulletin, Vol. 58, No. 633, pp.47-54.
- Lipkewich, M.P., and L. Borgman, 1969, "Two and Three Dimensional Pit Design Optimization Technique", A Decade in Digital Computing in the Mineral Industry, ed. Alfred Weiss, pp. 505-523, SME-AIME, New York.
- Marino, J.M., and J.P. Slama, 1973, "Ore Reserve Evaluation and Open Pit Planning", In 10th APCOM, pp. 139-144, Johannesburg, Republic of South Africa.
- Pana, M.T., 1965, "The Simulation Approach to Open Pit Design", In 5th APCOM, Tucson, Arizona.
- Pana, M.T., and T.R. Carlson, 1966, "A Description of a Computer Technique Used in Mine Planning of the Utah Mine of Kennecott Copper Corp.", In 6th APCOM, Pennsylvania State University, Pennsylvania.
- Phillips, D.A., 1973, "Optimum Design of an Open Pit", In 10th APCOM, pp. 145-147, Johannesburg, Republic of South Africa.

- Robinson, R.H., 1975, "Programming the Lerchs-Grossmann Algorithm for Open Pit Mine Design", In 13th APCOM, Claustal, Federal Republic of Germany.
- Ryckum, E.A., and T. Chen, 1979, "Open Pit Mine Feasibility Method and Application at Placer Development", Computer Method for the 80's in the Mineral Industry, ed. Alfred Weiss, pp. 304-309, SME-AIME, New York.
- Seymour, F.H., 1985, "Open Pit Limit Analysis on a Microcomputer", Master of Science Thesis, Colorado School of Mines, Golden, Colorado.
- Soderberg, A., and D.O. Rausch, 1968, "Pit planning and Layout", Surface Mining, ed. E.P. Pfleider, SME-AIME, New York.
- Stanley, B., 1979, "Mineral Model Construction: Principles of Ore-Body Modeling", Open Pit Mine Planning and Design, ed. J.T. Crawford and W.A. Hustrulid, pp.43-50, SME-AIME, New York.

APPENDIX A
COMPUTER CODE

```

C*****
C
C PROGRAM : SAMPIT.FOR
C This is the main program to determine the true
C optimum pit limit by using the following
C optimizing pit limit algorithms ;
C 1. maximum network flow algorithm,
C 2. graph theory (tree) algorithm,
C 3. modified-tree graph algorithm.
C The computer code is written in FORTRAN 77 and
C run on DEC VAX/VMS mainframe computer
C
C Programmed by : P.Huttagosol, 1987
C
C Subroutine called :
C BLOCKS - obtains the model & block dimensions
C OBTAIN - obtains the necessary filenames
C MODELS - generates the cone template
C MAXFLO - designs the pit limit by Max-flow
C GRAPHT - designs the pit limit by Graph
C MGRAPH - designs pit limit by Modified-tree
C OUTPUT - output the final pit to data file
C*****
C
C CHARACTER VALFIL*10,OUTFIL*10,ANS
C
C Title.....
C
C WRITE(*,10)
C 10 FORMAT(1X,////,
C & T5,'-----',/,/,
C & T5,'* Program SAMPIT.FOR :',/,/,
C & T5,'* Design the optimum pit limit by the',/,/,
C & T5,'* true optimizing pit limit algorithms.',/,/,
C & T5,'-----',/,/,
C & '//////////,T15,'HIT RETURN TO CONTINUE !')
C 20 READ(*,20) ANS
C FORMAT(A1)
C
C Interactively obtain the block and model information....
C CALL BLOCKS
C
C Interactively obtain the necessary file names.....
C CALL OBTAIN(VALFIL,OUTFIL)

```

```

C   Call a subroutine to generate the cone template.....
      CALL MODELS

C   Open the block model data files.....
      OPEN (UNIT=10,FILE=VALFIL,ACCESS='DIRECT',STATUS='OLD'
& ,RECL=1)

C   Select algorithm to design true optimum pit limit.....
40   WRITE(*,50)
50   FORMAT(1X,////////,
& T5,'-----',//,
& T5,'* Select option to design optimum pit limit *',//,
& T5,'* 1. maximum network flow algorithm, *',//,
& T5,'* 2. graph theory (tree) algorithm, *',//,
& T5,'* 3. modified-tree graph algorithm. *',//,
& T5,'-----',//,
& //////////,T15,'Option (1/2/3) = ')
      READ(*,*) INDEX
      IF (INDEX.EQ.1.OR.INDEX.EQ.2.OR.INDEX.EQ.3) GOTO 70
      WRITE(*,60)
60   FORMAT(1X,//,T10,'***Error!-Answer"1"or"2"or"3"***')
      GOTO 40

C   Design the true optimizing ultimate pit limit.....
70   IF (INDEX.EQ.1) THEN
      CALL MAXFLO
    ELSEIF (INDEX.EQ.2) THEN
      CALL GRAPHT
    ELSE
      CALL MGRAPH
    ENDIF

C   Close the working file.....
      CLOSE (UNIT=10)

C   Write out the pit topography into the output file.....
      CALL OUTPUT (OUTFIL)

      STOP
      END

```

```

C*****
C
C SUBROUTINE : BLOCKS *
C To obtain the information about the model size *
C block size, and allowable slope angles. *
C
C Variables used : *
C NX - total number of blocks on X-axis *
C NY - total number of blocks on Y-axis *
C NZ - total number of levels in the model *
C XDM - the dimension of block on X-axis *
C YDM - the dimension of block on Y-axis *
C ZDM - the dimension of block on Z-axis *
C ANGI - allowable slope angle along X-axis *
C ANGJ - allowable slope angle along Y-axis *
C
C Subroutine called : none *
C
C*****

```

SUBROUTINE BLOCKS

CHARACTER ANS

COMMON/BLK1/NX,NY,NZ,XDM,YDM,ZDM,IMAX,JMAX,TANI,TANJ

C Write the subroutine heading.....

```

5 WRITE(*,5)
  FORMAT(1X,/,T5,'Subroutine : BLOCKS',/,/,
& T10,'Obtain dimensions of block,
& model size and slope angles',/)

```

C Interactively obtain the block model information.....

```

10 WRITE(*,15)
15 FORMAT(/,T5,'Enter no.of blocks on each axis(X,Y,Z)')
30 READ(*,30) NX,NY,NZ
  FORMAT(3I3)

```

C Interactively obtain the dimensions of the mineral block

```

35 WRITE(*,35)
  FORMAT(/,T5,'Enter block dimension(ft.) on each
& axis(X,Y,Z) :')
  READ(*,*) XDM,YDM,ZDM

```

```

C   Interactively obtain the maximum allowable slope angles.

      WRITE(*,50)
50   FORMAT(/,
      & T5,'Enter allowable slope along X-axis(degrees) : ')
      READ(*,*) ANG1
      ANGI = ANG1*3.1415926/180.0
      TANI = TAN(ANGI)

      WRITE(*,55)
55   FORMAT(1X,
      & T5,'Enter allowable slope along Y-axis(degrees) : ')
      READ(*,*) ANG2
      ANGJ = ANG2*3.1415926/180.0
      TANJ = TAN(ANGJ)

60   WRITE(*,65)
65   FORMAT(//,T5,'Do you want to change any input
      & parameters(Y/N):')
      READ(*,70) ANS
70   FORMAT(A1)
      IF (ANS.EQ.'Y'.OR.ANS.EQ.'y') GO TO 10

      RETURN
      END

```

```

C*****
C
C   SUBROUTINE : OBTAIN
C       Obtain the necessary input&output filenames
C
C   Variables used :
C       TOPFIL - the input topography filename
C       VALFIL - the economic block model filename
C       OUTFIL - the optimum pit output filename
C       TOP(I,J) - original surface at column(I,J)
C
C   Subroutine called : none
C
C*****

```

```

SUBROUTINE OBTAIN(VALFIL,OUTFIL)

```

```

PARAMETER(MX=75)
INTEGER*2 TOP(MX,MX),PIT(MX,MX)
CHARACTER*10 TOPFIL,VALFIL,OUTFIL,ANS*1

```

```
COMMON/BLK1/NX,NY,NZ,XDM,YDM,ZDM,IMAX,JMAX,TANI,TANJ
COMMON/BLK2/TOP
COMMON/OPUT/PIT
```

C Write the headings.....

```
WRITE(*,2)
2   FORMAT(1X,/,T5,'Subroutine : OBTAIN',//,
&   T10,'Obtain necessary file names.....',//)
10  FORMAT(A10)
15  FORMAT(1X,/,T10,'*Error! File',A10,'does not exist.')
```

C Interactively obtain the TOP file name.....

```
20  WRITE(*,22)
22  FORMAT(1X,/,T5,'Enter the input surface file name:')
    READ(*,10) TOPFIL

    OPEN(UNIT=11,FILE=TOPFIL,ACCESS='SEQUENTIAL',
&    STATUS='OLD',FORM='FORMATTED',ERR=26)
    DO 24 I=1,NX
        READ(11,*) (TOP(I,J),J=1,NY)
24  CONTINUE
    CLOSE(UNIT=11)
    GO TO 30

26  WRITE(*,15) TOPFIL
    GO TO 20
```

C Initialize the intermediate pit surface.....

```
30  DO 35 I=1,NX
        DO 32 J=1,NY
            PIT(I,J) = TOP(I,J)
32  CONTINUE
35  CONTINUE
```

C Interactively obtain the BLOCK VALUE file name.....

```
40  WRITE(*,42)
42  FORMAT(1X,/,T5,'Enter the economic block model file
& name : ')
    READ(*,10) VALFIL
    OPEN(UNIT=10,FILE=VALFIL,ACCESS='DIRECT',
&    STATUS='OLD',RECL=1,ERR=44)
    CLOSE(UNIT=10)
    GO TO 50
```

```
44      WRITE(*,15) VALFIL
        GO TO 40
```

C Interactively obtain the OUTPUT file name.....

```
50      WRITE(*,52)
52      FORMAT(1X,///

```

```
C*****
C
C SUBROUTINE : MODELS
C      Generate the cone template according to the given
C      pit slope allowances along two perpendicular
C      horizontal axis X and Y.
C
C Programmed by : P.Huttagosol, 1987
C
C Variables used :
C      RI,RJ - radius of the ellipse on X and Y axis.
C      DI,DJ,DK - the rectilinear distance
C      ALLOW - allowable distance from cone center
C      SURF(I,J) - cone template in the first quarter
C
C Subroutine called : none
C
C*****
```

```
      SUBROUTINE MODELS
```

```
      PARAMETER(ML=40)
      INTEGER*2 SURF(ML,ML)
```

```
      COMMON/BLK1/NX,NY,NZ,XDM,YDM,ZDM,IMAX,JMAX,TANI,TANJ
      COMMON/BLK10/SURF
```

C Write the subroutine heading.....

```
      WRITE(*,5)
5      FORMAT(1X,///

```

C Calculate the necessary parameters.....

```
TANK = MAX(TANI,TANJ)
TANK = TANK/ZDM
IMAX = INT((NZ*ZDM)/(XDM*TANI))+1
JMAX = INT((NZ*ZDM)/(YDM*TANJ))+1
```

C Determine the cone pattern.....

```
DO 30 I=1,IMAX
  I1 = I-1
  DI = FLOAT(I1)*XDM
  DII = DI*DI

  DO 20 J=1,JMAX
    J1 = J-1
    DJ = FLOAT(J1)*YDM
    DJJ = DJ*DJ
    DIJ = SQRT(DII+DJJ)

    IF (DJ.NE.0.0) THEN
      TANP = (DI/DJ)**2
    ENDIF

    KMAX = INT(TANK*DIJ)+1
    DO 10 K=KMAX,1,-1
      DK = FLOAT(K)*ZDM
      RI = DK/TANI
      IF (DJ.EQ.0.0) THEN
        ALLOW = RI
      ELSE
        RII = RI*RI
        RJJ = (DK/TANJ)**2
        YY = (RII*RJJ)/(RII+RJJ*TANP)
        XX = YY*TANP
        ALLOW = SQRT(XX+YY)
      ENDIF

      IF (DIJ.GT.ALLOW) GOTO 20
      SURF(I,J) = K
10    CONTINUE

20    CONTINUE

30    CONTINUE

RETURN
END
```

```

C*****
C
C      SUBROUTINE : MAXFLO
C          This is the main subroutine that designs an
C          optimum pit limit using maximum flow method.
C
C      Programmed by : P.Huttagosol, 1987. Modified from
C          the original program written by Barnes(1982).
C
C      Variables used :
C          NOD - node number
C          ARC - arc number
C          SUMP(K) - sum of positive nodes on level K
C          SUMN(K) - sum of negative nodes on level K
C
C      Subroutine called :
C          INITIA - initialize state of variables
C          PLACER - generate the outer bound
C          ADD1 - add nodes and initialization
C          ADDARC - add arcs to the network
C          ALLOCA - initial allocation of flow
C          LISTS - put nodes in the scan list
C          LABELS - scanning and labeling
C          REMOV1 - remove nodes with access supply
C          INFORM - write information on the screen
C          FINPIT - update the optimum pit limit
C
C*****
      SUBROUTINE MAXFLO

      IMPLICIT INTEGER(A-Z)
      PARAMETER(MZ=40)
      INTEGER*2 SUMP(MZ),SUMN(MZ)

      COMMON/BLK15/SUMP
      COMMON/BLK17/SUMN
      DATA TMN1/1/

C      Initialize the state of variables.....

      CALL INITIA(NOD,NXND,SMND)
      ARC = 3
      NXTA = TMN1
      SMAC = 0

C      Generate the outer bound by placer method.....

      CALL PLACER(KTOP,KBOT)

```

```
C   Iterate through the levels : from level to bottom level.
      DO 150 K=KTOP,KBOT
        SUMP(K) = 0
        SUMN(K) = 0

C   Add node to the network.....
      CALL ADD1(K,NOD,NXND,SMND)

      AD1 = SUMP(K)
      AD2 = SUMN(K)
      IF (SUMP(K).EQ.0) GOTO 140
      IF (K.EQ.KTOP) GOTO 120

C   Add arcs from positive nodes to their overlying nodes...
      CALL ADDARC(K,KTOP,AD3,ARC,NXTA,SMAC)

C   Initial allocation of flow.....
      CALL ALLOCA(K)

C   Put nodes into scan lists.....
      CALL LISTS(K,KTOP)

C   Scanning and labeling.....
      CALL LABELS(K,KTOP)

C   Remove labeled nodes with access supply.....
120      CALL REMOV1(K,KTOP,MN1,MN2,MN3,NOD,NXND,SMND,ARC,
      & NXTA,SMAC)

C   Output desired information on the screen and next level.
140      IOPT = 1
      CALL INFORM(K,IOPT,AD1,AD2,AD3,MN1,MN2,MN3)
150      CONTINUE

C   Determine the final pit surface.....
      CALL FINPIT(KTOP,KBOT)

      RETURN
      END
```

```

C*****
C  SUBROUTINE : ADD1 *
C      This subroutine add nodes enclosed within the *
C      top surface and outer bound to the network *
C      and initilize some arrays. *
C *
C  Variables used : *
C      SINK - the imaginary sink node *
C      SORC - the imaginary source node *
C      QUAN(N) - quantity or capacity of node N *
C      NEXT(N) - list of next node after node N *
C      INUM(N),JNUM(N) - block ID number of node N *
C      PNP(K) - positive node pointer for level K *
C      NNP(K) - negative node pointer for level K *
C*****
C      SUBROUTINE ADD1 (K,NOD,NXND,SMND)

      IMPLICIT INTEGER(A-Z)
      PARAMETER (MX=75,MD=50000,MZ=40)
      REAL XDM,YDM,ZDM,TANI,TANJ
      INTEGER*2 PNP (MZ) ,NNP (MZ) ,SUMP (MZ) ,SUMN (MZ) ,QUAN (MD)
      INTEGER*2 TOP (MX,MX) ,BOT (MX,MX) ,INUM (MD) ,JNUM (MD) ,KVAL
      DIMENSION BOX1 (MD) ,BOX2 (MD) ,BOX3 (MD) ,BOX4 (MD) ,NEXT (MD)

      COMMON/BLK1/NX,NY,NZ,XDM,YDM,ZDM,IM,JM,TANI,TANJ
      COMMON/BLK2/TOP
      COMMON/BLK3/BOT
      COMMON/BLK5/BOX1
      COMMON/BLK6/BOX3
      COMMON/BLK7/BOX4
      COMMON/BLK8/QUAN
      COMMON/BLK11/INUM
      COMMON/BLK12/JNUM
      COMMON/BLK13/NEXT
      COMMON/BLK15/SUMP
      COMMON/BLK16/PNP
      COMMON/BLK17/SUMN
      COMMON/BLK18/NNP
      COMMON/BLK20/BOX2
      DATA SINK,SORC,TMN1/1,1,1/

C  Read block values on level K.....

      NXTP = TMN1
      NXTN = TMN1
      DO 12 I=1,NX
         I1 = (I-1)*NY

```

```

DO 10 J=1,NY
  IF (K.LE.TOP(I,J).OR.K.GT.BOT(I,J)) GOTO 10
  KRN = (I1+J-1)*NZ+K
  READ(10,REC=KRN) KVAL
  QUAN(NOD) = KVAL
  BOX2(NOD) = TMN1
  BOX3(NOD) = TMN1
  BOX4(NOD) = TMN1
  JNUM(NOD) = J

```

C Initialize array of positive node properties.....

```

  IF (KVAL.GT.0) THEN
    BOX1(NOD) = SORC
    NEXT(NOD) = NXTP
    INUM(NOD) = I
    SUMP(K) = SUMP(K)+1
    NXTP = NOD

```

C Initialize arrays of negative node properties.....

```

  ELSE
    BOX1(NOD) = SINK
    NEXT(NOD) = NXTN
    INUM(NOD) = -I
    SUMN(K) = SUMN(K)+1
    NXTN = NOD
  ENDIF

```

C Prepare the identification number for next node.....

```

  IF (SMND.EQ.0) THEN
    NOD = NOD+1
  ELSE
    NOD = NXND
    NXND = NEXT(NOD)
    SMND = SMND-1
  ENDIF

```

C Next block on the current level.....

```

10     CONTINUE
12     CONTINUE
      PNP(K) = NXTP
      NNP(K) = NXTN

      RETURN
      END

```

```

C*****
C  SUBROUTINE : ADDARC *
C    Place an arc connecting each positive node to *
C    all of its overlying negative nodes. *
C *
C  Programmed by : P.Huttagosol, 1987. Modified from the *
C    original program written by Barnes(1982). *
C *
C  Variables used : *
C    PNOD - positive node *
C    NNOD - negative node *
C    PARC - arc that connects from positive node *
C    NARC - arc that connects to negative node *
C    BOX1(N) - source node of labeled node N *
C    BOX2(N) - arc pointer for node N *
C    BOX5(A) - NNOD of arc A *
C    BOX6(A) - PARC next to arc A *
C    BOX7(A) - PNOD of arc A *
C    BOX8(A) - NARC next to arc A *
C    FLOW(A) - unit flow along arc A *
C *
C  Subroutine called : *
C    CONE - generates a cone on a positive node *
C*****
      SUBROUTINE ADDARC(K,KTOP,AD3,ARC,NXTA,SMAC)

      IMPLICIT INTEGER(A-Z)
      PARAMETER (MX=75,MD=50000,MA=250000,MZ=40)
      INTEGER*2 PNP(MZ),SUMP(MZ),SUMN(MZ),CON(MX,MX)
      INTEGER*2 INUM(MD),JNUM(MD),FLOW(MA),QUAN(MD),NNP(MZ)
      DIMENSION BOX1(MD),BOX2(MD),NEXT(MD)
      DIMENSION BOX5(MA),BOX6(MA),BOX7(MA),BOX8(MA)

      COMMON/BLK4/CON
      COMMON/BLK5/BOX1
      COMMON/BLK8/QUAN
      COMMON/BLK11/INUM
      COMMON/BLK12/JNUM
      COMMON/BLK13/NEXT
      COMMON/BLK15/SUMP
      COMMON/BLK16/PNP
      COMMON/BLK17/SUMN
      COMMON/BLK18/NNP
      COMMON/BLK19/FLOW
      COMMON/BLK20/BOX2
      COMMON/BLK21/BOX5
      COMMON/BLK22/BOX6
      COMMON/BLK23/BOX7

```

```
COMMON/BLK24/BOX8
DATA SINK, TMN1/1,1/
```

C Iterate through all positive nodes on current level.....

```
AD3 = 0
PNOD = PNP(K)
NUMP = SUMP(K)
DO 28 NP=1, NUMP
  IID = INUM(PNOD)
  JID = JNUM(PNOD)
  PARC = TMN1
```

C Generate an inverted cone with vertex on positive block.

```
CALL CONE(IID, JID, K, MNI, MXI, MNJ, MXJ)
```

C Find the overlying negative nodes within the cone.....

```
DO 26 KK=K-1, KTOP, -1
  IF (SUMN(KK).EQ.0) GOTO 26

  NNOD = NNP(KK)
  NUMN = SUMN(KK)
  DO 24 NN=1, NUMN
    I1 = IABS(INUM(NNOD))
    J1 = JNUM(NNOD)
    IF (KK.GT.CON(I1, J1)) GOTO 22
```

C Add arc and initialize flow.....

```
AD3 = AD3+1
FLOW(ARC) = 0
IF (BOX1(NNOD).NE.SINK) GOTO 20
IF (QUAN(PNOD).EQ.0) GOTO 20
IF (QUAN(NNOD).EQ.0) GOTO 20
FLOW(ARC) = MIN(QUAN(PNOD), -QUAN(NNOD))
QUAN(PNOD) = QUAN(PNOD) - FLOW(ARC)
QUAN(NNOD) = QUAN(NNOD) + FLOW(ARC)
```

C Store arc properties in the arrays.....

```
20      BOX5(ARC) = NNOD
      BOX6(ARC) = PARC
      BOX7(ARC) = PNOD
      BOX8(ARC) = BOX2(NNOD)
      PARC = ARC
      BOX2(NNOD) = ARC
```

```

C   Prepare identification number for next arc.....

          IF (SMAC.EQ.0) THEN
            ARC = ARC+1
          ELSE
            ARC = NXTA
            NXTA = BOX6(ARC)
            SMAC = SMAC-1
          ENDIF

C   Next negative node.....

22          NNOD = NEXT(NNOD)
24          CONTINUE
26          CONTINUE
           BOX2(PNOD) = PARC

C   Next positive node.....

           PNOD = NEXT(PNOD)
28          CONTINUE

           RETURN
           END

```

```

C*****
C
C   SUBROUTINE : ALLOCA
C           This subroutine allocates initial flow along
C           the new arcs and also along its continuous
C           path of three arcs.
C
C   Programmed by : P.Huttagosol, 1987. Modified from the
C           original program written by Barnes(1982).
C
C*****
SUBROUTINE ALLOCA(K)

  IMPLICIT INTEGER(A-Z)
  PARAMETER(MD=50000,MA=250000,MZ=40)
  INTEGER*2 PNP(MZ),SUMP(MZ),QUAN(MD),FLOW(MA)
  DIMENSION BOX1(MD),BOX2(MD),NEXT(MD)
  DIMENSION BOX5(MA),BOX6(MA),BOX7(MA),BOX8(MA)

  COMMON/BLK5/BOX1
  COMMON/BLK8/QUAN
  COMMON/BLK13/NEXT

```

```

COMMON/BLK15/SUMP
COMMON/BLK16/PNP
COMMON/BLK19/FLOW
COMMON/BLK20/BOX2
COMMON/BLK21/BOX5
COMMON/BLK22/BOX6
COMMON/BLK23/BOX7
COMMON/BLK24/BOX8
DATA SINK, TMN1/1,1/

```

C Iterate through all positive nodes on current level.....

```

PNOD = PNP(K)
NUMP = SUMP(K)
DO 38 NP=1, NUMP
  IF (QUAN(PNOD).EQ.0) GOTO 36

```

C The first path.....

```

30 TEM1 = BOX2(PNOD)
  IF (TEM1.EQ.TMN1) GOTO 36
  ARC1 = TEM1
  NOD1 = BOX5(ARC1)
  TEM1 = BOX6(ARC1)

```

C The second path.....

```

32 TEM2 = BOX2(NOD1)
  IF (TEM2.EQ.TMN1) GOTO 30
  ARC2 = TEM2
  NOD2 = BOX7(ARC2)
  TEM2 = BOX8(ARC2)
  IF (FLOW(ARC2).EQ.0.OR.NOD2.EQ.PNOD) GOTO 32

```

C The last path.....

```

34 TEM3 = BOX2(NOD2)
  IF (TEM3.EQ.TMN1) GOTO 32
  ARC3 = TEM3
  NNOD = BOX5(ARC3)
  TEM3 = BOX6(ARC3)
  IF (BOX1(NNOD).NE.SINK) GOTO 34
  IF (QUAN(NNOD).GE.0.OR.NNOD.EQ.NOD1) GOTO 34

```

C Allocate flow through three arcs on the continuous path..

```

FLOWM = MIN(QUAN(PNOD), FLOW(ARC2), -QUAN(NNOD))
FLOW(ARC1) = FLOW(ARC1) + FLOWM

```

```

FLOW(ARC2) = FLOW(ARC2)-FLOWM
FLOW(ARC3) = FLOW(ARC3)+FLOWM
QUAN(PNOD) = QUAN(PNOD)-FLOWM
QUAN(NNOD) = QUAN(NNOD)+FLOWM

```

```

IF (QUAN(PNOD).EQ.0) GOTO 36
IF (FLOW(ARC2).EQ.0) GOTO 32
GOTO 34

```

```

36     PNOD = NEXT(PNOD)
38     CONTINUE

```

```

RETURN
END

```

```

C*****
C
C SUBROUTINE : LISTS
C This subroutine determines nodes to be scanned
C and put into the scan list
C
C Variables used :
C BOX4(N) - list of the first scan list
C BOX3(N) - list of the re-scan list
C*****
SUBROUTINE LISTS(K,KTOP)

IMPLICIT INTEGER(A-Z)
PARAMETER(MD=50000,MZ=40)
INTEGER*2 PNP(MZ),NNP(MZ),SUMP(MZ),SUMN(MZ),QUAN(MD)
DIMENSION BOX1(MD),BOX3(MD),BOX4(MD),NEXT(MD)

COMMON/BLK5/BOX1
COMMON/BLK6/BOX3
COMMON/BLK7/BOX4
COMMON/BLK8/QUAN
COMMON/BLK13/NEXT
COMMON/BLK15/SUMP
COMMON/BLK16/PNP
COMMON/BLK17/SUMN
COMMON/BLK18/NNP
DATA SINK, TMN1, TMN2/1,1,2/

C Reinitialize positive node's scan list of upper level...

DO 42 KK=KTOP,K-1

```

```

      IF (SUMP(KK).EQ.0) GOTO 42
      PNOD = PNP(KK)
      NUMP = SUMP(KK)
      DO 40 NP=1,NUMP
        QUAN(PNOD) = 0
        BOX3(PNOD) = TMN1
        BOX4(PNOD) = TMN1
        PNOD = NEXT(PNOD)
40      CONTINUE
42      CONTINUE

C      Put positive nodes of current level with surplus
C      quantity into a scan list.....

      NLAB = TMN2
      PNOD = PNP(K)
      NUMP = SUMP(K)

      DO 44 NP=1,NUMP
        IF (QUAN(PNOD).GT.0) THEN
          BOX4(NLAB) = PNOD
          NLAB = PNOD
        ELSE
          BOX4(PNOD) = TMN1
        ENDIF
        PNOD = NEXT(PNOD)
44      CONTINUE

C      Put negative nodes that need more flow in the scan list.

      DO 52 KK=KTOP,K-1
        IF (SUMN(KK).EQ.0) GOTO 52
        NNOD = NNP(KK)
        NUMN = SUMN(KK)
        DO 50 NN=1,NUMN
          IF (BOX1(NNOD).EQ.SINK.AND.QUAN(NNOD).LT.0) THEN
            BOX4(NLAB) = NNOD
            NLAB = NNOD
          ELSE
            QUAN(NNOD) = 0
            BOX4(NNOD) = TMN1
          ENDIF
          NNOD = NEXT(NNOD)
50      CONTINUE
52      CONTINUE

      RETURN
      END

```

```

C*****
C
C SUBROUTINE : LABELS
C   Scanning and labeling process. If break-through
C   occurs, it augments the flow and re-scan.
C
C Programmed by : P.Huttagosol, 1987. Modified from the
C   original program written by Barnes(1982).
C
C Variables used :
C   SNOD,SCAN - node to be scanned
C   LNOD - labeled node
C   SARC - arc to be scanned
C   NLAB,NLBL - the last node to be labeled
C   FLOWM - augmented flow
C   IND - the sign indicator of scanned node
C   TND,TMN - terminal node or terminal arc
C*****
SUBROUTINE LABELS(K,KTOP)

IMPLICIT INTEGER(A-Z)
PARAMETER(MD=50000,MA=250000,MZ=40)
REAL A,B
INTEGER*2 PNP(MZ),NNP(MZ),SUMP(MZ),SUMN(MZ)
INTEGER*2 INUM(MD),FLOW(MA),QUAN(MD)
DIMENSION BOX1(MD),BOX2(MD),BOX3(MD),BOX4(MD)
DIMENSION BOX5(MA),BOX6(MA),BOX7(MA),BOX8(MA)
DIMENSION NEXT(MD)

COMMON/BLK5/BOX1
COMMON/BLK6/BOX3
COMMON/BLK7/BOX4
COMMON/BLK8/QUAN
COMMON/BLK11/INUM
COMMON/BLK13/NEXT
COMMON/BLK15/SUMP
COMMON/BLK16/PNP
COMMON/BLK17/SUMN
COMMON/BLK18/NNP
COMMON/BLK19/FLOW
COMMON/BLK20/BOX2
COMMON/BLK21/BOX5
COMMON/BLK22/BOX6
COMMON/BLK23/BOX7
COMMON/BLK24/BOX8
DATA SINK,SORC,TMN1,TMN2/1,1,1,2/

```

```

C -----
C First scanning and labeling.
C -----
60     BOX4(NLAB) = TMN1
        SNOD = TMN2

C Obtain node to be scanned from the list.....

62     NXTS = BOX4(SNOD)
        BOX4(SNOD) = TMN1
        IF (NXTS.EQ.TMN1) GOTO 100
        SNOD = NXTS
        IF (QUAN(SNOD).EQ.0) GOTO 62
        INDEX = ISIGN(1,INUM(SNOD))

C Find arc to be scanned.....

64     TEM1 = BOX2(SNOD)
        IF (TEM1.EQ.TMN1) GOTO 62
        SARC = TEM1
        IF (INDEX.LT.0) THEN
            LNOD = BOX7(SARC)
            TEM1 = BOX8(SARC)
        ELSE
            LNOD = BOX5(SARC)
            TEM1 = BOX6(SARC)
        ENDIF

C Determine potential flow along scanned arc.....

        FLOWM = QUAN(SNOD)*INDEX
        IF (FLOWM.LT.-FLOW(SARC)) FLOWM = -FLOW(SARC)
        IF (FLOWM.EQ.0) GOTO 64

C If breakthrough occurs goto augment routine.....

        A = FLOAT(QUAN(LNOD))
        B = FLOAT(QUAN(SNOD))
        IF (A*B) 70,66,64

C Labeling and put LNOD in the scan list.....

66     QUAN(LNOD) = FLOWM*INDEX
        BOX1(LNOD) = SNOD
        IF (BOX4(LNOD).NE.TMN1) GOTO 64
        BOX4(NLAB) = LNOD
        NLAB = LNOD
        GOTO 64

```

```

C -----
C Augment flow after breakthrough.
C -----

70   IF (IABS(FLOWM).GT.IABS(QUAN(LNOD))) THEN
      FLOWM=-INDEX*QUAN(LNOD)
      ENDIF
      IF (INDEX.LT.0) THEN
          NOD1 = LNOD
          NOD2 = SNOD
      ELSE
          NOD1 = SNOD
          NOD2 = LNOD
      ENDIF
      NOD3 = NOD1
      NOD4 = NOD2

C   Increase or decrease flow along the scanned arc.....

      FLOW(SARC) = FLOW(SARC)+FLOWM
      INDEX = ISIGN(1,FLOWM)
      FLOWM = IABS(FLOWM)
      IF (INDEX.LT.0) GOTO 74

C   Change flow quantity along the path from source to SNOD.

72   NOD2 = BOX1(NOD1)
      IF (NOD2.EQ.SORC) GOTO 76
      PARC = BOX2(NOD1)
      DOWHILE (BOX5(PARC).NE.NOD2)
          PARC = BOX6(PARC)
      ENDDO
      FLOW(PARC) = FLOW(PARC)-FLOWM

74   NOD1 = BOX1(NOD2)
      NARC = BOX2(NOD2)
      DOWHILE (BOX7(NARC).NE.NOD1)
          NARC = BOX8(NARC)
      ENDDO
      FLOW(NARC) = FLOW(NARC)+FLOWM
      GOTO 72

C   Change flow quantity along the path from LNOD to sink...

76   IF (INDEX.LE.0) GOTO 80
78   NOD3 = BOX1(NOD4)
      IF (NOD3.EQ.SINK) GOTO 82
      NARC = BOX2(NOD4)

```

```

      DOWHILE (BOX7(NARC).NE.NOD3)
        NARC = BOX8(NARC)
      ENDDO
      FLOW(NARC) = FLOW(NARC)-FLOWM

80     NOD4 = BOX1(NOD3)
      PARC = BOX2(NOD3)
      DOWHILE (BOX5(PARC).NE.NOD4)
        PARC = BOX6(PARC)
      ENDDO
      FLOW(PARC) = FLOW(PARC)+FLOWM
      GOTO 78

C     Calculate new capacity after flow augmentation.....

82     QUAN(NOD1) = QUAN(NOD1)-FLOWM
      QUAN(NOD4) = QUAN(NOD4)+FLOWM

C     -----
C     Re-scanning and Re-labeling after breakthrough.
C     -----

      BOX3(TMN2) = NOD1
      BOX3(NOD1) = NOD4
      BOX3(NOD4) = TMN1
      SCAN = TMN2
      NLBL = NOD4

C     Obtain node to be scanned from a re-scan list.....

90     NXTS = BOX3(SCAN)
      BOX3(SCAN) = TMN1
      IF (NXTS.EQ.TMN1) GOTO 94
      SCAN = NXTS
      INDEX = ISIGN(1,INUM(SCAN))

C     Find the arc to be scanned.....

      SARC = BOX2(SCAN)
92     IF (SARC.EQ.TMN1) GOTO 90
      FLOWA = FLOW(SARC)
      IF (INDEX.LT.0) THEN
        LNOD = BOX7(SARC)
        SARC = BOX8(SARC)
      ELSE
        LNOD = BOX5(SARC)
        SARC = BOX6(SARC)
      ENDIF

```

```

      IF (BOX1(LNOD).NE.SCAN) GOTO 92
C   Determine a potential amount of flow for labeling.....
      FLOWM = QUAN(SCAN)*INDEX
      IF (FLOWM.LT.-FLOWA) FLOWM = -FLOWA
      QUAN(LNOD) = FLOWM*INDEX
C   Label and put LNOD in the scan list.....
      IF (BOX3(LNOD).NE.TMN1) GOTO 92
      BOX3(NLBL) = LNOD
      NLBL = LNOD
      GOTO 92
C   Rescan SNOD.....
94   INDEX = ISIGN(1,INUM(SNOD))
      SARC = BOX2(SNOD)
96   IF (SARC.EQ.TMN1) GOTO 98
      FLOWA = FLOW(SARC)
      IF (INDEX.LT.0) THEN
          LNOD = BOX7(SARC)
          SARC = BOX8(SARC)
      ELSE
          LNOD = BOX5(SARC)
          SARC = BOX6(SARC)
      ENDIF
      IF (BOX1(LNOD).NE.SNOD) GOTO 96
C   Determine possible amount of flow for labeling.....
      FLOWM = QUAN(SNOD)*INDEX
      IF (FLOWM.LT.-FLOWA) FLOWM = -FLOWA
      QUAN(LNOD) = FLOWM*INDEX
      GOTO 96
C   Put SNOD into the original scan list and go back to
C   the first scanning process.....
98   BOX4(TMN2) = SNOD
      GOTO 60

```

ARTHUR LAKES LIBRARY
 COLORADO SCHOOL of MINES
 GOLDEN, COLORADO 80401

```

C -----
C Find the zero capacity nodes that can be labeled.
C -----

100   NLAB = TMN2
      INDEX = 0

      DO 116 KK=KTOP,K
        IF (SUMP(KK).EQ.0) GOTO 108

C Find positive node with zero capacity.....

      PNOD = PNP(KK)
      NUMP = SUMP(KK)
      DO 106 NP=1,NUMP
        IF (QUAN(PNOD).NE.0) GOTO 104

102   PARC = BOX2(PNOD)
      IF (PARC.EQ.TND1) GOTO 104
      FLOWA = FLOW(PARC)
      NNOD = BOX5(PARC)
      PARC = BOX6(PARC)

      IF (QUAN(NNOD).EQ.0) GOTO 102
      IF (BOX4(NNOD).NE.TMN1) GOTO 102

      FLOWM = QUAN(NNOD)
      IF (FLOWM.GT.FLOWA) FLOWM=FLOWA
      IF (FLOWM.EQ.0) GOTO 102

C Obtain unlabeled positive node and put in scan list.....

      INDEX = 1
      QUAN(PNOD) = FLOWM
      BOX1(PNOD) = NNOD

      IF (BOX4(PNOD).NE.TMN1) GOTO 104
      BOX4(NLAB) = PNOD
      NLAB = PNOD

C Next positive node with zero capacity.....

104   PNOD = NEXT(PNOD)
106   CONTINUE

```

C Find negative node with zero capacity.....

```

108     IF (SUMN(KK).EQ.0) GOTO 116
        IF (KK.EQ.K) GOTO 116

        NNOD = NNP(KK)
        NUMN = SUMN(KK)
        DO 114 NN=1,NUMN
            IF (QUAN(NNOD).NE.0) GOTO 112

        NARC = BOX2(NNOD)
110     IF (NARC.EQ.TND1) GOTO 112
        FLOWA = FLOW(NARC)
        PNOD = BOX7(NARC)
        NARC = BOX8(NARC)

        IF (QUAN(PNOD).EQ.0) GOTO 110
        IF (BOX4(PNOD).NE.TMN1) GOTO 110

        FLOWM = QUAN(PNOD)
        IF (FLOWM.LT.-FLOWA) FLOWM=-FLOWA
        IF (FLOWM.EQ.0) GOTO 110

```

C Obtain unlabeled negative node and put in label list....

```

        INDEX = 1
        QUAN(NNOD) = FLOWM
        BOX1(NNOD) = PNOD

        IF (BOX4(NNOD).NE.TMN1) GOTO 104
        BOX4(NLAB) = NNOD
        NLAB = NNOD

```

C Next negative node.....

```

112         NNOD = NEXT(NNOD)
114     CONTINUE

```

C Next level.....

```

116     CONTINUE

        IF (INDEX.EQ.1) GOTO 60

        RETURN
        END

```

```

C*****
C  SUBROUTINE : REMOV1                                     *
C      Remove labeled nodes with access supply           *
C                                                         *
C  Programmed by : P.Huttagosol, 1987. Modified from the *
C      original program written by Barnes(1982).         *
C                                                         *
C  Variables used :                                       *
C      SMND - counter for removed nodes                  *
C      SMAC - counter for removed arcs                   *
C                                                         *
C*****
      SUBROUTINE REMOV1(K,KT,MN1,MN2,MN3,NOD,NXND,SMND,ARC,
&  NXTA,SMAC)

      IMPLICIT INTEGER(A-Z)
      PARAMETER(MD=50000,MA=250000,MZ=40)
      INTEGER*2 PNP(MZ),NNP(MZ),SUMP(MZ),SUMN(MZ),QUAN(MD)
      DIMENSION BOX2(MD),BOX6(MA),BOX7(MA),BOX8(MA)
      DIMENSION NEXT(MD)

      COMMON/BLK8/QUAN
      COMMON/BLK13/NEXT
      COMMON/BLK15/SUMP
      COMMON/BLK16/PNP
      COMMON/BLK17/SUMN
      COMMON/BLK18/NNP
      COMMON/BLK20/BOX2
      COMMON/BLK22/BOX6
      COMMON/BLK23/BOX7
      COMMON/BLK24/BOX8
      DATA TMN1,TMN2/1,2/

C  Initialize counters for nodes and arc removed.....

      MN1 = 0
      MN2 = 0
      MN3 = 0

C  -----
C  Remove positive nodes with access supply.
C  -----

      DO 124 KK=KT,K
      IF (SUMP(KK).EQ.0) GOTO 124
      NEXT(TMN2) = TMN1
      TMP1 = TMN2
      ISUM = SUMP(KK)

```

```

PNOD = PNP(KK)
DO 122 NP=1,ISUM
  IF (QUAN(PNOD).LE.0) GOTO 121
  SUMP(KK) = SUMP(KK)-1
  TMP2 = NEXT(PNOD)
  NEXT(PNOD) = NOD
  NOD = PNOD
  PNOD = TMP2
  GOTO 122

C  Next positive node.....

121      NEXT(TMP1) = PNOD
        TMP1 = PNOD
        PNOD = NEXT(PNOD)
122      CONTINUE

C  Count number of positive blocks mined.....

        IDIF = ISUM-SUMP(KK)
        MN1 = MN1+IDIF
        SMND = SMND+IDIF

C  Next level.....

        NEXT(TMP1) = TMN1
        PNP(KK) = NEXT(TMN2)
124      CONTINUE

C  -----
C  Remove negative nodes with excess supply.
C  -----

DO 139 KK=KT,K-1
  IF (SUMN(KK).EQ.0) GOTO 139
  TMP1 = TMN2

  ISUM = SUMN(KK)
  NNOD = NNP(KK)
  DO 138 NN=1,ISUM
    IF (QUAN(NNOD).LE.0) GOTO 137
    SUMN(KK) = SUMN(KK)-1

C  Remove the arc connecting to the removed node.....

130      NARC = BOX2(NNOD)
        IF (NARC.EQ.TMN1) GOTO 136
        MN3 = MN3+1

```

```

        PNOD = BOX7(NARC)
        IF (QUAN(PNOD).GT.0) GOTO 134
        PARC = BOX2(PNOD)
        IF (PARC.NE.NARC) GOTO 132
        BOX2(PNOD) = BOX6(PARC)
        GOTO 134

132      TEM3 = PARC
        PARC = BOX6(TEM3)
        IF (PARC.NE.NARC) GOTO 132
        BOX6(TEM3) = BOX6(PARC)
134      BOX6(NARC) = ARC
        ARC = NARC
        SMAC = SMAC+1
        NARC = BOX8(NARC)
        GOTO 130

C      Reinitialize the negative node properties.....

136      TMP2 = NEXT(NNOD)
        NEXT(NNOD) = NOD
        NOD = NNOD
        NNOD = TMP2
        GOTO 138

C      Next negative node.....

137      NEXT(TMP1) = NNOD
        TMP1 = NNOD
        NNOD = NEXT(NNOD)
138      CONTINUE

C      Count number of negative blocks removed.....

        IDIF = ISUM-SUMN(KK)
        MN2 = MN2+IDIF
        SMND = SMND+IDIF

C      Next level.....

        NEXT(TMP1) = TMN1
        NNP(KK) = NEXT(TMN2)
139      CONTINUE
        NXTA = BOX6(ARC)
        NXND = NEXT(NOD)

        RETURN
        END

```

```

C*****
C
C SUBROUTINE : GRAPHT
C This is the subroutine that performs the graph
C theory algorithm (tree method) in order to
C design the optimum pit limit.
C
C Programmed by : Huttagosol, 1987. Modified from the
C original program written by Dagdelen(1985).
C
C Variables used :
C SNOD - strong node number, positive node
C WNOD - weak node number, negative node
C SUMP(K) - sum of positive nodes on level K
C SUMN(K) - sum of negative nodes on level K
C SNP(K) - positive node pointer for level K
C WNP(K) - negative node pointer for level K
C CON(I,J) - cone surface level at column(I,J)
C NI(1000) - index for each iteration
C
C Subroutine called :
C INITIA - initialize the state of variables
C PLACER - generate the outer bound
C ADD2 - add nodes and initialization
C TFORM1 - transformation and connection
C EVAL1 - evaluation and normalization
C REMOV2 - remove nodes with strong index
C CONE - to generate the cone surface pattern
C INFORM - output information on screen
C FINPIT - update the final pit design
C
C*****
SUBROUTINE GRAPHT

IMPLICIT INTEGER(A-Z)
PARAMETER (MX=75,MZ=40,MD=50000)
INTEGER*2 INUM(MD),JNUM(MD),CON(MX,MX),NI(1000)
INTEGER*2 SNP(MZ),SUMP(MZ),WNP(MZ),SUMN(MZ)
DIMENSION NEXT(MD)

COMMON/BLK4/CON
COMMON/BLK11/INUM
COMMON/BLK12/JNUM
COMMON/BLK13/NEXT
COMMON/BLK15/SUMP
COMMON/BLK16/SNP
COMMON/BLK17/SUMN
COMMON/BLK18/WNP

```

```

C   Initialize the state of variables.....
      CALL INITIA(NOD,NXND,SMND)
C   Generate the outer bound by using placer method.....
      CALL PLACER(KTOP,KBOT)
C   Iterate through the levels.....
      DO 200 K=KTOP,KBOT
        K1 = K-1
        SUMP(K) = 0
        SUMN(K) = 0
C   Add nodes to the network and initialize node properties.
      CALL ADD2(K,NOD,NXND,SMND)
      AD1 = SUMP(K)
      AD2 = SUMN(K)
      IF (SUMP(K).EQ.0) GOTO 180
      IF (K.EQ.KTOP) GOTO 130
C   Search for the overlying node of a strong positive node.
      DO 20 M=1,1000
        NI(M) = 0
20    CONTINUE
C   Iterations.....
      DO 120 M=1,1000
        IF (M.GT.1) THEN
          IF (NI(M-1).EQ.0) GOTO 130
        ENDIF
C   Obtain the strong positive node from the list.....
      DO 110 NK=K,KTOP+1,-1
        IF (SUMP(NK).EQ.0) GOTO 110
        SNOD = SNP(NK)

        NUMS = SUMP(NK)
        DO 100 NS=1,NUMS
          IF (INUM(SNOD).LE.0) GOTO 90
          IID = INUM(SNOD)
          JID = JNUM(SNOD)

```

```

C   Generate the cone whose base is on the positive block...
      CALL CONE(IID,JID,NK,MNI,MXI,MNJ,MXJ)
C   Find the overlying weak negative nodes within the cone..
      DO 80 KK=K1,KTOP,-1
        IF (SUMN(KK).EQ.0) GOTO 80

        WNOD = WNP(KK)
        NUMW = SUMN(KK)
        DO 75 NW=1,NUMW
          IF (INUM(WNOD).GT.0) GOTO 72

          I1 = IABS(INUM(WNOD))
          J1 = JNUM(WNOD)
          IF (KK.GT.CON(I1,J1)) GOTO 72
          NI(M) = 1
        75
      80
C   Transform the tree containing SNOD and connect to WNOD..
      CALL TFORM1(SNOD,WNOD,RNOD)
C   Evaluate the successive tree and normalize if required..
      CALL EVAL1(RNOD)
C   Next negative node.....
70           IF (INUM(SNOD).LE.0) GOTO 90
72           WNOD = NEXT(WNOD)
75           CONTINUE
80           CONTINUE
C   Next positive node.....
90           SNOD = NEXT(SNOD)
100          CONTINUE
110          CONTINUE
120          CONTINUE
C   Remove strong nodes with access supply.....
130          CALL REMOV2(K,KTOP,MN1,MN2,NOD,NXND,SMND)
C   Write information on screen and next level.....
180          IOPT = 2

```

```

                CALL INFORM(K, IOPT, AD1, AD2, AD3, MN1, MN2, MN3)
200          CONTINUE

```

```

C    Determine the final pit surface.....

```

```

                CALL FINPIT(KTOP, KBOT)

```

```

                RETURN
                END

```

```

C*****
C
C    SUBROUTINE : ADD2
C          Add nodes to the original tree algorithm and
C          initialization
C
C    Programmed by : P.Huttagosol, 1987
C
C    Variables used :
C          INUM(N) - block ID , ore/waste index
C          JNUM(N) - block ID on Y axis
C
C*****
C          SUBROUTINE ADD2(K, NOD, NXND, SMND)

```

```

          IMPLICIT INTEGER(A-Z)
          PARAMETER (MX=75, MZ=40, MD=50000)
          REAL XDM, YDM, ZDM, TANI, TANJ
          INTEGER*2 TOP (MX, MX), BOT (MX, MX), INUM (MD), JNUM (MD)
          INTEGER*2 SNP (MZ), SUMP (MZ), WNP (MZ), SUMN (MZ)
          INTEGER*2 VALU (MD), KVAL
          DIMENSION ROOT (MD), BNCH (MD), TWIG (MD), NEXT (MD)

```

```

          COMMON/BLK1/NX, NY, NZ, XDM, YDM, ZDM, IM, JM, TANI, TANJ
          COMMON/BLK2/TOP
          COMMON/BLK3/BOT
          COMMON/BLK5/ROOT
          COMMON/BLK6/BNCH
          COMMON/BLK7/TWIG
          COMMON/BLK8/VALU
          COMMON/BLK11/INUM
          COMMON/BLK12/JNUM
          COMMON/BLK13/NEXT
          COMMON/BLK15/SUMP
          COMMON/BLK16/SNP
          COMMON/BLK17/SUMN

```

```
COMMON/BLK18/WNP
DATA TMN1/1/
```

C Initialize variables.....

```
NXTS = TMN1
NXTW = TMN1
```

C Read block values on the current level K.....

```
DO 15 I=1,NX
  I1 = (I-1)*NY

  DO 10 J=1,NY
    IF (K.LE.TOP(I,J).OR.K.GT.BOT(I,J)) GOTO 10
    KRN = (I1+J-1)*NZ+K
    READ(10,REC=KRN) KVAL
```

C Initialize arrays of node properties.....

```
VALU(NOD) = KVAL
ROOT(NOD) = TMN1
BNCH(NOD) = TMN1
TWIG(NOD) = TMN1
JNUM(NOD) = J

IF (KVAL.GT.0) THEN
  NEXT(NOD) = NXTS
  INUM(NOD) = I
  SUMP(K) = SUMP(K)+1
  NXTS = NOD
ELSE
  NEXT(NOD) = NXTW
  INUM(NOD) = -I
  SUMN(K) = SUMN(K)+1
  NXTW = NOD
ENDIF
```

C Prepare the identification number for next node.....

```
IF (SMND.EQ.0) THEN
  NOD = NOD+1
ELSE
  NOD = NXND
  NXND = NEXT(NOD)
  SMND = SMND-1
ENDIF
```

```

C   Next node from the same level.....

10      CONTINUE
15      CONTINUE
        SNP(K) = NXTS
        WNP(K) = NXTW

        RETURN
        END

C*****
C
C   SUBROUTINE : TFORM1
C           Transformation and connection step of original
C           graph algorithm.
C
C   Programmed by : Huttagosol, 1987. Modified from the
C           original program written by Dagdelen(1985).
C
C   Variables used :
C           RNOD - extreme root node of a strong tree
C           ROOT(N) - root node or predecessor of node N
C           BNCH(N) - branch node or successor of node N
C           TWIG(N) - another successor of node ROOT(N)
C*****
C   SUBROUTINE TFORM1(SNOD,WNOD,RNOD)

        IMPLICIT INTEGER(A-Z)
        PARAMETER(MD=50000)
        DIMENSION ROOT(MD),BNCH(MD),TWIG(MD)

        COMMON/BLK5/ROOT
        COMMON/BLK6/BNCH
        COMMON/BLK7/TWIG
        DATA TMN1/1/

C   Find the extreme root node of the tree containing SNOD..

        RNOD = SNOD
        DOWHILE (ROOT(RNOD).NE.TMN1)
            RNOD = ROOT(RNOD)
        ENDDO

C   Transformation of a strong tree from RNOD to SNOD.....

        NOD1 = RNOD

```

```

30   IF (NOD1.EQ.SNOD) GOTO 38
      TMP1 = ROOT(NOD1)
      TMP2 = BNCH(NOD1)

      NOD2 = SNOD
35   TMP3 = NOD2
      NOD2 = ROOT(NOD2)
      IF (NOD2.NE.NOD1) GOTO 35

```

C The p-edge is transformed to be m-edge, and vice versa..

```

      ROOT(NOD1) = TMP3
      IF (NOD1.EQ.RNOD) THEN
        IF (TMP2.EQ.TMP3) THEN
          BNCH(NOD1) = TWIG(TMP2)
        ENDIF
      ELSE
        BNCH(NOD1) = TMP1
        IF (TMP2.EQ.TMP3) THEN
          TWIG(TMP1) = TWIG(TMP2)
        ELSE
          TWIG(TMP1) = TMP2
        ENDIF
      ENDIF

      IF (TMP2.NE.TMP3) THEN
        NOD2 = TMP2
        DOWHILE (TWIG(NOD2).NE.TMP3)
          NOD2 = TWIG(NOD2)
        ENDDO
        TWIG(NOD2) = TWIG(TMP3)
      ENDIF
      NOD1 = TMP3
      GOTO 30

```

C disconnect tree containing strong positive node from
C the dummy root and reconnect to the overlying weak node.

```

38   TMP1 = ROOT(SNOD)
      TMP2 = BNCH(SNOD)
      ROOT(SNOD) = WNOD
      BNCH(SNOD) = TMP1
      TWIG(SNOD) = BNCH(WNOD)
      BNCH(WNOD) = SNOD
      TWIG(TMP1) = TMP2

      RETURN
      END

```

```

C*****
C
C SUBROUTINE : EVAL1
C Evaluate the tree for each node and normalize
C the tree if strong edge is found not common
C to the dummy.
C
C Programmed by : Huttagosol, 1987. Modified from the
C original program written by Dagdelen(1985).
C
C Variables used :
C VALU(N) - original block value of node N
C CUMV(N) - cumulative value to root node N
C INDEX - strong/weak index
C
C*****
SUBROUTINE EVAL1(RNOD)

IMPLICIT INTEGER(A-Z)
PARAMETER(MD=50000)
INTEGER*2 VALU(MD),CUMV(MD),INUM(MD)
DIMENSION ROOT(MD),BNCH(MD),TWIG(MD)

COMMON/BLK5/ROOT
COMMON/BLK6/BNCH
COMMON/BLK7/TWIG
COMMON/BLK8/VALU
COMMON/BLK9/CUMV
COMMON/BLK11/INUM
DATA TMN1/1/

C Evaluation the cumulative value of each nodes.....

NOD1 = RNOD
40 IF (NOD1.EQ.TMN1) GOTO 50

IF (BNCH(NOD1).EQ.TMN1) THEN
CUMV(NOD1) = VALU(NOD1)
ELSE
NOD2 = BNCH(NOD1)
CUMV(NOD1) = VALU(NOD1)+CUMV(NOD2)

DOWHILE (TWIG(NOD2).NE.TMN1)
NOD2 = TWIG(NOD2)
CUMV(NOD1) = CUMV(NOD1)+CUMV(NOD2)
ENDDO
ENDIF

```

```
NOD1 = ROOT(NOD1)
GOTO 40
```

C Normalize the tree when a strong edge is found.....

```
50  NOD1 = RNOD
    RNOD = ROOT(NOD1)
    IF (RNOD.EQ.TMN1) GOTO 58

    IF (VALU(NOD1).GT.0.AND.CUMV(NOD1).LE.0) GOTO 52
    IF (VALU(NOD1).LE.0.AND.CUMV(NOD1).GT.0) GOTO 52
    GOTO 50
```

```
52  NOD2 = RNOD
54  IF (NOD2.EQ.TMN1) GOTO 56
    CUMV(NOD2) = CUMV(NOD2) - CUMV(NOD1)
    NOD2 = ROOT(NOD2)
    GOTO 54
```

C Remove strong edge and place it common to the dummy.....

```
56  IF (BNCH(RNOD).NE.NOD1) THEN
    NOD2 = BNCH(RNOD)
    DOWHILE (TWIG(NOD2).NE.NOD1)
    NOD2 = TWIG(NOD2)
    ENDDO
    TWIG(NOD2) = TWIG(NOD1)
ELSE
    BNCH(RNOD) = TWIG(NOD1)
ENDIF

    ROOT(NOD1) = TMN1
    TWIG(NOD1) = TMN1
```

C Specify the extreme root node to be weak or strong.....

```
58  IF (CUMV(NOD1).GT.0) THEN
    INDEX = 1
ELSE
    INDEX = -1
ENDIF
    IRN = IABS(INUM(NOD1))
    INUM(NOD1) = INDEX*IRN
```

C Re-evaluate the ore-waste index of the normalized tree..

```
IF (BNCH(NOD1).EQ.TMN1) GOTO 68
```

```

        NOD2 = BNCH(NOD1)
62      IRN = IABS(INUM(NOD2))
        INUM(NOD2) = INDEX*IRN
        IF (BNCH(NOD2).EQ.TMN1) GOTO 64
        NOD2 = BNCH(NOD2)
        GOTO 62

64      IF (TWIG(NOD2).NE.TMN1) THEN
            NOD2 = TWIG(NOD2)
        ELSE
66          NOD2 = ROOT(NOD2)
            IF (NOD2.EQ.NOD1) GOTO 68
            IF (TWIG(NOD2).EQ.TMN1) GOTO 66
            NOD2 = TWIG(NOD2)
        ENDIF
        GOTO 62

68      IF (RNOD.EQ.TMN1) GOTO 70
        GOTO 50

70      RETURN
        END

```

```

C*****
C
C   SUBROUTINE : REMOV2
C           Remove nodes with strong index.
C
C   Programmed by : P.Huttagosol, 1987
C
C   Variables used :
C           MIN1 - number of positive nodes removed
C           MIN2 - number of negative nodes removed
C
C*****
SUBROUTINE REMOV2(K,KTOP,MIN1,MIN2,NOD,NXND,SMND)

IMPLICIT INTEGER(A-Z)
PARAMETER(MZ=40,MD=50000)
DIMENSION NEXT(MD)
INTEGER*2 SNP(MZ),SUMP(MZ),WNP(MZ),SUMN(MZ),INUM(MD)

COMMON/BLK11/INUM
COMMON/BLK13/NEXT
COMMON/BLK15/SUMP
COMMON/BLK16/SNP

```

```

COMMON/BLK17/SUMN
COMMON/BLK18/WNP
DATA TMN1, TMN2/1, 2/

```

C Initialize variables.....

```

MIN1 = 0
MIN2 = 0

```

C Mine strong positive nodes and remove from the list.....

```

DO 170 KK=KTOP, K
  IF (SUMP(KK).EQ.0) GOTO 150

  NEXT(TMN2) = TMN1
  TMP1 = TMN2

  ISUM = SUMP(KK)
  SNOD = SNP(KK)

  DO 140 NS=1, ISUM
    IF (INUM(SNOD).LE.0) GOTO 135

    SUMP(KK) = SUMP(KK)-1
    TMP2 = NEXT(SNOD)
    NEXT(SNOD) = NOD
    NOD = SNOD
    SNOD = TMP2
    GOTO 140

```

C Next positive node.....

```

135     NEXT(TMP1) = SNOD
        TMP1 = SNOD
        SNOD = NEXT(SNOD)
140     CONTINUE

```

C Count number of positive blocks mined.....

```

IDIF = ISUM-SUMP(KK)
MIN1 = MIN1+IDIF
SMND = SMND+IDIF
NEXT(TMP1) = TMN1
SNP(KK) = NEXT(TMN2)

```

```
C   Iterate through negative nodes on the same level.....

150     IF (KK.EQ.K) GOTO 170
        IF (SUMN(KK).EQ.0) GOTO 170

        NEXT(TMN2) = TMN1
        TMP1 = TMN2

        ISUM = SUMN(KK)
        WNOD = WNP(KK)

        DO 160 NW=1,ISUM
            IF (INUM(WNOD).LE.0) GOTO 155

C   Mine strong negative nodes and remove from the list.....

        SUMN(KK) = SUMN(KK)-1
        TMP2 = NEXT(WNOD)
        NEXT(WNOD) = NOD
        NOD = WNOD
        WNOD = TMP2
        GOTO 160

C   Next negative node.....

155     NEXT(TMP1) = WNOD
        TMP1 = WNOD
        WNOD = NEXT(WNOD)

160     CONTINUE

C   Count number of negative blocks mined.....

        IDIF = ISUM-SUMN(KK)
        MIN2 = MIN2+IDIF
        SMND = SMND+IDIF

        NEXT(TMP1) = TMN1
        WNP(KK) = NEXT(TMN2)

170     CONTINUE
        NXND = NEXT(NOD)

        RETURN
        END
```

```

C*****
C
C      SUBROUTINE : MGRAPH
C          This is the main subroutine to determine the
C          optimum pit limit by using the modified-tree
C          graph algorithm.
C
C      Programmed by : P.Huttagosol, 1987
C
C      Variables used :
C          SUMP(K) - sum of positive nodes on level K
C          PNP(K) - positive node pointer for level K
C          CON(I,J) - generated cone surface at (I,J)
C          PIT(I,J) - intermediate pit surface at (I,J)
C          PNOD - weak positive node
C          SNOD - strong positive node
C          KRN - record number, negative block ID
C          IRN - column number of positive block
C          NI(INUM) - number of iterations
C
C      Subroutine called :
C          CONE - generate the cone with given slopes
C          PLACER - create a loose outer bound
C          RECORD - rearrange block ID number
C          ADD3 - add nodes to the tree network
C          STATUS - find status of the weak tree
C          TFORM2 - transform the tree containing SNOD
C          CONNec - connect SNOD to overlying node
C          EVAL2 - evaluation of the successive tree
C          REMOV3 - remove nodes and update pit surface
C          INFORM - write information on the screen
C
C*****
      SUBROUTINE MGRAPH

      IMPLICIT INTEGER(A-Z)
      PARAMETER(MX=75,MZ=40,MP=7000,MT=75000)
      INTEGER*2 BVAL(MT),BASE(MT),RNUM(MP),NEXT(MP)
      INTEGER*2 NI(1000),PNP(MZ),SUMP(MZ)
      INTEGER*2 TOP(MX,MX),CON(MX,MX),PIT(MX,MX)
      DIMENSION BID(MX,MX)
      REAL XDM,YDM,ZDM,TANI,TANJ

      COMMON/BLK1/NX,NY,NZ,XDM,YDM,ZDM,IM,JM,TANI,TANJ
      COMMON/BLK2/TOP
      COMMON/BLK4/CON
      COMMON/OPUT/PIT
      COMMON/TREC/BID

```

```

COMMON/BCK11/RNUM
COMMON/BCK13/NEXT
COMMON/BLK15/SUMP
COMMON/BLK16/PNP
COMMON/BCK19/BVAL
COMMON/BCK23/BASE

C   Initialize the state of variables.....
      CALL INITIA(NOD,NXND,SMND)

C   Generate the outer bound.....
      CALL PLACER(KTOP,KBOT)

C   Rearrange the block identification number.....
      CALL RECORD

C   Iterate through the levels.....
      DO 300 K=KTOP,KBOT
        K1 = K-1
        SUMP(K) = 0

C   Add nodes to the tree network.....
      CALL ADD3(K,NOD,NXND,SMND,AD2)
      AD1 = SUMP(K)

      IF (SUMP(K).EQ.0) GOTO 290
      IF (K.EQ.KTOP) GOTO 210

C   Iterate the algorithm from search to evaluation step....
      DO 10 M=1,1000
        NI(M) = 0
10    CONTINUE

      DO 200 M=1,1000
        IF (M.EQ.1000) THEN
          WRITE(*,*) 'ERROR! No.of iteration exceeds.'
          PAUSE
        ENDIF

        IF (M.GT.1) THEN
          IF (NI(M-1).EQ.0) GOTO 210
        ENDIF

```

```

C   Retrieve the strong positive node from the list and
C   generate the cone with the node as the cone base.....

      DO 190 NK=K,KTOP+1,-1
        IF (SUMP(NK).EQ.0) GOTO 190

        SNOD = PNP(NK)
        NUMP =SUMP(NK)

      DO 180 NS=1,NUMP
        IF (RNUM(SNOD).LE.0) GOTO 175

        IRN = RNUM(SNOD)
        JID = MOD(IRN,NY)
        IF (JID.EQ.0) JID=NY
        IID = (IRN-JID)/NY+1

        CALL CONE(IID,JID,NK,MNI,MXI,MNJ,MXJ)

C   Search for overlying weak nodes.....

      DO 170 I=MNI,MXI
        DO 160 J=MNJ,MXJ
          KMIN = PIT(I,J)+1
          KMAX = CON(I,J)

          DO 150 KK=KMIN,KMAX
            KRN = BID(I,J)-TOP(I,J)+KK
            IF (BVAL(KRN).GT.0) GOTO 150

            PNOD = BASE(KRN)
            IF (PNOD.EQ.SNOD) GOTO 150
            IF (RNUM(PNOD).GT.0) GOTO 150
            NI(M) =1

C   Find the status of the tree containing negative node KRN

            CALL STATUS(PNOD,QNOD,KRN,POINT)

C   Transformation the tree containing SNOD.....

            CALL TFORM2(SNOD,RNOD,KRN)

C   Connection of SNOD to its overlying negative node.....

            CALL CONNOC(PNOD,QNOD,RNOD,SNOD,
&                      KRN,POINT)

```

```
C   Evaluation and normalization if a strong edge occurs.....
                                CALL EVAL2(RNOD)
C   Next negative node.....
                                IF (RNUM(SNOD).LE.0) GOTO 175
150                               CONTINUE
160                               CONTINUE
170                               CONTINUE
C   Next strong positive node.....
175                               SNOD = NEXT(SNOD)
180                               CONTINUE
190                               CONTINUE
C   Next iteration.....
200                               CONTINUE
C   Remove strong positive nodes and update the pit.....
210                               CALL REMOV3(K,KTOP,MN1,MN2,NOD,NXND,SMND)
C   Output information on the screen and next level.....
290                               IOPT = 3
                                CALL INFORM(K,IOPT,AD1,AD2,AD3,MN1,MN2,MN3)
300                               CONTINUE

                                RETURN
                                END
```

```

C*****
C
C SUBROUTINE : ADD3
C Add nodes to the MGRAPH subroutine and
C initialization of level K
C
C Programmed by : P.Huttagosol, 1987
C
C Variables used :
C RNUM(N) - record no. and ore/waste index
C
C*****
SUBROUTINE ADD3(K,NOD,NXND,SMND,AD2)

IMPLICIT INTEGER(A-Z)
PARAMETER(MX=75,MZ=40,MP=7000,MT=75000)
DIMENSION NNOD(MP),BID(MX,MX)
INTEGER*2 BVAL(MT),BASE(MT),SUMV(MP),RNUM(MP)
INTEGER*2 ROOT(MP),BNCH(MP),TWIG(MP),VINE(MP),PNP(MZ)
INTEGER*2 TOP(MX,MX),BOT(MX,MX),NEXT(MP),SUMP(MZ)
REAL XDM,YDM,ZDM,TANI,TANJ

COMMON/BLK1/NX,NY,NZ,XDM,YDM,ZDM,IM,JM,TANI,TANJ
COMMON/BLK2/TOP
COMMON/BLK3/BOT
COMMON/TREC/BID
COMMON/BCK5/ROOT
COMMON/BCK6/BNCH
COMMON/BCK7/TWIG
COMMON/BCK8/SUMV
COMMON/BCK11/RNUM
COMMON/BCK12/VINE
COMMON/BCK13/NEXT
COMMON/BLK15/SUMP
COMMON/BLK16/PNP
COMMON/BCK19/BVAL
COMMON/BCK20/NNOD
COMMON/BCK23/BASE
DATA TMN1,TMN3/1,1/

C Initialize variables.....

NXTP = TMN1
AD2 = 0

C Read block model on level K.....

DO 10 I=1,NX

```

```

      I1 = (I-1)*NY
      DO 8 J=1,NY
        IF (K.LE.TOP(I,J).OR.K.GT.BOT(I,J)) GOTO 8
        IRN = I1+J
        KRN = BID(I,J)-TOP(I,J)+K
        IF (BVAL(KRN).GT.0) THEN
C      Initialize arrays of positive node properties.....

          SUMV(NOD) = BVAL(KRN)
          RNUM(NOD) = IRN
          ROOT(NOD) = TMN1
          BNCH(NOD) = TMN1
          TWIG(NOD) = TMN1
          VINE(NOD) = TMN1
          NNOD(NOD) = TMN3
          NEXT(NOD) = NXTP

          SUMP(K) = SUMP(K)+1
          NXTP = NOD

C      Prepare the identification number for next positive node

          IF (SMND.EQ.0) THEN
            NOD = NOD+1
          ELSE
            NOD = NXND
            NXND = NEXT(NOD)
            SMND = SMND-1
          ENDIF

C      Initialize negative block array.....

          ELSE
            BASE(KRN) = TMN1
            AD2 = AD2+1
          ENDIF

C      Next block.....

8          CONTINUE
10         CONTINUE
          PNP(K) = NXTP

          RETURN
          END

```

```

C*****
C
C SUBROUTINE : STATUS
C Define the status of the weak tree on which
C the overlying block KRN is located.
C
C Programmed by : P.Huttagosol, 1987
C
C Variables used :
C PNOD - positive weak node
C QNOD - successive node of PNOD
C POINT - indicator for status of weak tree
C*****
C SUBROUTINE STATUS(PNOD,QNOD,KRN,POINT)
C
C IMPLICIT INTEGER(A-Z)
C PARAMETER(MP=7000)
C DIMENSION NNOD(MP)
C INTEGER*2 BNCH(MP),TWIG(MP)
C
C COMMON/BCK6/BNCH
C COMMON/BCK7/TWIG
C COMMON/BCK20/NNOD
C DATA TMN1/1/
C
C The root node of KRN is the dummy vertex.....
C
C POINT = 0
C IF (PNOD.EQ.TMN1) GOTO 20
C
C PNOD and SNOD will be connected to the same node KRN....
C
C IF (NNOD(PNOD).NE.KRN) GOTO 10
C POINT = 1
C GOTO 20
C
C Negative node KRN is supported only by node PNOD.....
C
10 QNOD = BNCH(PNOD)
12 IF (QNOD.EQ.TMN1) GOTO 20
IF (NNOD(QNOD).EQ.KRN) GOTO 14
QNOD = TWIG(QNOD)
GOTO 12
C
C KRN is mutually supported by PNOD and QNOD.....
C
14 POINT = 2

```

```

      IF (BNCH(PNOD).EQ.QNOD) THEN
        BNCH(PNOD) = TWIG(QNOD)
      ELSE
        NOD1 = BNCH(PNOD)
16      IF (TWIG(NOD1).EQ.QNOD) GOTO 18
        NOD1 = TWIG(NOD1)
        GOTO 16
18      TWIG(NOD1) = TWIG(QNOD)
      ENDIF

20     RETURN
      END

```

```

C*****
C
C      SUBROUTINE : TFORM2
C      Transformation step of the modified-tree
C      graph algorithm.
C
C      Programmed by : P.Huttagosol, 1987
C
C      Variables used :
C      ROOT(N) - root node or predecessor of node N
C      BNCH(N) - branch node or successor of node N
C      TWIG(N) - another succeesor of node ROOT(N)
C      VINE(N) - vine node of node N, term M(N)
C      RNOD    - extreme root node of the transformation
C              chain
C
C*****

```

```

SUBROUTINE TFORM2(SNOD,RNOD,KRN)

```

```

IMPLICIT INTEGER(A-Z)
PARAMETER(MP=7000,MT=75000)
DIMENSION NNOD(MP)
INTEGER*2 SUMV(MP),BVAL(MT),BASE(MT)
INTEGER*2 ROOT(MP),BNCH(MP),TWIG(MP),VINE(MP)

```

```

COMMON/BCK5/ROOT
COMMON/BCK6/BNCH
COMMON/BCK7/TWIG
COMMON/BCK8/SUMV
COMMON/BCK12/VINE
COMMON/BCK19/BVAL
COMMON/BCK20/NNOD
COMMON/BCK23/BASE
DATA TMN1,TMN3/1,1/

```

```

C -----
C Find the extreme root node of the strong tree.
C -----

      RNOD = SNOD
22  IF (ROOT(RNOD).LE.TMN1) GOTO 25
      RNOD = ROOT(RNOD)
      GOTO 22

C -----
C Transformation of the strong tree at RNOD.
C -----

25  IF (RNOD.EQ.SNOD) GOTO 70
      TMP1 = VINE(RNOD)
      TMP2 = BNCH(RNOD)

C Find a successor of RNOD on the transformation chain....

      TMP3 = SNOD
30  IF (ROOT(TMP3).EQ.RNOD) GOTO 32
      TMP3 = ROOT(TMP3)
      GOTO 30

C Change P-edge to M-edge.....

32  NRN = NNOD(TMP3)
      RRN = NNOD(RNOD)
      ROOT(RNOD) = TMP3

      IF (TMP2.NE.TMP3) GOTO 34
      IF (TMP1.EQ.TMN1) THEN
          BNCH(RNOD) = TWIG(TMP2)
      ELSE
          BNCH(RNOD) = TMP1
          TWIG(TMP1) = TWIG(TMP2)
      ENDIF

      VINE(RNOD) = VINE(TMP2)
      NOD2 = RNOD
      GOTO 46

34  NOD2 = TMP2
36  IF (NOD2.EQ.TMN1) GOTO 40
      IF (TWIG(NOD2).EQ.TMP3) GOTO 38
      NOD2 = TWIG(NOD2)
      GOTO 36

```

```
38     IF (TMP1.EQ.TMN1) THEN
        TWIG(NOD2) = TWIG(TMP3)
    ELSE
        TWIG(NOD2) = TMP1
        TWIG(TMP1) = TWIG(TMP3)
    ENDIF
    VINE(RNOD) = VINE(TMP3)
    NOD2 = RNOD
    GOTO 46

40     IF (RRN.EQ.NRN) THEN
        NOD2 = RNOD
    ELSE
        NOD2 = TMP2
42     IF (NNOD(NOD2).EQ.NRN) GOTO 44
        NOD3 = NOD2
        NOD2 = TWIG(NOD2)
        GOTO 42
44     VINE(RNOD) = NOD2
        ROOT(NOD2) = TMP3
        IF (NOD2.EQ.TMP2) THEN
            IF (TMP1.EQ.TMN1) THEN
                BNCH(RNOD) = TWIG(TMP2)
            ELSE
                BNCH(RNOD) = TMP1
                TWIG(TMP1) = TWIG(TMP2)
            ENDIF
        ELSE
            IF (TMP1.EQ.TMN1) THEN
                TWIG(NOD3) = TWIG(NOD2)
            ELSE
                TWIG(NOD3) = TMP1
                TWIG(TMP1) = TWIG(NOD2)
            ENDIF
        ENDIF
    ENDIF

46     IF (VINE(NOD2).LE.TMN1) GOTO 48
        IF (VINE(NOD2).EQ.TMP3) VINE(NOD2)=VINE(TMP3)
        NOD2 = VINE(NOD2)
        IF (NOD2.EQ.TMN1) GOTO 48
        ROOT(NOD2) = TMP3
        GOTO 46

48     VINE(TMP3) = TMN1
        NNOD(RNOD) = NRN
```

```

C   Reevaluate SUMV for RNOD.....
      VAL2 = BVAL(RRN)
      IF (RRN.EQ.TMN3) VAL2=0
      IF (RRN.EQ.NRN) THEN
        SUMV(TMP3) = SUMV(TMP3)+VAL2
      ELSE
        SUMV(RNOD) = SUMV(RNOD)+VAL2
        VAL2 = BVAL(NRN)
        SUMV(RNOD) = SUMV(RNOD)-VAL2
        SUMV(TMP3) = SUMV(TMP3)+VAL2
      ENDIF
      BASE(NRN) = TMP3

C   -----
C   Transformation of the strong tree from RNOD to SNOD.
C   -----

50   NOD1 = TMP3
      IF (NOD1.EQ.SNOD) GOTO 70
      TMP1 = ROOT(NOD1)
      TMP2 = BNCH(NOD1)

C   Find a successor node NOD1 on the transformation chain..

      TMP3 = SNOD
52   IF (ROOT(TMP3).EQ.NOD1) GOTO 54
      TMP3 = ROOT(TMP3)
      GOTO 52

C   Change P-edge to M-edge and vice versa.....

54   NRN = NNOD(TMP3)
      ROOT(NOD1) = TMP3
      BNCH(NOD1) = TMP1

      IF (TMP2.NE.TMP3) GOTO 56
      TWIG(TMP1) = TWIG(TMP2)
      VINE(NOD1) = VINE(TMP2)
      NOD2 = NOD1
      GOTO 68

56   NOD2 = TMP2
58   IF (NOD2.EQ.TMN1) GOTO 62
      IF (TWIG(NOD2).EQ.TMP3) GOTO 60
      NOD2 = TWIG(NOD2)
      GOTO 58

```

```
60     TWIG(NOD2) = TWIG(TMP3)
        TWIG(TMP1) = TMP2
        VINE(NOD1) = VINE(TMP3)
        NOD2 = NOD1
        GOTO 68

62     NOD2 = TMP2
64     IF (NNOD(NOD2).EQ.NRN) GOTO 66
        NOD3 = NOD2
        NOD2 = TWIG(NOD2)
        GOTO 64

66     VINE(NOD1) = NOD2
        ROOT(NOD2) = TMP3
        IF (NOD2.EQ.TMP2) THEN
            TWIG(TMP1) = TWIG(TMP2)
        ELSE
            TWIG(NOD3) = TWIG(NOD2)
            TWIG(TMP1) = TMP2
        ENDIF

68     IF (VINE(NOD2).LE.TMN1) GOTO 69
        IF (VINE(NOD2).EQ.TMP3) VINE(NOD2)=VINE(TMP3)
        NOD2 = VINE(NOD2)
        IF (NOD2.EQ.TMN1) GOTO 69
        ROOT(NOD2) = TMP3
        GOTO 68

69     VINE(TMP3) = TMN1
        NNOD(NOD1) = NRN

C     Reevaluate term SUMV.....

        VAL2 = BVAL(NRN)
        SUMV(NOD1) = SUMV(NOD1)-VAL2
        SUMV(TMP3) = SUMV(TMP3)+VAL2
        BASE(NRN) = TMP3
        GOTO 50

70     RETURN
        END
```

```

C*****
C
C   SUBROUTINE : CONNEC
C           Connection step of the modified-tree graph
C
C   Programmed by : P.Huttagosol, 1987
C
C   Variables used :
C       BASE(i) - positive node that pays for the
C               removal of negative block i
C       NNOD(N) - the negative node that overlies
C               both node N and node ROOT(N).
C*****
C   SUBROUTINE CONNEC(PNOD,QNOD,RNOD,SNOD,KRN,POINT)
C
C       IMPLICIT INTEGER(A-Z)
C       PARAMETER(MP=7000,MT=75000)
C       DIMENSION NNOD(MP)
C       INTEGER*2 SUMV(MP),BVAL(MT),BASE(MT)
C       INTEGER*2 ROOT(MP),BNCH(MP),TWIG(MP),VINE(MP)
C
C       COMMON/BCK5/ROOT
C       COMMON/BCK6/BNCH
C       COMMON/BCK7/TWIG
C       COMMON/BCK8/SUMV
C       COMMON/BCK12/VINE
C       COMMON/BCK19/BVAL
C       COMMON/BCK20/NNOD
C       COMMON/BCK23/BASE
C       DATA TMN1,TMN3/1,1/
C
C   If VINE(SNOD) is not empty, put it in BNCH(SNOD).....
C
C       IF (VINE(SNOD).EQ.TMN1) GOTO 72
C       VNOD = VINE(SNOD)
C       TWIG(VNOD) = BNCH(SNOD)
C       BNCH(SNOD) = VNOD
C       VINE(SNOD) = TMN1
C
C   Connect tree containing SNOD to tree containing KRN.....
72   IF (PNOD.EQ.TMN1.AND.ROOT(SNOD).EQ.TMN1) GOTO 76
C
C       TMP1 = ROOT(SNOD)
C       TMP2 = BNCH(SNOD)

```

```

      IF (TMP1.NE.TMN1) THEN
        BNCH(SNOD) = TMP1
        TWIG(TMP1) = TMP2
      ENDIF
      IF (POINT.EQ.1) THEN
        ROOT(SNOD) = TMN1
        TWIG(SNOD) = TMN1
        VINE(SNOD) = PNOD
        VNOD = PNOD

74      IF (VNOD.LE.TMN1) GOTO 76
        ROOT(VNOD) = SNOD
        VNOD = VINE(VNOD)
        GOTO 74

      ELSE
        ROOT(SNOD) = PNOD

        IF (PNOD.EQ.TMN1) THEN
          TWIG(SNOD) = TMN1
        ELSE
          TWIG(SNOD) = BNCH(PNOD)
          BNCH(PNOD) = SNOD
        ENDIF
        IF (POINT.EQ.2) THEN
          VINE(SNOD) = QNOD
        ENDIF

      ENDIF

76      IF (PNOD.EQ.TMN1.OR.POINT.EQ.1) THEN
        BASE(KRN) = SNOD
      ENDIF

C      Recalculate the term SUMV for SNOD after connection.....

      IF (SNOD.EQ.RNOD) THEN
        NRN = NNOD(SNOD)
        VAL2 = BVAL(NRN)
        IF (NRN.EQ.TMN3) VAL2=0
        SUMV(SNOD) = SUMV(SNOD)+VAL2
      ENDIF
      NNOD(SNOD) = KRN

      RETURN
      END

```

```

C*****
C
C SUBROUTINE : EVAL2
C Evaluate the whole tree and normalization step
C for the modified-tree graph algorithm.
C
C Programmed by : P.Huttagosol, 1987
C
C Variables used :
C SUMV(N) - modified sum value of node N
C CUMV(N) - total mass supported by node N
C TOTV - total mass supported by a tree
C*****
C SUBROUTINE EVAL2(RNOD)
C
C IMPLICIT INTEGER(A-Z)
C PARAMETER(MP=7000,MT=75000)
C INTEGER*2 BVAL(MT),BASE(MT),SUMV(MP),CUMV(MP),RNUM(MP)
C INTEGER*2 ROOT(MP),BNCH(MP),TWIG(MP),VINE(MP)
C DIMENSION NNOD(MP)
C
C COMMON/BCK5/ROOT
C COMMON/BCK6/BNCH
C COMMON/BCK7/TWIG
C COMMON/BCK8/SUMV
C COMMON/BCK9/CUMV
C COMMON/BCK11/RNUM
C COMMON/BCK12/VINE
C COMMON/BCK19/BVAL
C COMMON/BCK20/NNOD
C COMMON/BCK23/BASE
C DATA TMN1,TMN3/1,1/
C
C -----
C Evaluate a term CUMV for all nodes in the tree.
C -----
C
C NOD1 = RNOD
80 IF (NOD1.EQ.TMN1) GOTO 90
IF (BNCH(NOD1).EQ.TMN1) THEN
CUMV(NOD1) = SUMV(NOD1)
ELSE
NOD2 = BNCH(NOD1)
CUMV(NOD1) = SUMV(NOD1)+CUMV(NOD2)
82 VNOD = NOD2
84 IF (VINE(VNOD).LE.TMN1) GOTO 86
VNOD = VINE(VNOD)

```

```

      CUMV(NOD1) = CUMV(NOD1)+CUMV(VNOD)
      GOTO 84
86    IF (TWIG(NOD2).EQ.TMN1) GOTO 88
      NOD2 = TWIG(NOD2)
      CUMV(NOD1) = CUMV(NOD1)+CUMV(NOD2)
      GOTO 82
      ENDIF
88    NOD1 = ROOT(NOD1)
      GOTO 80

C    -----
C    Normalize the tree if strong M-edges are found.
C    -----

90    NOD1 = RNOD
      RNOD = ROOT(NOD1)
      NRN = NNOD(NOD1)
      IF (NRN.EQ.TMN3) THEN
          TOTV = CUMV(NOD1)
          GOTO 134
      ENDIF
      RRN = NNOD(RNOD)
      IF (CUMV(NOD1).GT.0) GOTO 110

C    Check if the node to be normalized is the extreme root..

      IF (RNOD.NE.TMN1) GOTO 96
      IF (VINE(NOD1).EQ.TMN1) THEN
          BASE(NRN) = TMN1
      ELSE
          NOD2 = VINE(NOD1)
          ROOT(NOD2) = TMN1
          VNOD = VINE(NOD2)
92    IF (VNOD.LE.TMN1) GOTO 94
          ROOT(VNOD) = NOD2
          VNOD = VINE(VNOD)
          GOTO 92
94    VINE(NOD1) = TMN1
          BASE(NRN) = NOD2
      ENDIF
      NNOD(NOD1) = TMN3
      TOTV = CUMV(NOD1)
      GOTO 134

C    A strong M-edge is found.....

96    NOD2 = RNOD
      NOD3 = NOD1

```

```

98   IF (NNOD(NOD2).EQ.NNOD(NOD3)) GOTO 100
      CUMV(NOD2) = CUMV(NOD2)-CUMV(NOD1)
      NOD3 = NOD2
      NOD2 = ROOT(NOD2)
      IF (NOD2.EQ.TMN1) GOTO 100
      GOTO 98

100  IF (RRN.EQ.NRN) THEN
      NOD2 = RNOD
      ELSEIF (BNCH(RNOD).EQ.NOD1) THEN
        IF (VINE(NOD1).NE.TMN1) THEN
          VNOD = VINE(NOD1)
          BNCH(RNOD) = VNOD
          TWIG(VNOD) = TWIG(NOD1)
        ELSE
          BNCH(RNOD) = TWIG(NOD1)
        ENDIF
        GOTO 108
      ELSE
102    NOD2 = BNCH(RNOD)
        IF (NNOD(NOD2).EQ.NRN) GOTO 106
        IF (TWIG(NOD2).EQ.NOD1) GOTO 104
        NOD2 = TWIG(NOD2)
        GOTO 102
104    IF (VINE(NOD1).NE.TMN1) THEN
          VNOD = VINE(NOD1)
          TWIG(NOD2) = VNOD
          TWIG(VNOD) = TWIG(NOD1)
        ELSE
          TWIG(NOD2) = TWIG(NOD1)
        ENDIF
        GOTO 108
      ENDIF
106  IF (VINE(NOD2).EQ.NOD1) GOTO 107
      NOD2 = VINE(NOD2)
      GOTO 106
107  VINE(NOD2) = VINE(NOD1)

C   Establish an arc from the dummy to the normalized tree..

108  ROOT(NOD1) = TMN1
      TWIG(NOD1) = TMN1
      VINE(NOD1) = TMN1
      NNOD(NOD1) = TMN3

C   Evaluate the normalized tree as weak.....

      INDEX = -1

```

```

      IRN = ABS(RNUM(NOD1))
      RNUM(NOD1) = INDEX*IRN
      IF (BNCH(NOD1).EQ.TMN1) GOTO 110
      CALL REVALM(NOD1,INDEX)

```

```

C -----
C Normalize the tree if strong P-edges are found.
C -----

```

```

110   VAL3 = BVAL(NRN)
      TOTV = VAL3

```

```

      IF (RNOD.EQ.TMN1) GOTO 130
      IF (RRN.EQ.NRN) THEN
        NOD1 = RNOD
        GOTO 130
      ENDIF

```

```

112   NOD2 = BNCH(RNOD)
      IF (NOD2.EQ.TMN1) GOTO 116
      IF (NNOD(NOD2).EQ.NRN) GOTO 114
      NOD2 = TWIG(NOD2)
      GOTO 112

```

```

114   IF (NOD2.LE.TMN1) GOTO 116
      TOTV = TOTV+CUMV(NOD2)
      NOD2 = VINE(NOD2)
      GOTO 114

```

```

116   IF (TOTV.LE.0) GOTO 90

```

```

C   A strong P-edge is found.....

```

```

      SUMV(RNOD) = SUMV(RNOD)-VAL3
      NOD2 = RNOD
118   CUMV(NOD2) = CUMV(NOD2)-TOTV
      NOD3 = NOD2
      NOD2 = ROOT(NOD2)
      IF (NOD2.EQ.TMN1) GOTO 120
      IF (NNOD(NOD2).EQ.NNOD(NOD3)) GOTO 120
      GOTO 118

```

```

120   NOD1 = BNCH(RNOD)
      IF (NNOD(NOD1).EQ.NRN) THEN
        BNCH(RNOD) = TWIG(NOD1)
      ELSE

```

```

122   IF (NNOD(NOD1).EQ.NRN) GOTO 124
      NOD2 = NOD1
      NOD1 = TWIG(NOD1)

```

```

          GOTO 122
124      TWIG(NOD2) = TWIG(NOD1)
        ENDIF

C      Establish an arc from the dummy to the normalized tree..

          ROOT(NOD1) = TMN1
          TWIG(NOD1) = TMN1
          BASE(NRN) = NOD1
          VNOD = VINE(NOD1)
126     IF (VNOD.LE.TMN1) GOTO 128
          ROOT(VNOD) = NOD1
          VNOD = VINE(VNOD)
          GOTO 126

C      Reevaluate the normalized tree as strong.....

128     INDEX = 1
          IRN = ABS(RNUM(NOD1))
          RNUM(NOD1) = INDEX*IRN

          IF (BNCH(NOD1).LE.TMN1.AND.VINE(NOD1).LE.TMN1) GOTO 90
          CALL REVALM(NOD1,INDEX)
          GOTO 90

C      -----
C      Evaluate the whole tree at its extreme root.
C      -----

130     VNOD = NOD1
132     IF (VNOD.LE.TMN1) GOTO 134
          TOTV = TOTV+CUMV(VNOD)
          VNOD = VINE(VNOD)
          GOTO 132

134     IF (TOTV.GT.0) THEN
          INDEX = 1
        ELSE
          INDEX = -1
        ENDIF
          IRN = ABS(RNUM(NOD1))
          RNUM(NOD1) = INDEX*IRN

          IF (BNCH(NOD1).LE.TMN1.AND.VINE(NOD1).LE.TMN1) GOTO 140
          CALL REVALM(NOD1,INDEX)

140     RETURN
        END

```

```

C*****
C
C      SUBROUTINE : REVALM
C          To re-evaluate the ore-waste index of the
C          normalized tree and the successive tree.
C
C      Programmed by : P.Huttagosol, 1987
C
C      Variables used :
C          VNUM      - counter for VNOD, VINE(NODE)
C          STOR(VNUM)- storage for VNOD
C*****

```

```

      SUBROUTINE REVALM(NODE,INDEX)

```

```

      IMPLICIT INTEGER(A-Z)
      PARAMETER(MP=7000)
      INTEGER*2 ROOT(MP),BNCH(MP),TWIG(MP),VINE(MP)
      INTEGER*2 RNUM(MP),STOR(2000)

```

```

      COMMON/BCK5/ROOT
      COMMON/BCK6/BNCH
      COMMON/BCK7/TWIG
      COMMON/BCK11/RNUM
      COMMON/BCK12/VINE
      DATA TMN1/1/

```

```

C      Evaluate the ore-waste index of all successor nodes.....

```

```

      NOD1 = NODE
      VNUM = 0

```

```

      VNOD = VINE(NOD1)
10      IF (VNOD.LE.TMN1) GOTO 20
      IRN = ABS(RNUM(VNOD))
      RNUM(VNOD) = INDEX*IRN

```

```

      IF (BNCH(VNOD).LE.TMN1) GOTO 15
      VNUM = VNUM+1
      STOR(VNUM) = VNOD
15      VNOD = VINE(VNOD)
      GOTO 10

```

```

20      IF (BNCH(NOD1).LE.TMN1) GOTO 80
      NOD2 = BNCH(NOD1)

```

```
30     IRN = ABS(RNUM(NOD2))
      RNUM(NOD2) = INDEX*IRN

      VNOD = VINE(NOD2)
40     IF (VNOD.LE.TMN1) GOTO 50
      IRN = ABS(RNUM(VNOD))
      RNUM(VNOD) = INDEX*IRN

      IF (BNCH(VNOD).LE.TMN1) GOTO 45
      VNUM = VNUM+1
      STOR(VNUM) = VNOD
45     VNOD = VINE(VNOD)
      GOTO 40

50     IF (BNCH(NOD2).LE.TMN1) GOTO 60
      NOD2 = BNCH(NOD2)
      GOTO 30

60     IF (TWIG(NOD2).GT.TMN1) THEN
      NOD2 = TWIG(NOD2)
      ELSE
70     NOD2 = ROOT(NOD2)
      IF (NOD2.EQ.NOD1) GOTO 80
      IF (TWIG(NOD2).LE.TMN1) GOTO 70
      NOD2 = TWIG(NOD2)
      ENDIF
      GOTO 30

80     IF (VNUM.LE.0) GOTO 100
      NOD1 = STOR(VNUM)
      VNUM = VNUM-1
      GOTO 20

100    RETURN
      END
```

```

C*****
C
C   SUBROUTINE : REMOV3
C       Remove positive nodes with strong index and
C       update the intermediate pit surface
C       including all overlying negative nodes.
C
C   Programmed by : P.Huttagosol, 1987
C
C   Variable used :
C       PIT(I,J) - intermediate pit surface at (I,J)
C*****
C   SUBROUTINE REMOV3(K,KTOP,MN1,MN2,NOD,NXND,SMND)

C       IMPLICIT INTEGER(A-Z)
C       PARAMETER(MX=75,MZ=40,MP=7000)
C       INTEGER*2 RNUM(MP),NEXT(MP),PNP(MZ),SUMP(MZ)
C       INTEGER*2 CON(MX,MX),PIT(MX,MX)
C       REAL XDM,YDM,ZDM,TANI,TANJ

C       COMMON/BLK1/NX,NY,NZ,XDM,YDM,ZDM,IM,JM,TANI,TANJ
C       COMMON/BLK4/CON
C       COMMON/OPUT/PIT
C       COMMON/BCK11/RNUM
C       COMMON/BCK13/NEXT
C       COMMON/BLK15/SUMP
C       COMMON/BLK16/PNP
C       DATA TMN1,TMN2/1,2/

C   Initialize counters for removed nodes.....

C       MN1 = 0
C       MN2 = 0

C   Iterate through the level.....

C       DO 280 NK=KTOP,K
C           IF (SUMP(NK).EQ.0) GOTO 280

C           TMP1 = TMN2
C           PNOD = PNP(NK)
C           NUMP = SUMP(NK)

C           DO 250 NP=1,NUMP
C               IF (RNUM(PNOD).LE.0) GOTO 240

```

C Remove negative blocks enclosed within a strong cone....

```

      IRN = RNUM(PNOD)
      JID = MOD(IRN,NY)
      IF (JID.EQ.0) JID=NY
      IID = (IRN-JID)/NY+1

      CALL CONE(IID,JID,NK,MNI,MXI,MNJ,MXJ)

      DO 230 I=1,NX
        DO 220 J=1,NY
          MN2 = MN2+CON(I,J)-PIT(I,J)
          PIT(I,J) = CON(I,J)
220      CONTINUE
230      CONTINUE

```

C Remove a strong positive node at the cone base.....

```

      PIT(IID,JID) = NK
      SUMP(NK) = SUMP(NK)-1
      TMP2 = NEXT(PNOD)
      NEXT(PNOD) = NOD
      NOD = PNOD
      PNOD = TMP2
      GOTO 250

240      NEXT(TMP1) = PNOD
          TMP1 = PNOD

          PNOD = NEXT(PNOD)
250      CONTINUE

          IDIF = NUMP-SUMP(NK)
          MN1 = MN1+IDIF
          SMND = SMND+IDIF

          NEXT(TMP1) = TMN1
          PNP(NK) = NEXT(TMN2)
280      CONTINUE
          NXND = NEXT(NOD)

      RETURN
      END

```

```

C*****
C
C SUBROUTINE : PLACER
C   Generate the outer bound by using the placer
C   bounding method.
C
C Programmed by : P.Huttagosol, 1987
C
C Variables used :
C   BOT(I,J) - outer bound at column(I,J)
C   KTOP      - the most top level of interest
C   KBOT      - the most bottom level of interest
C*****
C SUBROUTINE PLACER(KTOP,KBOT)
C
C   IMPLICIT INTEGER(A-Z)
C   PARAMETER(MX=75)
C   REAL XDM,YDM,ZDM,TANI,TANJ
C   INTEGER*2 TOP(MX,MX),BOT(MX,MX),CON(MX,MX),KVAL
C
C   COMMON/BLK1/NX,NY,NZ,XDM,YDM,ZDM,IM,JM,TANI,TANJ
C   COMMON/BLK2/TOP
C   COMMON/BLK3/BOT
C   COMMON/BLK4/CON
C
C   Initialize the outer bound.....
C
C     DO 10 I=1,NX
C       DO 5 J=1,NY
C         BOT(I,J) = TOP(I,J)
C     5   CONTINUE
C   10   CONTINUE
C
C   Iterate form bottom to top level.....
C
C     DO 50 K=NZ,1,-1
C
C       DO 40 I=1,NX
C         I1 = (I-1)*NY
C
C         DO 30 J=1,NY
C           IF (K.LE.BOT(I,J)) GOTO 30
C           KRN = (I1+J-1)*NZ+K
C           READ(10,REC=KRN) KVAL

```

C Generate the cone with positive block on its base.....

```
      IF (KVAL.LE.0) GOTO 30
      CALL CONE(I,J,K,MNI,MXI,MNJ,MXJ)
```

C Update the outer bound.....

```
      DO 20 IX=MNI,MXI
        DO 15 JY=MNJ,MXJ
          IF (BOT(IX,JY).GE.CON(IX,JY)) GOTO 15
          BOT(IX,JY) = CON(IX,JY)
15      CONTINUE
20      CONTINUE
      BOT(I,J) = K
```

C Next block.....

```
30      CONTINUE
40      CONTINUE
```

C Next level.....

```
50      CONTINUE
```

C Find the most top and bottom level of interests.....

```
      KTOP = NZ
      KBOT = 1

      DO 70 I=1,NX
        DO 60 J=1,NY
          IF (KTOP.GT.TOP(I,J)) KTOP=TOP(I,J)
          IF (KBOT.LT.BOT(I,J)) KBOT=BOT(I,J)
60      CONTINUE
70      CONTINUE

      KTOP = KTOP+1

      RETURN
      END
```

```

C*****
C
C   SUBROUTINE : CONE
C       To determine surface of an inverted cone given
C       that the cone base is a positive block and the
C       cone angles respect maximum pit slope allowance.
C
C   Programmed by : P.Huttagosol, 1987
C
C   Variables used :
C       CON(I,J) - generated cone surface at column(I,J)
C       IID,JID,KID - ID of a cone base block
C       MNI,MXI - top cone perimeter along I-axis
C       MNJ,MXJ - top cone perimeter along J-axis
C
C   Subroutine called : none
C
C*****

      SUBROUTINE CONE(IID,JID,KID,MNI,MXI,MNJ,MXJ)

      PARAMETER (MX=75,ML=40)
      INTEGER*2 CON (MX,MX) , PIT (MX,MX) , SURF (ML,ML)

      COMMON/BLK1/NX,NY,NZ,XDM,YDM,ZDM,IMAX,JMAX,TANI,TANJ
      COMMON/BLK4/CON
      COMMON/OPUT/PIT
      COMMON/BLK10/SURF

C   Initialize the top cone boundary and cone surface.....

      MNI = NX
      MXI = 0
      MNJ = NY
      MXJ = 0

      DO 30 I=1,NX
        DO 20 J=1,NY
          CON(I,J) = PIT(I,J)
20      CONTINUE
30      CONTINUE

C   Determine the cone surface.....

      DO 60 I=1,NX
        II = IABS(IID-I)+1
        IF (II.GT.IMAX) GOTO 60

```

```

      DO 50 J=1,NY
        JJ = IABS(JID-J)+1
        IF (JJ.GT.JMAX) GOTO 50

        KK = KID-SURF(II,JJ)
        IF (KK.LT.PIT(I,J)) GOTO 50
        CON(I,J) = KK

        IF (MNI.GT.I) MNI=I
        IF (MXI.LT.I) MXI=I
        IF (MNJ.GT.J) MNJ=J
        IF (MXJ.LT.J) MXJ=J
50      CONTINUE

60      CONTINUE

      RETURN
      END

```

```

C*****
C
C   SUBROUTINE : INITIA
C       Initialize the counters for nodes and arcs
C       added and removed by the algorithm.
C
C*****
      SUBROUTINE INITIA(NOD,NXND,SMND)

      INTEGER SMND,SMA1,SMA2,SMA3,SMM1,SMM2,SMM3

      COMMON/BLK14/SMA1,SMA2,SMA3,SMM1,SMM2,SMM3

      NOD = 3
      NXND = 1
      SMND = 0
      SMA1 = 0
      SMA2 = 0
      SMA3 = 0
      SMM1 = 0
      SMM2 = 0
      SMM3 = 0

      RETURN
      END

```

```

C*****
C
C   SUBROUTINE RECORD
C       Rearrange the block identification number such
C       that the total number of blocks is reduced
C       from the total number of blocks in original
C       model to only the total number of blocks in
C       between the top surface TOP(I,J) and the outer
C       bound BOT(I,J). It is called only from the
C       MGRAPH optimizing routine.
C
C   Programmed by : P.Huttagosol, 1987
C
C   Variables used :
C       BID(I,J) - the starting block identification
C                   number for column(I,J)
C       BVAL(N)  - block value for block N
C       KRN      - old record number
C       NEWREC   - new record number
C*****
C   SUBROUTINE RECORD

C       PARAMETER (MX=75,MT=75000)
C       INTEGER BID (MX,MX)
C       INTEGER*2 TOP (MX,MX) , BOT (MX,MX) , BVAL (MT) , KVAL

C       COMMON/BLK1/NX,NY,NZ,XDM,YDM,ZDM,IM,JM,TANI,TANJ
C       COMMON/BLK2/TOP
C       COMMON/BLK3/BOT
C       COMMON/TREC/BID
C       COMMON/BCK19/BVAL

C   Initialize variable and store starting block number for
C   each column.....

C       NEWREC = 3

C       DO 80 I=1,NX
C           I1 = (I-1)*NY
C           DO 60 J=1,NY
C               J1 = (I1+J-1)*NZ
C               BID(I,J) = NEWREC

C   Read block between the original surface and outer bound.

C       KT = TOP(I,J)+1
C       KB = BOT(I,J)

```

```

          DO 40 K=KT,KB
             KRN = J1+K
             READ(10,REC=KRN) KVAL

C   Store block value in the arrays of new block number.....

             NEWREC = NEWREC+1
             BVAL(NEWREC) = KVAL

C   Next block.....

40          CONTINUE
60          CONTINUE
80          CONTINUE

          RETURN
          END

```

```

C*****
C
C   SUBROUTINE : FINPIT
C       Generate the final pit according to blocks
C       removed by the MAXFLO and GRAPHT routines.
C
C*****
          SUBROUTINE FINPIT(KTOP,KBOT)

          IMPLICIT INTEGER(A-Z)
          PARAMETER(MX=75,MZ=40,MD=50000)
          REAL XDM,YDM,ZDM,TANI,TANJ
          INTEGER*2 PNP(MZ),SUMP(MZ),NNP(MZ),SUMN(MZ)
          INTEGER*2 PIT(MX,MX),BOT(MX,MX),INUM(MD),JNUM(MD)
          DIMENSION NEXT(MD)

          COMMON/BLK1/NX,NY,NZ,XDM,YDM,ZDM,IM,JM,TANI,TANJ
          COMMON/OPUT/PIT
          COMMON/BLK3/BOT
          COMMON/BLK11/INUM
          COMMON/BLK12/JNUM
          COMMON/BLK13/NEXT
          COMMON/BLK15/SUMP
          COMMON/BLK16/PNP
          COMMON/BLK17/SUMN
          COMMON/BLK18/NNP

```

```

C   initialize the final pit surface.....
      DO 310 I=1,NX
        DO 300 J=1,NY
          PIT(I,J) = BOT(I,J)
300   CONTINUE
310   CONTINUE

C   Iterate through the level.....
      DO 400 K=KBOT,KTOP,-1
        K1=K-1

C   Relocate the surface level down to position of removed
C   positive node.....
          IF (SUMP(K).EQ.0) GOTO 330

          PNOD = PNP(K)
          NUMP = SUMP(K)

          DO 320 NP=1,NUMP
            I1 = IABS(INUM(PNOD))
            J1 = JNUM(PNOD)
            PIT(I1,J1) = K1
            PNOD = NEXT(PNOD)
320   CONTINUE

C   Relocate the surface level to position of removed
C   negative block.....
330   IF (SUMN(K).EQ.0) GOTO 400

          NNOD = NNP(K)
          NUMN = SUMN(K)

          DO 340 NN=1,NUMN
            I1 = IABS(INUM(NNOD))
            J1 = JNUM(NNOD)
            PIT(I1,J1) = K1
            NNOD = NEXT(NNOD)
340   CONTINUE

400   CONTINUE

      RETURN
      END

```

```

C*****
C
C SUBROUTINE INFORM
C     Show the results on screen after ending each
C     level.
C
C*****
      SUBROUTINE INFORM(K,IOPT,AD1,AD2,AD3,MN1,MN2,MN3)
      INTEGER AD1,AD2,AD3,SMA1,SMA2,SMA3,SMM1,SMM2,SMM3
      COMMON/BLK14/SMA1,SMA2,SMA3,SMM1,SMM2,SMM3
C Calculate sum of nodes added and removed.....
      SMA1 = SMA1+AD1
      SMA2 = SMA2+AD2
      SMA3 = SMA3+AD3
      SMM1 = SMM1+MN1
      SMM2 = SMM2+MN2
      SMM3 = SMM3+MN3
C Write information on screen.....
      WRITE(*,292) K
292  FORMAT(1X,///,T10,'*****END OF LEVEL ',I2,'*****')
      WRITE(*,294)
294  FORMAT(/,T22,'ADDED',T32,'MINED',T42,'CUM ADDED',T52,
& 'CUM MINED')
      WRITE(*,296) 'Positive nodes',AD1,MN1,SMA1,SMM1
296  FORMAT(1X,T5,A14,T20,I6,T30,I6,T40,I6,T50,I6)
      WRITE(*,296) 'Negative nodes',AD2,MN2,SMA2,SMM2
      IF (IOPT.EQ.1) THEN
          WRITE(*,296) 'Directed arcs',AD3,MN3,SMA3,SMM3
      ENDIF
      RETURN
      END

```

```

C*****
C
C   SUBROUTINE OUTPUT
C       Output the final pit surface to the desired
C       output file.
C
C   Variables used :
C       PIT(I,J) - the final pit surface
C
C*****
      SUBROUTINE OUTPUT(OUTFIL)

      PARAMETER(MX=75)
      INTEGER*2 PIT(MX,MX)
      CHARACTER*10 OUTFIL

      COMMON/BLK1/NX,NY,NZ,XDM,YDM,ZDM,IMAX,JMAX,TANI,TANJ
      COMMON/OPUT/PIT

      OPEN(UNIT=20,FILE=OUTFIL,ACCESS='SEQUENTIAL',
& STATUS='NEW')

      DO 90 I=1,NX
          WRITE(20,80) (PIT(I,J),J=1,NY)
          FORMAT(20(1X,I2))
80      CONTINUE
90

      CLOSE(UNIT=20)

      RETURN
      END

```

APPENDIX B
PROGRAM DESCRIPTIONS

The program SAMPIT.FOR as given in Appendix A includes the following routines:

MAIN PROGRAM The main program SAMPIT.FOR is the main subroutine driver. It inputs the true pit optimizing techniques selected by a user. The three optimizing routines are :

MAXFLO GRAPHT MGRAPH

SUBROUTINE BLOCKS This subroutine interactively obtains the information about block model (NX,NY,NZ), block dimensions (XDM,YDM,ZDM), and slope constraints in two main directions (ANGI,ANGJ).

SUBROUTINE OBTAIN This subroutine interactively obtains the necessary file names and reads in the original surface file.

A two-dimensional array TOP(I,J) defines a surface for a block model, where I and J indices correspond to row and column number on X, and Y axis of the block model respectively. A value TOP(I,J), corresponding to a level-index in the block model, must be an integer. The original surface can be read in as:

```

      DO ii I=1,NX
          READ(u,*) (TOP(I,J),J =1,NY)
      ii CONTINUE

```

After assigning values to an array TOP(I,J), an array PIT(I,J) defining the intermediate pit surface is

initialized as;

$$\text{PIT}(I,J) = \text{TOP}(I,J)$$

The data file containing an economic block value must be defined such that each record represents a mining value of a rectangular block of material or air. If I, J, and K indices are used to represent the location of a block on each axis X, Y, and Z respectively (where I=1 to NX, J=1 to NY, and K=1 to NZ; NZ, NY are numbers of blocks on each horizontal axis; and NZ is a number of levels, e.g., K=1 is the top level and K=NZ is the bottom level), the block value VAL of each block, which must be an integer, can be read in as:

$$W = ((I-1)*NY) + (J-1)*NZ + K$$

READ(u,REC=W) VAL

SUBROUTINE MODELS This subroutine generates a cone template corresponding to given slope angles in two main directions (X and Y). An interger SURF(IX,JY) is used to recognize the cone template in the first quadrant where IX and JY represent a relative rectilinear distance from a cone-based block. This array is employed in the subroutine CONE to generate an inverted cone surface in all four quadrants.

SUBROUTINE MAXFLO This is the main subroutine that calculates ultimate pit limit using the maximum network flow method developed by Johnson (1968). This subroutine is also

modified from the original code written by Barnes (1982).

This subroutine calls the following routines.

INITIA PLACER ADD1 AADARC ALLOCA
LISTS LABELS REMOVI INFORM FINPIT

Due to the dimensioned arrays of this subroutine and some of its main subroutines called, the total number of nodes, including positive and negative nodes, added to the network at each iteration cannot exceed 50,000. Also, the total number of arcs cannot exceed 250,000.

SUBROUTINE ADD1 This subroutine adds nodes to the maximum flow optimizing routine. It also classifies each node into positive or negative category.

SUBROUTINE ADDARC This subroutine connects new added positive node to its overlying negative node by placing arcs linking to the negative nodes. It is called only by the MAXFLO subroutine.

SUBROUTINE ALLOCA This routine initializes flow for each new arc added to the MAXFLO routine. The amount of flow depends on arc capacity.

SUBROUTINE LISTS This routine searches for nodes to be scanned by the MAXFLO routine and put them in the scan list. The array BOX4(N) is used to store nodes to be scanned after node N in the first scanning process. The array BOX3(N) is also used as a re-scan list of nodes after breakthrough and

flow augmentation.

SUBROUTINE LABELS This subroutine performs the scanning and labeling processes. It is an application of the labeling algorithm which is a part of the maximum network flow method. The flow augmentation step after breakthrough is also carried out by this subroutine.

SUBROUTINE REMOVE1 After ending an iteration at each level, this subroutine removes nodes with access supplies from the maximum flow network. The arcs connecting to and from those removed nodes are also deleted.

SUBROUTINE GRAPHT This is the main subroutine that calculates an ultimate pit limit by using the graph method developed by Lerchs and Grossmann (1968). The subroutine is modified from the original code written by Dagdelen (1985). This subroutine calls the following important routines:

INITIA PLACER ADD2 TFORM1 EVAL1
REMOVE2 INFORM FINPIT

Due to the dimensioned arrays of this subroutine and some of its subroutine called, the total number of nodes, including positive and negative nodes, added to the network at each iteration cannot exceed 50,000 unless the dimension is changed.

SUBROUTINE ADD2 This subroutine adds nodes to the graph optimizing routine. It also classifies each node into

positive or negative category.

SUBROUTINE TFORM1 This subroutine is an application of the transformation and connection step of the original graph algorithm as outlined in Section 3.5.1.

SUBROUTINE EVAL1 This subroutine performs the evaluation and normalization process of the graph algorithm as given in Section 3.5.1.

SUBROUTINE REMOV2 This subroutine removes strong nodes from the tree network of the GRAPHT routine at the end of each level.

SUBROUTINE MGRAPH This main subroutine calculates an ultimate pit limit using the modified-tree graph algorithm developed in this thesis. The important routines called by this subroutine includes:

INITIA	PLACER	RECORD	ADD3	STATUS
TFORM2	CONNEC	EVAL2	REMOV3	INFORM

The arrays for storing of positive node's properties at each iteration of the modified-tree optimizing routines are dimensioned at 7,000. These arrays contain the relationship between positive nodes in the modified-tree as described in the program code. There are some arrays that memorized the negative node's properties, but these arrays are dimensioned at 75,000 which is the maximum number of blocks between the original surface and the outer bound. So, the total number

of blocks between the original surface and the outer bound cannot exceed 75,000 unless the dimension is changed.

SUBROUTINE ADD3 This subroutine adds nodes to the modified-tree graph optimizing routine. It also classifies each node into positive or negative category.

SUBROUTINE STATUS Define the status of the weak tree containing an overlying negative node to be connected to a strong positive node. This weak tree may contain only one node which is that negative node, or this negative node may be already supported by some other positive nodes in the tree.

SUBROUTINE TFORM2 This subroutine is an application of the transformation step of the modified-tree graph algorithm as given in Section 4.3.

SUBROUTINE CONNec This subroutine is an application of the connection step of the modified-tree graph algorithm as outlined in Section 4.3.

SUBROUTINE EVAL2 This subroutine performs the evaluation and normalization process of the modified-tree graph algorithm as outlined in Section 4.3.

SUBROUTINE REVAL This subroutine reevaluates the successive tree after the normalization step of the modified-tree graph algorithm. This subroutine is called by the EVAL2 subroutine.

SUBROUTINE REMOV3 This subroutine removes strong positive nodes from the modified-tree network of the MGRAPH routine at the end of each level. The routine also updates the arrays PIT(I,J) defining an intermediate pit surface to the new values resulting from the removal of positive nodes and their overlying waste blocks.

SUBROUTINE RECORD This subroutine rearranges the block numbers for the MGRAPH optimizing routine. The blocks in between the original surface TOP(I,J) and the outer bound are renumbered in a more natural one dimensional notation, so that the dimension of some arrays storing the negative node's properties (see Section 4.3) can be reduced.

SUBROUTINE INITIA This subroutine initializes variables that is used for keeping track of total numbers of nodes and arcs added or removed by the pit optimizing routines.

SUBROUTINE PLACER This subroutine generates the outer bound, BOT(I,J), using the bounding method as explained in Section 5.3. This routine is called by all of the three optimizing routines.

SUBROUTINE CONE This subroutine generates a cone surface CON(I,J) given that the inverted cone has its base located on a particular positive block and the cone template SURF(IX,JY) is utilized. The parameters I and J of CON(I,J) are the same indices as used in the block model.

SUBROUTINE FINPIT This subroutine generates the final pit surface PIT(I,J) resulting from the removal of positive and negative nodes by the MAXFLOW and GRAPHT subroutines.

SUBROUTINE INFORM This subroutine outputs information about number of blocks added and mined after the execution of each level on the screen.

SUBROUTINE OUTPUT This subroutine writes the output of final surface, PIT(I,J), to the desired output file.

APPENDIX C

BLOCK GRADE DISTRIBUTION BY LEVEL

BLOCK GRADE DISTRIBUTION BY LEVEL

LEVEL	NO. OF BLOCKS					
	0.0-0.2	0.2-0.4	0.4-0.6	[%] Cu 0.6-0.8	0.8-1.0	> 1.0
1	2474	0	0	0	0	0
2	5512	78	35	0	0	0
3	5471	61	53	40	0	0
4	5453	82	54	36	0	0
5	5365	183	77	0	0	0
6	5066	346	140	37	36	0
7	4816	405	241	115	13	35
8	4818	613	194	0	0	0
9	4742	622	261	0	0	0
10	4749	583	268	25	0	0
11	4648	634	343	0	0	0
12	4595	685	344	1	0	0
13	4552	651	417	5	0	0
14	4527	577	436	84	1	0
15	4486	584	303	90	22	140
16	4486	722	254	47	39	77
17	4463	792	188	51	42	89
18	4473	559	360	66	57	110
19	4460	441	410	181	79	54
20	4461	392	555	170	30	17

LEVEL	NO. OF BLOCKS					
	% Cu					
	0.0-0.2	0.2-0.4	0.4-0.6	0.6-0.8	0.8-1.0	> 1.0
21	4461	427	465	192	50	30
22	4461	159	793	124	35	53
23	4461	375	617	131	15	26
24	4466	170	861	91	25	12
25	4460	296	675	185	9	0
26	4460	182	701	235	47	0
27	4460	219	515	366	49	16
28	4460	98	478	507	40	42
29	4460	229	394	355	142	45
30	4460	330	305	334	107	89
31	4531	282	353	258	95	106
32	4461	109	397	375	144	139
33	4497	105	226	328	236	233
34	4505	112	221	296	187	304
35	4542	265	385	226	118	89
36	4544	449	241	149	84	158
37	4601	373	269	194	54	134
38	4706	262	266	218	77	96
39	4722	402	278	109	56	58
40	4764	463	128	145	49	76

APPENDIX D

SUMMATION OF ORE RESERVES

Each true optimization algorithm generates the same optimum pit limit for each case study. Ore reserves enclosed within optimum pit limits are summarized as shown below :

***** CASE I *****

% Cu	WASTE (TONS)	AVE GRADE	ORE (TONS)	AVE GRADE
0.00-0.19	6288334	.07	0	.00
0.20-0.39	2410226	.27	3044496	.36
0.40-0.59	0	.00	8317998	.48
0.60-0.79	0	.00	3624400	.68
0.80-0.99	0	.00	380562	.86
1.00-	0	.00	652392	1.10
TOTAL	8698560	.12	16019848	.53

***** CASE II *****

% Cu	WASTE (TONS)	AVE GRADE	ORE (TONS)	AVE GRADE
0.00-0.19	31713500	.03	0	.00
0.20-0.39	7683728	.31	7430020	.38
0.40-0.59	0	.00	26439998	.46
0.60-0.79	0	.00	2156518	.64
0.80-0.99	0	.00	0	.00
1.00-	0	.00	0	.00
TOTAL	39397228	.09	36026536	.45

***** CASE III *****

% Cu	WASTE (TONS)	AVE GRADE	ORE (TONS)	AVE GRADE
0.00-0.19	175149130	.05	0	.00
0.20-0.39	81005340	.28	24392212	.37
0.40-0.59	0	.00	87656114	.49
0.60-0.79	0	.00	38110566	.69
0.80-0.99	0	.00	10402028	.89
1.00-	0	.00	12631034	1.32
TOTAL	256154470	.12	173191954	.60

***** CASE IV *****

% Cu	WASTE (TONS)	AVE GRADE	ORE (TONS)	AVE GRADE
0.00-0.19	231744136	.00	0	.00
0.20-0.39	52263848	.30	20622836	.37
0.40-0.59	0	.00	75188178	.49
0.60-0.79	0	.00	45413732	.68
0.80-0.99	0	.00	14406990	.88
1.00-	0	.00	13935818	1.24
TOTAL	284007984	.06	169567554	.62