INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
Algorithms for optimum design and planning of open pit mines

Zhao, Yixian, Ph.D.
The University of Arizona, 1992
ALGORITHMS
FOR OPTIMUM DESIGN AND PLANNING
OF OPEN PIT MINES

by
Yixian Zhao

A Dissertation Submitted to the Faculty of the
DEPARTMENT OF MINING AND GEOLOGICAL ENGINEERING
In partial Fulfillment of the Requirements
For the Degree of
DOCTOR OF PHILOSOPHY
WITH A MAJOR IN MINING ENGINEERING
In the Graduate College
THE UNIVERSITY OF ARIZONA

1992
As members of the Final Examination Committee, we certify that we have read the dissertation prepared by Yixian Zhao entitled Algorithms for Optimum Design and Planning of Open Pit Mines and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

Young C. Kim  
4-28-92  
Date

Charles E. Glass  
4-28-92  
Date

Satya Harpalani  
4-28-92  
Date

Donald E. Myers  
4-28-92  
Date

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copy of the dissertation to the Graduate College.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

Young C. Kim  
Dissertation Director  
5-6-92  
Date
STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: [Signature]
ACKNOWLEDGEMENTS

I would like to thank Dr. Young C. Kim for his guidance and advice in preparing this dissertation. I am especially grateful to the time that Dr. Kim spent in the discussions of dissertation subjects as well as in the evaluation and English corrections of this dissertation.

I also would like to thank Dr. Charles E. Glass, Dr. John L. Denny, Dr. Satya Harpalani and Dr. Donald E. Myers for being my committee members. I appreciate the comments on the dissertation subjects and the corrections on my English from my committee members.

I thank the Ministry of Education of the People’s Republic of China, for providing a scholarship that enabled me to study in the United States.
# TABLE OF CONTENTS

**LIST OF ILLUSTRATIONS**

**LIST OF TABLES**

**ABSTRACT**

## 1 INTRODUCTION

1.1 Overview .............................................. 16
1.2 Statement of Ultimate Pit Limit Problems .................. 19
1.3 Statement of Long Range Mine Planning Problems ............ 21
   1.3.1 Proposed Mathematical Optimization Approaches ........ 22
   1.3.1 Popular “Pushback” Mine Sequence .................... 23
1.4 Statement of Grade Control Problems ........................ 24
1.5 Scope of Work ......................................... 26

## 2 REVIEW OF PREVIOUS WORK FOR ULTIMATE PIT LIMIT DESIGN

2.1 Overview .............................................. 28
2.2 Lerchs-Grossmann’n 3-D Graph Algorithm .................... 29
2.3 Maximum Network Flow Algorithm .......................... 33
2.4 Linear Programming Model ................................ 36
2.5 2-D Dynamic Programming Model .......................... 38
2.6 Family of Moving Cone Algorithms ........................ 43
2.7 Parametric Function Approach ............................ 48
2.8 Other Algorithms ...................................... 49
2.9 Conclusion ............................................ 49
# TABLE OF CONTENTS - (Continued)

## 3 A NEW GRAPH THEORY ALGORITHM FOR OPTIMUM ULTIMATE PIT LIMIT DESIGN

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Introduction</td>
<td>50</td>
</tr>
<tr>
<td>3.2 Definitions</td>
<td>51</td>
</tr>
<tr>
<td>3.3 Rules of Arc Generation</td>
<td>53</td>
</tr>
<tr>
<td>3.3.1 Computer Representation of an Arc</td>
<td>53</td>
</tr>
<tr>
<td>3.3.2 Rules of Arc Generation</td>
<td>55</td>
</tr>
<tr>
<td>3.4 Properties of the Tree</td>
<td>60</td>
</tr>
<tr>
<td>3.5 Joint Support and Re-allocation</td>
<td>67</td>
</tr>
<tr>
<td>3.6 The Specific Steps of the Algorithm</td>
<td>71</td>
</tr>
<tr>
<td>3.7 Optimality and Convergence</td>
<td>72</td>
</tr>
<tr>
<td>3.7.1 Optimality Theorem</td>
<td>72</td>
</tr>
<tr>
<td>3.7.2 Convergence Theorem</td>
<td>81</td>
</tr>
<tr>
<td>3.8 Programming Experience</td>
<td>83</td>
</tr>
<tr>
<td>3.8.1 Shell Cone Template</td>
<td>83</td>
</tr>
<tr>
<td>3.8.2 Model Reduction</td>
<td>85</td>
</tr>
<tr>
<td>3.8.3 Search Procedure</td>
<td>88</td>
</tr>
<tr>
<td>3.8.4 Optimization by Layers</td>
<td>89</td>
</tr>
<tr>
<td>3.9 Case Studies</td>
<td>92</td>
</tr>
<tr>
<td>3.10 Program Validation</td>
<td>93</td>
</tr>
<tr>
<td>3.10 Comparison Between the New Graph Algorithm and Lerchs-Grossmann</td>
<td></td>
</tr>
<tr>
<td>Graph Algorithm</td>
<td>94</td>
</tr>
<tr>
<td>3.11 Summary Remarks</td>
<td>97</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS - (Continued)

### 4 INVESTIGATION OF COMPUTERIZED TECHNIQUES IN MINE PRODUCTION SCHEDULING

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Overview</td>
<td>98</td>
</tr>
<tr>
<td>4.2 Investigation of the Approach Combining the Lagrangian Relaxation</td>
<td></td>
</tr>
<tr>
<td>with the Ultimate Pit Limit Algorithm</td>
<td>105</td>
</tr>
<tr>
<td>4.2.1 Zero-one Programming Model for Mine Production Scheduling</td>
<td>106</td>
</tr>
<tr>
<td>4.2.2 Everett's Generalized Lagrange Multiplier Method</td>
<td>110</td>
</tr>
<tr>
<td>4.2.3 Application of the Generalized Lagrange Multiplier Method</td>
<td>114</td>
</tr>
<tr>
<td>to Mine Production Scheduling</td>
<td></td>
</tr>
<tr>
<td>4.2.4 General Condition of Non-Convergence</td>
<td>122</td>
</tr>
<tr>
<td>4.2.5 Non-Convergence due to the Inability to Detect the Redundant</td>
<td></td>
</tr>
<tr>
<td>Optimal Solutions</td>
<td>123</td>
</tr>
<tr>
<td>4.2.6 Non-convergence due to the Inability to Handle the Required</td>
<td></td>
</tr>
<tr>
<td>Advanced Stripping</td>
<td>129</td>
</tr>
<tr>
<td>4.3 Pushback Approach</td>
<td>130</td>
</tr>
<tr>
<td>4.3.1 Pushback Generation</td>
<td>130</td>
</tr>
<tr>
<td>4.3.2 Mine Sequencing among Pushbacks</td>
<td>134</td>
</tr>
<tr>
<td>4.3.3 Optimizing Production Schedule</td>
<td>135</td>
</tr>
<tr>
<td>4.4 Summary</td>
<td>136</td>
</tr>
</tbody>
</table>

### 5 THE IMPLEMENTATION ASPECT OF BLAST HOLE KRIGING

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Introduction</td>
<td>137</td>
</tr>
<tr>
<td>5.2 Polygon Manipulation Algorithms</td>
<td>144</td>
</tr>
<tr>
<td>5.2.1 Point-in-Polygon Algorithms</td>
<td>145</td>
</tr>
<tr>
<td>5.2.2 Algorithm of Calculating the Area of the Polygon</td>
<td>152</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS - (Continued)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.3 Polygon Clipping Algorithm</td>
<td>160</td>
</tr>
<tr>
<td>5.3 Blast Hole Handling and Database Maintenance</td>
<td>166</td>
</tr>
<tr>
<td>5.3.1 Blast Hole Editor</td>
<td>166</td>
</tr>
<tr>
<td>5.3.2 Blast Hole Database</td>
<td>172</td>
</tr>
<tr>
<td>5.3.3 Rock Type Database</td>
<td>178</td>
</tr>
<tr>
<td>5.4 Interactive Graphics Planning</td>
<td>180</td>
</tr>
<tr>
<td>5.4.1 Kriging System</td>
<td>180</td>
</tr>
<tr>
<td>5.4.2 Interactive Ore-Waste Delineations</td>
<td>188</td>
</tr>
<tr>
<td>5.5 Summary and Future Enhancements</td>
<td>192</td>
</tr>
<tr>
<td>6 CONCLUSIONS AND RECOMMENDATIONS</td>
<td>193</td>
</tr>
<tr>
<td>6.1 Conclusions</td>
<td>193</td>
</tr>
<tr>
<td>6.2 Recommendations for Future Research</td>
<td>197</td>
</tr>
</tbody>
</table>

APPENDIX A

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pit Limits Corresponding to Different Penalty Parameters for a 2-D Cross Section Example</td>
<td>199</td>
</tr>
</tbody>
</table>

APPENDIX B

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pushback Contour Maps of a Gold Deposit</td>
<td>208</td>
</tr>
</tbody>
</table>

SELECTED BIBLIOGRAPHY

213
## LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 A Typical 3-D Mine Model</td>
<td>19</td>
</tr>
<tr>
<td>1.2 Circular Nature of Mine Evaluation Process</td>
<td>21</td>
</tr>
<tr>
<td>2.1 An Un-normalized Tree</td>
<td>30</td>
</tr>
<tr>
<td>2.2 A Normalized Tree</td>
<td>31</td>
</tr>
<tr>
<td>2.3 Vertex Y of the Weak Branch Is a Restricting Vertex to Vertex X of the Strong Branch</td>
<td>32</td>
</tr>
<tr>
<td>2.4 The Tree Obtained by Combining the Weak Branch and the Strong Branch</td>
<td>32</td>
</tr>
<tr>
<td>2.5 Network of Pit Limit Design</td>
<td>35</td>
</tr>
<tr>
<td>2.6 A 2-D Cross Section Example</td>
<td>37</td>
</tr>
<tr>
<td>2.7 $m_{ij}$ Tableau</td>
<td>39</td>
</tr>
<tr>
<td>2.8 $M_{ij}$ Tableau</td>
<td>39</td>
</tr>
<tr>
<td>2.9 $P_{ij}$ Tableau</td>
<td>40</td>
</tr>
<tr>
<td>2.10 Optimal Contour</td>
<td>41</td>
</tr>
<tr>
<td>2.11 Inverted Cone Frustum</td>
<td>44</td>
</tr>
<tr>
<td>2.12 Approximation of the Pit Limit by Overlapping of the Cones</td>
<td>45</td>
</tr>
<tr>
<td>2.13 Example of Moving Cone Method Missing Shared Contribution and Over-mining at the Same Time</td>
<td>45</td>
</tr>
<tr>
<td>3.1 Process of Generating Arc</td>
<td>54</td>
</tr>
<tr>
<td>3.2 Process of Removing Arc</td>
<td>55</td>
</tr>
<tr>
<td>3.3 An Example of a Positive Arc</td>
<td>56</td>
</tr>
<tr>
<td>3.4 An Example of a Negative Arc</td>
<td>57</td>
</tr>
<tr>
<td>3.5 An Ore Vertex Supporting Several Waste Vertices</td>
<td>57</td>
</tr>
<tr>
<td>3.6 Several Ore Vertices Supporting One Waste Vertex</td>
<td>58</td>
</tr>
<tr>
<td>3.7 A 2-D Example of Arc Generation</td>
<td>59</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>3.8</td>
<td>Changing Non-root Vertex into Root Vertex by Re-arranging Arcs</td>
</tr>
<tr>
<td>3.9</td>
<td>An Example of Re-arranging Arcs</td>
</tr>
<tr>
<td>3.10</td>
<td>Different Supporting Patterns of Negative trees</td>
</tr>
<tr>
<td>3.11</td>
<td>Different Supporting Patterns of Positive trees</td>
</tr>
<tr>
<td>3.12</td>
<td>An Example of Re-allocation</td>
</tr>
<tr>
<td>3.13</td>
<td>Waste Vertex ( v_i ) of the Negative Tree is A Restricting Vertex to Ore</td>
</tr>
<tr>
<td>3.14</td>
<td>The Structure of the Chain of a Tree (Relationship between Ore Vertices and</td>
</tr>
<tr>
<td></td>
<td>Waste Vertices)</td>
</tr>
<tr>
<td>3.15</td>
<td>A Tree Intersecting the Border between ( Z ) and ( V - Z )</td>
</tr>
<tr>
<td>3.16</td>
<td>Relationship between Vertices of a Positive Tree with the Maximum Closure</td>
</tr>
<tr>
<td>3.17</td>
<td>Relationship between Vertices of a Negative Tree with The Maximum Closure</td>
</tr>
<tr>
<td>3.18</td>
<td>Shell Cone Template</td>
</tr>
<tr>
<td>3.19</td>
<td>Model Size Reduction</td>
</tr>
<tr>
<td>3.20</td>
<td>The Blocks Used for Optimization</td>
</tr>
<tr>
<td>3.21</td>
<td>Cone ( B ) is Complete within Cone ( A )</td>
</tr>
<tr>
<td>3.22</td>
<td>Pit Limit Optimization by Layers</td>
</tr>
<tr>
<td>3.23</td>
<td>Computation Time of Optimization by Layers</td>
</tr>
<tr>
<td>4.1</td>
<td>An Example of Existence of Redundant Optimal Solutions</td>
</tr>
<tr>
<td>4.2</td>
<td>Possible Pit Limits for Different Penalty Parameters</td>
</tr>
<tr>
<td>4.3</td>
<td>Cross Section View of the Next Best Mineralization</td>
</tr>
<tr>
<td>4.4</td>
<td>Sequence Access Concept between Pushbacks</td>
</tr>
<tr>
<td>4.5</td>
<td>Stripping Ratio of Different Pushbacks</td>
</tr>
</tbody>
</table>
### LIST OF ILLUSTRATIONS - (Continued)

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Simplified Flow Chart of BHK System</td>
<td>139</td>
</tr>
<tr>
<td>5.2 Hardware Requirements for BHK System</td>
<td>141</td>
</tr>
<tr>
<td>5.3 The Internal Angles Subtended by All the Boundary Segments</td>
<td>146</td>
</tr>
<tr>
<td>5.4 Vectors Formed by Three Points</td>
<td>146</td>
</tr>
<tr>
<td>5.5 Right Hand Rule</td>
<td>146</td>
</tr>
<tr>
<td>5.6 Line Intersection Method to Determine If a Point Is Inside or Outside the Polygon Boundary</td>
<td>149</td>
</tr>
<tr>
<td>5.7 Point in Polygon Test by Horizontal Line</td>
<td>150</td>
</tr>
<tr>
<td>5.8 Point in Polygon Test by Swathing Method</td>
<td>151</td>
</tr>
<tr>
<td>5.9 Geometry of Region ( \mathcal{R} ) and its Perimeter ( C ) in Counterclockwise Direction</td>
<td>153</td>
</tr>
<tr>
<td>5.10 A Triangle</td>
<td>155</td>
</tr>
<tr>
<td>5.11 Examples of Counterclockwise and Clockwise Triangles</td>
<td>156</td>
</tr>
<tr>
<td>5.12 Decomposition of Polygons into Triangles</td>
<td>156</td>
</tr>
<tr>
<td>5.13 Point Inside Polygon and Component Triangles not Overlapping Each Other</td>
<td>157</td>
</tr>
<tr>
<td>5.14 Point Inside Polygon and Component Triangles Overlapping Each Other</td>
<td>158</td>
</tr>
<tr>
<td>5.15 Summing up the Positive and Negative Component Triangles (Point Inside the Polygon)</td>
<td>158</td>
</tr>
<tr>
<td>5.16 Summing up the Positive and Negative Component Triangles (Point Outside the Polygon)</td>
<td>159</td>
</tr>
<tr>
<td>5.17 Intersection, Union and Complement between Two Polygons</td>
<td>160</td>
</tr>
<tr>
<td>5.18 Three Configurations of Polygon Sets</td>
<td>162</td>
</tr>
<tr>
<td>5.19 One Polygon Inside Another</td>
<td>162</td>
</tr>
</tbody>
</table>
LIST OF ILLUSTRATIONS - (Continued)

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.20 Intersections between Polygon A and B</td>
<td>164</td>
</tr>
<tr>
<td>5.21 Union of Polygon A and B</td>
<td>164</td>
</tr>
<tr>
<td>5.22 Complement of Polygon B with respect to Polygon A</td>
<td>165</td>
</tr>
<tr>
<td>5.23 Main Menu of the Blast Hole Editor</td>
<td>167</td>
</tr>
<tr>
<td>5.24 An Example of the Output File from Option 1</td>
<td>168</td>
</tr>
<tr>
<td>5.25 Appending Window of the Blast Hole Editor</td>
<td>169</td>
</tr>
<tr>
<td>5.26 Full Screen Editing Window of the Blast Hole Editor</td>
<td>170</td>
</tr>
<tr>
<td>5.27 Graphics Display of the Blast Holes</td>
<td>171</td>
</tr>
<tr>
<td>5.28 Dividing Orebody by Regular Grids for Data Structure Purpose</td>
<td>172</td>
</tr>
<tr>
<td>5.29 Rock Type Polygons</td>
<td>179</td>
</tr>
<tr>
<td>5.30 Variogram Editor</td>
<td>182</td>
</tr>
<tr>
<td>5.31 Editing Panels for Setting Up Kriging Parameters</td>
<td>184</td>
</tr>
<tr>
<td>5.32 The Graphics Display of the Shot Boundary</td>
<td>185</td>
</tr>
<tr>
<td>5.33 Example Summary Output from Kriging</td>
<td>187</td>
</tr>
<tr>
<td>5.34 Interactive Ore-Waste Delineations</td>
<td>189</td>
</tr>
<tr>
<td>5.35 An Example Output from the Graphics Ore-Waste Delinement</td>
<td>190</td>
</tr>
<tr>
<td>5.36 An Example Plot of Ore-Waste Delineation</td>
<td>190</td>
</tr>
<tr>
<td>5.37 A Numerical Map of the Estimated Block Grades</td>
<td>191</td>
</tr>
<tr>
<td>A.1 Pit Limits Corresponding to Different Penalty Parameters for a 2-D</td>
<td></td>
</tr>
<tr>
<td>Cross Section Example</td>
<td>199</td>
</tr>
<tr>
<td>A.2 Pushback Contour Maps of a Gold Deposit</td>
<td>208</td>
</tr>
</tbody>
</table>
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Computation Time of Optimization by Layers</td>
<td>91</td>
</tr>
<tr>
<td>3.2 Model Descriptions and Processing Time for Each Case</td>
<td>93</td>
</tr>
<tr>
<td>4.1 Summary of Blocks of Two Alternatives</td>
<td>126</td>
</tr>
<tr>
<td>4.2 Existence of Pit Limits for Different Penalty Parameters</td>
<td>127</td>
</tr>
<tr>
<td>4.3 Bench by Bench Ore and Waste Inventories for Each Pushback</td>
<td>133</td>
</tr>
</tbody>
</table>
ABSTRACT

A graph theory oriented algorithm for optimal ultimate pit limit design is developed. Mathematical proofs of optimality and convergency are given. The algorithm works on a 3-D block mine model and formulates the model into a directed graph consisting many trees. The vertices in the graph are identified with the blocks in the model and the imposed arcs in the graph represent pit slope constraints. The formation of each directed tree is based more on the ore-waste support concept than geometric constraints alone. The algorithm efficiently handles the joint support and re-allocation problems. The theoretical proof shows that the new algorithm is consistently faster than the well known Lerchs-Grossmann's (LG) algorithm, which is the only algorithm developed in the past one-quarter century capable of producing a true optimum pit limit. The case study results show that the new algorithm is able to generate the optimal ultimate pit limit for a model with 80 columns x 80 rows x 40 levels on an IBM PC AT 80286 microcomputer in 115 minutes. The indirect comparison was made between the results of the new algorithm and the results obtained by P. Huttagosol (1988,1989) using the LG algorithm. P. Huttagosol optimized a mine model, which was little smaller than the model optimized by the new algorithm, in 535 minutes of VAX8600 CPU time. The comparison between 535 minutes of VAX8600 CPU time for a smaller model with 115 minutes PC AT processing time for a bigger model clearly indicates that the new algorithm is significantly faster the LG algorithm.

This study also investigates both proposed mathematical optimization approaches and the popular trial and error "pushback" approach to long range mine planning. Both the theoretical analysis and numerical examples demonstrate it is
impossible to obtain the optimal solution to mine production scheduling by the approach combining the Lagrangian relaxation with the ultimate pit limit algorithm. The non-convergence due to redundant optimal solutions and the non-convergence due to the requirement of advanced stripping are identified with the proposed approach. The investigation clarifies the long-time misunderstood concept and proves the impossibility of such a research direction itself.

Finally, some problem solving techniques which play important roles in the computerized mine planning and grade control are developed and discussed. Specifically, they are: 1) point-in-polygon algorithm, 2) polygon area algorithm, 3) polygon clipping algorithm, 4) blast hole data collection, validation and database maintenance, and 5) the interactive graphics ore-waste delineation.
CHAPTER 1

INTRODUCTION

1.1 Overview

Some ore bodies are mined through open pit methods and some others are mined by underground methods. However, the investments on the deposits that can be mined by the open pit method are more attractive, than the ones that must be mined through underground methods to the investment companies, since open pit mining has many advantages over underground mining such as: lower production cost, higher productivity, higher safety, higher recovery, flexibility of operation, etc. The data from U.S. Bureau of Mines indicated that about 68 percent of world crude minerals, and 83 percent of U.S. overall minerals were mined from surface or open pit mines (Allsman, 1973). Since this dissertation only focuses on the problems of open pit mines, the terminology of mine design and planning throughout this dissertation only refers to open pit mine design and planning.

The primary objective of mine planning is to develop a schedule which tells when and where mining should take place over successive time periods until all the ore reserves are depleted. Since the net present value (NPV) analysis is the most widely used technique to evaluate mining venture, the fundamental objective of a mining company is to maximize the NPV produced by its mining projects. Therefore, the design of a mine project is optimal when the maximum NPV is realized.

In reality, there is a great difficulty in reaching the true optimum mine plan due to the nature of long production life, great uncertainty of ore reserve estimation and some other uncontrollable risks, such as: geological risks, engineering risks,
economic risks, and political risks. A mineral reserve must be mined in a certain time horizon. Therefore, it can not be turned into cash all at once. In fact it is very common for a mine to last more than 50 years. It is not hard to imagine the difficulties in the prediction of economic risks for 50 years into the future. One of the economic risks is that associated with mineral markets since prices fluctuate significantly and precipitously for mineral commodities. Grade distributions and tonnages are estimated based on the available samples. However, even geostatistical methods can not generate very reliable estimates without a "sufficient" number of samples. During long range mine planning, the number of samples based on exploratory diamond drill hole (DDH) assays may be far from "sufficient". Both the grade distributions and mineral market prices have great influence on the cash flow, which will directly affect the NPV.

In spite of the above difficulties, the best possible mine plan must be worked out at great efforts, since an investment company can not afford the cost of mistakes in mine planning. Mining ventures are extremely capital intensive. Usually, new mines may require financial commitments ranging from 500 million to as much as 10 billion dollars (Gentry and O'neil, 1984). The development of an optimal (or nearly so) mine plan is a very complex procedure even for a given set of parameters and constraints, due to the very large number of variables and constraints involved. However, this complexity creates an ideal environment for the application of sophisticated operations research and computer techniques.

Since the mid 60's, the increased application of computers in mine design and planning has made great benefits to mining industry. The mine pit limit design can be accomplished completely by computers using today's techniques and computing facilities. However, mine production scheduling is still carried out partially by computers and partially by humans. A typical open pit mine design and planning usually involves the following seven main tasks:

1. geological survey,
2. rock mechanics study,
3. mineralization modeling,
4. ultimate pit limit design,
5. long range mine planning,
6. short range mine planning,
7. grade control in daily production.

Usually, geological survey is not the main responsibility of mining engineers. This part is done by geological and surveying personnel. The rock mechanics study is itself an important branch of mining engineering. This part of the study is usually carried out by the people who specialize in this field.

A process transforming the known features of a real deposit into a mathematical model which can be manipulated by computers is called mineralization modelling. The commonly applied model is 3-D fixed block model (Kim 1978) which is developed by partitioning the orebody into fixed size blocks. The following basic tasks should be performed in the block model development: (1) initialize a 3-D mine model in the computer, (2) input surface topography, (3) input geology information, and (4) assign grades to each block in the mine model. Figure 1.1 shows a typical 3-D fixed block mine model.

The short range planning is mainly concerned with the monthly, weekly or daily control of operations. It translates the long term strategies proposed by the long range planning into a workable and effective production plan while complying with restrictions imposed by the long range plans, plant capacities, inventory restrictions, equipment availability and existing mining situations.
Figure 1.1 A Typical 3-D Mine Model (from Crawford)

The scope of this dissertation primarily encompasses the following topics: ultimate pit limit algorithms, approaches to long range mine planning and sequence development, and computerized grade control in daily production.

1.2 Statement of Ultimate Pit Limit Problems

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. The objective in ultimate
pit limit design is to maximize the difference between the total value obtained from the ore material to be extracted and the total cost of mining that ore material and removing its overlying waste material without considering the time value of money, while satisfying pit slope stability and operational constraints. The ultimate pit limit design gives a maximum possible mining boundary in three dimensions or the maximal mineable ore reserves. After the ultimate pit limit is determined, all the subsequent mine planning works are performed within the pit limit.

Since the ultimate pit limit plan is the basic foundation of all mine planning, the need for establishing the ultimate pit limit is undeniable although the most important economical information is controlled by the planning of the mining sequence. The ultimate pit plan serves as an aid in the evaluation of the economic potential and the initial feasibility study of a mineral deposit. It is also essential for estimating the capacity of processing plants, determining the locations for the processing plants, waste dumps, tailings ponds and other elements complementary to the mining operation. The ultimate pit limit plan may have an effect on property acquisition and the layout of a community near or surrounding the mine property.

Many variables should be considered in the ultimate pit limit design. Kim (1979) lists the main variables that enter into the design of the ultimate pit limit, those that remain after excluding the practical operating considerations. These variables are:

1. grade of ore deposit,
2. its spatial location, i.e., grade distributions,
3. recoverability of the resource,
4. market prices,
5. mining rate,
6. milling rate,
7. mining sequence,
8. ultimate pit wall slope,
9. mining cost,
10. processing and refining costs, etc.

These variables are highly interrelated. For instance, mining costs influence total ore reserves such that the higher mining costs will result in a smaller pit and vice versa. Total ore reserve in turn influences the mining rate, capital investment, and mining sequences, which may in turn alter the mining costs, leading to a circular process of mine evaluation as shown in figure 1.2. However, if some variables are fixed at some predicted values, that is, after some assumptions are made, there exists an optimum ultimate pit limit which maximizes the economic criteria.

![Diagram of Circular Nature of Mine Evaluation Process]

Figure 1.2    Circular Nature of Mine Evaluation Process

1.3 Statement of Long Range Mine Planning Problems

Long range mine planning refers to developing a multi-period mining sequence which will deplete the mineable ore reserves from the initial condition of the deposit to the ultimate pit limit. The time span of each planning period in a long range mine plan could range from one year to several years.
The main objective of the mining sequence development is to maximize the net present value of the cash flows throughout the mine life while maintaining a reasonable degree of operational continuity such as: to meet grade-tonnage requirements for each period, to meet precedence requirements among selected sequences, and to guarantee haul road accessibility. The long range planning provides the conceptual framework for the mining activities and deals with the future strategies. Mining sequence development has a great influence on the profitability of a mining venture. The present value of a mining operation can often be improved by starting with a higher cutoff grade in the early years of mining and tapering to a lower cutoff near the end of the mine life (Lane, 1964; Blackwell, 1970; Marek and Welhener, 1985).

1.3.1 Proposed Mathematical Optimization Approaches

It is always very attractive to obtain a true optimum solution by some mathematical optimization methods. In practice, the direct application of available standard operations research techniques, such as integer linear programming, to mine production scheduling problems becomes infeasible due to the large number of variables and constraints. Past attempts have focused on indirect approaches, utilizing the special structure of the mine production scheduling and making use of the structure in the solution algorithm. In production scheduling, certain practical constraints such as ore and waste tonnage requirements for each period must be reasonably taken into account in addition to the sequencing constraints. The only difference between the ultimate pit limit problem and the production scheduling problem is that latter must satisfy certain additional constraints. After these additional constraints are removed through Lagrangian relaxation method while keeping the sequencing constraints, the mathematical representation of the production scheduling problem then has the same form as that of the ultimate pit limit problem, which in turn can be solved by the available ultimate pit limit algorithm. Based on the above idea, many attempts have been made to develop
true mathematical optimization methods (Johnson, 1968; Davis-Williams, 1973; Dagdelen, 1985; and Elevli, et al., 1989) to solve the mine production scheduling problem. Unfortunately, the proposed approaches have not enjoyed application in practice due to some critical limitations of their own.

1.3.2 Popular “Pushback” Mine Sequence

The “pushback” mine sequence, which is a heuristic optimization approach, has been widely accepted by the mining industry, since it is the most practical method for long range mine planning. Mathieson (1982), Iles and Perry (1981) and Couzens (1978) give excellent descriptions of the pushback (or phase) sequencing. The main steps in the pushback sequence approach are the determination of the pushback (or phase) and the development of the mining sequence among the pushbacks.

A series of nested pits can be obtained by repeatedly applying the pit limit design algorithm by gradually decreasing the product prices while keeping the production costs constant. A cutoff grade is associated with each price in such a way that a higher product price will correspond to a lower cutoff grade and vice versa. In general, the highest cutoff grade then defines the inner most pit, whereas the lowest cutoff grade defines the outer most pit. If the horizontal distance between any two of those nested pits exceeds the minimum mining width, these nested pits can serve as a series of pushbacks. Since each outer pit increment is less profitable than all inner pits, these series of nested pits represent the “next best” mineralization with inner pit being the “best” and its outer pit being the “next best”. The boundary of the outer most pit serves as the ultimate pit limit. The mining sequences should be pushed from the inner to outer pits, generated by gradually decreasing the cutoff grades. Therefore, this method is called “PUSHBACK” approach.

Since mining may not be able to proceed following an inner pit to outer pit sequence due to practical constraints, a mining sequence among the pushbacks
must be developed subject to sequential requirements among benches, ore, waste tonnage requirements, grade blending requirement, and road access requirement.

Alternative mining sequences can be generated by repeatedly applying the above procedure. Corresponding to each cutoff grade strategy, a set of pushbacks can be generated. A particular mining schedule can in turn be worked out based on the pushbacks, and a net present value analysis can be carried out. Then, the particular sequence yielding the highest net present value (NPV) is chosen as the optimal solution.

1.4 Statement of Grade Control Problems

Grade control is at the bottom of the mine planning hierarchy. It deals with ore-waste selection in daily production. After each blast, a mining engineer must determine material routing, which should be sent to the processing plant and which should go to waste dump. If waste is sent to processing plant, the unnecessary processing costs have to be paid. On the other hand, if ore is sent to waste dump, it will cause great losses, too. This feature is particularly important because, unlike in mine planning and ore reserve estimation, the error of mis-classification is non-compensating, that is, one can never recover the cost of misrouted material.

It is well known that kriging estimation is nearly conditionally unbiased with the minimum error variance. In the long run, the use of kriging will result in the least amount of mis-classification between ore and waste categories. At production stage, blast hole assays are used for grade estimation. Therefore, it is also called Blast Hole Kriging. The kriging process for grade control using blast hole assays is identical to kriging for orebody modeling. However, a successful implementation of the computerized grade control system highly depends on successfully solving logistics problems, such as, (1) polygon manipulation, (2) database handling, and (3) interactive graphics planning.
Although polygon manipulation algorithms (i.e., point-in-polygon algorithms, polygon area algorithms, and polygon clipping algorithms) do not belong to the domain of mining technologies, these algorithms are very important part of the basic tools used in mine planning, particularly, in interactive graphics planning. This is because all geometries in mine planning, such as, planning boundaries, surface contours, haul road access, and existing production constraints, have to be specified and handled mostly by polygons.

Data handling is the most time consuming task in computerized grade control. A lot of blast hole data become available every day from various sources such as the assay laboratory, survey station, survey and ore control personnel, and blast hole maps. These data must be systematically recorded since they contain the most basic and valuable information for ore-waste classification. Commercially available editors or word processors, and database managers are not suitable for handling the blast hole and geology data since they do not have the functions to efficiently process these data. Moreover, they can not be linked to the hardware and software used in grade control. A special data handling system including a blast hole editor, blast hole database manager, and rock type database manager is vitally important to the successful implementation of the computerized grade control.

Interactive graphics planning should be the major part of the computerized grade control since the best ore-waste delineation results can be obtained in the interactive graphics planning mode. In the interactive graphics planning mode, the user can input the ore-waste boundaries on the screen using a mouse or the keyboard, as it is to be flagged in the pit in actual production. Alternative ore-waste delineations can be investigated very quickly in order to obtain the best grade-tonnage combination.
1.5 Scope of Work

The overall objective of mine planning is to work out a mine plan which results in a maximum NPV throughout the life of the mine project. Although this objective is clear, its achievement is not straightforward and involves many interrelated parameters such as: optimization with respect to mining rate, optimization with respect to capital investment, optimization with respect to mining sequence. As it can be seen from the previous statements, the task of mine design and planning is a huge and complicated one. The specific objectives of this dissertation are:

1. to develop a new graph theory oriented algorithm for ultimate pit limit design,
2. to investigate the current approaches to long range mine production scheduling,
3. to discuss the problem solving techniques for the practical problems involved in development of a computerized, interactive, grade control system.

This dissertation proceeds as follows: In chapter 2, previous pit design techniques are reviewed. Different methods for designing the ultimate pit limit are briefly discussed and their shortcomings and limitations are explained. Chapter 3 describes the development of a new graph theory oriented algorithm for ultimate pit limit design. The general procedure of the new algorithm will be provided, and the proofs of its optimality and convergence will be given in chapter 3. Chapter 4 investigates the current approaches to long range mine production scheduling. The emphasis is placed on explaining the impossibility applying the Lagrangian relaxation method to obtain a "true optimum" solution using today's computing facilities and operations research techniques. Chapter 5 will stress the implementation aspect of a highly automated ore-waste delineation system called Blast Hole Kriging (BHK), in order to improve the daily grade control procedure
in open pit mines. Practical problem solving techniques in the implementation of such a grade control system, such as: polygon manipulation algorithms, database handling, and interactive graphics planning, will be discussed. Chapter 6 contains the conclusions of this study and future research recommendations.
CHAPTER 2
REVIEW OF PREVIOUS WORK FOR ULTIMATE PIT LIMIT DESIGN

2.1 Overview

Since early 1960's, the application of computers in solving the ultimate pit limit design problem has received a great attention in both academic and research institutions and many mining companies. Many research and case study papers have been published and several algorithms have been developed.

Kim (1978) classified the various pit design techniques into two categories, rigorous optimizing algorithms and heuristic optimizing algorithms. The word "rigorous" implies the availability of mathematical proof of optimality and the word "heuristic" is used to describe an algorithm which works well in nearly all cases but which lacks rigorous mathematical proof of optimality. The ultimate pit limit design algorithms so far can be listed as follows:

1. Theoretical optimizing algorithms
   a) Lerchs-Grossmann's 3-D graph algorithm
   b) Maximum network flow algorithm
   c) Linear programming model

2. Heuristic algorithms
   a) Dynamic programming
   b) Family of moving cone algorithms
   c) Parametric function approach
   d) Other algorithms
Although the true optimizing algorithms have been investigated for more than twenty years, they are not commonly utilized by mining companies since they are difficult to understand and hard to program. In addition they require huge computer memory and high computation costs. The majority of the mining companies are using the moving cone heuristic algorithm in their pit limit design despite the fact that the moving cone method has obvious shortcomings. To understand how these methods can be used in pit optimization, it is necessary to review and discuss them in little more details.

2.2 Lerchs-Grossmann’s 3-D Graph Algorithm

The work of Lerchs and Grossmann (Lerchs and Grossmann, 1965) was the first mathematical approach to the problem of determining the optimum ultimate pit limit. Detailed discussion on Lerchs-grossmann algorithm can be found in Robinson, R. H., 1975; Lipkewich, et al., 1969; and Huttagosol, et al., 1989.

The algorithm formulates the block model into a special graph, that is, a directed tree. The vertices in the tree are identified with blocks and the imposed directed arcs represent pit slope constraints. These directed arcs indicate the relationship between waste blocks that must be removed in order to mine a particular ore block. Since any feasible contour of a pit is contained by a closure of a graph, the optimum pit limit design problem becomes the determination of a closure of a graph with maximum mass.

A tree may contain p-branches (plus-branch) and m-branches (minus-branch). A p-branch is the one linked by an arc pointing away from the root of the tree, while an m-branch is the one linked by an arc pointing toward it. A p-branch is strong if it supports a total positive value. Otherwise, it is weak. An m-branch is strong if it supports a zero or negative value. Otherwise, it is weak.

Normalization plays the key role in solving the re-allocation problem in this algorithm. The main purposes of normalization are: 1) to avoid supporting a
branch whose total value is positive by other blocks, and 2) to prevent supporting other blocks by a branch with a non-positive total value. In figure 2.1, obviously, it is not necessary to support branch 1 by $arc_{(10,6)}$ since the total mass of branch 1 itself is already positive (+2). Therefore, normalization is obtained by removing $arc_{(10,6)}$ and connecting branch 1 to the dummy root $X_0$ by $arc_{(x_0,5)}$ (see figure 2.2). Similarly, it does not make any sense to support block 7 by branch 2 through $arc_{(11,7)}$ since branch 2 itself is already negative (-1). Therefore, normalization has to be performed by removing $arc_{(11,7)}$ and connecting branch 2 to the dummy root $X_0$ by $arc_{(x_0,11)}$. Figure 2.2 is the normalized tree obtained from figure 2.1. In the normalized tree, each strong branch is connected to the dummy root. The normalization processes are depicted quite easily in the example. However, this extremely difficult to program.

![Diagram of an un-normalized tree](image)

**Figure 2.1**  An Un-Normalized Tree
The algorithm utilizes an iterative process that transforms a normalized tree into a new normalized tree. The algorithm ends when the set of strong branches of the normalized tree becomes a closure of the directed graph.

Algorithm:

Step 1. Initialization: A dummy node $X_0$ is added to the tree and will be used as a reference vertex. The entire graph is initially normalized by connecting the dummy node $X_0$ to every vertex in the mine model.

Step 2. Search: For a vertex $X$ belonging to a strong branch, if there exists an overlying vertex $Y$ belonging to a weak branch which must be mined to expose vertex $X$ (see figure 2.3), identify the root vertex, $X_r$, of the strong branch and then go to step 3. Otherwise, go to step 5.
Figure 2.3  Vertex Y of the Weak Branch Is A Restricting Vertex to Vertex X of the Strong Branch

Figure 2.4  The Tree Obtained by Combining the Weak Branch and the Strong Branch
Step 3. Connection and Transformation: The two branches found in step 2 are combined into one by removing the arc between $X$, and dummy root $X_0$ and connecting vertex $X$ to vertex $Y$ (see figure 2.4). In addition, all the arcs on the chain $(X, \ldots, X_r)$ have changed their status. That is, a $p$-edge becomes an $m$-edge and vice versa.

Step 4. Normalization: The combined branch must be re-evaluated for every node in it. Normalization is then performed for the resultant branch by removing any strong arc from the branch and connecting it to the dummy root. Go to step 2.

Step 5. Stop the iteration. The maximum closure consists of all the vertices on the strong branches of the final normalized tree.

The Lerchs-Grossmann algorithm is a systematical method which leads to the optimal ultimate pit limit. Optimality and convergence proofs were given by Lerchs and Grossmann in 1965. The main disadvantages of this technique are hard to understand and difficult to program. The normalization process, which splits up blocks that had been previously grouped together, is extremely time consuming and very hard to implement in a computer program.

2.3 Maximum Network Flow Algorithm

The network flow method was proposed for pit limit optimization by Johnson in 1968. The key idea to apply this technique to pit limit design is to transform a 3-D mine block model into a bipartite network. Then, the standard methods in Operations Research (OR) can be used to solve a maximum flow problem.

A bipartite network is formulated as follows: Basically, each block is denoted by one node. All ore blocks and an imaginary dummy source are put on one side and all waste blocks and an imaginary dummy sink are put on the other side. Each ore block is connected to all the waste blocks that must be removed before mining that ore block according to pit slope requirements. Every positive block
is connected to the dummy source and every negative block is connected to the dummy sink. The arc direction determines the flow direction. The directions of the arcs in the network are drawn as follows: All the arcs connecting the dummy source and the ore blocks point from the source to the ore blocks. All the arcs connecting the waste blocks and the dummy sink point from the waste blocks to the sink. Finally, the arcs connecting the ore blocks and the waste blocks point from the ore blocks to the waste blocks. The arc capacities are defined as follows: the dummy source has an unlimited capacity to send. The dummy sink has an unlimited capacity to receive. The arcs connecting the source and the ore blocks have capacities of the corresponding ore block values so that no more flows than the ore block values are allowed. The arcs connecting the waste blocks and the sink have capacities of the corresponding waste block values so that no more flows than the waste block values are allowed. Finally, the arcs connecting the ore blocks and the waste blocks have unlimited capacities because if a waste block is a restricting block to an ore block, it must be mined out at any cost in order to mine the ore block below. Figure 2.5 illustrates such a bipartite network.

In general, pit limit design based on the network flow approach involves the following steps:

Step 1. Transform a 3-D mine block model into a network flow problem such as the one shown in figure 2.5.

Step 2. Solve this network flow problem for maximum flow by OR techniques such as labeling algorithm.

Step 3. Delete all the waste nodes whose arcs are unsaturated from the network, together with all nodes that are linked with these waste nodes.

Step 4. The nodes in the current network constitute the blocks within the optimum pit limit. The sum of the residual values of the arcs connecting the ore blocks and the source is the profit from pit design.
Conceptually, this technique is easy to understand. However, several difficulties have been identified with respect to the application of network flow to the pit limit design problem. These are: 1) huge computer memory requirements and computation costs, and 2) difficult re-allocation of flow from one block to another. A typical pit limit design problem may involve one million blocks which would require one million nodes to represent them. There may be up to hundreds or thousands possible successors or predecessors for some nodes in the network, which should be represented by directed arcs. The capacity and flow must be recorded for each arc in the network. The re-allocation is extremely difficult for a network with so many nodes and arcs. It is a very tough job even for mainframe computers. As it can be seen, all these difficulties are certainly big hurdles in application of the

Figure 2.5  Network of Pit Limit Design
maximum flow algorithm to pit design. Due to the above difficulties, this technique has never been accepted by the mining companies.

Suppose that one can solve the maximum network flow problem. But step 3 of the algorithm is still not an easy job. Step 3 is to delete all the waste nodes, whose arcs are unsaturated, from the network, together with all nodes that are linked with these waste nodes. For each unsaturated waste node, say node $i$, it may be supported by many underlying ore nodes. Each of these underlying ore nodes may support many other overlying waste nodes. And each of these overlying waste nodes has the same structure as node $i$. There is no simple way to delete all the nodes that are linked with these unsaturated waste nodes. Theoretically, the above problem may be solved by the iterative approach to solve the maximum netflow problem many times. That is, after the netflow problem is solved, all the unsaturated waste nodes and all the ore nodes, which are directly linked to the unsaturated waste nodes, are removed from the network. Then, the maximum netflow problem consisting of the remaining nodes is restarted. The above processes are repeated until no more unsaturated nodes are left. However, it is not feasible to the practical pit limit problem using the above approach, because it is even not possible to solve the maximum netflow problem for just once.

2.4 Linear Programming Model

Linear programming is a well known operations research technique. This method has been used by several academic researchers for the ultimate pit limit problems (Myers, 1966; Johnson, 1968; and Koenigsberg, 1982). The ultimate pit limit problem can be formulated as an integer linear programming problem with the following structure:

$$\text{Maximize} \quad \sum_{i=1}^{N} c_i x_i$$

$$\text{Subject to:} \quad -x_i + x_i \leq 0 \quad i = 1, N$$

$$x_i = 0, 1 \quad i = 1, N$$
Where,

\[ N = \text{total number of blocks of the mine model}; \]
\[ i = \text{block number in a more natural 3-D notation}; \]
\[ t = \text{set of overlying blocks which must be removed before mining block } i; \]
\[ c_i = \text{net value of block } i; \]
\[ x_i = 1 \text{ if block } i \text{ is mined}; \]
\[ = 0 \text{ otherwise}. \]

Theoretically, it is not difficult to formulate the pit limit design problem into a standard integer linear programming model. But the application of integer linear programming to pit limit design is impossible, at least in the near future.

In formulation of pit limit design problem into linear programming, the total number of sequencing constraints for a single particular block, that is, the mathematical representation that the block can not be mined unless all the overlying blocks have been removed, is equal to the total number of overlying blocks. It can be illustrated by a simple 2-D example as shown in figure 2.6, where there are eight restricting blocks above block 13.

![Figure 2.6 A 2-D Cross Section Example](image)

Therefore, the above sequencing constraints with respect to block \( x_{13} \) can be represented by eight mathematical constraints as follows:
For a 100 (column) x 100 (row) x 50 (bench) mine model with cubic blocks, there are up to 126,633 constraints for a single block at the bottom of the model, and there may be totally up to 12,412,277,152 (more than 10 billion) constraints required to formulate the entire model into linear programming problem! In the application of linear programming technique, the number of constraints and number of variables must not be too large because the required computer time for solutions in proportional to the third power of the number of constraints (Fytas and Calder, 1986). Even supposing that computation time is not critical, a computer can not handle an integer linear programming problem with several million constraints due to its precision. As such, the difficulty of application of this algorithms in the pit limit design can be quite easily appreciated. In fact, except the original academic papers given by the respective authors, no one has tried to implement it.

2.5 2-D Dynamic Programming Model

An original dynamic programming algorithm was proposed by Lerchs-Grossmann (1965) to determine the optimal configuration of blocks to be removed in a two dimensional cross section. The main procedure of this method is first to
construct a tableau based on the block values for a cross section, then to apply the standard dynamic programming technique on it. The algorithm mainly involves the following steps:

Step 1. For a given cross section, let \( m_{ij} \) represent the net value of the block in level \( i \) and column \( j \) (figure 2.7).

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
1 & -1 & -2 & -3 & -2 & -3 & -2 & -1 \\
2 & -2 & -1 & +3 & +3 & +4 & +2 & -3 \\
3 & -3 & +1 & +5 & -1 & +6 & +1 & -4 \\
\end{array}
\]

Figure 2.7 \( m_{ij} \) Tableau

Step 2. Construct a tableau of \( M_{ij} \) based on the block values \( m_{ij} \) as follows:

\[
M_{ij} = \sum_{k=1}^{i} m_{kj}
\]

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
1 & -1 & -2 & -3 & -2 & -3 & -2 & -1 \\
2 & -3 & -3 & 0 & +1 & +1 & 0 & -4 \\
3 & -6 & -2 & +5 & 0 & +7 & +1 & -8 \\
\end{array}
\]

Figure 2.8 \( M_{ij} \) Tableau
$M_{ij}$ represents the cumulative net value realized in extracting a single column $j$ from the top down to level $i$. Then, overlay a network on the $M_{ij}$ tableau as shown in figure 2.8, where, each cell represents a block of the mine model and arcs from a particular block to its previous column indicate the possible paths from that block to its previous column.

Step 3. Construct a tableau of $P_{ij}$ by applying the standard dynamic programming technique as follows:

$$P_{0j} = 0$$

$$P_{ij} = M_{ij} + \max\{P_{i-1,j-1}, P_{i,j-1}, P_{i+1,j-1}\}$$

Figure 2.9 $P_{ij}$ Tableau

$P_{ij}$ is the maximum possible contribution of any feasible pit from column 1 to column $j$. The path with the maximum value is indicated by an arrow from each block to its previous column as shown in figure 2.9.
Step 4. Find a $P_{\text{max}}$ which represent the maximum value of the pit as:

$$P_{\text{max}} = \text{Max}\{P_{ij}\} \quad \text{for all } j$$

If $P_{\text{max}}$ is non-positive, then there exists no contour with positive profit. If $P_{\text{max}}$ is positive, then the optimum contour is obtained by following the arrows starting at the block where $P_{\text{max}}$ is found to the left elements as shown by the thick arrows in figure 2.9. Figure 2.10 shows the pit limit for this given cross section example.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>-1</td>
<td>+3</td>
<td>+3</td>
<td>+4</td>
<td>+2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-2</td>
<td>+5</td>
<td>+6</td>
<td></td>
<td></td>
<td>-3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-3</td>
<td>+1</td>
<td>-4</td>
<td>+1</td>
<td>-4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.10  Optimal Contour

For a two dimensional pit limit problem, the above dynamic programming is simple and fast. However, this two dimensional dynamic programming approach becomes impractical in real three dimensional cases since it is able to generate the contour only in a given two dimensional cross section. The final pit geometry of all cross sections, when they are assembled together, does not yield the optimum ultimate pit limit and may violate the maximum allowable pit slopes.

Since then, many attempts have been made to extend the Lerchs-Grossmann 2-D dynamic programming algorithm in order to solve the 3-D ultimate pit limit

Johnson's works (1970 and 1971) are still close to the original 2-D algorithm except that they also consider the block value information from 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. The satisfactory results are not achieved by the above improvement since it can not generate the optimal solution and it fails to follow 3-D pit slope constraints, which, in turn, requires to manually smooth the resulting pit limit. The main disadvantage of the algorithm is that it does not consider waste blocks which must be removed in order to mine an ore block other than those along the cross section and longitudinal section. This is referred to as corner effect because in developing ultimate pit limit configurations with this technique, waste blocks in the corner are totally neglected.

In Koenigsberg (1982), a 3-D dynamic programming is proposed for solving a 3-D case of ultimate pit limit problem. The algorithm searches the block value information in the interested cross section and investigates the relationship between levels within a column, and between columns within a section. However, a generalized pit slope constraint is still infeasible with his algorithm.

Besides the above disadvantages, none of the methods based on dynamic programming can not handle the variable slope problem. That is, the pit walls must have the same slopes in all directions. In practice, all mines require the possibility of different slope angles for different directions.

In summary, dynamic programming methods are still impractical for determining the true optimum ultimate pit limit with three dimensional geometry and generalized pit slope requirements.
2.6 Family of Moving Cone Algorithms

This kind of heuristic techniques started with the well known moving cone method of Kennecott (Pana, 1965). The moving cone algorithm has been widely used by most mining companies to determine ultimate pit limits. This is probably due to the fact that it is simple, fast, and very easy to understand and implement.

There are many variations of this method (Mario and Slama, 1973; Robinson, R. H., 1975; Lemieux, 1979; Hanson, J., 1986; and Cai, W., 1989). Basically, these authors tried to improve the original moving cone method, but they still used the "cone concept". Although each author suggested a different manner for handling the joint support problem using the cone concept instead of using the definitions of predecessors and successors, the main feature of these modified techniques is still to simulate the removal of material in cone increments just as the moving cone algorithm. Since there are no clear definitions of the predecessors and successors for each block in the mine model, the above techniques only have the ability to partially handle the joint support problem and have no ability to solve the re-allocation problem. Their work improved the original moving cone algorithm in certain degrees, but the modified algorithms never became optimum ones. Since all versions of the modified algorithms are more complicated than the original moving cone algorithm (Pana, 1965) and they can not generate the true optimum designs, these suggested modifications are not used by most mining companies.

The essential mechanics of the original moving cone algorithm are:

1. to define a volume of material to be considered for potential mining by superimposing a frustum of an inverted cone into the ore body model as shown in figure 2.11,

2. to sum the total values of all the blocks whose centers lie inside the cone,

3. to remove all the blocks if its total value is positive,
4. to move the base of the cone from ore block to ore block from the top bench to bottom bench until every ore block has its turn in the base. The process is continued until it is not possible to find an increment with positive value.

Figure 2.11    Inverted Cone Frustum (from Crawford)

Through the use of the cone concept any shape can be approximated by the overlapping of the cones as shown in figure 2.12.
Figure 2.12  Approximation of the Pit Limit by Overlapping of the Cones

Figure 2.13  Example of Moving Cone Method Missing Shared Contribution and Over-mining at the Same Time
It can be seen that this modified moving cone method simulates the actual mining process. The main shortcoming of the moving cone algorithm is that it does not generate the true optimal solution because it does not have the capability to handle the joint support and re-allocation problems. Figure 2.13 shows an example where the moving cone method fails to detect a shared contribution and over mines at the same time. The deviation from optimum may be quite big for ore bodies of irregular or spotty shape. In figure 2.13, there are three positive valued (ore) blocks represented by $A$, $B$ and $C$. An inverted cone can be generated above each ore block. The moving cone method will mine all the blocks within the cone if the total value of the blocks inside the cone is positive.

The following shortcomings prevent the method from yielding the optimal solution:

1. Under-mining due to a missing shared contribution. This method evaluates one cone at a time; therefore, it can not detect shared contributions. It is profitable if cones $A$ and $B$ are evaluated together (see figure 2.13). But, cones $A$ and $B$ are not profitable if they are considered individually. Therefore, neither $A$ nor $B$ is mined by the moving cone method.

\[
\text{Profit of cone } (A + B) = Value(A) + Value(B) + Value(D) + Value(E) + Value(F) = +7
\]
\[
\text{Profit of cone } A = Value(A) + Value(D) + Value(E) = -1
\]
\[
\text{Profit of cone } B = Value(B) + Value(F) + Value(E) = -1
\]

2. Over-mining due to "free ride" effect. This method will mine all the blocks within a cone if the cone is profitable. Cones $A$ and $B$ are not mined since they are not individually profitable. After a cone is set up at ore block $C$, blocks $A$, $B$, $C$, $D$, $E$, $F$ and $G$ are mined together since the total value of cone $C$ is positive (+4). Therefore, +4 unit profit can be realized by using
moving cone method. But the maximum net value obtained from the pattern is +7 instead of +4 by mining blocks $A$, $B$, $D$, $E$ and $F$ and leaving blocks $C$ and $G$ as unmined. Blocks $C$ and $G$ are mined by the moving cone method due to the "free ride" effect provided by upper bench ore blocks $A$ and $B$.

Profit of cone $C = Value(A) + Value(B) + Value(C) + Value(D) + Value(E) + Value(F) + Value(G) = +4$

In 1986, J. Hanson proposed a modified network flow algorithm which is similar to the "Allocating Moving Cone Algorithm" proposed by Robinson in 1975. Although Hanson claims it is a "Modified Network Flow Algorithm", it is mostly a moving cone type simulation method. The algorithm can not deal with the reallocation problem, while it can partially handle the joint support problem. Case studies by Kim et al. (1987) show that Hanson's algorithm missed the optimum solutions in all cases, while the algorithm generated more profit than moving cone algorithm in most cases and it also occasionally generated less profit than the multi-pass moving cone method.

In 1989, W. Cai failed to modify Hanson's algorithm into an optimum one and concluded that it is impossible to make such a simulation oriented algorithm optimal since there are no clear definitions of predecessors and successors in this kind of algorithm. Clear definitions of predecessors and successors with respect to each block are required in order to completely solve the joint support and reallocation problems, both of which are necessary to achieve the optimal pit limit.

### 2.7 Parametric Function Approach

This method was originally suggested by G. Matheron in 1975. It was reported by Francois-Bongarcon and Marechal in 1976. Further details were given in D. Myers (1980).
The basic idea of this method is to find a parametrizing function whose isovalue curves give the desired solution by determining if a block is within the ultimate pit limits for a given set of economic parameters. This method works with the original block grades and determines a parametric function defined on all the blocks in the mine model. With the aid of such a parametric function a series of pits with different sizes, each containing the maximum quantity of metal for its size, can be found by comparing the parametric function value of each block with the specified cutoff grade. The parametrizing function is based on the idea of penalty parameters, i.e., a higher production cost will result in a smaller pit, and vice versa.

The set of all the blocks in a mine model is denoted by $N$. Let $X_i$ be the center coordinates of block $i$, for any $i \in N$. A real-valued function $\Lambda$ is defined on $X_i$, for any $i \in N$. For any given cutoff grade $\lambda$, $\Lambda(X_i) > \lambda$ indicates that block $i$ is within the optimal ultimate pit limit. Therefore, the set of blocks satisfying the condition $\Lambda(X) > \lambda$ is an optimal contour. With such a function it would be possible to easily determine optimal contours for a variety of possible economic conditions.

This method only generates the heuristic solution because of some mathematical transformations. The geometrical constraints of the circular cone are approximated by the intersection of two triangular cones. Unlike other heuristic algorithms, this approach is mathematically very complex. Interested readers are referred to the cited references (D. Myers, 1980).

2.8 Other Algorithms

There are several other suggested algorithms. They are:

(a) Heuristic algorithm by Phillips (1972)

(b) Heuristic algorithm by Korobov (1974)
Method (a) is a pure trial and error method which tries to find the best combinations on each bench. Method (b) uses the idea of allocating valuations between ore blocks and all the overlying waste blocks.

These algorithms are not optimal, and in practice, they are not as good as the heuristic moving cone algorithm. Therefore, they are not popular. Since these algorithms were brought out, they have not been used.

2.9 Conclusion

Since the early 1960's, there were many efforts to develop the optimal algorithms for ultimate pit limit design, and a lot of attempts were tried to modify existing heuristic algorithms into the optimal ones. Unfortunately, the available optimal algorithms are not practical and the practical algorithms are not optimal. Subsequently, there has been considerable debate concerning optimal and heuristic algorithms. Essentially, some people favor the optimal algorithms since they yield true optimal pit designs, and some others favor the moving cone technique since it is simple and fast. Is it possible to find an algorithm which is both optimal and practical? The next chapter will answer this question.
CHAPTER 3

A NEW GRAPH THEORY ALGORITHM FOR OPTIMUM ULTIMATE PIT LIMIT DESIGN

3.1 Introduction

The main purpose of this chapter is to introduce a new graph theory algorithm for optimum ultimate pit limit design. This algorithm will produce a true optimal solution and maximize the total undiscounted net profit for a given 3-D block mine model. The algorithm performs much better than the well known Lerchs–Grossmann algorithm, in terms of a reduction in computation time and in computer memory requirements. More importantly, the new algorithm is easier to understand and program than the previous optimal pit limit design algorithms. The computer representation of arcs and the rules of arc generation which dictate the formation of trees in the graph, the mechanism of the graph transformation, the properties and the general procedures of the algorithm, the mathematical proofs of its optimality and convergence, the suggested specific steps to program the algorithm, comparison between the new algorithm with Lerchs–Grossmann algorithm, program validation, some experience and strategies in implementation of the algorithm and two case study results are given and discussed in this chapter.

In practice, there are two key problems in optimum ultimate pit limit design. These two problems involve joint support and re-allocation. Joint support means that a group of ore blocks together support another group of waste blocks. Re-allocation means to shift the already allocated value from one block to another one. In order for an algorithm to be a truly optimum one, these two problems must be handled properly. An algorithm can not be optimal, unless these two problems are completely solved. The new algorithm does handle these two problems efficiently.
A 3-D fixed block mineralization model must be prepared before applying this algorithm. A block is classified as either ore or waste according to its net value. A block is considered as a possible ore block if the net profit value of that block is positive. Otherwise, the block is considered as a waste block. The algorithm formulates the 3-D block model into a directed graph consisting of many trees. The vertices in the graph are equated to the blocks in the mine model and the imposed arcs in the graph represent pit slope constraints. The formation of each directed tree is based more on the ore–waste support concept than the geometry constraints alone. That is, arcs are only generated between either ore to waste vertices, or waste to ore vertices, and no arcs are generated between ore vertices nor between the waste vertices. Each directed arc which points from an ore vertex to an overlying waste vertex or points from a waste vertex to an underlying ore vertex indicates that the removal of the overlying waste block is necessary before mining the underlying ore block. Initially, the values of vertices are set equal to the values of corresponding blocks. As the graph is transformed, the values of the vertices keep changing to indicate the current status of the vertices within a given tree. The algorithm determines a maximum closure, that is, a feasible contour of a pit with maximum mass, thus achieving the optimal pit limit.

3.2 Definitions

This chapter uses not only the terminology of the common graph theory, but also terms defined for the new algorithm. Some terms are defined as follows:

**Vertex:** A vertex is an element of the graph representing the location and value of a block. **Ore vertex** is the one corresponding to an ore block. **Waste vertex** is the one corresponding to a waste block. The notation $v_i$ is used to denote vertex $i$.

**Arc:** An arc is an ordered pairs of two vertices such as $a_{i,j} = (v_i, v_j)$, which means that there is an arrow drawn from vertex $v_i$ to vertex $v_j$. In the algorithm,
arcs are used to represent the mining constraints. Positive arc is the one starting at an ore vertex towards an overlying waste vertex. Negative arc is the one starting at a waste vertex towards an underlying ore vertex.

Directed graph: A directed graph $G = (V, A)$ is defined by a set of vertices $V$ of $G$, together with a set of arcs $A$ of $G$.

Subgraph: A graph $G_s = (V_s, A_s)$ is a subgraph of $G = (V, A)$, if $V_s \subseteq V$ and $A_s \subseteq A$. That is, $G_s$ is defined by a set of vertices $V_s \subseteq V$ and all the arcs that connect vertices $V_s$ in $G$.

Partial graph: A graph $G_p = (V_p, A_p)$ is a partial graph of $G = (V, A)$, if $V_p = V$ and $A_p \subseteq A$. That is, $G_p$ contains all the vertices $V$ and a set of arcs $A_p \subseteq A$.

Path: A path is a sequence of arcs $(a_1, 2, a_2, 3, a_3, 4, \ldots, a_{n-1}, n)$ such that the terminal vertex of each arc corresponds to the initial vertex of the succeeding arc.

Tree: A tree is a connected and directed graph containing no cycles and there is a unique path between any two vertices in the tree. A single vertex can be a tree of itself. Positive tree is the one with total positive mass. Negative tree is the one with total non-positive mass.

Predecessor and successor: Between any two vertices linked by a directed arc, the vertex on the starting end of the arc is the predecessor, whereas the other one on the terminal end of the arc is the successor.

Root: A root is a vertex without a predecessor in a tree.

Closure: A closure is any surface satisfying the slope constraints. The main purpose of pit limit optimization is to find a closure with maximum mass, which is also called as maximum closure. The null set can also be a closure.
3.3 Rules of Arc Generation

Tree is the basic element of the graph constructed by the algorithm, whereas arc generation is the basic process to construct the trees. The efficiency of arc generation will directly impact the efficiency of the algorithm. Therefore, it is extremely important to efficiently add or delete arcs and create trees in computer representation. This section will discuss the computer representation of an arc and define the rules to generate arcs.

3.3.1 Computer Representation of an Arc

Two arrays, $V(i)$ and $L(i)$, are used to represent the tree in computer representation, where $V(i)$ is the current value of vertex $i$, and $L(i)$ is the predecessor (or link pointer) of vertex $i$.

In the algorithm, the process of removing arcs is just as important as the process of adding arcs, since the algorithm utilizes the iterative method to find the best pattern of arc generation. That is, in each iteration, some new arcs will be added and some existing arcs may be removed due to the process of re-allocation. The main purpose of this subsection is to describe the computer representation of adding arcs and removing arcs.

[1] Adding Arcs

As an arc $a_{i,j}$ is generated from vertex $i$ to vertex $j$, the following operations must be performed in the computer representation:

$L(i)$ and $V(j)$ are unchanged

$L(j) = i$

$V'(i) = V^0(i) + V^0(j)$

where, $V'(i)$ is the value of vertex $i$ after the arc is added.  
$V^0(i)$ is the value of vertex $i$ before the arc is added.
Figure 3.1 illustrates the process to generate an arc, $a_{i,j}$, from vertex $i$ to vertex $j$. In figure 3.1, (a) shows two vertices, $v_i$ and $v_j$, (b) shows the physical arc, $a_{i,j}$, added from $v_i$ to $v_j$, and (c) shows the operations required in computer representation to add arc $a_{i,j}$.

![Diagram](image)

- **Before adding the arc** (a)
  - $V(i)$
  - $L(j) = 0$
  - $v_i$
  - $V(j)$
  - $L(i) = i$
- **After adding the arc** (b)
  - $V(i)$
  - $V'(i) = V^0(i) + V^0(j)$
  - $L(i)$
- **Computer representation** (c)
  - Operations in computer representation to add arc $a_{i,j}$:
    - $V(i)$: unchanged
    - $L(j) = i$
    - $V'(i) = V^0(i) + V^0(j)$
    - $L(i)$: unchanged

**Figure 3.1 Process of Generating an Arc**

[2] **Removing Arcs**

As an arc $a_{i,j}$ is removed between vertex $i$ and vertex $j$, the following reverse operations are required in the computer representation:

- $L(i)$ and $V(j)$ are unchanged
- $L(j) = 0$
- $V'(i) = V^0(i) - V^0(j)$

where, $V'(i)$ is the value of vertex $i$ after the arc is removed.
- $V^0(i)$ is the value of vertex $i$ before the arc is removed.
Figure 3.2 illustrates the process for removing arc $a_{i,j}$ between vertex $i$ and vertex $j$. In Figure 3.2, (a) shows a pattern of two vertices, $v_i$ and $v_j$, connected by arc $a_{i,j}$, (b) is the status of the two vertices after the arc, $a_{i,j}$, is removed, and (c) shows the operations required in computer representation to remove arc $a_{i,j}$.

![Figure 3.2 Process of Removing an Arc](image)

### 3.3.2 Rules of Arc Generation

The rules which dictate whether a positive arc or a negative arc should be generated are as follows:

1. **A positive arc means complete support and is upward.**

   A positive arc is generated from an ore vertex to an overlying waste vertex if the current value of the ore vertex is greater than the current absolute value of the waste vertex, that is, the ore vertex is big enough to completely offset the overlying waste vertex.

   Figure 3.3 gives an example of a positive arc. Initially, waste vertex $v_j$ has a value of $-5$ and ore vertex $v_i$ has a value of $+10$. In figure 3.3, waste vertex $v_j$ is supported by ore vertex $v_i$ through arc $a_{i,j}$ and, after allocating 5 units of profit to offset waste vertex $v_j$, ore vertex $v_i$ still has 5 units of profit left.
[2] A negative arc means partial support and is downward.

A negative arc is generated from a waste vertex to an underlying ore vertex if the current value of the ore vertex is less than or equal to the current absolute value of the waste vertex, that is, the ore vertex is not big enough to completely offset the overlying waste vertex, but it is good enough as partial support.

Figure 3.4 gives an example of a negative arc. Initially, waste vertex $v_j$ has a value of $-10$ and ore vertex $v_i$ has a value of $+5$. In figure 3.4, ore vertex $v_i$ is used to partially support waste vertex $v_j$ through arc $a_{j,i}$ and, after receiving 5 units of profit from ore vertex $v_i$, waste vertex $v_j$ still has $-5$ units of profit left.

In summary, if it is a complete support, a positive arc (upward arc) is generated from an ore vertex to its overlying waste vertex; if it is a partial support, a negative arc (downward arc) is added from a waste vertex to its underlying ore vertex.
When the above rules of arc generation are applied, a vertex may have more than one successor, but it can only have one predecessor. In a situation where the value of an ore vertex is big enough to support several overlying waste vertices, this ore vertex is likely to have several successors (see figure 3.5). In figure 3.5, ore vertex \( v_{o1} \) has three successors \( v_{w1}, v_{w2}, v_{w3} \), and one predecessor \( v_{w4} \).

\[
\begin{align*}
V(i) &= -10 \\
L(i) &= 0
\end{align*}
\]

\[
\begin{align*}
V(j) &= -5 \\
L(j) &= 0
\end{align*}
\]

Operations in computer representation to add negative arc \( a_{ji} \):

\[
\begin{align*}
V(j) &= -10 + 5 = -5 \\
L(j) &: \text{unchanged} \\\nV(i) &: \text{unchanged} \\
L(i) &= j
\end{align*}
\]

Figure 3.4 An Example of a Negative Arc

\[
\begin{align*}
V(i) &= 5 \\
L(i) &= 0
\end{align*}
\]

\[
\begin{align*}
V(j) &= 5 \\
L(j) &= j
\end{align*}
\]

(a) Before adding the arc  (b) After adding the arc  (c) Computer representation

Figure 3.5 An Ore Vertex Supporting Several Waste Vertices
Likewise, in a situation where the value of a waste vertex is big enough to require several underlying ore vertices to support it, this waste vertex may have several successors (see figure 3.6). In figure 3.6, waste vertex \( v_{w1} \) has three successors \( v_{o1}, v_{o2} \) and \( v_{o3} \), and one predecessor \( v_{o4} \).

![Figure 3.6 Several Ore Vertices Supporting One Waste Vertex](image)

Figure 3.6 (a) shows an eight block 2-D model problem, where the dashed arrows represent the assumed slope constraints in order to expose the bottom ore blocks. Since ore vertex \( v_6 \) can completely support both waste vertices \( v_1 \) and \( v_2 \), two positive arcs, \( a_{6,1} \) and \( a_{6,2} \), are generated from \( v_6 \) to \( v_1 \) and \( v_2 \). After offsetting waste vertices \( v_1 \) (-1) and \( v_2 \) (-5), the current value of ore vertex \( v_6 \) is updated by \( 8+(-1)+(-5)=2 \). Therefore, the remaining value of \( v_6 \) becomes 2 (see figure 3.7 (b)). After the above two arcs are added, the sum of the current values of ore vertices \( v_6 \) and \( v_7 \), which is 2+3=5, is not enough to offset waste vertex \( v_3 \) which has a current value -9. Therefore, two negative arcs, \( a_{3,6} \) and \( a_{3,7} \), are generated from \( v_3 \) to \( v_6 \) and \( v_7 \) and the current value of vertex \( v_3 \) is updated by \(-9+2+3=-4 \) (see figure 3.7 (c)). Now the current value of waste vertex \( v_3 \) is -4, which can be completely offset by ore vertex \( v_8 \). A positive arc, \( a_{8,3} \), is next added from \( v_8 \) to \( v_3 \) in figure 3.7 (d).
a. Mining constraints

b. After adding arcs #1 and #2

c. After adding arcs #1, #2, #3 and #4

d. The final graph

Figure 3.7  A 2-D Example of Arc Generation
In the final tree shown in figure 3.7 (d), ore vertex $v_6$ is used to completely offset both waste vertices $v_1$ and $v_2$, and its remaining value is used as partial support to waste vertex $v_3$. Waste vertex $v_3$ is partially supported by both ore vertices $v_6$ and $v_7$, and its remaining value is completely offset by ore vertex $v_6$.

The final graph shown in figure 3.7 (d) consists of three trees which are rooted at vertices $v_4$, $v_5$ and $v_8$, respectively (please see that $L(4)=L(5)=L(8)=0$). Of the three trees, vertices $v_4$ and $v_5$ are trees themselves. All the three trees are positive trees.

3.4 Properties of The Tree

The trees constructed by the above rules of arc generation have the following properties which are all very important in understanding the new graph theory algorithm.

**Property 1.** Ore vertex always has positive current value and waste vertex always has negative or zero current value.

Suppose that $v_i$ is an ore vertex and $v_j$ is an overlying waste vertex. Initially, the ore vertex has a current value $V(i)$ equal to the ore block value which is positive, and the waste vertex has a current value $V(j)$ equal to the waste block value which is negative.

According to the rules of arc generation, the current value of ore vertex $v_i$ will be updated by $V'(i) = V^0(i) + V^0(j)$, if and only if a positive arc, $a_{i,j}$, is generated from ore vertex $v_i$ to its overlying waste vertex $v_j$. But, positive arc $a_{i,j}$ will be added only if $V^0(i) > |V^0(j)|$. Therefore, after adding arc $a_{i,j}$, the current value of ore vertex $v_i$ is still positive (see figure 3.7 (b)).

Likewise, the current value of waste vertex $v_j$ will be updated by $V'(j) = V^0(j) + V^0(i)$, if and only if a negative arc, $a_{j,i}$, is generated from waste vertex $v_j$ to its underlying ore vertex $v_i$. But, negative arc $a_{j,i}$ will be added only if
Therefore, after adding arc $a_{ji}$, the current value of waste vertex $v_j$ is still non-positive (see figure 3.7 (c)).

The importance of this property is that it can be used to easily identify ore or waste vertices in the process to construct trees.

**Property 2.** A positive tree is always rooted at an ore vertex and a negative tree is always rooted at a waste vertex.

Initially, any waste vertex is a negative tree itself and any ore vertex is a positive tree itself. From the rules of arc generation, after a negative arc is added from a waste vertex to an underlying ore vertex, the ore vertex becomes a non-root vertex and the waste vertex will be the root vertex of a new negative tree obtained by adding the arc from that waste vertex to that ore vertex. In figure 3.7 (c), negative arcs $a_{3,6}$ and $a_{3,7}$ change ore vertices $V_6$ and $V_7$ into non-root vertices and make waste vertex $V_3$ into the root vertex of the new negative tree.

Likewise, after a positive arc is added from an ore vertex to an overlying waste vertex, the waste vertex becomes a non-root vertex and the ore vertex will be the root vertex of a new positive tree obtained by generating the arc from that ore vertex to that waste vertex. In figure 3.7 (d), positive arc $a_{8,3}$ changes waste vertex $v_3$ into non-root vertex and makes ore vertex $V_8$ into the root vertex of the combined positive tree.

This property can be used to identify whether a vertex belongs to a positive or negative tree by checking its root vertex. That is, if the tree is rooted at an ore vertex, all the vertices on the tree belong to the positive tree. Otherwise, all the vertices on the tree belong to the negative tree.

**Property 3.** Each tree has only one root. Only the root vertex of a tree has the null predecessor, i.e., the predecessor array element of the root vertex is not used. The total mass of a tree is equal to the current value of its root vertex, i.e., the total mass of a tree is located at its root.
In the tree constructed by the algorithm, a vertex can have only one predecessor. If the predecessor of the vertex is null, the vertex is the root. Therefore, each tree has only one root.

There is a single chain from any vertex to the root vertex in a tree. From the rules of arc generation, for any arc generated from a vertex on the chain, the current value of that vertex becomes the cumulative value of all successors of that vertex plus the value of that vertex. Therefore, the current value of the root vertex is the cumulative value of all the vertices on the tree.

**Property 4.** A predecessor can be traced from its successors, but successors cannot be traced from their predecessors, because the algorithm uses the single chain linker.

From the rules of arc generation, when an arc is added from vertex $v_i$ to vertex $v_j$, the predecessor is kept by setting $L(j) = i$ and the successor is not recorded. Since each vertex can have only one predecessor, it is easy to keep the predecessor information for each vertex. But each vertex may have many successors. Theoretically, the information of the successors can be recorded; practically, it is limited by computer memory. Therefore, the new algorithm has to use the iterative process to obtain the optimum solution.

**Property 5.** If a vertex has successors, the initial value of the vertex can be obtained by subtracting all the current values of its successors from its current value of the vertex under consideration.

If an ore vertex has successors, which are waste vertices, the current value of the ore vertex is the residual value after supporting its successors. If a waste vertex has predecessor, which is ore vertex, the current value of the waste vertex is already offset by its predecessor. In figure 3.7 (b), waste vertices $v_1$ and $v_2$ are
successors of ore vertex $v_6$. After offsetting waste vertices $v_1 (-1)$ and $v_2 (-5)$, the current value of ore vertex $v_6$ is $2 (2=-8+(-1)+(-5))$. If $V(1) = -1$ and $V(2) = -2$ are subtracted from the current value of $v_6$, $V(6) = 2$, that is, $2 - (-1) - (-5) = 8$, the original value of vertex $v_6$ before supporting waste vertices $v_1$ and $v_2$ is easily obtained.

Similarly, if a waste vertex has successors, which are ore vertices, the current value of the waste vertex is the remaining value after having been partially supported by its successors. If an ore vertex has predecessor, which is waste vertex, the current value of the ore vertex is already used to partially support its predecessor. In figure 3.7 (c), ore vertices $v_6$ and $v_7$ are successors of waste vertex $v_3$. After being partially supported by ore vertices $v_6 (+2)$ and $v_7 (+3)$, the current value of waste vertex $v_3$ is $-4 (-4=-9+2+3)$. If $V(6) = +2$ and $V(7) = +3$ are subtracted from the current value of $v_3$, $V(3) = -4$, that is, $-4 - (+2) - (+3) = -9$, the original value of vertex $v_3$ before it is partially supported by ore vertices $v_6$ and $v_7$ is easily obtained.

This property is used to remove the arcs between a vertex and its successors.

**Property 6.** In a positive tree, any ore vertex can be turned into the root of the positive tree by rearranging the arcs on the chain starting from its root vertex and ending at the interested ore vertex. In a negative tree, any waste vertex can be turned into the root of the negative tree by rearranging the arcs on the chain starting from its root vertex and ending at the interested waste vertex. This rearranging procedure is as follows:

1. Locate the chain starting at vertex $v_i$, which you want to turn into the root, ending at the root vertex, $v_{root}$.

2. Remove all the arcs on the chain by applying property 5. This can be done by successively subtracting the successor’s current value from its
predecessor's current value for all the vertices on the chain, and then, setting all the predecessors of the vertices on this chain as null.

3. Re-generate the arcs for all the vertices on the chain starting at the original root vertex $v_{\text{root}}$ and ending at vertex $v_i$ by applying the rules of arc generation.

Figure 3.8 (a) shows a simple positive tree rooted at ore vertex $v_{o1}$. Suppose that it is desired to turn ore vertex $v_{o3}$ into the root. The graph as shown in figure 3.8 (b) can be obtained by removing arcs on the chain $v_{o3} - v_{w1} - v_{o1}$.

After the arcs are re-generated along the chain $v_{o1} - v_{w1} - v_{o3}$ on the graph of figure 3.8 (b) by applying the rules of arc generation, two possible graphs as shown in figure 3.8 (c) and (d) can be obtained. If ore vertex $v_{o1}$ can not completely offset waste vertex $v_{w1}$, negative arc $a_{w1,o1}$ is first added, and positive arc $a_{o3,w1}$ is next added. After that, the graph shown in figure 3.8 (c) is obtained. If ore vertex $v_{o1}$ can completely offset waste vertex $v_{w1}$, positive arc $a_{o1,w1}$ is added, and no arc should be added between vertices $v_{o3}$ and $v_{w1}$ since the both vertices belong to the positive trees. In such situation, the graph shown in figure 3.8 (d) is obtained. In either case, ore vertex $v_{o3}$ is the root of the new graph.

After ore vertex $v_{o3}$ becomes root, ore vertex $v_{o4}$ can be made into the root vertex in the same way by re-arranging arcs on chain $v_{o4} - v_{w3} - v_{o3}$. The same comments apply to the negative trees. Therefore, the procedure assures that any ore vertex in a positive tree can be made into the root of a new positive tree and any waste vertex in a negative tree can be turned into the root of a new negative tree by either reversing the arcs on the chain or cutting the chain into several trees or both.

It is important to emphasize that nothing is done to the vertices which are not on the chain in the re-arranging process.

A numerical example is given below. Figure 3.9 (a) shows a negative tree rooted at waste vertex $v_5$. Suppose that it is desirable to make waste vertex $v_2$
into the root of the negative tree. The chain from vertex \( v_2 \) to the root \( v_6 \) is 2-6-3-8-5. Figure 3.9 (b) is the graph obtained by removing the arcs on the chain 2-6-3-8-5. Now the arcs should be re-generated for the vertices on the above chain in the sequence of 5-8-3-6-2. A negative arc \( a_{5,8} \) is first added since ore vertex \( v_8 \) (+7) can not completely offset waste vertex \( v_8 \) (-8). After that, since ore vertex \( v_8 \) already belongs to a negative tree, no arc should be added between vertices \( v_8 \) and \( v_3 \). Next, the current value of vertex \( v_6 \) is updated by 7+(-6)=1 since the positive arc \( Q_{6,3} \) is added. Finally, a negative arc \( a_{2,6} \) is generated and vertex \( v_2 \) is made into the root vertex of a new negative tree. Figure 3.9 (c) shows the rearranged graph which consists of two negative trees. One is rooted at waste vertex \( v_2 \) and the other is rooted at waste vertex \( v_5 \).

Figure 3.8 Changing Non-root Vertex into Root Vertex by Re-arranging Arcs
(a) A negative tree rooted at vertex 5

(b) After removing the arcs on chain 5-8-3-6-2

(c) After re-arranging the arcs on chain 5-8-3-6-2

Figure 3.9 An Example of Re-arranging Arcs
3.5 Joint Support and Re-allocation

The rules of arc generation given in section 3.3.2 automatically solve the joint support problem. Therefore, when the rules of arc generation are applied, there is no extra effort needed to handle the joint support problem.

Most ultimate pit limit design algorithms such as floating cone method and maximum network flow algorithm had difficulty in handling joint support and re-allocation problems both of which are necessary to achieve the optimal pit limit. In the new algorithm, property 6 plays a very important role in solving re-allocation problems.

When a non-root waste vertex, say vertex $v_j$, of a negative tree can be jointly supported by another non-root ore vertex, say vertex $v_i$, of a positive tree, an arc can not be simply added between waste vertex $v_j$ and ore vertex $v_i$, since each vertex can have only one predecessor and the predecessors of non-root vertices are already used. Prior to connecting an arc between these two vertices, they must be transformed into the roots of the new trees. Only after vertex $v_j$ is made into the root of the negative tree and vertex $v_i$ is made into the root of the positive tree, an arc can then be added between the two root vertices according to the rules of arc generation. It is very important to realize that generation of an arc for any root vertex only requires updating the current value and the predecessor for this root vertex, without any influence on all other non-root vertices in the same tree.

According to property 3, the current value of the root vertex of a tree is the total mass of that tree. By rearranging the arcs to transform the desired non-root vertex into the root vertex, the total mass of the vertices on the new tree is transferred to the interested vertex, new root, thus solving the re-allocation problem.

Figure 3.10 (a) shows a 4 block layout. The waste vertices $v_1, v_2$ and $v_3$ can be supported by ore vertex $v_4$ in one of the three patterns as shown in figures 3.10 (b), (c) and (d), which are all negative trees with total mass of $-1$ located at their
root vertices $v_3$, $v_2$ and $v_1$, respectively. Although each tree can be obtained from another one through rearranging arcs by applying property 6, each one of the three trees has different manner of allocation. For example, tree in figure 3.10 (b) means that waste vertices $v_1$ and $v_2$ are completely offset by ore vertex $v_4$ and waste vertex $v_3$ still has $-1$ unit profit left. After it is rearranged into the pattern of figure 3.10 (c) by making vertex $v_2$ into root vertex, the remaining $-1$ unit profit is transferred to waste vertex $v_2$. Similarly, the remaining $-1$ unit profit can be transferred to waste vertex $v_1$ by turning $v_1$ into root vertex in figure 3.10 (d).

Figures 3.11 (a) through (d) shows an identical example for positive trees. The same comments apply to figure 3.11 as they did in figure 3.10, that is, the total mass (+2) of the tree can be easily transferred to any one of the three ore vertices by rearranging arcs.

Figures 3.12 (a) consists of two trees. Tree 1 is a negative tree with total mass of $-1$ located at its root vertex $v_2$ and tree 2 is a positive tree with total mass of +2 located at its root vertex $v_3$.

After the two trees in figure 3.12 (a) are created, it is found that waste vertex $v_3$ in negative tree 1 can be jointly supported by ore vertex $v_6$ in positive tree 2. An arc can not be directly added between $v_3$ and $v_6$, because the total mass ($-1$) of negative tree 1 is located at its root vertex $v_2$ instead of vertex $v_3$, and the total mass (+2) of positive tree 2 is located at its root vertex $v_8$ instead of vertex $v_6$. But, by applying property 6, vertices $v_3$ and $v_6$ can be easily turned into the root vertices through reversing arcs on chains $v_2 - v_4 - v_3$ in tree 1 and $v_8 - v_5 - v_6$ in tree 2. By doing that, the total mass of tree 1 is re-allocated to its new root vertex $v_3$ in tree 1" and the total mass of tree 2 is re-allocated to its new root vertex $v_6$ in tree 2" (see figure 3.12 (b)). Now, an arc can be added from ore vertex $v_6$ to waste vertex $v_3$. Figure 3.12 (c) shows the combined tree. Vertex $v_6$ is the root of the combined tree with total mass of $+1$. 

Figure 3.10  Different Supporting Patterns of Negative Trees

Figure 3.11  Different Supporting Patterns of Positive Trees
Figure 3.12 An Example of Re-allocation
In figure 3.12 (c), waste vertex $v_3$ is jointly supported by ore vertices $v_4$ and $v_6$, waste vertex $v_5$ is jointly supported by ore vertices $v_6$, $v_7$ and $v_8$, and ore vertex $v_6$ is shared by waste vertices $v_3$ and $v_5$. This example sufficiently demonstrates that this new algorithm can adequately handle both the joint support and re-allocation problems.

### 3.6 The Specific Steps of The Algorithm

The initial directed graph is constructed by setting the value of each vertex equal to the corresponding block value and setting the predecessor of each vertex equal to null, that is, each vertex is a small tree of itself.

The algorithm uses an iterative process that transforms the initial directed graph into many directed trees. The algorithm stops when the set of the positive trees becomes the maximum closure of the initial 3-D mine block model.

The iterative process consists of five steps.

**Step 1.** If there exists a waste vertex, say vertex $v_j$, which belongs to a negative tree and must be removed in order to expose an underlying ore vertex, say vertex $v_i$, which belongs to a positive tree (see figure 3.13), go to step 2. Otherwise, go to step 5.

**Step 2.** If waste vertex $v_j$ is the root of the negative tree, then go to step 3. Otherwise, apply property 6 mentioned earlier to rearrange the arcs on the chain from vertex $v_j$ to the root vertex of the negative tree to transform waste vertex $v_j$ into the root of the negative tree.

**Step 3.** If ore vertex $v_i$ is the root of the positive tree, then go to step 4. Otherwise, rearrange the arcs on the chain from vertex $v_i$ to the root vertex of the positive tree to convert ore vertex $v_i$ into the root of the positive tree.

**Step 4.** Add an arc between ore vertex $v_i$ and waste vertex $v_j$ by applying
the rules of arc generation. That is, add a positive arc from \( v_i \) to \( v_j \) if the current value of ore vertex \( v_i \) is greater than the current absolute value of waste vertex \( v_j \). Otherwise, add a negative arc from waste vertex \( v_j \) to ore vertex \( v_i \). Go to step 1.

Step 5. Stop the iteration. The final graph is defined as the stable graph. The set of the vertices of all the positive trees in the stable graph is a maximum closure of the mine model.

\[ \text{Figure 3.13 Waste Vertex } v_j \text{ of the Negative Tree is A Restricting Vertex to Ore Vertex } v_i \text{ of the Positive Tree} \]

3.7 Optimality and Convergence

3.7.1 Optimality Theorem

Optimality Theorem:

*In the final stable graph, the set of all the vertices which belong to the positive trees is a maximum closure of both the final stable graph and the original mine model.*
Some notations are defined as follows:

\( G = (V, A) \) is the graph used to describe the mine model.

\( V \) is the set of all the vertices used to represent the blocks in the mine model.

\( A \) is the set of all the arcs generated by the algorithm.

\( G^t \) is the graph after iteration \( t \).

\( G_p = (V_p, A_p) \) is a subgraph defined by all the vertices \( V_p \) of the positive trees in the final stable graph.

\( V_p \) is the set of the vertices of all the positive trees in the final stable graph.

\( G_n = (V_n, A_n) \) is a subgraph defined by all the vertices \( V_n \) of the negative trees in the final stable graph.

\( V_n \) is the set of the vertices of all the negative trees in the final stable graph.

\( Z \) is the set of all the vertices of the maximum closure of the final stable graph.

\( V - Z \) is the set of all the vertices not belonging to the maximum closure of the final stable graph.

\( \Gamma(v_i) \) = all vertices that must be removed in order to expose vertex \( i \).

\( V \) can be partitioned into two subsets in two ways, namely, \( V_p \) and \( V_n \) or \( Z \) and \( V - Z \). \( \Gamma(v_i) \) is a mapping function which maps vertex \( v_i \) into all the restricting vertices to vertex \( v_i \).

Proof:

[1] First, we shall prove that the set of the vertices of all the positive trees in the final stable graph is a closure of both the final stable graph and the original mine model.
By definition, a closure is any surface satisfying the slope constraints. That is, the blocks above the surface defined by the closure satisfy the slope constraints. When the initial graph is constructed and the algorithm (iterative process) is applied, iteration \( t + 1 \) transforms a directed graph \( G^t \) into a new directed graph \( G^{t+1} \) with some new arcs being added and some old arcs being removed. The iterative process continues until the graph becomes stable. That is, no more negative trees are "above" positive trees. In the stable graph, the following relationship holds:

\[ \Gamma(v_i) \subseteq V_p, \text{ for all } v_i \subseteq V_p \]

Then, it immediately follows that \( V_p \), the set of the vertices of the positive trees in the final stable graph, is a closure.

[2] Next, we shall prove that if a vertex \( v_k \) of a positive tree \( T \) belongs to the maximum closure \( Z \) of the final stable graph, then all the vertices of this tree also belong to \( Z \). The above proof proceeds by showing that if vertex \( v_i \) of the positive tree \( T \) does not belong to \( Z \), then \( Z \) is not a maximum closure. We assume \( v_k \subseteq Z; v_i \subseteq V - Z; v_k \) and \( v_i \) are vertices of positive tree \( T \) in the final stable graph.

Any chain of a tree constructed by the algorithm has the following structure:

\[ \ldots, v_{ore} - v_{waste} - v_{ore} - v_{waste} - \ldots \]

That is, the ore and waste vertices appear interchangeably on the chain. In any pair of the \( v_{ore} - v_{waste} \) vertices linked by an arc, the waste vertex \( v_{waste} \) is always at the top end of the arc and the ore vertex \( v_{ore} \) is always at the bottom end of the arc (see figure 3.14), regardless of complete support or partial support. Also, the overlying waste vertex \( v_{waste} \) must be a restricting vertex to the underlying ore vertex \( v_{ore} \). Otherwise, the arc between \( v_{ore} \) and \( v_{waste} \) will not be generated by the algorithm.

In figure 3.15, \( Z \) consists of all the vertices within the maximum closure, and \( V - Z \) is the complement of \( Z \). As the chain from \( v_k \) to \( v_i \) passes the border
between \( Z \) and \( V - Z \), there must be one arc to connect the first vertex \( v_p \) on \( Z \) side and the first vertex \( v_q \) on \( V - Z \) side (possibly \( v_p = v_k \) and \( v_q = v_l \)). Then, \( v_p \) must be a waste vertex and \( v_q \) must be an ore vertex since \( Z \) is a closure and also due to the structure shown in figure 3.14. After removing the arc between vertices \( v_p \) and \( v_q \), the original positive tree \( T \) will be cut into two parts. The part of \( T \) containing vertex \( v_q \) inside \( V - Z \) is strictly positive. Otherwise, the arc between vertices \( v_p \) and \( v_q \) would not have been added. The tree \( T \) may intersect the border between \( Z \) and \( V - Z \) several times. But as each time it passes the border, the first vertex of tree \( T \) on \( Z \) side must be a waste vertex and the first vertex of tree \( T \) on \( V - Z \) side must be an ore vertex in order to keep \( Z \) as a closure. In any case, the part of tree \( T \) inside \( V - Z \) is strictly positive. Therefore, \( Z \) is not a maximum closure if some vertices of a positive tree is not included in \( Z \) since a larger mass can be reached by including all the vertices inside \( V - Z \) of the positive tree.

![Diagram](image)

- \( \bullet \) = waste vertex
- \( \circ \) = ore vertex

Figure 3.14 The Structure of the Chain of A Tree
(Relationship between Ore Vertices and Waste Vertices)
Figure 3.15  A Tree Intersecting the Border Between $Z$ and $V - Z$

A simple example as shown in figure 3.16 can be used to clarify the above relationship. In figure 3.16 (a), $Z$ is a maximum closure with total mass of positive 8. After positive vertex $A$ is put back to $V - Z$ as shown in figure 3.16 (b), the remaining vertices of $Z'$ are still a closure, since all the vertices within $Z'$ satisfy the slope constraints. But, $Z'$ is not a maximum closure since the total mass of $Z'$ is 3, which is less than 8 given by including positive block $A$ into the closure. But, negative vertex $B$ can not be excluded alone from the closure since vertex $B$ is a restricting vertex to positive vertex $A$. The remaining vertices of $Z''$ as shown in figure 3.16 (c) are not a closure although a larger total mass can be realized by excluding negative vertex $B$. The above example says that since removal of some positive vertices of positive trees within the maximum closure, such as vertex $A$ in figure 3.16 (b), may not violate the slope constraints, the remaining vertices of $Z'$ may still be a closure although it will not be the maximum closure. However, after any negative vertex of a positive tree within the maximum closure, such as vertex $B$ in figure 3.16 (c), is excluded from the closure, the remaining vertices of $Z''$ will not be a closure any more.
Figure 3.16  Relationship between Vertices of A Positive Tree with the Maximum Closure.
In summary, a negative vertex of a positive tree belonging to the maximum closure can not be excluded due to the slope constraints. Even if removal of a positive vertex of a positive tree belonging to the maximum closure may not violate the slope constraints, the remaining vertices will have smaller total mass.

[3] Now, we prove that if Z is a maximum closure, then, no vertex of a negative tree is inside Z. This can be proven by showing that if a vertex, \( v_k \), of a negative tree T is contained within Z, Z will not be a maximum closure.

Due to the structure shown in figure 3.14, we can not simply include the ore vertices of a negative tree into Z, since Z is a closure. The same arguments used in [2] will apply here too. As a negative tree T intersects the border between Z and \( V - Z \), the first vertex on Z side must be a waste vertex and the first vertex on \( V - Z \) side must be an ore vertex as Z is a closure. The part of tree T inside Z must be non-positive. Z may have a larger mass by removing the part of tree T inside Z. As such, the maximum closure Z should not have any vertices of a negative tree.

Figure 3.17 is an example with the same blocks as figure 3.16. After negative vertex C is included into Z as shown in figure 3.17 (b), the augmented vertices of \( Z' \) are still a closure, since adding vertex C to Z will not violate the slope constraints. But, \( Z' \) is not a maximum closure since the total mass of \( Z' \) is 6, which is less than 8 as before vertex C is added. On the other hand, positive vertex D can not be included into Z alone since vertex D is restricted by negative vertex C. The augmented vertices of \( Z'' \) as shown in figure 3.17 (c) are not a closure although a larger total mass can be realized by including vertex D along. The above example shows that if a negative vertex of a negative tree, such as vertex C in figure 3.17 (b), is included into the maximum closure, the augmented vertices of \( Z' \) may still be a closure although it will not be the maximum closure. However, if any positive vertex of a negative tree, such as vertex D in figure 3.17 (c), is included into the closure, the augmented vertices of \( Z'' \) will not be a closure any more.
Figure 3.17  Relationship between Vertices of A Negative Tree with the Maximum Closure.
In summary, a positive vertex of a negative tree can not be included into the maximum closure due to the slope constraints. Even if adding of a negative vertex of a negative tree into the maximum closure may not violate the slope constraints, the augmented vertices will have smaller total mass.

[4] If we put [1], [2] and [3] together, we can immediately draw the conclusion that the maximum closure of the final stable graph is the set of the vertices of all the positive trees in the final stable graph, that is, \( Z = V_p \).

[5] Finally, we will show that the maximum closure of the final stable graph is also the maximum closure of the original mine model. The mining constraints in the original mine model can be represented by arcs from each ore vertex to its all restricting waste vertices. We can treat the final stable graph as a partial graph of the original mine model due to the following reasons: (a) all the arcs in the final stable graph are generated according to the mining constraints, (b) the final stable graph contains the same vertices as the original mine model, and (c) the final stable graph has less restricting arcs than the original mine model. Since (1) \( V_p \) is a closure of the original mine model, (2) \( V_p \) is a maximum closure of the final stable graph, which is a partial graph of the original mine model, and (3) the mass of the maximum closure of the partial graph is always greater than or equal to that of the original mine model, the set of the vertices of all the positive trees in the final stable graph must be a maximum closure of the original mine model.

If the set of the vertices of all the positive trees in the final stable graph is empty, that is, \( V_p = \phi \), then the maximum closure is empty.

This completes the proof of optimality theorem.
3.7.2 Convergence Theorem

Transformations:

The outline of the algorithm given in section 3.6 does not indicate the amount of calculations involved in each iteration. To clarify these points we shall analyze the operations required in each step of the algorithm.

[1] Step 1 is the search step to find if there exists a waste vertex of a negative tree, which is a restricting vertex to an ore vertex of a positive tree. The operations required by this search step mainly depend on the location of the interested ore vertex. The maximum number of vertices to be searched is equal to the total number of the restricting vertices to that ore vertex, which is certainly a finite number. This step will find a pair of vertices, namely, waste vertex $v_j$ of a negative tree and ore vertex $v_i$ of a positive tree (see figure 3.13).

[2] Steps 2 and 3 are re-arranging steps. Suppose that there are $n_1$ arcs on the chain from waste vertex $v_j$ to the root vertex, $v_{\text{root-neg}}$, of the negative tree and $n_2$ arcs on the chain from ore vertex $v_i$ to the root vertex, $v_{\text{root-pos}}$, of the positive tree. By applying property 6, a maximum of $4 \times (n_1 + n_2)$ operations will transform waste vertex $v_j$ into the root vertex of the new negative tree and ore vertex $v_i$ into the root vertex of the new positive tree.

[3] Step 4 requires 2 operations to combine the two trees rooted at vertices $v_j$ and $v_i$ into one new tree.

After the above examination, it is clear that each iteration will require a finite number of operations. But the above analysis does not prove that the process will terminate in a finite number of iterations. Next, we shall prove that a maximum closure will be found in a finite number of iterations.
Convergence Theorem:

In applying the steps of the algorithm, a maximum closure of the original mine model is obtained in a finite number of iterations.

Some notations are defined as follows:

- $M_p^t$ is the total mass of the vertices of all the positive trees after iteration $t$.
- $V_p^t$ is the set of the vertices of all the positive trees after iteration $t$.
- $V(j)$ is the current value of waste vertex $v_j$ which is identified in step 1 of the algorithm, after it is transformed into the root vertex by step 2.
- $V(i)$ is the current value of ore vertex $v_i$ which is identified in step 1 of the algorithm, after it is transformed into the root vertex by step 3.
- $X_j$ is the set of the vertices of the negative tree rooted at waste vertex $v_j$.
- $X_i$ is the set of the vertices of the positive tree rooted at ore vertex $v_i$.

Proof:

The graph $G^t$ can be characterized by $V_p^t$, the set of the vertices of all the positive trees, and the mass $M_p^t$ of this set. In order to prove the theorem, we only have to show that the set of all positive trees cannot repeat itself in the sequence $V_p^0, V_p^1, \ldots, V_p^n$. We shall show that either $M_p$ decreases during an iteration or else $M_p$ stays constant but the set $V_p$ increases.

Initially, $M_p^0$ is the sum of the values of all the ore vertices and $V_p^0$ contains all the ore vertices. The iteration $t + 1$ will update $M_p^t$ and $V_p^t$ as follows:

- if $V(i) \leq |V(j)|$ then $M_p^{t+1} = M_p^t - V(i)$
  
  $V_p^{t+1} = V_p^t - X_i$ 

- if $V(i) > |V(j)|$ then $M_p^{t+1} = M_p^t + V(j)$

  $V_p^{t+1} = V_p^t + X_j$
That is, after re-arranging arcs by applying steps 2 and 3, if ore vertex \( v_i \) can completely offset waste vertex \( v_j \), a positive arc, \( a_{ij} \), combines the two trees into one positive tree and all the vertices connected with the waste vertex \( v_j \) will be included in the set of the positive trees; otherwise, a negative arc, \( a_{ji} \), combines the two trees into one negative tree and all the vertices connected with ore vertex \( v_i \) have to leave the set of the positive trees.

In any case, \( M_t^{t+1} \leq M_t \) because of \( V(j) \leq 0 \) and \( V(i) > 0 \). When \( M_t^{t+1} = M_t \) due to \( V(j) = 0 \), the set \( V_p^{t+1} \) is larger than the set \( V_p^t \) since the set \( X_j \) is included in the set \( V_p^{t+1} \).

This completes the proof.

3.8 Programming Experience

Since different programmers may have different programming styles and skills, the algorithm described above can be implemented in many different ways. However, the following experiences and strategies may be universally useful to make the program more efficient.

3.8.1 Shell Cone Template

Due to the mining sequence requirements and the variable slope angles for different sectors in the pit, one has to identify which waste block in the model should be removed before a particular ore block can be mined. Many iterations are needed to determine the overburden for an ore block. This procedure has to be repeated for all the ore blocks in the mine model for many times in order to establish the optimal pit limit. Therefore, it is the main operation of the algorithm. A shell cone template is introduced in this section. For the sake of simplicity, 45 degree slopes at all directions and cubic blocks in the mine model are assumed in the following figures. It should be emphasized that the shell cone template works well for variable slopes and rectangular blocks.
Figure 3.18  Shell Cone Template
Figure 3.18 (a) shows a shell cone template which can be used to determine the overburden for an ore block. One only needs to place the shell cone template on a specific ore block and to check all the waste blocks within the shell. The above idea can be realized by performing a vertical search inside the shell cone by applying the following properties:

1. If a block is within the cone template, then, all the above blocks on the same column are within the cone template.

2. The lower levels always penetrate the upper levels in the shell cone template.

Figure 3.18 (b) shows the computer representation of the shell cone template. One array is used for the relative index coordinates in X-direction and the another one is used for the relative index coordinates in Y-direction. The vertical direction increments can be easily applied by DO-LOOP in computer program. For a particular ore block, cone can be generated by adding the relative X and Y index coordinates and the vertical increments to the index coordinates of that ore block. One block within the cone can be determined by only three addition operations.

3.8.2 Model Reduction

The main purpose of model reduction is to reduce the physical size of the model which still contains the optimum pit limit. Nowadays, there is an increasing use of micro-computers in mine design. Therefore, efficient use of the limited computer memory is more important.

The model reduction method recommended is very simple, fast and easy to implement. It assumes that each ore block can support all the overburden above it. The reduced model contains all the ore blocks in the mine model and the waste blocks which are the restricting blocks to the ore blocks. The procedure of the model reduction is as follows:
(a) Locate the first ore block in a *bottom-up* search fashion.

(b) Set up a cone on this ore block using the shell cone template and remove all the blocks within the cone by lowering the initial surface without calculating the profitability of the cone.

(c) Find the next ore block, which is not removed by the previous cones, in the *bottom-up* search fashion and place a cone on this ore block and lower the previous surface to the current surface.

(d) Repeat step (c) until all ore blocks are removed.

The final surface is used as the maximum possible pit limit (or the outer bound). Only the blocks between the initial surface and the maximum possible pit limit are used for pit optimization. The above model reduction is very substantial. Usually, only 20 to 40 percent of the entire model falls between the initial surface and the maximum possible pit limit. Figure 3.19 schematically shows the model size reduction in a 2-D cross section.

The bottom-up search strategy is very efficient in model size reduction, since when a cone is set up at the most bottom ore block, all the blocks (regardless of ore or waste) within the cone can be removed at once. The maximum number of the cones required to reduce the model is the number of the vertical columns in the model. In the example of figure 3.18, only three cones, that is, one cone at ore block A, one cone at ore block B and the other one at ore block C, are required to reduce the model. Figure 3.20 shows the reduced model. Usually, it only takes several minutes to reduce a practical, large mine model.
Figure 3.19 Model Size Reduction

Figure 3.20 The Blocks Used for Optimization
3.8.3 Search Procedure

After model reduction, there are two surfaces, i.e., the initial surface and the maximum possible pit limit. The blocks between the two surfaces should be stored into the computer memory by vertical columns. After the initial directed graph is constructed, the following search procedure can be used to search the ore and waste vertex pairs in the process to add arcs:

(a) Find the first ore vertex which belongs to a positive tree for a particular column in a bottom-up search fashion.

(b) Set up a search cone on this ore block by applying the shell cone template.

(c) Find a waste vertex within this cone which belongs to a negative tree. If such a waste vertex is not found, go to (e).

(d) Add an arc between the ore vertex and the waste vertex according to steps 2, 3 and 4 as described by the algorithm. After the arc is added, if the ore vertex still belongs to a positive tree, go to (c). Otherwise, find the next ore vertex belonging to a positive tree on the same column in the bottom-up search fashion. If such an ore vertex is found, go to (b). Otherwise, go to (e).

(e) Move to the next column and go to (a).

The above procedure is repeated until the maximum closure is found. The bottom-up search procedure can considerably reduce execution times.

In figure 3.21, A and B are two ore vertices on the same column. Vertex A is under vertex B. By applying the shell cone template on the two blocks, two cones, i.e., cones A and B, can be generated, respectively. If all the waste vertices in the cone A already belong to the positive trees, it is not necessary to set up cone B to search the waste vertices within it since cone B is completely within cone A. Whenever the above situation occurs, after searching the cone A, the search
procedure will go to the next column instead of generating cone B. Therefore, this kind of unnecessary repetition can be avoided by the above search procedure. Usually, most of the arcs are added in the first several iterations. It is very common that only a few of arcs are added in each of the later iterations. By avoiding all the unnecessary repetitions, the above search procedure will speed up the convergence for the optimum solution.

![Diagram showing Cone B is Completely within Cone A](image)

Figure 3.21 Cone B is Completely within Cone A

If the moving cone algorithm is used, a top-bottom search procedure must be used. Otherwise, moving cone algorithm will have serious problem of over-mining due to the "free riding effect", since moving cone method will remove all the blocks within a cone if the cone is profitable. It must be emphasized that, in the new algorithm, a bottom-up search process will not cause the over-mining problem, because no single block is removed by the new algorithm before the final optimum results are obtained.

3.8.4 Optimization by Layers

It was found that optimization by layers required less computation time than optimization of the whole mine model at once for all the cases. Suppose
that the total number of benches in the mine model is equal to \( N_z \). The so-called "optimization by one run" approach is simply to run the program only once for the entire model from the first bench to the most bottom bench. While the "optimization by layers" approach requires dividing the total number of benches \( N_z \) into \( N_{\text{run}} \) increments and then running the program \( N_{\text{run}} \) times for the entire model such that the ultimate pit limit from the previous run is used as the input surface for the next run as shown in figure 3.22. The reduction in the computation time by use of the method of optimization by layers depends on the mine model as well as the number of benches of each sub-run. When the number of benches of each sub-run is less than 10, optimization by layers can reduce the total computation time by 50 percent to 75 percent as compared to the time required by optimizing the whole model at once. Table 3.1 summarizes the cumulative computation times required to optimize the whole mine models described in table 3.2 by layers versus the different number of benches of each sub-run. Figure 3.23 graphically illustrates the above summary of the total computation times.

![Figure 3.22 Pit Limit Optimization by Layers](image)
Table 3.1  Computation Time of Optimization by Layers

Cumulative computation times used to optimize the whole models by layers

<table>
<thead>
<tr>
<th>Number of benches of each sub-run</th>
<th>Cumulative computation times (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Model 1</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
</tr>
<tr>
<td>4</td>
<td>83</td>
</tr>
<tr>
<td>6</td>
<td>71</td>
</tr>
<tr>
<td>8</td>
<td>74</td>
</tr>
<tr>
<td>10</td>
<td>99</td>
</tr>
<tr>
<td>20</td>
<td>160</td>
</tr>
<tr>
<td>40</td>
<td>209</td>
</tr>
</tbody>
</table>

Figure 3.23  Computation Time of Optimization by Layers
Optimization by layers requires less computation time than optimization of the whole mine model at once. This is particularly important if the microcomputer is used for very large mine model. If the method of optimization by layers is used, it is always possible to store only the necessary information using the memory for each sub-run. Actually, it is not hard to let the program automatically input the previous ultimate pit limit as the input surface for the next run, load the necessary blocks into the memory and carry out the each sub-run until the whole model is optimized. In this manner, the user only needs to execute the program once although the entire model is optimized by several sub-runs inside the program.

3.9 Case Studies

An operational computer program called "MINER6B" for the ultimate pit limit design was implemented using the new graph theory algorithm. The computer program was coded in FORTRAN 77. Two case studies on two distinct real deposits were carried out for the following situations:

Computer:

IBM AT 80286 compatible with a 12 mega hertz processor.

These two deposits are:

1. a massive copper molybdenum deposit,
2. a vein type gold deposit.

The main inputs to the program:

1. the initial surface file,
2. the economic file which contains the net profit value of each individual block in the 3-D mine model.
The slope angle:

45 degree in all direction.

The model descriptions and the output summary are given in the table 3.2.

Table 3.2 Model Descriptions and the Summary Output

<table>
<thead>
<tr>
<th>Item</th>
<th>Case 1 (Cu-Mo)</th>
<th>Case 2 (Gold)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Block dimension</td>
<td>80'x 80'x 40'</td>
<td>40'x 40'x 20'</td>
</tr>
<tr>
<td>2. Initial model size</td>
<td>50 x 50 x 40</td>
<td>80 x 80 x 40</td>
</tr>
<tr>
<td>-ore blocks</td>
<td>9,522</td>
<td>24,031</td>
</tr>
<tr>
<td>-waste blocks</td>
<td>59,702</td>
<td>120,211</td>
</tr>
<tr>
<td>-air blocks</td>
<td>30,776</td>
<td>111,785</td>
</tr>
<tr>
<td>-total blocks</td>
<td>100,000</td>
<td>256,000</td>
</tr>
<tr>
<td>3. Reduced model size</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-ore blocks</td>
<td>9,522</td>
<td>24,031</td>
</tr>
<tr>
<td>-waste blocks</td>
<td>15,497</td>
<td>28,294</td>
</tr>
<tr>
<td>-total blocks</td>
<td>25,019</td>
<td>52,325</td>
</tr>
<tr>
<td>4. Blocks within optimum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pit limit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-ore blocks</td>
<td>8,802</td>
<td>21,152</td>
</tr>
<tr>
<td>-waste blocks</td>
<td>8,663</td>
<td>14,934</td>
</tr>
<tr>
<td>-total blocks</td>
<td>17,465</td>
<td>36,086</td>
</tr>
<tr>
<td>5. Processing time (in minutes)</td>
<td>71</td>
<td>115</td>
</tr>
</tbody>
</table>

3.10 Program Validation

Although the algorithm is optimal, it does not necessarily mean that the program is optimal. Some validations were carried out to check if the results from the program were optimal. One most important means of validation was to compare the results of "optimization by one run" with that of "optimization by layers". Since the results obtained from the two kinds of runs are exactly the same for all the cases, it seems that the program is also optimal.
3.11 Comparison Between The New Graph Algorithm and Lerchs-Grossmann Graph Algorithm

The Lerchs-Grossmann (LG) algorithm is a graph algorithm which formulates the block mine model into a special graph consisting of a big tree. We will show why the new graph algorithm is better than the LG algorithm by comparing the steps of each algorithm.

The general steps of the LG algorithm given in section 2.2 are as follows:

1. initialization (step 1),
2. search (step 2),
3. connection and transformation (step 3),
4. normalization (step 4),
5. stop (step 5).

The new algorithm given in section 3.6 consists of the following steps:

1. search (step 1),
2. re-arranging arcs (step 2 and 3),
3. connection (step 4),
4. stop (step 5).

The step 2 of the LG algorithm is completely equivalent to that of the new algorithm step 1. But, the new algorithm is able to do this job much more efficiently than LG algorithm since the new algorithm can utilize properties 1 and 2 given in section 3.4 to easily recognize the ore or waste vertices by checking if the current values of the vertices are positive or negative, and identify the positive or negative
trees by simply checking if their root vertices are ore or waste vertices. The only way for LG algorithm to recognize the ore or waste vertices is to check their initial values. It means that LG algorithm must use additional memory to keep the initial values. But, the memory shortage is always a problem in pit limit optimization since a large number of blocks are usually required in mine model. In addition, LG algorithm has to somehow sum all the values of the vertices of a branch in order to find if the branch is positive or negative, which is certainly more time consuming than the way it is performed by the new algorithm.

Steps 2, 3 and 4 of the new algorithm have the same function as LG algorithm step 2. The new algorithm takes \([4 \times (n_1 + n_2) + 2]\) operations (see section 3.7.2) to re-arrange the arcs and connect the two trees into one new tree. LG step 2 will remove the arc between vertex \(v_{zr}\) and \(x_0\) and add an arc from vertex \(v_{zr}\) to vertex \(v_y\) (see figure 2.3 and 2.4). Next, LG step 2 has to perform the transformation of masses for all the vertex on chain \((v_{y1}, \ldots, v_y, v_{z1}, \ldots, v_{zr})\) (figure 2.4), and to change the status for all the edges on chain \((v_{z1}, \ldots, v_{zr})\) from p-edge to m-edge and vice versa. Since LG algorithm’s p-edge and m-edge, which are determined by the relative direction to the dummy root, are more complex to handle than the notations of positive or negative arcs, which are directly defined by the support concept (i.e., complete or partial support) used by the new algorithm, there is no doubt that LG algorithm will need more time to do the same job.

The normalization step is the biggest drawback of the LG algorithm since this step valuates all the vertices on the chain \((x_0, v_{y1}, \ldots, v_y, v_{z1}, \ldots, v_{zr})\) (figure 2.4) to check if each vertex is connected to a strong edge. All the strong edges on the above chain must be removed one by one. After a strong edge, say edge \(e_n = (x_a, x_b)\), is removed from the branch and connected to the dummy root \(x_0\), the mass of that strong branch must be subtracted from all the remaining edges on the chain \((x_0, \ldots, x_b)\). The above process is repeated until the last strong edge has been removed from the chain. As it can be seen, besides programming difficulties, the processing time required by the normalization step is considerable. However,
this main step is completely eliminated in the new algorithm. As a matter of fact, the elimination of the normalization step not only considerably reduces the computation time, but also greatly simplifies the programming.

From the preceding discussion, it is quite easy to find why the new graph theory algorithm is much faster and simpler than the well known Lerchs-Grossmann graph theory algorithm.

Direct comparison between the new graph theory algorithm with the LG algorithm is not possible since the available program of LG algorithm at the University of Arizona is the one from Whittle Programming (Whittle, 1986), which can not handle the mine models described in table 3.1 on the PC AT microcomputer. But an indirect comparison can be made if the results obtained from the new graph algorithm are compared with the results obtained by P.Huttagosol (1988, 1989) using LG graph algorithm. His model size and block dimension are as follows:

Model size (column x row x level): 75 x 75 x 40
Block dimension: 80' x 80' x 40'

While his model size is little smaller than that of case 2 in table 3.1, his block dimension has the same ratio of block length to block height as the model of case 2 in table 3.1, and thus satisfying the condition for comparing the new graph algorithm with the LG graph algorithm.

P.Huttagosol ran his programs of the modified LG algorithm, which he claimed was much faster that the original LG algorithm, on a mainframe computer, a DEC VAX8600, to optimize the pit limit for his model. He obtained the pit limit using both the original LG algorithm and his modified version of LG algorithm in 1,628 and 535 minute VAX8600 CPU times, respectively. The comparison between 535 minute VAX8600 CPU time for a smaller model (225,000 blocks) with 115 minute PC AT 80286 processing time for a bigger model (256,000 blocks)
strongly indicates that the new algorithm is significantly faster than LG graph algorithm, although it is not known exactly how to convert the VAX CPU to PC AT processing time.

3.12 Summary Remarks

As illustrated by the above case studies, the new graph algorithm is able to generate an optimal ultimate limit for the relatively large model with 256,000 blocks on PC AT 80286 type microcomputer in less than two hours. If it is run on mainframe computers, it may solve the pit limit optimization problem for any practical, large mine model within several hours. The importance of programming skills should not be overlooked, since a good data structure in the program can make the algorithm more efficient or vice versa. Although some strategies and experiences are recommended in the suggested mode of implementation, it does not mean that the algorithm must be programmed in the same way.

In mine planning, although the most important economical information is the production scheduling, the importance of an optimal ultimate pit limit algorithm is undeniable. It appears that the final pit limit is a prerequisite for the determination of the long term planning. Moreover, the ultimate pit limit algorithm is always used as a basic tool in long term planning. As will be seen in the next chapter, the ultimate pit limit algorithm plays a key role in either the proposed "true optimum" production scheduling approach or the most popular "pushback" approach.
CHAPTER 4

INVESTIGATION OF COMPUTERIZED TECHNIQUES IN MINE PRODUCTION SCHEDULING

4.1 Overview

After the ultimate pit limit is defined in 3-D space through an ultimate pit limit algorithm, the next step in mine planning is to design an extraction sequence to deplete the mineable ore reserves from the initial conditions of the deposit to the ultimate pit limit. The process, of developing a multi period extraction sequence over successive time periods, is usually referred to as long range mine planning, or mine production scheduling. The purpose of this chapter is to investigate the use of computerized techniques in mine production scheduling with the emphasis on the demonstration of the impossibility of applying the approach combining the Lagrangian relaxation with ultimate pit limit algorithm to obtain a “true optimum” solution. The concepts used to apply the Lagrangian relaxation method to mine production scheduling are not new, but this chapter will investigate how far the proposed Lagrangian relaxation method can go.

The main objective of the long range mine planning is to develop a most efficient extraction schedule which would provide the maximum profit for the mining company under the fixed mine capacity (or mining rate), existing economic conditions and technological constraints. The time span of each planning period in a long range mine plan can range from one year to several years. The most widely used project evaluation technique is the net present value (NPV) analysis. Since different extraction sequences will lead to different cash flows throughout the mine life,
the extraction sequences will directly influence the NPV of the mining investment. Thus, mine production scheduling is of vital importance to mine management.

The traditional trial and error approach used in production scheduling, prior to the application of computers, was done by hand. With advances in computer technology, most of the mine planning is now performed with the aids of computers. Although great efforts have been made to work out true optimization algorithms, the computerized trial and error "pushback" approach is still the most popular mine sequencing practice. It is very interesting that ultimate pit limit algorithms play the key role in both the proposed "true optimization" approach and the popularly used trial and error pushback approach.

Because of the importance of mine production scheduling, many attempts have been made to develop true optimization methods (Johnson, 1968; Davis-Williams, 1973; Dagdelen, 1985; and Elevli, et al., 1989) to solve this problem. It is always very desirable to generate the optimal solution by mathematical optimization methods. Unfortunately, a practical optimum algorithm for mine production scheduling has not been found. It is not difficult to fit the mine production scheduling problem into standard mathematical optimization models such as large scale integer linear programming. But in practice, the direct application of integer linear programming technique to the mine production scheduling problems is not possible due to the immense amount of variables and constraints. In order to solve the production scheduling problem, past attempts have focused on indirect approaches utilizing the special structure of the mine production scheduling and making use of the structure in the solution algorithm.

As discussed in chapters 2 and 3, the sequencing constraints must be satisfied in ultimate pit limit design. That is, a particular block can not be mined unless all the restricting blocks above defined by the specified slope angles are removed first. In production scheduling, in addition to the sequencing constraints, certain practical constraints, such as ore and waste tonnage requirements for each period
must be reasonably taken into account in order to efficiently utilize the equipment, manpower and resources, and to keep a smooth production process. As it can be seen, the only difference between the ultimate pit limit problem and the production scheduling problem is that the latter must satisfy certain additional capacity constraints. After these additional capacity constraints are removed through some mathematical manipulations while keeping the sequencing constraints, the mathematical representation of the production scheduling problem then has the same form as that of the ultimate pit limit problem. This special structure of the mine production scheduling problem provides the researchers some chance and hope to obtain the optimal solution to the production scheduling through available ultimate pit limit algorithms. The above idea has been considered as the possible direction for the researchers (Johnson, 1968; Davis-Williams, 1973; Dagdelen, 1985; and Elevli, et al., 1989) to develop true optimization approaches for the mine production scheduling.

In 1968, Johnson formulated the production scheduling problem as a large scale linear programming model, but the existing solution algorithms can not handle such a big linear programming problem due to the very large number of variables and constraints. Then, Johnson (1968) applied the Dantzig-Wolfe decomposition principle to decompose the problem into two separated tasks; namely the master problem which includes the capacity constraints (such as ore/waste tonnage requirements) and the subproblem which includes only the mining sequencing constraints. As a result, the subproblem is the ultimate pit limit problem that can be solved by the existing ultimate pit limit algorithms. In this manner, the optimal solution to the original production scheduling problem can be obtained by solving repeatedly the ultimate pit limit problems many times for successive iterations until the termination conditions are satisfied. Johnson also formulated the subproblem (ultimate pit limit problem) into a maximum network flow problem and suggested using available operations research techniques to solve the maximum network flow problem. Johnson furthermore suggested to decompose the T-multi-period mine
production scheduling problem into $T$ individual, smaller maximum network flow problems with each one corresponding to a given production period. The memory and running time requirements of Johnson's method are too excessive to be realistically applied in mine production scheduling. Johnson's work was probably the first attempt to achieve an optimal solution to the mine production scheduling subject to the capacity constraints. Although his method has never been implemented due to its obvious shortcomings since it was proposed two decades ago, his work is still an important contribution to mining industry.

In 1973, Davis and Williams tried to solve the multi-period production scheduling problem by taking the special structure of the problem into consideration. Their purpose was to determine which blocks of ore and waste should be mined in each production period subject to the mining and milling constraints. Since they realized that it was not practical to solve the optimization problem with respect to all constraints of all mining periods simultaneously, their approach was to solve each single period scheduling problem one at a time. At each period, their procedure was to divide the production scheduling into two separated problems by using an undetermined penalty parameters (or Lagrange multipliers). One of the problems is the ultimate pit limit problem, and the other involves the selection of the penalty parameters to the coefficients of the objective function. In their approach, several alternative schedules that satisfy the constraint limits are generated for each period, the multi-period schedules are then obtained by selecting the best combinations that could maximize the discounted cash flow over all the periods of the mine life using dynamic programming. Their application of Lagrange multiplier to production scheduling problem was quite crude and they did not provide any theoretical proof of optimality, but they indeed used the idea of penalty parameters to adjust the coefficients of the objective function in order to obtain mine pit limits of different sizes. Their work may be the first attempt to use Lagrange multipliers to systematically adjust the monetary values of the mining blocks. They realized the fact that the pit sizes varied inversely with the penalty parameters. As a re-
sult, the pits with different tonnages could be obtained by changing the penalty parameters.

Dagdelen (1985), and Elevli, et al. (1989) attempted to find an efficient and applicable solution algorithm for the linear programming model formulated by Johnson (1968) rather than restating the mine production scheduling problem in a different form. Although the basic idea of their approaches is exactly the same as that suggested by Davis and Williams (1973), they explained the mine production scheduling problem more mathematically than Davis and Williams did. Dagdelen (1985) divided mine production scheduling into two separate problems by applying Everett’s generalized Lagrange multipliers technique. One of the problems is the ultimate pit limit problem which can be solved by the existing ultimate pit limit algorithm, and the other involves the determination of a set of Lagrange multipliers using subgradient optimization technique proposed by Held-Karp (1973). Dagdelen also tried to solve the multi-period production scheduling problem defined by Johnson (1968) in order to take into the effect of the time value of money. Instead of solving the whole T-multi-period production scheduling problem at once, Dagdelen decomposed the T-multi-period problem into T individual problems with one problem for each period through manipulations of the economic data of mining blocks. In view of the difficulties in application of Dagdelen’s method for multi-period production scheduling, Elevli, et al. (1989) focused on the single period production scheduling problem by utilizing the Lagrange multiplier technique. Elevli’s work focused on the improvement of the procedure in determination of the Lagrange multipliers by utilizing the relationship between the Lagrange multipliers and the block values.

There is no doubt that the application of the Lagrange multiplier method has added another step to the optimal solution of the mine production scheduling. At least, the proposed Lagrange multiplier method helps people to understand the mine production scheduling problem more mathematically. But, at the
same time, it brings more unresolved problems. A practical approach to determine the Lagrange multipliers has not yet been found. The general condition of non-convergence (section 4.2.4) and the non-convergence due to the inability to detect the redundant optimal solutions (section 4.2.5) greatly limit the application of the proposed Lagrange multiplier method to the real situation. Although the above researchers claimed that they successfully developed the optimal solution algorithms for mine production scheduling problem, their algorithms have not been adopted by the mining industry. The difficulties involved in the determination of Lagrange multipliers are certainly hurdles for the new technique to be accepted by the mining industry, but the proposed approaches up to date have some critical limitations of their own as well.

Since the proposed mathematical optimization approaches for mine production scheduling so far can not really handle the practical problems, the most mining companies are using the computerized “pushback” approach in their long range mine design although such an approach can not guarantee the true optimum solution.

The computerized “pushback” approach has been used in mine production scheduling for a long time. Mathieson (1982) summarized the methodology of the overall pushback mine planning approach with emphasis on the design of an optimal sequence of open pit expansions. This type of pushback approach is also reported by Crawford (1976), Couzens (1978), Iles and Perry (1981), Mathieson (1982), Fiore (1986), Whittle (1988), Cai (1989), and Faust (1989).

The pushback approach involves repeatedly running an ultimate pit limit algorithm on a mineralization block model to obtain a series of nested pits by systematically changing cutoff grades or product prices. This approach is based on the fact that increasing or decreasing either cutoff grades or product prices, the size of the pit can be expanded or contracted. By successively changing cutoff grades,
a number of nested incremental pits can be generated which can in turn be used as the general directions to be followed in development of the pit.

In general, a larger pit is associated with lower cutoff grade and a smaller pit is associated with higher cutoff grade. The pits will expand outwards as the cutoff grades decrease. The inner most pit is the most profitable one because it is based on highest cutoff grade. Each outer pit increment is less profitable than its inner pits. Therefore, profitability decreases from the inner most pit to the outer most incremental pit. If the horizontal distance between any two of those nested pits is greater than the minimum mining width, these nested pits can serve as a series of pushbacks. A higher NPV can be achieved if higher return from a mining venture can be made as early as possible. This is the exact idea of "next best" philosophy of the pushback approach.

Although the pushbacks define a general direction for mine planning, i.e., where is the next best mineralization, practical situations may not permit simply mining the next best reserves following an inner pit to outer pit sequence. One constraints is the stripping ratio. As discussed by Mathieson (1982), the stripping ratio in one pushback may be small but can go sky high in the next pushback immediately following. To maintain a relatively smooth stripping ratio, some advance stripping may be necessary.

After the pushbacks are generated, a mining sequence must be somehow developed on the pushbacks through trial-and-error. For this purpose, a single period zero-one programming model is proposed by Cai (1989). The proposed model provides some systematic feature in mining sequence development and reduces the trial-and-error process to some extent. Multi-period mining schedules can be worked out by defining a sub-optimization objective for each mining period, then proceeding scheduling period by period.

Section 4.2 will briefly discuss the Lagrangian relaxation method in mine
production scheduling and the problems associated with its application. Section 4.3 will discuss the general procedure of pushback approach.

4.2 Investigation of the Approach Combining the Lagrangian Relaxation with the Ultimate Pit Limit Algorithm

The fundamental idea behind the Lagrange multiplier method is the incorporation of selected constraints in the linear programming model into the objective function in order to relax the original problem. The relaxed problem is called Lagrangian problem, which is designed to be easier to solve than the original problem. In the case of mine production scheduling, after the capacity constraints are combined into the objective function, the relaxed production scheduling problem will have the same form as that of the ultimate pit limit problem, which can be solved readily using an available ultimate pit limit algorithm. Therefore, the above solution scheme is called as the approach combining the Lagrangian relaxation with the ultimate pit limit algorithm.

The previous researchers already realized the existence of the general gap-condition (see section 4.2.4) which will cause the non-convergency. However, they have not realized that there are two other major problems associated with the application of the approach combining the Lagrangian relaxation with the ultimate pit limit algorithm in the mine production scheduling. One problem is the non-convergence due to the redundant (duplicated) optimal solutions (see section 4.2.5) in a given period for the specified ore and waste tonnage constraints. And the other one is the non-convergence due to the required advanced stripping (see section 4.2.6). The existence of redundant optimal solutions and the required advanced stripping always lead to non-convergence. Unless these two problems are solved, the proposed approach combining the Lagrangian relaxation with the ultimate pit limit algorithm cannot be used in mine production scheduling. This chapter will prove that there is no way to solve these two problems, if the ultimate pit limit
algorithm is used to solve the relaxed mine production scheduling problem.

Since it is not practical to solve the multi period production scheduling problem simultaneously, the multi period scheduling problem is eventually decomposed into a set of single period scheduling problems, with each problem for a period (Johnson, 1968; Davis-Williams, 1973; and Dagdelen, 1985). Then, the single period scheduling problems are solved one by one, with each period at a time. It is not the intent of this dissertation to discuss how to decompose the multi period scheduling problem into many single period scheduling problems. The interested readers are referred to the cited references. This section will focus only on the problem of single period scheduling.

This section will cover the following topics: formulation of the single period production scheduling problem into the mathematical optimization model, the theoretical background of the Lagrange multiplier method, application of the Lagrange multiplier method to mine production scheduling, non-convergence due to redundant optimal solutions, and non-convergence due to the practical requirement of advanced stripping.

4.2.1 Zero-one Programming Model for Mine Production Scheduling

The mineable blocks (or the blocks in 3-D mine model) are the basic design variables if the mathematical optimization model is used to formulate the mine production scheduling problem. There is a decision variable associated with each block in the mine model. Since the decision with respect to a block has only two choices, i.e., either mined or not mined, this kind of decision can be represented by decision variables that are restricted to just two values. Usually, these types of problems can be formulated into the standard zero-one programming models in which zero or one are chosen to be the values of the decision variables. The decision variable takes the value of one if the associated block is mined. Otherwise, the decision variable is equal to zero.
Mathematically, the standard zero-one programming problem can be stated in the following form:

\[
\text{Maximize } \sum_{j=1}^{n} c_j x_j \\
\text{Subject } \sum_{j=1}^{n} a_{ij} x_j \leq b_i
\]

\[(4.1)\]

where, 
- \(n\) is the number of decision variables;
- \(m\) is the number of constraints;
- \(c_j\)'s are the coefficients in the objective function;
- \(x_j\)'s are the decision variables of the problem;
- \(a_{ij}\)'s are the coefficients of the constraints;
- \(b_i\)'s are the right hand side of the constraints;

A feasible solution is a vector \(X = (x_1, x_2, \ldots, x_n)^T\) which satisfies the constraints in (4.1); here \(x_j\)'s will be either zero or one. An optimum solution is the feasible solution which maximizes the objective function.

In mine production scheduling, there are two types of constraints. The first type constraints involve sequential relations due to slope angles. The second type constraints involve design capacities, such as mining and milling capacities. Usually, the capacity constraints for the mine production scheduling problem are not considered as very rigid, and these constraints do not have to be exactly satisfied.
A reasonable error tolerance can be accepted in most cases. That is, any solution within certain error tolerance limits is accepted as an optimal solution. To formulate the mine production scheduling problem into zero-one programming model, the necessary variables and coefficients are defined as follows:

Let

$X_i$ be block $i$ in the mine model,

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

$C_i$ be the coefficient in the objective function representing monetary value of block $i$,

$$C_i = \begin{cases} 
\text{Revenue of block } i - \text{Costs of block } i, & \text{if block } i \text{ is ore}, \\
\text{Stripping cost (negative)}, & \text{if block } i \text{ is waste}.
\end{cases}$$

$I_i^O$ be the ore indicator factor for block $i$,

$$I_i^O = \begin{cases} 
1 & \text{if block } i \text{ is mined as ore}, \\
0 & \text{if block } i \text{ is mined as waste}.
\end{cases}$$

$I_i^W$ be the waste indicator factor for block $i$,

$$I_i^W = \begin{cases} 
1 & \text{if block } i \text{ is mined as waste}, \\
0 & \text{if block } i \text{ is mined as ore}.
\end{cases}$$

$T_i$ be the tonnage of block $i$.

$O^T$ be the ore tonnage requirement.

$W^T$ be the waste tonnage requirement.

$\varepsilon^O$ be the tolerance of ore tonnage requirement.

$\varepsilon^W$ be the tolerance of waste tonnage requirement.

$N$ be the total number of blocks in the mine model.

$\Gamma_i$ be the subset of blocks which must be mined in order to expose block $i$.

The above definitions of variables and coefficients permit the formulation of the objective function and the constraints of the production scheduling problem.
Objective function: \( \text{Max } Z(X) = \sum_{i=1}^{N} C_i X_i \) \hspace{1cm} (4.2)

Ore tonnage requirement: \( \sum_{i=1}^{N} I_i^O T_i X_i = O^T \pm \epsilon^O \)

Waste tonnage requirement: \( \sum_{i=1}^{N} I_i^W T_i X_i = W^T \pm \epsilon^W \)

Sequence constraints: \( X_i - X_j \leq 0, \text{ all } j \in \Gamma_i \)
\( X_i = 0, 1 \)
\( 1 \leq i \leq N \)

Problem (4.2) can be written into the form of (4.3), which is more commonly used in the zero-one integer programming problems.

Objective function: \( \text{Max } Z(X) = \sum_{i=1}^{N} C_i X_i \) \hspace{1cm} (4.3)

Ore tonnage requirement: \( O^T - \epsilon^O \leq \sum_{i=1}^{N} I_i^O T_i X_i \leq O^T + \epsilon^O \)

Waste tonnage requirement: \( W^T - \epsilon^W \leq \sum_{i=1}^{N} I_i^W T_i X_i \leq W^T + \epsilon^W \)

Sequence constraints: \( X_i - X_j \leq 0, \text{ all } j \in \Gamma_i \)
\( X_i = 0, 1 \)
\( 1 \leq i \leq N \)

Problem (4.3) is a zero-one integer programming problem for which general purpose algorithms have been developed. However, in most practical instances, this is a large scale constrained optimization problem which may involve hundreds of thousands of variables, and millions of sequencing constraints. Therefore, direct application of the general zero-one integer programming algorithms is not feasible. The only possible way to solve such an optimization problem is to employ some indirect approaches. The Lagrangian relaxation is usually used to remove the sequencing constraints. The relaxed problem is then solved by some special algorithms, such as the ultimate pit limit algorithm.
4.2.2 Everett’s Generalized Lagrange Multiplier Method

The Lagrange multiplier method is commonly used to solve constrained problems. The general procedure of the Lagrange multiplier method is to combine the constraints into the objective function and then to take the derivatives of the objective function with respect to the variables. Everett (1963) has extended the application of the Lagrange multiplier method to optimization problems involving non differentiable functions. This application is slightly different than the classical Lagrange multiplier method. Therefore, it is called the generalized Lagrange multiplier method.

The procedure is first to identify those constraints which are to be handled by this method, next multiply each constraint by an undetermined lagrange multiplier and then subtract the product from the original objective function. The relaxed problem is designed to be easier to solve than the original one.

Everett’s generalized Lagrange multiplier method can be used to solve the following maximization problems:

\[
\begin{align*}
\text{Max } & Z(X) \\
& AX \leq b
\end{align*}
\]  

\[ (4.4) \]

Theorem 4.1 (Everett): If the following two conditions are satisfied for problem (4.4):

1. \( \lambda^T = (\lambda^1, \lambda^2, \ldots, \lambda^n) \) are non-negative real valued multipliers,
2. \( X^* \in S \) maximizes the function \( Z(X) - \lambda^T AX \), over all \( X \in S \),

then,

3. \( X^* \) maximizes \( Z(X) \) over all those \( X \in S \) such that \( AX \leq AX^* \).
**Proof (Everett):**

By assumptions (1) and (2), $X^* \in S$ maximizes

$$Z(X) - \lambda^TAX, \text{ over all } X \in S,$$

this means that,

$$Z(X^*) - \lambda^TAX^* \geq Z(X) - \lambda^TAX, \text{ over all } X \in S,$$

and hence, that

$$Z(X^*) \geq Z(X) + \lambda^T(AX^* - AX), \text{ over all } X \in S$$

But if the latter inequality is true for all $X \in S$, it is necessarily true for any subset of $S$, and hence true on a particular subset $S^*$ of $S$ for which the capacity constraints satisfy $AX \leq AX^*$. Then, on the subset $S^*$ the term

$$\lambda^T(AX^* - AX)$$

is non-negative by definition of the subset $S^*$ on which $AX \leq AX^*$ and the non-negativity of the $\lambda^T$. Thus, the inequality reduces to

$$Z(X^*) \geq Z(X) \text{ for all } X \in S^*$$

The theorem is proved.
Discussion:

Everett’s theorem says that for any choice of non-negative $\lambda^T$, a maximum for the Lagrangian function $(Z(X) - \lambda^T AX)$ can be found when $X = X^*$, which is the best solution to the following constrained problem:

$$\begin{align*}
Max & 
Z(X) \\
AX & \leq AX^*
\end{align*}$$

(4.5)

In problem (4.5), the right hand side of the constraints is $AX^*$ instead of $b$ as in original problem (4.4). If $AX^* \leq b$, then a feasible solution to original problem (4.3) is obtained. When $AX^* = b$, the optimal solution to original problem (4.3) is obtained. $X^*$ and also $AX^*$ varies as $\lambda^T = (\lambda^1, \lambda^2, \ldots, \lambda^n)$ take different values. That, the Lagrange multiplier method generates a mapping of the space of $\lambda^T$ vectors into the space of constraint $AX^*$ vectors. The solution procedure becomes the matter of finding $X^*$ associated with the proper $\lambda^T = (\lambda^1, \lambda^2, \ldots, \lambda^n)$ such that $AX^* = b$.

Problem (4.3), which defines the mine production scheduling problem, is slightly different from problem (4.4) which is used in Everett’s theorem. Problem (4.4) has only the upper bound on the constraints. However, problem (4.3) has both upper and lower bounds on the constraints. Therefore, Everett’s theorem should be slightly modified in order to fit problem (4.3). Problem (4.3) can be written in the more general form (4.6)

$$\begin{align*}
Max & 
Z(X) \\
l & \leq AX \leq u
\end{align*}$$

(4.6)
Theorem 4.2: If the following two conditions are true for problem (4.6):

1. \( \lambda^T = (\lambda^1, \lambda^2, \ldots, \lambda^n) \) are real valued multipliers,
2. \( X^* \in S \) maximizes the function

\[
Z(X) - \lambda^T A X, \quad \text{over all } X \in S,
\]

then,

3. \( X^* \) maximizes \( Z(X) \) over all those \( X \in S \) such that \( AX = AX^* \).

Proof

The proof of theorem (4.2) is analogous to that of theorem (4.1). By assumptions (1) and (2), \( X^* \in S \) maximizes

\[
Z(X) - \lambda^T A X, \quad \text{over all } X \in S,
\]

it means that,

\[
Z(X^*) - \lambda^T A X^* \geq Z(X) - \lambda^T A X, \quad \text{over all } X \in S,
\]

and hence, that

\[
Z(X^*) \geq Z(X) + \lambda^T (AX^* - AX), \quad \text{over all } X \in S
\]

But if the latter inequality is true for all \( X \in S \), it is necessarily true for any subset of \( S \), and hence true on a particular subset \( S^* \) of \( S \) for which the capacity constraints satisfy \( AX = AX^* \). Thus, the inequality reduces to

\[
Z(X^*) \geq Z(X) \quad \text{for all } X \in S^*
\]

The theorem is proved.
Discussion:

The non-negative $\lambda^T$ required by theorem (4.1) is not required by theorem (4.2). Theorem (4.2) says that for any choice of $\lambda^T$, a maximum for the Lagrangian function $(Z(X) - \lambda^TAX)$ can be found when $X = X^*$, which is the best solution to the following constrained problem:

$$\begin{align*}
\text{Max } & Z(X) \\
\text{subject to } & AX = AX^*
\end{align*}$$

(4.7)

In problem (4.7), the original constraints $l \leq AX \leq u$ is replaced by the constraints $AX = AX^*$. If $l \leq AX^* \leq u$, the optimal solution to original problem (4.6) is obtained. $X^*$ and also $AX^*$ varies as $\lambda^T = (\lambda^1, \lambda^2, \ldots, \lambda^n)$ take different values. Therefore, the Lagrange multiplier method generates a mapping of the space of $\lambda^T$ vectors into the space of constraint $AX^*$ vectors. After realizing the above facts, the solution procedure becomes the matter of finding $X^*$ associated with the proper $\lambda^T = (\lambda^1, \lambda^2, \ldots, \lambda^n)$ such that $l \leq AX^* \leq u$.

4.2.3 Application of the Generalized Lagrange Multiplier Method to Mine Production Scheduling

The mine production scheduling can be formulated into problem (4.3), but this problem can not be solved by direct application of the general zero-one programming techniques due to the large amount of the constraints and variables. The re-formulation of the problem can be accomplished by introducing Lagrange multiplier $\lambda^T = (\lambda^O, \lambda^W)$. This is achieved through multiplying the capacity constraints by Lagrange multipliers $\lambda^T = (\lambda^O, \lambda^W)$ and subtracting these products from the original objective function as follows:
\[ \text{Max } L(X, \lambda^T) = \sum_{i=1}^{N} C_i X_i - \lambda^O \sum_{i=1}^{N} I_i^O T_i X_i - \lambda^W \sum_{i=1}^{N} I_i^W T_i X_i \] (4.8)
\[ X_i - X_j \leq 0, \text{ all } j \in \Gamma_i \]
\[ X_i = 0, 1 \]
\[ 1 \leq i \leq N \]

where all the variables and coefficients are defined the same as in problem (4.2). Problem (4.8) is called as the Lagrangian problem.

Problem (4.8) can be simplified into the following form:

\[ \text{Max } L(X, \lambda^T) = \sum_{i=1}^{N} (C_i - \lambda^O I_i^O T_i - \lambda^W I_i^W T_i) X_i \] (4.9)
\[ X_i - X_j \leq 0, \text{ all } j \in \Gamma_i \]
\[ X_i = 0, 1 \]
\[ 1 \leq i \leq N \]

For a given set \( \lambda^T = (\lambda^O, \lambda^W) \), problem (4.9) is an ultimate pit limit problem, with the original block values \( C_i \)'s replaced by the the modified block values \( (C_i - \lambda^O I_i^O T_i - \lambda^W I_i^W T_i)'s \). Problem (4.9) can be solved using an existing pit limit algorithm. However, the solution obtained is unlikely to satisfy the capacity constraints. The problem now is to determine these particular \( \lambda^T = (\lambda^O, \lambda^W) \) which will also satisfy these capacity constraints. Therefore, after an ultimate pit limit is found through the ultimate pit limit algorithm for a given set of Lagrange multipliers, the blocks within the pit limit are checked to see if the ore and waste tonnage constraints are all satisfied. If yes, then the optimal solution to the production scheduling as defined in problem (4.3) is obtained. If the ore or waste tonnage constraints are not satisfied, next iteration should be performed by updating Lagrangian multipliers and determining the associated pit limit. The above processes
are continued until a pit limit which satisfies the ore and waste tonnage constraints is found.

Lagrange multipliers are used to change the block monetary values. In problem (4.9), \( (\lambda^O_i T_i) \) is the penalty to block \( i \) if it is mined as ore, and \( (\lambda^W_i T_i) \) is the penalty to block \( i \) if it is mined as waste. Theoretically, both \( \lambda^O \) and \( \lambda^W \) could take either positive or negative values as stated in theorem (4.2). Since the penalty parameters change the block values, different pit size can be obtained by varying \( \lambda^T \).

Four methods have been brought out to find the Lagrange multipliers in mine production scheduling:

1. simplex method
2. parametric adjustment
3. subgradient optimization
4. combination of parametric adjustment and subgradient optimization

[1] Simplex Method

Johnson (1968) adopted the Dantzig and Wolfe's (1960) decomposition principle for large scale linear programs to mine production scheduling problems. Johnson's method is to decompose the mine production scheduling problem into two problems by including the capacity constraints in the master problem and the mining sequencing constraints in the subproblem. As a result, solving the original mine production scheduling problem becomes a matter of solving the subproblem many times by an ultimate pit limit algorithm for successive iterations until termination conditions are satisfied. The coefficients of the objective function (i.e., the block monetary values) of the subproblem are modified in each iteration by the
current Lagrange multipliers. Johnson suggested to use the simplex algorithm and
the decomposition principle to determine the Lagrange multipliers. The general
procedure of this approach consists of the following steps:

Step 1. Decompose the mine production scheduling problem into two separated linear programming problems. They are the master problem which includes only the capacity constraints (such as ore and waste tonnage requirements) and the subproblem which includes only the mining sequencing constraints. Construct an initial feasible solution (Dantzig and Wolfe, 1960) to the master problems.

Step 2. Calculate the Lagrange multipliers based on the current solution.

Step 3. Modify the block values using the Lagrange multipliers obtained in step 2 as the penalty parameters.

Step 4. Determine the pit limit by applying a pit limit algorithm on the modified block values.

Step 5. Check the termination conditions using the Lagrange multipliers obtained in step 2 and the solution obtained in step 4. If the termination conditions are satisfied, the optimal solution has been obtained. Otherwise, go back to step 2 and repeat the above process.

Johnson's work gave some mathematical thoughts about the mine production scheduling problem. However, his approach has never been implemented including himself, since it was brought out in two decades ago.

[2] Parametric Adjustment

Davis-Williams (1973) proposed a parametric adjustment method which increases or decreases Lagrange multipliers parametrically by 1 for the most violated constraint. Their procedure includes the following steps:
Step 1. Initialize multipliers, \((\lambda = 0)\).

Step 2. Calculate the modified block values for the given \(\lambda\) and determine the associated pit limit.

Step 3. Examine the violation of any lower bound of the constraints. If the lower bound of the constraints are satisfied, go to step 5.

Step 4. Reduce the multiplier by 1 (i.e., \(\Delta \lambda = -1\)) for the most violated constraint identified by step 3 and then go to step 1.

Step 5. Examine the upper bounds of the constraints. If they are satisfied, the current solution is optimal, so stop. If there is violations, then increase the multiplier by 1 (i.e., \(\Delta \lambda = +1\)) for the most violated constraint and go to step 1.

The above procedure is repeated until the obtained pit limit satisfies both the lower and upper bounds of the constraints.

Actually, the parametric adjustment is a pure trial and error method.

[3] Subgradient Optimization

The subgradient optimization method to the Lagrangian problem was proposed by Held-Karp (1970,1971). Dagdelen (1985) re-introduced the method to mine production scheduling to adjust Lagrange multiplies. This method starts with an initial set of the Lagrange multipliers equal to zero (i.e., \(\lambda = 0\)), and then computes a direction and a step size to get new multipliers based on the degree of the violations. The multipliers are updated by the rule:

\[
\lambda^{k+1} = \lambda^k + d_k (AX^k - b)
\]

where, \(X^k\) is a solution to \(L(X^k, \lambda^k)\) and \(d_k\) is a positive scalar step size. The limit on the approximate step size is given by Held-Karp (1971) as follows:
where,

\[ d_k = \alpha_k \frac{(L(X^*, \lambda^*) - L(X^k, \lambda^k))}{\| S^k \|^2} \]

\[ \| S^k \| \text{ denotes an Euclidean norm.} \]

\[ S^k = (AX^k - b). \]

\[ 0 \leq \alpha \leq 2. \]

\[ L(X^*, \lambda^*) \text{ is the best estimated value of the objective function.} \]

The great difficulty in applying the subgradient method to improve the Lagrange multipliers is how to calculate the scalar step size \( d_k \), which relies on \( (L(X^*, \lambda^*) - L(X^k, \lambda^k)) \). Held-Karp called \( L(X^*, \lambda^*) \) as the best target. However, it is impossible to determine the best value of the objective function (or revenue) in advance subject to the projected ore and waste tonnages without knowing the ore grade. Ore grade information will become available only after the exact location to be mined is determined. In mining industry, the revenue mainly depends on the ore grade, the ore tonnage to be mined and the waste tonnage to be removed. Of them, ore grade may be the most important factor to influence the revenue. \( L(X^*, \lambda^*) \) can not be computed before the ore grade information is available. Although Dagdelen (1985) theoretically suggested to use the Held-Karp’s (1970, 1971) subgradient method to adjust the Lagrange multipliers in mine planning, he did not practically use the approach proposed by himself in his own example program. A pure trial and error scheme was applied in Dagdelen’s example program (Dagdelen, 1985, pp. 237-239). Such a trial and error method may only work for some very small, hypothetical problems. Even for Dagdelen’s very small model (about 20 ore blocks to be scheduled in a time period), only ore tonnages were specified. That is, waste tonnage requirements were not used as the mining constraints, which is totally not acceptable in mining practice. If both ore and waste tonnage targets are put into
consideration, the proposed subgradient method may not work at all for practical, large mine models.

[4] Combination of Parametric Adjustment and Subgradient Method

Elevli (1989) proposed an approach to determine the Lagrange multipliers for production scheduling. His procedure is somehow to combine the parametric adjustment and subgradient optimization methods, using cumulative distribution curve of the ore block values. His basic idea is that if the penalty to an ore block is greater than its original value of the ore block, the modified value of the ore block will be negative. As more and more ore blocks become negative due to the penalty, the pit limit will get smaller and smaller. He assumes that the percentage of the reduction in the pit limit is approximately the same as the percentage of the ore blocks which become negative. Suppose that 40 percent of the ore block have the net values equal to or less than $V_{40}$, these 40 percent of the ore blocks will have the negative values after $V_{40}$ is subtracted from the original net values of all ore blocks. As a result, the pit size could be reduced approximately by 40 percent. The pit size varies inversely as the penalty parameter applied to the values of the ore blocks. His method consists of the following steps:

Step 1. Construct the cumulative curve of the positive blocks in the mine model. The $X$ axis of the curve represents the possible values of the positive blocks. The range of the $X$ axis should be from 0 to the maximum value of the ore blocks. The $Y$ axis of the curve represents the percentage of the ore blocks which have the net values equal to or less than the corresponding $X$ coordinate. Suppose that $P(x_i, y_i)$ is a point on the cumulative curve of the positive blocks. Then, point $P(x_i, y_i)$ indicates that there are $y_i$ percent of the ore blocks with the net values equal to or less than $x_i$.

Step 2. Initialize multiplier, $(\lambda = 0)$.
Step 3. Modify the block values using the current $\lambda$ and determine the associated pit limit using an ultimate pit limit algorithm.

Step 4. Examine the ore tonnage within the obtained pit limit. If the ore tonnage violation is within an acceptable error tolerance, the solution is considered as optimal. Otherwise, adjust the Lagrange multiplier with the aids of the cumulative curve and go back to step 3.

The Elevli's assumption that the amount of the reduction in pit size is the same as the percentage of the ore blocks which become negative is not true because of the following reasons:

1. Before an ore block is turned into a negative valued block, the ore block is used to offset its overlying waste blocks. After the ore block is modified into a negative valued block, it is treated the same as a waste block and must be supported by other ore blocks.

2. The assumption totally ignores the support relationship between ore blocks and waste blocks. Some ore blocks may be covered by more waste blocks, and some other ore blocks may be covered by less waste blocks.

3. Some ore blocks are able to offset many waste blocks if they have large profit values. While some others may have very little ability to support the overlying waste blocks because they have very small profit values.

Elevli's approach provided some new thoughts about the relationship between the pit size and ore blocks. However, the inability to handle the waste tonnage almost makes Elevli's method meaningless, since this method completely ignores the waste tonnage constraint. A mine production scheduling will not make much sense if the required waste tonnage can not be reasonably satisfied. In mine production scheduling, besides the ore and waste tonnage requirements, there may
be other constraints, such as feed grade requirement. But, the ore and waste tonnage constraints are the most important and basic ones which must be reasonably satisfied. Otherwise, the mine production scheduling is not much useful.

In summary, of the four approaches to determine the Lagrange multipliers discussed above, three of them are limited in practical use because of their inherent shortcomings. The first approach (Johnson's simplex method) includes mining partial blocks. The third approach (Dagdelen's subgradient method) requires the best value of the objective function, which is impossible to be estimated in advance. The fourth one (Elevli's combination of parametric adjustment and subgradient method) ignores the waste tonnage constraint. Only the second one, parametric adjustment by Davis-Williams (1973), has some chance to find the Lagrange multipliers at the price of the excessive computation time. The parametric adjustment approach does not involve much mathematics. It simply increases or decreases 1 to one of the Lagrange multipliers in each iteration. For practical, large mine models, more than several hundred iterations may be needed to work out a single period production scheduling by the parametric adjustment approach. Each iteration may require several hours or over night processing time. Even so, there is still no a priori guarantee to produce the optimal solution because of the general condition of non-convergence (section 4.2.4), non-convergence due to the inability to detect the redundant optimal solutions (section 4.2.5), and non-convergence due to the inability to handle the required advanced stripping (section 4.2.6).

4.2.4 General Condition of Non-Convergence

Everett (1963) discussed the possibility of non-convergence to the optimum solution with the Lagrangian method in some cases. Everett states as follows:

"... There is no a priori guarantee, however, that this mapping is onto--for a given problem there may be inaccessible regions (called gaps) con-"
sisting of constraint vectors that are not generated by any $\lambda$ vectors. Optimum payoffs for constraints inside such inaccessible regions can therefore not be discovered by straightforward application of the Lagrange multiplier method, and must hence be sought by other means."

By the above statements, Everett says that the algorithm may not converge to the optimum solution of the problem for some given set of constraints. This non-convergence can take place when there is no set of Lagrange multipliers satisfying the specified constraints of the original problem can be found. When it is not possible to find such a set of Lagrange multipliers to satisfy the constraints of the original problem, then it is said that the condition of gaps exists for the problem being solved.

4.2.5 Non-Convergence due to the Inability to Detect the Redundant Optimal Solutions

The non-convergence problem which will be discussed in this section is not the general cases mentioned by Everett (section 4.2.4). But, it is due to the fact that relaxed problem (4.9) is solved by an ultimate pit limit algorithm. When there exist redundant optimal solutions, the proposed approach combining the Lagrangian relaxation with the ultimate pit limit algorithm can not identify any of them. Instead, the approach will provide a solution which is either the union of all the redundant solutions or empty. Since such a solution does not satisfy the ore and waste constraints, the approach will try to modify the Lagrangian multipliers. Therefore, the approach will go into "dead" loop.

Figure 4.1 (a) is a simple 2-D cross section of hypothetical mine model. The numbers in the figure represent the monetary values from mining the blocks. Now, suppose that the following ore and waste tonnage requirements are required:
Ore tonnage requirement: 7 ore blocks,
Waste tonnage requirement: 9 waste blocks.

Corresponding to the above tonnage requirements, there are two alternatives. That is, one is alternative 1 shown in figure 4.1 (b), and the other one is alternative 2 shown in figure 4.1 (c). The summary of blocks of both alternatives is given in table 4.1. When the proposed approach of combination of the Lagrangian relaxation with the ultimate pit limit algorithm is used, the pit limit will be the union of two alternatives as shown in figure 4.2 (a) if the penalty parameters are small, and the pit limit will be empty as shown in figure 4.2 (b) if the penalty parameters are too big. There do not exist such Lagrange multipliers which generate a pit limit with the specified ore tonnage (7 blocks) and waste tonnage (9 blocks). Appendix A gives the pit limits corresponding to different penalty parameters (Lagrangian multipliers). From table 4.2, one can find the range of the penalty parameters within which the pit limits are the union of the two alternatives.

If an integer linear programming algorithm is used to solve the relaxed problem, either alternative 1 or alternative 2 will be the optimal solution, since the integer linear programming has the capability to systematically search all the redundant optimal solutions one by one. Hence, the non-convergency problem will not occur in this case if standard integer linear programming algorithms are used to the relaxed problem. It is clear that the above non-convergence is caused by the application of the ultimate pit limit algorithm. Since the standard integer linear programming algorithms can not handle such a big problem of pit limit design, ultimate pit limit algorithm is the only available choice. This non-convergency problem can not be overcome by improving the technique to determine the Lagrange multipliers, since it is not caused by the method of Lagrangian relaxation, but the application of the ultimate pit limit algorithm to the relaxed problem.
Figure 4.1  An Example of Existence of Redundant Optimal Solutions
Table 4.1  Summary of Blocks of Two Alternatives

<table>
<thead>
<tr>
<th></th>
<th>Alternative 1</th>
<th>Alternative 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of ore blocks</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>No. of waste blocks</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Tot. value of ore block</td>
<td>58</td>
<td>60</td>
</tr>
<tr>
<td>Tot. value of waste blocks</td>
<td>-9</td>
<td>-11</td>
</tr>
<tr>
<td>Net profit</td>
<td>49</td>
<td>49</td>
</tr>
</tbody>
</table>

\[ \lambda^0 = 3 \quad \lambda^w = 3 \]

(a) Pit Limit Is the Union of Two Alternatives

\[ \lambda^0 = 4 \quad \lambda^w = 4 \]

(b) Pit Limit Is Empty

Figure 4.2  Possible Pit Limits for Different Penalty Parameters
No one can imagine how many combinations there are in a practical mine model with hundred thousands of mining blocks. And no one can guarantee that there are no redundant optimal solutions for the specified constraints. As a matter of the fact, it is very likely to have the redundant optimal solutions in mine planning, since all the waste blocks on the same bench will have the same value and all the ore blocks on the same bench with the same grade will have the same profit value. As a result, the application of the Lagrangian relaxation technique to mine production scheduling is bogged down by the application of the ultimate pit limit algorithm.

Table 4.2 Existence of Pit Limits for Different Penalty Parameters

<table>
<thead>
<tr>
<th>Λ</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

1: Pit limit is the union of two alternatives.

0: Pit limit is empty (i.e., no economic pit limit can be found for the penalty parameters).
Theorem 4.3: If there exist redundant optimal solutions, the approach of combination of the Lagrangian relaxation and with ultimate pit limit algorithm will give the solution which is either the union of all the redundant solutions or empty.

Proof:

Suppose that there are $n$ redundant optimal solutions.

Let

- $T_i^O$ be the ore tonnage of alternative $i$, $i = 1, 2, \ldots, n$,
- $T_i^W$ be the waste tonnage of alternative $i$, $i = 1, 2, \ldots, n$,
- $V_i^O$ be the cumulative value of all the ore blocks of alternative $i$, $i = 1, 2, \ldots, n$,
- $V_i^W$ be the cumulative value of all the waste blocks of alternative $i$, $i = 1, 2, \ldots, n$,
- $\lambda^O$ be the Lagrange multiplier for ore tonnage constraint,
- $\lambda^W$ be the Lagrange multiplier for waste tonnage constraint.

Then the value of the Lagrangian objective function, $L_i(X, \lambda^T)$, for alternative $i$ is:

$$L_i(X, \lambda^T) = V_i^O - V_i^W - \lambda^O T_i^O - \lambda^W T_i^W, \quad i = 1, 2, \ldots, n.$$

By the assumptions that all alternatives are optimal solutions, that is, both the ore and waste tonnages of all the alternatives are equal to the required ore and waste tonnages, the total profits (i.e., $V_i^O - V_i^W$) are the same for all the alternatives. Therefore, the following statements must be true:

- $T_1^O = T_2^O = \ldots = T_n^O = \text{required ore tonnage},$
- $T_1^W = T_2^W = \ldots = T_n^W = \text{required waste tonnage},$
- $V_1^O - V_1^W = V_2^O - V_2^W = \ldots = V_n^O - V_n^W.$
Therefore, the values of the Lagrangian objective function for all the redundant optimal solutions are exactly the same, no matter what Lagrange multipliers are chosen. That is:

\[ L_1(X, \lambda^T) = L_2(X, \lambda^T) = \ldots = L_n(X, \lambda^T) \]

Not like the standard integer linear programming algorithms, the ultimate pit limit algorithms do not have any capability to detect any one of the redundant optimal solutions. Since all the redundant optimal solutions are still equally good even if different Lagrangian multipliers are applied, the ultimate pit limit algorithms will either pick up all the redundant optimal solutions or discard all the redundant optimal solutions. If all the redundant optimal solutions are included in the solution set, the ore and waste tonnages must be greater than the amount specified by the constraints. Therefore, the problem is not solved. On the other hand, if all the redundant optimal solutions are excluded, the problem is still not solved. In such a situation, the proposed approach will not converge for given ore and waste tonnage constraints, since the solution is either the union of all the redundant optimal solutions or the empty, no matter how the Lagrangian multipliers are changed (or perturbed).

4.2.6 Non-convergence due to the Inability to Handle the Required Advanced Stripping

This problem arises from the change in the spaces on which the original mine production scheduling problem and the ultimate pit limit problem are defined. The production scheduling problem is defined on the space which includes all the blocks (all positive blocks and all negative blocks) in a mine model. But, the ultimate pit limit problem is defined on the space which includes all the positive blocks and some negative blocks, which are the restricting blocks to the positive blocks. When a negative block is not a restricting block to any positive block, this negative block will never be evaluated in the ultimate pit limit computation. However, the above
situation is not true in the production scheduling problem. Some waste blocks, which may not be the restricting blocks to the ore blocks to be scheduled in the current period, must be mined in the current period in order to meet the waste tonnage requirement. The above requirement of advanced stripping is employed in almost all production scheduling to keep a reasonable stripping ratio and to expose enough ore material for subsequent production periods. Since the ultimate pit limit algorithm can not remove the waste blocks which are not the restricting blocks to the ore blocks to be scheduled in the current period, the approach combining the Lagrangian relaxation with the ultimate pit limit algorithm will not converge if advanced stripping is required. In practice, advanced stripping is always required. The extent of the advanced stripping depends on the nature of the orebody, surface topography, and the management philosophy regarding smoothing of equipment utilization and manpower requirements.

4.3 Pushback Approach

"Pushback" sequence approach is the most popularly used by mining industry in mine production scheduling. This approach uses an ultimate pit limit algorithm to obtain a series of nested pits by varying cutoff grades or product prices. After a set of pushbacks are generated, a particular mining sequence can be worked out based on ore reserves in all the pushbacks. Finally, by varying the cutoff grades, production capacities, different pushbacks can be tried and in turn different alternative mining sequences can be obtained. The mining sequence which provides maximum net present value is the desired mining sequence.

4.3.1 Pushback Generation

Although the pushback approach has been used by most of the mining companies, Mathieson's paper (1982) is probably the first one discussing the overall
pushback mine planning methodology in details. The philosophy of mining the "next best" is the central idea of the pushback approach.

The pushbacks have an important feature, i.e., the profitability declines from inner pushbacks to outer pushbacks since each outer incremental pit is generated by gradually decreasing cutoff grades. If the horizontal distance between any two of those nested pits is greater than the minimum allowable mining width, these nested pits represent the "next best" mineralization with inner pit being the "best" and the outer pit being the "next best". Such an inner pushback to outer pushback sequence provides a general direction for mining advancing. Figure 4.3 shows a cross section of such "next best" incremental pits on a hypothetical deposit. The numerical number in each incremental pit in figure 4.3 is the order of pushbacks. Usually, the pushback patterns of real situation are more complicated than that as shown in figure 4.3. Appendix B contains some pushback contour maps from a real gold deposit.

Figure 4.3  Cross Section View of the Next Best Mineralization

(After Mathieson, 1982)
Many articles have reported on the application of pushback approach to long range mine planning (Crawford, 1976; Couzens, 1978; Iles and Perry, 1981; Mathieson, 1982; Fiore, 1986; Whittle, 1988; Cai, 1989; and Faust, 1989).

In pushback development, access to each active bench must be provided besides the constraint that horizontal distance between any two nested pits is greater than the minimum allowable mining width. Figure 4.4 shows the sequential access concept between pushbacks.

After a series of nested pits is generated for a particular set of cutoff grades, bench by bench inventories of ore and waste tonnages are calculated for each incremental pushback. Table 4.3 provides such an example of the bench by bench inventories of ore and waste tonnages for a real gold deposit.

If mining can proceed in exactly the way the nested pits are generated, maximum net present value will be achieved. This is because the phasing order
defined by the pushbacks represents a philosophy of always mining the "next best" ore. Unfortunately, mining may not be able to proceed following an inner pit to outer pit sequence due to practical constraints. One of the constraints is stripping ratio. Figure 4.5 shows the stripping ratio of the first three pushbacks for this gold deposit. The strategy of advance stripping is not only for the purpose to smooth the stripping ratio among production periods, but also for the purpose to satisfy the minimum ore exposure for the following production periods. Therefore, a more detailed mining sequence must be somehow worked out among the pushbacks. As discussed by Mathieson (1982), "During the mine life, typically three or more pushbacks are active at any production period: one perhaps nearing exhaustion on ore supply, another beginning to be mined for ore, and another one or two being stripped".

![Figure 4.5 Stripping Ratio of Different Pushbacks](image)
Table 4.3  Bench by Bench Ore and Waste Tonnage Inventories for Each
Pushback (Unit of Tonnage is ton)

<table>
<thead>
<tr>
<th>bench number</th>
<th>pushback #1 ore</th>
<th>pushback #1 waste</th>
<th>Pushback #2 ore</th>
<th>Pushback #2 waste</th>
<th>pushback #3 ore</th>
<th>pushback #3 waste</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3956</td>
<td>533</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>24400</td>
<td>1689</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>36311</td>
<td>40756</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>70800</td>
<td>104133</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>147778</td>
<td>149200</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>2267</td>
<td>18533</td>
<td>225022</td>
<td>201200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>86800</td>
<td>80311</td>
<td>285644</td>
<td>322311</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>120889</td>
<td>173778</td>
<td>224089</td>
<td>590178</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>164889</td>
<td>133733</td>
<td>223600</td>
<td>877822</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>199111</td>
<td>60000</td>
<td>1333</td>
<td>168489</td>
<td>246511</td>
<td>1178089</td>
</tr>
<tr>
<td>16</td>
<td>193333</td>
<td>19822</td>
<td>6667</td>
<td>351511</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>158667</td>
<td>13956</td>
<td>13333</td>
<td>659200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>146178</td>
<td>116467</td>
<td>7556</td>
<td>746089</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>138222</td>
<td>268667</td>
<td>24444</td>
<td>839911</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>90667</td>
<td>247911</td>
<td>69333</td>
<td>842533</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>151556</td>
<td>132444</td>
<td>148889</td>
<td>729022</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>188444</td>
<td>46667</td>
<td>275556</td>
<td>624444</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>165333</td>
<td>19556</td>
<td>356444</td>
<td>44000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>421778</td>
<td>280000</td>
<td>413778</td>
<td>199111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>408889</td>
<td>117333</td>
<td>369111</td>
<td>86667</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>315556</td>
<td>66667</td>
<td>261778</td>
<td>57333</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>1806356</td>
<td>1334311</td>
<td>3184445</td>
<td>6157023</td>
<td>1484445</td>
<td>3465911</td>
</tr>
</tbody>
</table>
4.3.2 Mine Sequencing among Pushbacks

Although there is not a standard approach to mining sequence among pushbacks, there is indeed a general direction as how to proceed. This general direction is mainly defined by two precedence requirements. The first one is the sequential requirement among benches within the same pushback, i.e., within the same pushback, benches from higher elevation have higher priority to be mined than benches from lower elevation. This sequential requirement should be satisfied because the lower benches usually cannot be mined before the higher benches are removed within the same pushbacks. The second one is the precedence requirement among the benches having the same elevation from different pushbacks. The benches in inner pushbacks have higher priority to be mined than the benches in outer pushbacks if those benches are on the same level. The mining schedule should follow the order from the inner pushbacks to outer pushbacks as closely as possible while seeking to satisfy practical constraints. Besides the above two precedence requirements, some other constraints should also reasonably be satisfied, such as the mining rate, grade blending, and road access requirements.

The most current approaches to mining sequence among different pushbacks are characterized entirely by trial-and-error. To improve the procedure of mining sequence among pushbacks, a zero-one programming model for single period production scheduling is proposed by Cai (1989). Since his model is designed for the single period production scheduling, the multi-period production scheduling development is carried out period by period with each run for one production period.

4.3.3 Optimizing Production Schedule

The optimization of mine production scheduling is realized by generating alternative ore and waste removal schedules and choosing the best one. Corresponding to each cutoff grade strategy, a set of pushbacks can be worked out, a particular mining schedule can in turn be developed based on the pushbacks, and
a net present value (NPV) analysis can be carried out. For each variation of cut-off grade strategy, the above processes, i.e., (1) pushback generation, (2) mining sequence among pushbacks, and (3) NPV analysis, are repeated once. Although there is a general direction as to how to proceed, there is not a strict rule how to do. The more alternatives are explored and the more likely the "optimal" mining sequence will be detected. The "optimal" pit mining sequence is the one having the maximum net present value.

4.4 Summary

The importance of this chapter is not as much as to contribute to the solution technique of mine production scheduling, but to clarify the long-time misunderstood concept to apply the approach combining the Lagrangian relaxation with the ultimate pit limit algorithm to mine production scheduling problem and prove its impossibility of the research direction itself.

The direct "optimum" mine production scheduling so far has not enjoyed wide application in mining industry. And it will not prevail in industry using today's computing facilities and operations research techniques. Moreover, it is very difficult to incorporate some operating constraints, such as minimum working space, minimum amount of ore exposed at the end of each period, and minimum shovel moves, into the optimal scheduling model.

The "pushback" mine sequencing is so far the most popular approach in mining sequence development. This is because finding the "next best" ore seems to be the most logical approach in a mining project. Unfortunately, in pushback mine sequencing which requires to explore alternative mining sequences based on different cutoff grade strategy, each mining sequence must be worked out by a trial-and-error approach.
CHAPTER 5

THE IMPLEMENTATION ASPECT OF THE BLAST HOLE KRIGING

5.1 Introduction

The purpose of this chapter is to describe the implementation aspect of highly automated computerized ore-waste delineation procedure to optimize daily grade control in open pit mines. For this reason, the problem solving techniques required for the implementation of such a system are discussed in this chapter.

The so called grade control is to properly classify the mined material into different categories, such as, ore, leach and waste types. The primary goal of the computerized grade control system is to improve grade control and the ore-waste selection procedure at an operating mine so that the ore control personnel can make the correct selection with regards to the various classes of ore and waste. The grade control in mine operations is a logical application for computers since the same tasks are performed repetitively every day. A complete computer program package called Blast Hole Kriging (BHK) system was implemented for a personnel computer which takes full advantages of the PC computer to perform the functions of blast hole handling, geology information handling, blast hole kriging, grade control (or ore-waste delineations) in daily production.

The accuracy of grade estimation in daily ore-waste delineations is particularly important since any errors made with respect to the classification of material will result in a non-compensating financial loss, that is, either a loss of potential profit from ore being sent to waste dump or unwarranted processing costs by sending waste to processing plant.
The more commonly used ore-waste delineation procedure in the past based on the central blast hole assay and the polygonal grade assignment around this hole does not optimally utilize the information provided by the available blast hole assays since this method ignores the spatial correlation between the samples. The polygonal method simply extends or assigns the grade of one sample to a large volume with the error variance equal to the so called extension variance. From the experience of geostatistics, it is known that the extension variance is always larger than the kriging variance. As such, the quality of the estimates produced by the polygonal method are not as good as that of kriged results.

It is well known that the kriging method gives a conditionally unbiased estimate (or nearly so) with the minimum error variance. In the long run, blast hole kriging will produce the least amount of mis-classification between ore, leach and waste categories as one mines according to the kriged estimate rather than the actual assay results of the central blast hole.

The BHK system was developed for IBM/PC AT or XT computer, using Fortran language and supplemented by a few Assembly language routines for bitmap manipulation. The entire system consists of ten (10) computer programs. The database handling, grade estimation by kriging technique and interactive selection of ore–waste boundaries are the major functions of the system. Figure 5.1 is the simplified flow chart of the BHK system.

Although the BHK system is a sophisticated program package, the system is very simple to operate. All the data files used by the BHK system are created by the system itself. If the user follows the logical series of the steps shown in figure 5.1 to run the programs, each program will automatically accept the data files created by the previous programs and generate the data files for the following programs, without the user having to think about where the files are and how the files are created. This procedure not only greatly simplifies the operations of the programs by reducing the entry of data, but also greatly decreases the chance for user errors.
Figure 5.1  Simplified Flow Chart of BHK System
The software in the BHK system has the capabilities to efficiently handle the following tasks that are being performed at operating mines:

1. The easy and rapid mode of collection and validation of blast hole coordinates, assay values, and geology as the information becomes available from various sources.

2. Database handling capability which provides the user with an efficient way to add, delete, modify and retrieve the blast holes and the geology information.

3. The efficient way to handle and utilize geology information in grade estimation.

4. The capability of polygon manipulations.

5. Fully interactive graphics planning.

6. Color graphics capability to display the blast hole locations, assay values, block grades, shot boundaries, ore-waste boundaries and geology information on a color monitor and also to zoom any particular interested area or an object.

7. The easy mode of map generation using a plotter or a printer.

8. Preparation of data or maps for material routing and flagging.

The successful implementation of the BHK system is highly dependent upon a proper hardware configuration. The BHK system was developed for the IBM PC/AT or XT type computers. Linked to the PC computer are some peripherals which constitute the grade control work station. It consists of a color graphics monitor, digitizer, printer, plotter, mouse, modem and survey station as shown in figure 5.2.
Although figure 5.2 shows the hardware requirements, the BHK system was designed to provide the user with some very flexible ways to operate and run the programs with the minimum hardware requirements. The system runs on both AT and XT computers. The color-coded graphics displays on the screen can be ZOOMED as well as plotted by a plotter or a printer. The mouse is used to control the graphics display and interactive planning. If a mouse is not installed, the same results can be achieved using the arrow keys on the keyboard. Any one of the following three options can be used in any programs in the BHK system to input the coordinates: 1) digitizer, 2) keyboard, and 3) data file.
Conceptually, blast hole kriging is identical to kriging for orebody modeling. However, there is a world of difference in its implementation due to its on-line and production oriented nature of the problem. Here, solving the logistics problem is the key to a successful implementation of blast hole kriging at an operating mine. Sections 5.2, 5.3 and 5.4 will discuss what these logistics problems are and how these logistics problems are solved in order to make the BHK system into a practical and user friendly operational tool in daily production.

Section 5.2 develops the applied geometry concepts that underlie the polygon manipulation algorithms used by the BHK system. These polygon manipulation algorithms are:

1. point-in-polygon algorithms,

2. polygon area algorithm,

3. polygon clipping algorithm.

The above algorithms play a key role in mine planning and map generation since the most geology constraints, planning boundaries and surface contours are handled by polygons. In fact, it is impossible to develop the interactive graphics planning package without the aids of these polygon manipulation algorithms.

Section 5.3 discusses the techniques used to solve the problems in the data collection, validation and database maintenance. Commonly, there are two types of data, that is, blast hole data and geology data, which must be dealt with in daily grade control. The blast hole data are made available in bits and pieces from various sources such as assay laboratory, survey station, grade control personnel and blast hole maps. Usually, coordinates and assay values are collected separately, and they must be merged and verified. Blast hole data entry and verification are very time consuming. To reduce and simplify the tasks to get blast hole data into the computer and verified, a blast hole editor was designed for this particular
purpose. Because of the relatively large amount of blast hole data inherent in a blast hole database, an efficient means of data storage and retrieval is essential. The blast hole assays for the entire mine are stored in the blast hole database. Rock types should be the important parameters in grade estimation. Hence, a rock type database is designed to store and retrieve the rock type information for the entire mine. The rock type database provides an efficient way to handle and utilize the rock type information in grade estimation. The data structures and the design considerations of both the blast hole database and rock type database will be discussed in section 5.3.

Section 5.4 describes the following general steps which constitute the interactive graphics planning part of the BHK system:

1. Store the variogram parameters for each rock type, each pushback and each assay variable in database format.

2. Update the job control parameters used in kriging.

3. Input the shot boundary and retrieve the blast holes within the shot boundary and the neighborhood from both the active bench and the above benches.

4. Assign the rock type code to each blast hole and each block.

5. Perform grade estimation by kriging technique.

6. Display the estimated grades and the shot boundary by color codes.

7. Delineate ore-waste boundaries in an interactive mode.

8. Generate maps which can be used by pit foreman and the shovel operators for the purpose of material routing.

9. Prepare the summary report of the estimated tonnages and grades for each ore-waste polygon boundary within the shot.
5.2 Polygon Manipulation Algorithms

The computer color graphics hardware developed recently provides the mining engineers with a new concept in mine planning, which allows the engineers to directly view their data and interactively perform graphics planning in a fashion that was not possible just a few years ago. Increasingly, the computer interactive graphics planning is used by more and more mining companies since it has dramatically increased the mining engineer's mine planning productivity, while providing more accurate results.

In order to properly assess the different scheduling alternatives, mining engineers have to be able to provide feasible plans in a timely fashion. These include pushback (or phase mining) generations, annual plans, monthly plans, and daily ore-waste delineations. Besides the planning boundaries, the current surface contour, haul road access, and existing production constraints have to be specified and handled mostly by polygons. Hence, the polygon manipulation algorithms play a key role in accurately evaluating grades and tonnages from long range planning to daily grade control.

This section develops the applied geometry concept that underlies these polygon manipulation algorithms. Section 5.2.1 introduces and compares three point-in-polygon algorithms, section 5.2.2 gives the mathematical derivations of the polygon area algorithm in two different approaches, and section 5.2.3 develops a polygon clipping algorithm to determine the intersection, union, and complement between polygon sets.

Although the point-in-polygon algorithms, polygon area algorithm and polygon clipping algorithm to be introduced in this section do not belong to the domain of mining technologies, these algorithms are very important part of the basic tools used in mine planning, particularly, in interactive graphics planning. As it will be seen in section 5.4, it is impossible to develop the interactive graphics planning package without the aids of these polygon manipulation techniques.
5.2.1 Point in Polygon Algorithms

The need for an efficient algorithm which determines the position of a given point relative to a closed polygon boundary arises very frequently in mine planning in order to check if a point (or block) is within the planning boundary. The point-in-polygon test is also a very basic function required in polygon manipulations and computer graphics. Such algorithms therefore have been studied extensively (Shimrat, 1962; Hall, 1975; Anderson, 1976; and Salomon, 1978). The purpose of this section is to discuss and compare the following three point-in-polygon algorithms. They are interior angle method, line intersection method and swathing approach.

[1] Interior Angle Method

The idea of this method is quite simple. In this method, the sum of all possible interior angles subtended by all the boundary segments is found (see figure 5.3 (a) and (b) ), using the point of interest as the vertex and any two adjacent boundary points to complete the angle. Right hand rule is used as a convention in determining the direction (or sign) of the angles. The counterclockwise angles are assigned a positive sign, and the clockwise angles are assigned a negative sign. If the sum of the interior angles is ±360 degrees (figure 5.3 (a) ), then the point is inside the polygon. If the sum is zero (figure 5.3 (b) ), the point is outside the polygon. When the polygon boundary has \( n \) node points, this method requires \( n \) interior angles to be calculated.

If \( \mathbf{A} \) and \( \mathbf{B} \) are the two vectors drawn from the given point \( P_p(x_p, y_p) \) to points \( P_a(x_a, y_a) \) and \( P_b(x_b, y_b) \) as shown in figure 5.4, then we can write

\[
\mathbf{A} = (x_a - x_p)i + (y_a - y_p)j = X_1i + Y_1j \quad (5.1)
\]
\[
\mathbf{B} = (x_b - x_p)i + (y_b - y_p)j = X_2i + Y_2j \quad (5.2)
\]
Figure 5.3  The Internal Angles Subtended by All the Boundary Segments

Figure 5.4  Vectors Formed by Three Points

Figure 5.5  Right Hand Rule
Now the vector dot product \( \mathbf{A} \cdot \mathbf{B} \) and the vector cross product \( \mathbf{A} \times \mathbf{B} \) are defined as:

\[
\mathbf{A} \cdot \mathbf{B} = |\mathbf{A}| |\mathbf{B}| \cos \theta = X_1 X_2 + Y_1 Y_2 \tag{5.3}
\]
\[
\mathbf{A} \times \mathbf{B} = |\mathbf{A}| |\mathbf{B}| \sin \theta \mathbf{k} = (X_1 Y_2 - X_2 Y_1) \mathbf{k} \tag{5.4}
\]

And the above equations can be transformed into the following forms:

\[
\cos \theta = \frac{X_1 X_2 + Y_1 Y_2}{|\mathbf{A}| |\mathbf{B}|} \tag{5.5}
\]
\[
\sin \theta = \frac{X_1 Y_2 - X_2 Y_1}{|\mathbf{A}| |\mathbf{B}|} \tag{5.6}
\]

Then, the value of \( \theta \) can be found by the ordinary library trigonometric functions ACOS or ASIN. The vector cross product defined by equation (5.4) is also a vector. Because the direction of vector cross product is defined by the so called right hand rule (see figure 5.5), the sign of the angle can also be determined by the right hand rule. Hence, if \( \sin \theta \) computed by equation (5.6) is positive, the angle is counterclockwise. Otherwise, the angle is clockwise.

The above procedures are not difficult to understand nor hard to implement. The problem with the above approach is that the use of library functions ACOS or ASIN would impose considerable time penalties in evaluation of each angles.

In 1975 Hall proposed a method which only requires calculating \( \sin(\sum_{i=1}^{n} \theta_i) \) and \( \cos(\sum_{i=1}^{n} \theta_i) \) without evaluation of the actual angles \( \theta_i \) for \( i = 1 \) to \( n \). Knowing \( \sin \theta_i \) and \( \cos \theta_i \) for each successive pair of boundary points, it is possible to compute \( \sin(\sum_{i=1}^{m+1} \theta_i) \) and \( \cos(\sum_{i=1}^{m+1} \theta_i) \) for \( 1 \leq m \leq n \) at each segment through the following relationships:

\[
\sin(\sum_{i=1}^{m+1} \theta_i) = \sin(\sum_{i=1}^{m} \theta_i + \theta_{m+1})
\]
\[
= \sin(\sum_{i=1}^{m} \theta_i) \cos \theta_{m+1} + \sin \theta_{m+1} \cos(\sum_{i=1}^{m} \theta_i) \tag{5.7}
\]
cos(\sum_{i=1}^{m+1} \theta_i) = cos(\sum_{i=1}^{m} \theta_i + \theta_{m+1})
= cos(\sum_{i=1}^{m} \theta_i)cos(\theta_{m+1}) - sin(\sum_{i=1}^{m} \theta_i)sin(\theta_{m+1}) \tag{5.8}

In order to determine whether \(\sum_{i=1}^{m} \theta_i\) sums to \(\pm360^\circ\) or to \(0^\circ\), \(sin(\sum_{i=1}^{m} \theta_i)\) is monitored. Since \(sin(0^\circ) = sin(\pm180^\circ) = sin(\pm360^\circ) = 0\), \(sin(\sum_{i=1}^{m} \theta_i) > 0\) if \(0^\circ < \sum_{i=1}^{m} \theta_i < 180^\circ\) and \(sin(\sum_{i=1}^{m} \theta_i) < 0\) if \(180^\circ < \sum_{i=1}^{m} \theta_i < 360^\circ\). The maximum angle subtended by any segment cannot exceed \(+180^\circ\) or \(-180^\circ\), and the job of monitoring \(\sum_{i=1}^{m} \theta_i\) is done by keeping track of the number of times that \(\sum_{i=1}^{m} \theta_i\) passes through \(0^\circ\) or \(\pm180^\circ\) and then testing the value of \(sin(\sum_{i=1}^{m-1} \theta_i)\) just before summation of the last segment's subtended angle to determine if \(\sum_{i=1}^{m} \theta_i\) is \(0^\circ\) or \(\pm360^\circ\). Finally, the sense of the direction of the boundary is obtained by examining the sign of \(sin(\sum_{i=1}^{m} \theta_i)\) on the last iteration to determine whether the accumulated angle is positive (counterclockwise) or negative (clockwise).

This algorithm is also capable of detecting if the given point is on one of the boundary points or on one of the boundary segments.

(1) Point falling on one of the boundary points

The lengths of the vectors \(A\) and \(B\) are computed for the angle subtended by each boundary segment in order to calculate \(sin(\theta_i)\) and \(cos(\theta_i)\). These lengths are checked to see if they take zero value. In the event that one of them is zero, the given point then is known to coincide with that particular boundary point.

(2) Point falling on one of the boundary segments

In order to detect if a point falls on the boundary segment, \(cos(\theta_i)\) is compared with \(cos(180^\circ) = -1\). Because of round off errors caused by the finite length of the computer word, the cosine is actually compared with \(-0.9999999998\) which is \(0.001^\circ\) off from \(180^\circ\). If \(cos\theta_i\) is less than or equal to \(-0.9999999998\), it is assumed that the point lies on the boundary.

The processes involved in this algorithm are very time-consuming, but the
algorithm has the following remarkable advantages: (1) it works for the polygons of any shape, (2) it can detect if the point is on the polygon node points or on the boundary segments and (3) it determines the direction of the boundary outline. Due to the above advantages, the interior angle method is quite popular although it is a very slow algorithm.

[2] Line Intersection Method

In this method, the number of intersections between the polygon boundary outline and a line extending from the given point to infinity in any direction is counted (see figure 5.6). If there is an even number of intersections (figure 5.6 (a)), the point is outside the polygon; whereas an odd number of intersections (figure 5.6 (b)) means that the point is inside. Since a line can be projected from the given point to infinity in any direction, it is convenient to select a line along the x-axis direction as shown in figure 5.7 in order to simplify the computation. In figure 5.7, point a is outside the polygon because the horizontal line a extending from point a intersects the boundary segments twice (even). Whereas point b is inside as line b intersects the boundary segments three times (odd).

Figure 5.6 Line Intersection Method to Determine If a Point Is Inside or Outside the Polygon Boundary
Shimrat (1962) was probably the first person to introduce the line intersection method. Then, Anderson re-introduced the algorithm in 1976.

This algorithm is fast and simple, but it has the following restrictions: (1) points on the boundary segments can not be detected, (2) the points can not be coincident with the boundary node points and (3) the points can not be on the horizontal segments of the boundary outline. In order to make the original algorithm into a very general one, the restrictions must be overcome and the algorithm must be extended. Therefore, the above problems should be handled separately as the special cases. Although such modifications will make the algorithm a little bit complicated and increase the processing time, this algorithm is still one order of magnitude faster than the interior angle method.

[3] Swathing Method

This algorithm is not concerned with the original polygon, but with a processed version of it whereby it is represented by a set of horizontal swaths. Each swath contains those sides spanning the interval of the y-axis between two successive vertices when they are ordered by decreasing Y-endpoints. For example,
the polygon shown in figure 5.8 can be divided into six (6) swaths. Swath1 contains segments 5 and 6; \( y\text{-interval}_1 = [y_1, y_2] \). Swath2 contains segments 3 and 6; \( y\text{-interval}_2 = [y_2, y_3] \). … Swath6 contains segments 7 and 8; \( y\text{-interval}_6 = [y_6, y_7] \).

![Figure 5.8 Point in Polygon Test by Swathing Method](image)

After the original polygon is divided into swaths, whether a point is inside or outside the polygon can be determined simply by counting the number of polygon segments within the appropriate swath intersecting a right-directed ray emanating from the point. To determine the number of intersections, the swathing method only checks the polygon segments within the appropriate swath instead of all the segments as the line intersection method does. If the number of the intersections is odd, the point is within the polygon. Otherwise, an even number of intersections indicates that the point is outside the polygon. For instance, since point \( a \) is within swath2 which contains segments 3 and 6, and there are two (even) intersections between linea and segments 3 and 6, then the conclusion can be easily drawn that point \( a \) is outside the polygon. Similarly, point \( b \) falls in swath3 which contains segments 1, 2, 3 and 6, and there are three (odd) intersections between lineb and the segments within swath3, therefore, point \( b \) is inside.
In fact, this approach is a modified version of the **Line Intersection Method**. The following cases are dealt with separately as the special situations: (1) the points coincide with the boundary node points and (2) the points are on the horizontal segments of the boundary outline.

The swathing method requires the overhead time to sort the polygon segments in order to prepare the original polygon into the appropriate swaths. If there is only one point under consideration to determine its position relative to a polygon, it may not be worthwhile to create the swaths. However, this algorithm becomes very efficient when the polygon has large number of segments and there are large number of points being considered. This technique is best suitable to such a situation since the swaths are created just once, then, all the points can utilize the same swaths.

### 5.2.2 Algorithm of Calculating the Area of the Polygon

Computation of the area of an irregular polygon is a common task in all stages of mining planning. The applications of such an algorithm include computing the ore reserve within a polygonal boundaries and calculating the average grade weighted by the mining blocks or partial mining blocks within the planning boundary. An algorithm to calculate the exact area of any irregular polygon using the vertex coordinates is introduced in this section. The mathematical derivations of this algorithm are given in two different approaches.

[1] **Green’s Integration Method**

**Green’s Theorem in the Plane**, which is explained in most of the advanced calculus text book, is one of the main topics of advanced calculus. This theorem gives an exact relation between a line integral taken around the curve $C$ (or curves) forming the boundary of region $\mathcal{R}$ and a certain double integral taken over the region. With the aid of this theorem we can derive the basic equation for
for computing the exact area of an irregular polygon.

The area of region $\mathcal{R}$ enclosed by an outline of curve $C$ can be calculated with the following line integral when the perimeter is traversed in the counterclockwise direction (Spiegel, M.R., 1971; Taylor, A.S. and Mann, W.R., 1983),

$$ A = \int_{\mathcal{R}} dxdy = \frac{1}{2} \oint_{C} (xdy - ydx) \quad (5.9) $$

Where $A$ is the area of region $\mathcal{R}$, $x$ and $y$ are the coordinate values of a point on the perimeter of region $\mathcal{R}$, and $C$ is the perimeter of the region $\mathcal{R}$ in the counterclockwise direction (see figure 5.9).

![Figure 5.9](image)

**Figure 5.9**  Geometry of Region $\mathcal{R}$ and Its Perimeter $C$ in Counterclockwise Direction

By discretizing, $x_i$ can be substituted for $x$, $y_i$ for $y$, $x_{i+1} - x_i$ for $dx$, $y_{i+1} - y_i$ for $dy$, and $n$ distinct line segments for the continuous perimeter $C$. Discretizing equation (5.9) produces,

$$ A = \frac{1}{2} \sum_{i=1}^{n} [x_i(y_{i+1} - y_i) - y_i(x_{i+1} - x_i)] \quad (5.10) $$

Expanding equation (5.10) yields,

$$ A = \frac{1}{2} \sum_{i=1}^{n} (x_iy_{i+1} - x_iy_i) - x_{i+1}y_i + x_iy_i) \quad (5.11) $$
Cancelling the second and fourth terms in equation (5.11) leaves,

\[ A = \frac{1}{2} \sum_{i=1}^{n} (x_i y_{i+1} - x_{i+1} y_i) \]  

(5.12)

Equation (5.12) uses \( x_{n+1} \) and \( y_{n+1} \) which are not in the range of points from 1 to \( n \). Because the polygon is a continuous loop, it is not hard to realize that \( x_{n+1} = x_1 \) and \( y_{n+1} = y_1 \).

Equation (5.12) can be used to compute the area of any polygon provided that the points \([ (x_i, y_i) \] for \( i = 1 \) to \( n \)\) are the coordinates of the polygon vertices and they are specified in the counterclockwise direction. If the node points are clockwise, the area will be correct but it will have a negative value.

[2] Summing up the Areas of the Component Triangles

The area of an irregular polygon can be computed by summing up the areas of its component triangles since any polygon can be decomposed into a set of triangles and it is quite easy to calculate the area of a triangle. The area of a triangle is equal to \((\text{height} \times \text{base})/2\). For a triangle with vertices \([ P_A(x_1, y_1), P_B(x_2, y_2), P_C(x_3, y_3) \] as shown in figure 5.10 (a), its area is \((h \times b)/2\). Let vectors \( \mathbf{V}_1 \) and \( \mathbf{V}_2 \) be as follows:

\[ \mathbf{V}_1 = \mathbf{CA} = (x_1 - x_3)\mathbf{i} + (y_1 - y_3)\mathbf{j}, \]

\[ \mathbf{V}_2 = \mathbf{CB} = (x_2 - x_3)\mathbf{i} + (y_2 - y_3)\mathbf{j}. \]

Then,

\[ h = |\mathbf{V}_2| \sin \theta, \]

\[ b = |\mathbf{V}_1|. \]

Where \( \theta \) is the angle between \( \mathbf{V}_1 \) and \( \mathbf{V}_2 \) (see figure 5.10 (b)).
Consequently, the area of the triangle, $A_\Delta$, can be computed as follows:

\[
A_\Delta = \frac{1}{2} (h \times b) = \frac{1}{2} |V_1||V_2| \sin \theta = \frac{1}{2} |V_1 \times V_2| \\
= \frac{1}{2} |[(x_1 - x_3)i + (y_1 - y_3)j] \times [(x_2 - x_3)i + (y_2 - y_3)j]| \\
= \frac{1}{2} |[(x_1 - x_3)(y_2 - y_3) - (y_1 - y_3)(x_2 - x_3)]k| \\
= \frac{1}{2} |(x_1 - x_3)(y_2 - y_3) - (y_1 - y_3)(x_2 - x_3)|
\]

(5.13)

In the special case that point $(x_3, y_3) = (0, 0)$, the area of the triangle consisting of the points $[(x_1, y_1), (x_2, y_2), (0, 0)]$ can be computed using the simpler formula given by equation (5.14).

\[
A_\Delta = \frac{1}{2} |x_1y_2 - x_2y_1| = \frac{1}{2} |x_2y_1 - x_1y_2|
\]

(5.14)

In equation (5.14), $(x_1y_2 - x_2y_1)$ and $(x_2y_1 - x_1y_2)$ have the same absolute value, while they differ by signs. That is, if $(x_1y_2 - x_2y_1)$ is positive, then, $(x_2y_1 - x_1y_2)$ must be negative and vice versa. In practice, it does not matter whether $(x_1y_2 - x_2y_1)$ or $(x_2y_1 - x_1y_2)$ is used since the two expressions are relative to the
of the triangle vertices. To be consistent with section 5.2.1, we will use equation (5.15) as the standard formula to calculate the area of the triangle with one vertex located at the origin.

$$A_{\Delta} = \frac{1}{2}(x_1y_2 - x_2y_1)$$

(5.15)

When formula (5.15) is used to calculate the area of the triangle with one vertex located at origin, the area $A_{\Delta}$ is positive if the triangle vertices are counterclockwise. Otherwise, it is negative (see figure 5.11).

Figure 5.11 Examples of Counterclockwise and Clockwise Triangles

$$\text{Area} = \frac{(x_1y_2 - x_2y_1)}{2}$$

(a) Area = $+9$

(b) Area = $-9$

Figure 5.12 Decomposition of Polygons into Triangles

Since it is always possible to connect all the vertices of a polygon to a common point $(0,0)$ and to decompose the polygon into a set triangles as shown.
in figures 5.12 (a), (b) or (c), formula (5.15) can be used to compute each one of the triangles. If the origin point \((0, 0)\) is within the polygon boundary, the pattern shown in figures 5.12 (a) or (b) should be used in decomposition. If the origin \((0, 0)\) is outside the polygon boundary, the pattern of figure 5.12 (c) should be used.

Next, we will discuss how to obtain the area of the polygon by summing up its component triangles in different cases. To make it easier to explain, it is assumed that the polygon vertices \([(x_i, y_i), \text{for } i = 1 \text{ to } n]\) are pre-arranged into counterclockwise direction and also \(x_{n+1} = x_1\) and \(y_{n+1} = y_1\).

(1) The origin is within the polygon and the component triangles do not overlap each other (the pattern shown in figure 5.12 (a)).

![Figure 5.13: Point Inside Polygon and Component Triangles not Overlapping Each Other](image)

In the case that there are no overlapping component triangles, the areas of all the component triangles must have the same sign. It is easy and simple to sum up the areas \(A_{\Delta_i}\) of the component triangles for this case to obtain the area of the polygon. In figure 5.13, the polygon is divided into five non–overlapping triangles. Since the areas of the component triangles are all positive, the area of the polygon is simply the sum of areas of its component triangles.

\[
A = \sum_{i=1}^{n} A_{\Delta_i} = \frac{1}{2} \sum_{i=1}^{n} (x_i y_{i+1} - x_{i+1} y_i)
\]  

(5.16)
(2) The origin is within the polygon and the component triangles overlap each other (the pattern shown in figure 5.12 (b)).

\[ \Delta_{abo} \text{ is counterclockwise, } A_{abo} \text{ is positive} \]
\[ \Delta_{bco} \text{ is clockwise, } A_{bco} \text{ is negative} \]
\[ \Delta_{cdo} \text{ is counterclockwise, } A_{cdo} \text{ is positive} \]

Figure 5.14  Point Inside Polygon and Component Triangles Overlapping Each Other

\[ A_{\Delta_{abo}} + A_{\Delta_{bco}} + A_{\Delta_{cdo}} = A_{abcd} \]

This case differs from case (1) only in that some component triangles may overlap each other. Therefore, we only discuss the effect caused by the overlapping triangles. In figure 5.14, after connecting each node points to the origin point, six triangles are generated. Of the six triangles, triangles $\Delta_{abo}$, $\Delta_{bco}$ and $\Delta_{cdo}$ overlap each other. Since $\Delta_{abo}$ and $\Delta_{cdo}$ are counterclockwise and $\Delta_{bco}$ is clockwise, $A_{\Delta_{abo}}$ and $A_{\Delta_{cdo}}$, the areas of $\Delta_{abo}$ and $\Delta_{cdo}$, are positive and $A_{\Delta_{bco}}$, the area of $\Delta_{bco}$, is negative. When the positive and negative area values are added together, the contributions of the overlapped areas are automatically cancelled out. That is, $A_{\Delta_{abo}} + A_{\Delta_{bco}} + A_{\Delta_{cdo}} = A_{abcd}$. This is best illustrated in figure 5.15. As it can

Figure 5.15  Summing up the Positive and Negative Component Triangles (Point Inside the Polygon)
be seen, formula (5.16) can still be used even if some component triangles overlap each other.

(3) The origin is outside the polygon (the pattern shown in figure 5.12 (c)).

In this case, after connecting the two end points of each segment on the polygon to the origin point, a triangle will be formed between the segment and the origin. If the segment triangle is counterclockwise, the area value is positive. Otherwise, it is negative (see figure 5.16). The area of the polygon is obtained by summing up of these positive and negative segment triangle areas as it is illustrated in figure 5.16. Therefore, formula (5.16) is still applicable for this case.

Figure 5.16 Summing up the Positive and Negative Component Triangles (Point Outside the Polygon)

In summary, formula (5.16) is applicable in computing the areas of the polygons of any shape. We utilized the concept of decomposition of a polygon into a set of triangles in deriving formula (5.16). However, formula (5.16) only requires the coordinates of the polygon vertices. In practice, user only needs to input the coordinates of the polygon vertices into formula (5.16) without thinking how to decompose the polygon into a set of the triangles. If the polygon is counterclockwise, the sum is the positive area of the polygon. If the polygon is clockwise, the sum is the negative area of the polygon.
It is quite interesting that formula (5.12) and (5.16) are exactly the same, although they are derived from two completely different approaches. That is, formula (5.12) is derived from Green's Theorem in the Plan and formula (5.16) is derived from summing up the areas of the component triangles. Since formula (5.16) is simple to use, easy to understand and suitable for computer program environment, it is the general form of the method used to calculate the area of the polygon by node coordinates.

5.2.3 Polygon Clipping Algorithm

This section presents a polygon clipping algorithm used to determine the intersection and union of two polygons, and the complement of one polygon with respect to the other. The most common application of polygon clipping is either to find the intersection between two polygons (K. B. Salomon, 1978) or to calculate areas of intersection between a set of polygons and a given grid of rectangular blocks (B. J. Larkin, 1988 and R. W. Barbaro, 1985). However, in the interactive graphics mine planning application, the functions of intersection, union and complement are all required. In figure 5.17, if polygon $A$ represents the planning boundary and $B$ represents the cut polygon, configuration (a) is used to calculate the intersection between the planning boundary and the cut polygon, (b) is for unmine (or filling back), and (c) is used to determine the remaining area after the cut polygon is removed.

![Figure 5.17 Intersection, Union and Complement between Two Polygons](image)

Figure 5.17 Intersection, Union and Complement between Two Polygons
The most important design criteria for the algorithm are simplicity, generality and efficiency. The algorithm described here has the capability to handle the polygons of any shape or size subject only to the following condition: the region common to both polygons must consist of a single contiguous region. That is, the configurations of figures 5.18 (a) and (b) are allowed and the pattern of figure 5.18 (c) is disallowed. However, the problem shown in figure 5.18 (c) can be easily solved by applying the algorithm twice.

![Figure 5.18 Three Configurations for Pairs of Polygons](image)

(a) allowed  (b) allowed  (c) disallowed

Figure 5.18 Three Configurations for Pairs of Polygons
(a) and (b) allowed, (c) disallowed

Any two polygons in the same plane will satisfy one of the following relationships:

1. they completely coincide;
2. one polygon is entirely within the second one;
3. they are mutually exclusive;
4. they intersect.

Before the polygon clipping algorithm is applied, the relationship of the two polygons must be identified. Case (1) or (2) above is identified by repeatedly applying the point-in-polygon algorithm. If all the vertices of the first polygon
are coincident with those of the second one and vice versa, case (1) is true. If all the vertices of the first polygon are completely within the second one and all the vertices of the second one are completely outside the first one, case (2) is true. Cases (3) and (4) can be determined by checking if the segments of the two polygons intersect. If there is no intersection between the segments of the two polygons, case (3) is true. Otherwise, case (4) is true.

If case (1) is found, the intersection and union are the same as either of the two polygons and both complements are the empty. Case (1) is easy to deal with. However, this case does not have much practical use in mine planning.

Case (2) has its application in mine planning when the entire planning boundary under consideration is represented by large polygon $A$ and a particular cut is represented by small polygon $B$. This situation is illustrated in figure 5.19. In this case, the intersection is the same as polygon $B$, the union is the same as polygon $A$, the complement of polygon $A$, that is, $(B \not\in A)$, is the empty, and the complement of polygon $B$ ($A \not\in B$) is the area inside polygon $A$ and outside polygon $B$. The set $(A \not\in B)$ is defined by the boundaries of the two polygons to represent the remaining area after cut $B$ is removed.

![Figure 5.19 One Polygon Inside Another](image)

Case (3) is also easy to handle. In this case, the intersection is empty, the union is defined by the two disjoint polygons, and the complement of one polygon is the other polygon. As it can be seen, case (3) does not have much application in mine planning.
Following the above discussion, it can be seen that there is no actual polygon clipping calculation involved in processing cases (1), (2) and (3). The polygon clipping algorithm is mainly concerned with case (4), i.e., they intersect.

The polygon clipping algorithm is used to determine the intersection, union or complement of two polygons of any shape or size subject to the condition that the region common to both polygons must consist of a single contiguous region (see figure 5.18 (a) and (b)). The basic idea of the algorithm is that the intersection, union or complement will be new polygons with some vertices drawn from the intersections between the segments of the two polygons, certain vertices directly from the original polygons. The first step of the algorithm is to prepare both polygons into counterclockwise direction and then find all the intersections between the segments of the two polygons regardless of intersection or union or complement to be determined.

The intersection of two polygons will be the shaded area as shown in figure 5.20 (a) or (b). To determine the intersection between the two polygons is equivalent to determine the border segments of the shaded areas, or to determine the vertices of the shaded areas. The process starts with a segment from polygon A which is inside polygon B, traverses the A segments forwards until the border of B is encountered, then switches over to the B segments and traverses them (these B segments must be inside A or on the border of A) forwards until the border of A is encountered, continues alternating between A segments inside B and B segments inside A in forward direction until the starting segment is encountered (figure 5.20). All the segments (or vertices) traversed by the above path make up the intersection of the two polygons. The two patterns shown in figure 5.20 (a) and (b) are different, but, the process to determine the intersections uses the exact same LOOP in the computer program. The intersection consists of only one polygon since this algorithm is developed for the situation where the region common to both polygons consists of a single contiguous region.
The union of polygons $A$ and $B$ is defined by the border segments of the shaded areas as shown in figure 5.21 (a) or (b). To determine the union of the two polygons, the process starts with a segment from polygon $A$ which is outside polygon $B$, traverses the $A$ segments forwards until the border of $B$ is encountered, then switches over to the $B$ segments and traverses them (these $B$ segments must be outside $A$ or on the border of $A$) forwards until the border of $A$ is encountered, continues alternating between $A$ segments outside $B$ and $B$ segments outside $A$ in forward direction until the starting segment is encountered (see figure 5.21). The union always consists of a single contiguous polygon.

Figure 5.20 Intersections between Polygons $A$ and $B$

Figure 5.21 Unions of Polygons $A$ and $B$
The complement of polygon $B$ with respect to polygon $A$ (that is, $A$ not $B$) (see figure 5.22 (a) or (b)) is defined by the border segments (or vertices) of the shaded areas. The complement of $B$ with respect to $A$ can be determined in the quite similar way as to determine the intersection or union of the two polygons. One only needs to identify the border segments (or vertices) of the shaded areas in figure 5.22 (a) or (b). The process starts with a segment from polygon $A$ which is outside polygon $B$, traverses the $A$ segments forward until the border of $B$ is encountered, then switches over to the $B$ segments and traverses them (these $B$ segments must be inside $A$ or on the border of $A$) backward until the border of $A$ is encountered, continues alternating between $A$ segments outside $B$ in forward direction and $B$ segments inside $A$ in backward direction until the starting segment is encountered (figure 5.22). The above procedure must be repeated until all $A$ segments outside $B$, all $B$ segments inside $A$, and coincident segments of the opposite direction have been traversed. It should be noticed that the complement (that is, $A$ not $B$) may consist of either one contiguous polygon as shown in figure 5.22 (a), or two or more disjoint polygons as the situation shown in figure 5.22 (b).

![Figure 5.22 Complement of Polygon $B$ with respect to Polygon $A$](image_url)
5.3 Blast Hole Handling and Database Maintenance

Data handling is a very important part of the BHK system. It is also the most time consuming task in computerized grade control. Commercially available editors or word processors, and database are not suitable for handling blast hole data and geology information since they do not have the functions to efficiently process blast hole and geology data and. Moreover, they can not be linked to the hardware peripherals shown in figure 5.2. To reduce the effort required to process the large amount of blast hole data and the complicated geology information, a special data handling system was designed. It consists of three parts, namely, a blast hole editor, a blast hole database and a rock type database. This section stresses the practical aspects of solving the problems in data handling and database maintenance in the BHK system.

The primary role of the blast hole editor is to enter, edit, display and validate the blast hole locations and assay values. The blast hole database manager creates and maintains the database for all the blast holes in a mine, with the basic functions to add, update, delete and retrieve the blast holes. The rock type database has the capability to store and retrieve all the rock type information for an operating mine. The rock type database makes it possible for the programs in BHK system to automatically assign rock type codes to the blast holes and mining blocks in grade estimation.

5.3.1 Blast Hole Editor

At an operating mine, blast hole data are made available every day. These blast hole data must be recorded since they provide the most basic and valuable information for ore-waste classification. But, a great effort is required to get the blast hole data into computer since these data are collected from various sources such as assay laboratory, survey station, survey personnel and blast hole maps in bits and pieces. Moreover, coordinates and assay values are often collected
separately and they must be merged and verified.

The blast hole editor is the most frequently used program in the BHK system, since all the blast holes can be processed very efficiently by this specially designed editor. The blast hole editor stores all the collected information in a temporary database called [TMPBASE.DAT], which is actually a working file to handle the blast hole coordinates and assay values. The blast holes having their coordinates and assay values completed and verified can be interactively selected and automatically saved into the permanent blast hole database.

Since the assay laboratory and survey station are able to put the assayed values and surveyed coordinates either in data files on diskette or in paper format, it is very important for the blast hole editor to have the capability to automatically read in these files. In the event of a failure of automatically reading in the files, the blast hole coordinates and the assays should be entered through the keyboard manually. A digitizer always provides a convenient way to collect the coordinates from the blast hole maps. A visual graphics display or color map of the blast hole data is a good way to check the possible errors.

The blast hole editor was purposely designed to meet the requirements cited above. This editor provides the user with the flexible alternatives to collect and process the blast hole data. According to the available information and equipment, the user can choose any one of the options from the main menu of the blast hole editor shown in figure 5.23 to enter, edit and verify the blast hole data.

<table>
<thead>
<tr>
<th>Main Menu of Assay Data Editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1 Load Lab Assay Data File to the Working File</td>
</tr>
<tr>
<td>F2 Load Coordinate File to the Working File</td>
</tr>
<tr>
<td>F3 Append Assay Data to the Working File Using Keyboard</td>
</tr>
<tr>
<td>F4 Digitize HV / Edit / Plot / Load Data to Database</td>
</tr>
<tr>
<td>F5 Edit / Plot / Load Data to Database</td>
</tr>
<tr>
<td>F6 Exit</td>
</tr>
</tbody>
</table>

Figure 5.23 Main Menu of the Blast Hole Editor
The blast hole editor was specially designed with the following functions in order to efficiently solve the particular problems in blast hole handling.

[1] Read assay data files received from the assay laboratory.

If the coordinate already exists for an assay, the editor will automatically merge the assay with the existing coordinate. Otherwise, the editor saves the assay as a new entry. In merging the assays with the coordinates, the key reference is the hole number and the bench elevation. If the editor finds an existing hole having the same hole number on the same bench, the editor will assign the assay values to that existing hole. Figure 5.24 is a listing of output from using the first option shown in figure 5.23.

<table>
<thead>
<tr>
<th>Elev.</th>
<th>H-ID</th>
<th>Drl</th>
<th>O/W</th>
<th>Color</th>
<th>Cu</th>
<th>Mo</th>
<th>AS-Cu</th>
<th>EQUI.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2800.0</td>
<td>121</td>
<td>22</td>
<td>Ore</td>
<td>Pink</td>
<td>.600</td>
<td>.101</td>
<td>.000</td>
<td>1.305</td>
</tr>
<tr>
<td>122</td>
<td></td>
<td></td>
<td>Ore</td>
<td>Pink</td>
<td>.860</td>
<td>.042</td>
<td>.000</td>
<td>1.070</td>
</tr>
<tr>
<td>123</td>
<td></td>
<td></td>
<td>Waste</td>
<td>Blue</td>
<td>.250</td>
<td>.011</td>
<td>.000</td>
<td>.305</td>
</tr>
<tr>
<td>124</td>
<td></td>
<td></td>
<td>Leach</td>
<td>Green</td>
<td>.360</td>
<td>.014</td>
<td>.000</td>
<td>.410</td>
</tr>
<tr>
<td>1241</td>
<td></td>
<td></td>
<td>Leach</td>
<td>Yellow</td>
<td>.550</td>
<td>.030</td>
<td>.000</td>
<td>.700</td>
</tr>
<tr>
<td>2760.0</td>
<td>131</td>
<td>32</td>
<td>Ore</td>
<td>Red</td>
<td>1.100</td>
<td>.110</td>
<td>.002</td>
<td>1.650</td>
</tr>
<tr>
<td>132</td>
<td></td>
<td></td>
<td>Ore</td>
<td>Pink</td>
<td>.920</td>
<td>.040</td>
<td>.000</td>
<td>1.120</td>
</tr>
<tr>
<td>133</td>
<td></td>
<td></td>
<td>Ore</td>
<td>Pink</td>
<td>.840</td>
<td>.010</td>
<td>.001</td>
<td>.890</td>
</tr>
<tr>
<td>134</td>
<td></td>
<td></td>
<td>Ore</td>
<td>Red</td>
<td>1.200</td>
<td>.091</td>
<td>.001</td>
<td>1.655</td>
</tr>
<tr>
<td>1341</td>
<td></td>
<td></td>
<td>Leach</td>
<td>Yellow</td>
<td>.750</td>
<td>.016</td>
<td>.001</td>
<td>.730</td>
</tr>
</tbody>
</table>

Figure 5.24 An Example of the Output File from Option 1

[2] Load the files of the blast hole coordinates collected by the survey station or data collector.

If the assay values already exist for a blast hole, the editor will automatically merge the coordinate with the existing assays. Otherwise, the program saves the coordinate as a new entry. In the same way as mentioned in [1], the key reference is the hole number and the bench elevation in merging the coordinates with the assays.
Enter coordinates and assay values for the new blast holes manually using the keyboard.

This auxiliary function is necessary since it is needed in the event of a failure of automatically reading in the files. The fixed input fields for each attribute makes the data entry easier. Figure 5.25 is the screen used to append new assay data to the working file using the keyboard. Coordinates can be either entered in the corresponding fields or left as unknowns, since it is faster to input the coordinates by a digitizer.

![Figure 5.25 Appending Window of the Blast Hole Editor](image)

Edit or update the data entry on the full screen editor.

The interactive and full screen editing capability with fixed fields allows the user to edit or update the hole numbers, coordinates and assay values if corrections are needed. Figure 5.26 is the screen used for editing purpose. Coordinates can be either added to the data or updated during blast hole editing using the keyboard or a digitizer.
Assign coordinates to the blast holes using a digitizer.

Sometimes, the grade control personnel has to obtain the blast hole coordinates from the blast hole maps. In such a case, a digitizer is the most commonly used device to get the coordinates. Therefore, the blast hole editor was designed with the capability to drive the digitizer.

Display and plot the locations and the assay values using color codes for the user specified blast holes.

The graphics display will provide an intuitive way to check the possible errors. Figure 5.27 is an example of graphics display of the blast holes. A hard copy can be obtained for whatever is displayed on the screen through a plotter or a printer.
Figure 5.27  Graphics Display of the Blast Holes

[7] Load blast hole assays and coordinates to the permanent database.

The blast hole editor works on the temporary database file called [TMPBASE.DAT] to process the blast hole data. After the blast holes have the complete information entered and verified, they can be saved into the permanent blast hole database for future uses. For example, the kriging program will pick up the data according to the input shot boundary from the permanent database instead of the temporary file [TMPBASE.DAT]. It is necessary for the editor to have the function to send the blast hole data into the permanent database.

[8] Check and verify that all input values are within the pre-assumed ranges. It also detects the duplicated blast holes.

The validation of information can be expedited partly through the default parameters and their limits which are stored in the project control file and partly through computer matching of related names and/or variables. Of course, the final validation should be made by the user through displayed information.
5.3.2 Blast Hole Database

The stand alone blast hole database manager creates and maintains the database for all the blast holes collected in an operating mine. The basic data structure within the database is a "double direction linked list". The sample information can be quickly obtained by the linked list relationship. Packing technique is used to store the assay values in order to save the computer disk space.

Since there may be up to several hundred thousand blast holes for a mine, it is not wise nor realistic to save all the blast hole data in one single data chain. The data structure of the blast hole database consists of many data chains. The entire orebody and its surrounding area are divided into regular blocks (see figure 5.28) in order to systematically and efficiently organize and access the data chains. All the blast holes within a particular block are stored in the same data chain.

![Figure 5.28 Dividing Orebody by Regular Grids for Data Structure Purpose](image-url)
In figure 5.28, \( N_X \) is the number of blocks in the east–west direction, \( N_Y \) is the number of blocks in the north–south direction and \( N_Z \) is the number of benches. The total number of the blocks is equal to \((N_X \times N_Y \times N_Z)\). It must be emphasized that the concept of blocks used here is purely for data structure purpose. After the orebody is divided into the regular blocks as shown in figure 5.28, any one block with the grid coordinates \((i_x, i_y, i_z)\) can be easily addressed by the following formula:

\[
ID(i_x, i_y, i_z) = i_x + (i_y - 1) \times N_X + (i_z - 1) \times N_X \times N_Y
\]  

(5.17)

The blast hole database consists of two direct access files. They are [BHINDEX.DAT] and [BHASSAY.DAT]. The file [BHINDEX.DAT] is used to keep the starting record for each data chain, whereas file [BHASSAY.DAT] is used to store all the data information. The starting record for a block (or a data chain) saved in file [BHINDEX.DAT] is the record number at which the first blast hole of that block is stored in file [BHASSAY.DAT]. Since all the blast holes within a block are stored on the same data chain in file [BHASSAY.DAT], each block has only one starting record where the first blast hole of this block is stored. After the record number of the first blast hole for a block is known, all the blast holes within that block can be reached following the successor or predecessor linking pointers since all the blast holes of each block is linked on the same data chain by the double direction linking pointers.

The file [BHINDEX.DAT] has \((N_X \times N_Y \times N_Z)\) records so that to every record in file [BHINDEX.DAT] there corresponds a block. The record \(ID(i_x, i_y, i_z)\) in file [BHINDEX.DAT] stores only one single integer number denoted by \(irec(i_x, i_y, i_z)\), which is the record number of the first blast hole of the data chain for block \((i_x, i_y, i_z)\) in file [BHASSAY.DAT]. Initially, the starting record numbers of all the blocks are set to zero. Therefore, a zero starting record number
indicates that there is no data saved in assay file for the respective block.

The size of file [BHINDEX.DAT] will not change as more blast hole data are entered into the database. The size of file [BHASSAY.DAT] is getting bigger and bigger as more and more blast hole data become available.

In summary, there exist the following relationships between \( ID(ix, iy, iz) \), \( irec(ix, iy, iz) \), [BHINDEX.DAT], [BHASSAY.DAT], and the blocks in figure 5.28:

1. \( ID(ix, iy, iz) \) computed using formula (5.17) is the identification number for the block located at grid \((ix, iy, iz)\).

2. \( irec(ix, iy, iz) \) is the record number at which the first blast hole of the data chain for block \((ix, iy, ix)\) is stored in file [BHASSAY.DAT].

3. \( ID(ix, iy, iz) \) is also equal to the record number at which the starting record \( irec(ix, iy, iz) \) for block \((ix, iy, ix)\) is stored in file [BHINDEX.DAT].

In file [BHASSAY.DAT], each blast hole takes one record which contains the following items:

1. predecessor of the current record,
2. successor of the current record,
3. northing coordinate of the blast hole,
4. easting coordinate of the blast hole,
5. elevation of the blast hole,
6. assay values of all the variables.

A zero value of the predecessor of a record in the file [BHASSAY.DAT] indicates that the current record is the first blast hole of a data chain. Whereas a zero value of the successor of a record means the current record is the last blast hole of a data chain.
Although the block size can vary arbitrarily, to choose a reasonable block size is very important. If the block size is too big, too many data will be in each data chain, which will cause the problem for efficiently getting to access each datum on the chain. On the other hand, if the block size is too small, it will need too much disk space to keep the starting records for all the chains. In practice, the best block size should be determined by the blast hole spacing. By the author's experience, each chain should have around 100 to 150 blast holes on it. The vertical dimension of the blocks should correspond to the bench height. If the blast hole spacing is (30 x 30 ) feet, the horizontal dimensions of the blocks should be around 400 feet. In such a case, each data chain will have about 150 blast holes on it.

The blast hole database was designed to have the functions to add new data to database, modify, delete and retrieve the existing data in the assay database. The mechanics of the functions of the blast hole database is discussed below:


In order to add a blast hole to the database, the following things have to be done:

1. Determine the grid coordinates \((ix, iy, iz)\) of the block to which the blast hole to be added belongs.

2. Utilize formula (5.17) to calculate \(ID(ix, iy, iz)\), which is the record number in file [BHINDEX.DAT] to store the starting record of the data chain for block \((ix, iy, iz)\) in file [BHASSAY.DAT].

3. Read the starting record number \(irec(ix, iy, iz)\) for block \((ix, iy, iz)\) saved in record \(ID(ix, iy, iz)\) in file [BHINDEX.DAT].

4. If \(irec(ix, iy, iz) = 0\), i.e, there is no blast hole in assay file for block \((ix, iy, iz)\), add this blast hole at the end of assay file [BHASSAY.DAT]. At the same time, set the predecessor and successor pointers equal to zero, and update
the starting record of the data chain for this block by setting \( irec(ix, iy, iz) \) equal to the record number at which this blast hole is stored. Then go to the next blast hole. If \( irec(ix, iy, iz) > 0 \), i.e., some blast holes of this block already exist in the database, then go to step 5.

5. Read the link pointers of the first blast hole of block \((ix, iy, iz)\) stored at record \( irec(ix, iy, iz) \) in assay file \([BHASSAY.DAT]\).

6. Follow the successor pointer one by one to find the last blast hole for block \((ix, iy, iz)\).

7. Update the successor pointer of the last blast hole just found since after the new hole is added, the previously last hole on the data chain will no longer be the last one, but the one next to the last hole.

8. Add the new blast hole at the end of assay file \([BHASSAY.DAT]\), let the predecessor pointer equal to record number at which the previously last hole of the data chain is stored and let the successor pointer equal to zero.

The above process is repeated for each one of the blast holes to be added. Before running the program to add the new blast holes to the database, the user must prepare a data file containing the coordinates and the assay values of the blast holes to be added.

[2] Update the data within the database.

The procedure to update the data is as follows:

1. Determine the grid coordinates \((ix, iy, iz)\) of the block to which the blast hole to be added belongs.

2. Calculate \( ID(ix, iy, iz) \) using formula (5.17).

3. Read the starting record number \( irec(ix, iy, iz) \) for block \((ix, iy, iz)\) saved in record \( ID(ix, iy, iz) \) in file \([BHINDEX.DAT]\).
4. Start at record $irec(ix, iy, iz)$ in assay file [BHASSAY.DAT], follow the successor pointer to read the blast hole data, identify the blast hole that has exactly the same coordinates as the one to be updated.

5. Replace the old assay values by the new assay values.

To update an existing assay values, the user should provide the coordinate of this datum and the new assay values. The coordinates can not be modified.

[3] Delete the data within the database.

The procedure to delete the data within the database is different from that to update the data. As each blast hole is removed from the database, the very last blast hole in the database must be moved to the record which is originally taken by the deleted blast hole. This will require to update the link pointers of two data chains. One is the chain containing the blast hole to be deleted and the other is the chain containing the very last blast hole in the database.

1. Determine the grid coordinates $(ix, iy, iz)$ of the block to which the blast hole to be added belongs.

2. Calculate $ID(ix, iy, iz)$ using formula (5.17).

3. Read the starting record number $irec(ix, iy, iz)$ for block $(ix, iy, iz)$ saved in record $ID(ix, iy, iz)$ in file [BHINDEX.DAT].

4. Start at record $irec(ix, iy, iz)$ in assay file [BHASSAY.DAT], follow the successor pointer to read the blast hole data, identify the blast hole that has exactly the same coordinates as the one to be deleted.

5. Move the blast hole at the bottom of assay file [BHASSAY.DAT] to the record which was occupied previously by the blast hole to be deleted.
6. Update the link pointers of the two data chains. They are the chain containing the blast hole to be deleted and the chain containing the very last blast hole in the assay database.

In order to delete data from the database, the program provides the user with two options: (a) deleting all data that have the same coordinates as in the user provided data list, or (b) deleting all the data within the user specified polygon.

[4] Retrieve the data contained in the database.

This procedure is simple since it does not need to change the link pointers if some data in the database are retrieved. To retrieve blast holes, a boundary polygon is needed. The program will list all the data within the specified polygon.

1. Determine the blocks that are within the user specified polygon.

2. List all the data for each of the blocks just identified above and use point-in-polygon algorithm to check if each one of the data is within the polygon.

5.3.3 Rock Type Database

Since assay variables in different rock type may have different mineralization, it may be desired to estimate the grade for a block only using the blast holes from the same rock type in order to get a more accurate estimation results, particularly in a precious metal mine. However, it is difficult for mine geologists to gather geological data in a form which can be conveniently added to blast hole data. Therefore, a rock type database is designed to keep the geologic boundaries for the entire mine. The rock type database program creates and maintains the database for all collected rock type information.

Polygons are used to specify the boundaries for different rock types. That is, one polygon is used to define a given rock type. The rock type database allows the user to extend and update the database as more rock type information becomes
available. Rock type polygons can be overlapped. In the overlapped area, rock type will be assigned by the last input polygon. Therefore, the user can always use the latest available rock type information to update the rock type database by specifying and inputting the new rock type polygons to the existing rock type database. The rock type database can display or plot the rock type boundaries, which provides the user with an easy way to check the input rock type polygons.

The rock type information provided by the geological map shown in figure 5.29 can be easily input to the rock type database by several polygons.

The data structure of the rock type database is organized relative to the bench. All the rock type polygons in a bench are stored on the same data chain. Therefore, each bench has only one data chain.

The assignment of the rock type to each blast hole or a block to be estimated is accomplished automatically by the rock type database using the point-in-polygon algorithm. If it is desired to utilize the available geologic information in grade estimation, the user only needs to set the corresponding switch on without thinking about the mechanics. Then, the program will automatically assign geologic codes to both individual blast holes and the blocks to be estimated.
5.4 Interactive Graphics Planning

The interactive graphics planning is the major part of the BHK system since the tasks of grade control are done in the interactive graphics planning mode. To make the computerized grade control process as smooth and as rigorous as possible, it is very important to define a proper way to perform all the tasks of the interactive graphics planning. The programs used for the interactive graphics planning can be classified into two categories according to the function of each program; namely, 1) kriging system, and 2) interactive ore-waste delineations. This section will discuss how the problems are solved for the interactive graphics planning.

5.4.1 Kriging System

To make the kriging program more user friendly and to reduce the chance of user errors, four sub-programs are designed to perform the following four tasks: 1) editing variogram parameters, 2) editing the run control parameters for the kriging system, 3) specifying the shot boundary to be kriged and retrieving the blast holes which will be used for grade estimation, and 4) the actual process of kriging.

The entry of the variogram and the run control parameters can be minimized and simplified through the use of the variogram editor and kriging parameter editor, since it usually presents difficulty for new personnel to create the run files for various options and parameters. Both of the above editors present full screen panels for the user to enter or update the run parameters. The editors always display the existing parameters which can be accepted or modified as needed.

It is important to verify the shot boundary to be modeled and the blast holes used for grade estimation before further work is done. Therefore, this part of the work is done by an individual program just prior to running the kriging program.

Although the major inputs (variograms, run control parameters, shot boundary and blast holes) are prepared by the above three auxiliary programs, the kriging program still has to retrieve rock type polygons from the database, assign the rock
type codes to both the blast holes and the blocks and load the correct variogram for each block according to the rock type of that block and the specified pushback before a block can be kriged. All the above processes are fully automated.

[1] Variogram Editor

Assay variables in different areas and different rock types may have different spatial correlation structures and the different mineralizations. If kriging method is used to estimate the grade of the interested areas, different variograms should be used for different areas and the different rock types in order to obtain a more accurate estimation result. Variograms belonging to different areas can be distinguished by pushback number used in mine production. Variograms for each rock type can be computed for each pushback that is currently underway. Usually, there are more than one working area (pushback) in production, there are more than one assay variable under consideration and there are several rock types involved. As a consequence, there may be up to $N(\text{assay variables}) \times M(\text{rock types}) \times K(\text{pushback numbers})$ variograms needed in grade estimation.

It may cause great confusion and the chance of user errors if each variogram is stored in an individual data file. Therefore, there is one program served as an interactive, full-screen, menu guided editor purposely designed for the BHK system to provide a systematic way to enter, update and store the variogram parameters for all assay variables, all rock types and all production areas in a single file.

Figure 5.30 (a) is the main menu of the variogram editor. The user can choose the assay variable for which to edit the variograms from this menu. After an assay variable is selected, the second menu as shown in figure 5.30 (b) will appear on the screen. Then, the user can choose the rock type for which to edit the variograms from the second menu. Finally, the main editing panels as shown in figure 5.30 (c) are used to enter and update the parameters for the selected assay variable and the rock type.
Assay Variable Selection Menu
1. Edit variogram parameters for assay: Cu
2. Edit variogram parameters for assay: No
3. Edit variogram parameters for assay: As-Cu
4. Save and Exit
5. Quit

(a) Assay variable selection menu

Rock Type Variable Selection Menu

(b) Rock type variable selection menu

Main Editing Screen

<table>
<thead>
<tr>
<th>Pushback number</th>
<th>100</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variogram type</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Angle of anisotropy</td>
<td>135.09000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>Horizontal anisotropy factor</td>
<td>1.100000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>Vertical anisotropy factor</td>
<td>2.000000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>Nugget</td>
<td>0.000000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>Sill of 1st nested model A1</td>
<td>0.000000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>Range of 1st nested model A2</td>
<td>400.000000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>Sill of 2nd nested model A3</td>
<td>0.000000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>Range of 3rd nested model A4</td>
<td>0.000000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>Sill of 4th nested model A5</td>
<td>0.000000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>Range of 5th nested model A6</td>
<td>0.000000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
</tbody>
</table>

1) Spherical 2) Linear 3) Exponential 4) Gaussian 5) Ex-wijian 6) Cubic

Press Esc key to go back to the main menu

(c) Main editing screen

Figure 5.30 Variogram Editor
Such a variogram editor makes the work simple and efficient to enter, update and maintain all the variogram parameters. When performing grade estimation, the user only needs to specify the pushback number. Kriging program will automatically load the correct variogram for each block to be kriged according to the rock type of the block and the specified pushback number.

As production progresses and more blast holes become available, variograms should be remodeled periodically, such as yearly. After each modification of the variograms, the variogram editor should be run to update the variogram parameters.

[2] **Kriging Parameter Editor**

All the run control parameters used by the kriging program should be updated before each shot is modeled. To simplify the editing work and to minimize the user error, an interactive, full-screen, menu guided editor was designed. The following parameters are entered and updated by this editor.

1. Pushback number which tells the kriging program to automatically load the corresponding variogram(s) from the variogram parameter file.
2. Elevation of the active bench (only one bench can be processed for each run).
3. Size of the selective mining unit, i.e., size of blocks to be kriged.
4. Maximum search radius to include the samples.
5. Maximum number of holes to be used in estimation.
6. Minimum number of holes needed before a block is estimated.
7. Number of above benches to include to search samples.
8. Specification of rock type information to be used in grade estimation.
9. Specification of interpolation method used for each assay variable in grade estimation. Two interpolation methods are available, namely, ordinary kriging and inverse distance squared weighting (IDS). Different interpolation methods can be used for each assay variable in the same run. For example, the first variable is estimated by kriging and the second variable is estimated by IDS.

<table>
<thead>
<tr>
<th>Descriptive title of this run: Test Run BHSETUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pushback number</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>Bench elevation</td>
</tr>
<tr>
<td>2960.0</td>
</tr>
<tr>
<td>X-direction length of mining block</td>
</tr>
<tr>
<td>30.0</td>
</tr>
<tr>
<td>Y-direction length of mining block</td>
</tr>
<tr>
<td>30.0</td>
</tr>
<tr>
<td>Maximum search radius to include sample</td>
</tr>
<tr>
<td>200.0</td>
</tr>
<tr>
<td>Maximum number of holes to be used in grade estimation</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>Minimum number of holes for a block to be estimated</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>Number of active benches to be included for sample selection</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>Do you want to use rock type in grade estimation (Y/N)?</td>
</tr>
<tr>
<td>Yes</td>
</tr>
<tr>
<td>Shot number</td>
</tr>
<tr>
<td>200</td>
</tr>
<tr>
<td>Date at which this shot was pulled (dd/mm/yyyy)</td>
</tr>
<tr>
<td>7/20/1989</td>
</tr>
</tbody>
</table>

Figure 5.31 Editing Panels for Setting up Kriging Parameters

Figure 5.31 shows the first screen of the editing menu for setting up the control parameters.


The area to be modeled in the daily grade control is defined by the shot boundary, which is usually digitized from a working bench map. But, to be sure that the system will work under any circumstance, the program also provides the user with the options to input the shot boundary through a data file or to interactively input from the keyboard.

After the shot boundary is specified, the existing blast holes within the shot boundary as well as within the neighborhood from both the active bench and the
above benches that have been previously mined will be automatically retrieved from the permanent blast hole database. These blast holes will be saved in a file which can be used as the input to the grade estimation program. Then the program will display and plot both the shot boundary and the blast holes to provide an easy way to check if there is any mistake. Figure 5.32 shows a graphics display after a shot boundary is input.

Figure 5.32   The Graphics Display of the Shot Boundary

The shot boundary should be as accurate as possible in order to get an accurate tonnage estimation, since the BHK system does not keep the record for the previously mined areas. Therefore, all the blocks within the shot boundary will be estimated. The tonnage will be calculated based on the exact geometry of the input shot boundary.
Grade Estimation

Two estimation techniques (ordinary kriging and IDS) are available to assign the average grade to each selective mining unit (SMU) within the user specified shot boundary for any of the assay variables that are associated with the mine model. The program also computes the estimated total tonnages and average grades of each material type (e.g., ore, leach, waste) contained within the specified shot boundary. The kriging method is highly recommended since it gives the unbiased estimate with the minimum error variance. The IDS method provides a good starting point when the variograms are not available.

To obtain a more accurate estimate, geology should be an important parameter, particularly in precious metal mines. If it is desired to use rock type information in grade estimation, the user only needs to set the corresponding switch on without thinking about the mechanics. In such a case, the program will first automatically retrieve all the rock type polygons stored in the rock type database which intersect the shot boundary from the current bench and the above benches, assign the rock type to all the blast holes within the shot boundary as well as within the neighborhood from both the active bench and the above benches using the point-in-polygon algorithm, and also assign the rock type to each one of the blocks within the shot boundary. Then, the program will estimate the grades for the blocks within the shot boundary one at time. Before a block is kriged, the program will identify the rock type of the block being kriged, then pick up the correct variogram which is calculated by the samples of the same rock type within the specified pushback, and search the nearest blast holes having the same rock type.

In order to make the program work at any case, some necessary assumptions are made as follows:

1. The kriging program estimates the grade for a block using only the blast holes from the same rock type if the block was already assigned a rock type.
2. For those blocks where rock type information is not available, the kriging program automatically uses all the blast holes within the search radius to perform the grade estimation.

3. For those blocks where rock types are not available, the program uses the variogram of the first rock type in kriging.

4. If a rock type does not have a variogram for the specified pushback, the program uses a variogram of the same rock type for other pushbacks if a variogram is available for other pushbacks. If there is no variogram available for a rock type for any pushback, the program utilizes IDS method instead of kriging method in grade estimation.

If rock type information is not required in kriging, the program always uses the first variogram in the list for the current pushback.

Figure 5.33 is a sample summary output from kriging for a shot boundary.

<table>
<thead>
<tr>
<th>GRADE INTERVAL</th>
<th>AREA (ft*ft)</th>
<th>VOLUME (cubic ft)</th>
<th>TONNAGE (1000*T)</th>
<th>EQUIVALENT GRADE</th>
</tr>
</thead>
<tbody>
<tr>
<td>.0000-.1000</td>
<td>4400.0</td>
<td>220000.0</td>
<td>17.60</td>
<td>.3148</td>
</tr>
<tr>
<td>.1000-.2000</td>
<td>7600.0</td>
<td>380000.0</td>
<td>30.40</td>
<td>.4924</td>
</tr>
<tr>
<td>.2000-.3000</td>
<td>8000.0</td>
<td>400000.0</td>
<td>32.00</td>
<td>.7206</td>
</tr>
<tr>
<td>.3000-.4000</td>
<td>15600.0</td>
<td>780000.1</td>
<td>62.40</td>
<td>.9558</td>
</tr>
<tr>
<td>&gt;4000</td>
<td>&gt;4000</td>
<td>&gt;200000.0</td>
<td>&gt;1.60</td>
<td>&gt;1.6167</td>
</tr>
</tbody>
</table>

Waste: 4400.0 220000.0 17.60 .3148
Leach: 15600.0 780000.0 62.40 .6054
Ore: 16000.0 600000.1 64.00 1.0152

Figure 5.33 Example Summary Output from Kriging
5.4.2 Interactive Ore–Waste Delineations

The job of ore–waste delineations is done in an interactive graphics planning mode. The interactive graphics planning program is very efficient and powerful for ore-waste delineations after blasting of each shot in daily production. The program displays the kriged grades of the blocks for a shot area using color codes for five interval cutoff grades (see figure 5.34). The first and the second color codes represent high grade and low grade ore material. The third and the fourth color codes represent high grade and low grade leach material. The fifth color code is for waste. The thresholds of the five color codes are input by the user specified cutoff grade file. Blast holes can be optionally superimposed on the estimated block grades. Moreover, the interactive graphics planning program is a sophisticated grade-tonnage calculator which allows the user to interactively and rapidly input the ore-waste boundaries on the screen using a mouse or the keyboard, as it is to be flagged in the pit in actual production.

The program calculates the exact tonnages for any shaped polygons utilizing the polygon clipping algorithm which was developed in section 5.2. After each cut polygon is input, the program automatically determines and mines the intersection of the shot boundary and the cut polygon on the screen. Each cut is maintained by the nodal coordinates of the intersection of the shot boundary and the cut polygon, which is saved for later use. The grade and tonnage summary is calculated and displayed simultaneously as each cut polygon is input (see figure 5.34). The user can either accept the current ore-waste boundaries or re-try different boundaries during the same run. Different alternatives of the ore-waste delineations can be investigated in order to obtain the best grade-tonnage combination.

Classification of the material to be mined as ore, leach or waste material cannot be done solely on the estimated grade of gridded blocks. Mining constraints, including the orientation of the mining face and the selectivity of the mining equipment, must be taken into account.
At the end of a planning session, the program generates the following outputs:

1. A data file which only contains the coordinates of the ore-waste polygons and can be fed back to survey station to directly flag the ore-waste boundaries through survey station.

2. A summary report of the estimated tonnages and grades for each ore-waste polygon boundary within the shot, which is useful for the production scheduling to determine the daily availability of ore and waste material (figure 5.35).
## REPORT OF ORE-WASTE DELINEATION

<table>
<thead>
<tr>
<th>POLYGON #: 1</th>
<th>POLYGON #: 2</th>
<th>POLYGON #: 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>TONNAGE = 38125</td>
<td>TONNAGE = 36875</td>
<td>TONNAGE = 24844</td>
</tr>
<tr>
<td>EQU-GRADE = 1.1625</td>
<td>EQU-GRADE = 1.1194</td>
<td>EQU-GRADE = 0.5471</td>
</tr>
<tr>
<td>Cu GRADE = 1.1194</td>
<td>Cu GRADE = 0.5263</td>
<td>Cu GRADE = 0.3466</td>
</tr>
<tr>
<td>Mo GRADE = 0.0093</td>
<td>Mo GRADE = 0.0053</td>
<td>Mo GRADE = 0.0022</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>POLYGON NODES (X,Y)</th>
<th>POLYGON NODES (X,Y)</th>
<th>POLYGON NODES (X,Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2140.0 422.1</td>
<td>2103.4 278.8</td>
<td>2227.1 365.3</td>
</tr>
<tr>
<td>2140.0 520.0</td>
<td>2050.0 285.0</td>
<td>2225.0 350.0</td>
</tr>
<tr>
<td>2220.0 520.0</td>
<td>2140.0 265.0</td>
<td>2180.0 270.0</td>
</tr>
<tr>
<td>2240.0 460.0</td>
<td>2140.0 422.1</td>
<td>2103.4 278.8</td>
</tr>
<tr>
<td>2234.5 419.4</td>
<td>2234.5 419.4</td>
<td>2172.8 359.5</td>
</tr>
<tr>
<td>2234.5 419.4</td>
<td>2234.5 419.4</td>
<td>2172.8 359.5</td>
</tr>
<tr>
<td>2172.8 359.5</td>
<td>2172.8 359.5</td>
<td>2172.8 359.5</td>
</tr>
</tbody>
</table>

Figure 5.35 An Example Output File from the Interactive Graphics Ore-Waste Delineation

Figure 5.36 An Example Plot of Ore-Waste Delineation
3. A map including the shot boundary, estimated grades of blocks, blast holes (optional) and the ore-waste boundaries which can be used by the pit foreman for flagging different zones of ore and waste and by the shovel operator for material routing (see figure 5.36).

4. If desired, a numerical map of the estimated grades of the blocks can be generated by the BHK system (figure 5.37). Each square represents a block and two values are printed in each square.

| ASSAY VARIABLE #1: Cu, SCALE FACTOR: 1000.0 |
| ASSAY VARIABLE #2: Mo, SCALE FACTOR: 1000.0 |

![Figure 5.37  A Numerical Map of the Estimated Block Grades](image-url)
5.5 **Summary and Future Enhancements**

The primary goal of the BHK system is to increase the ore tonnages and to achieve a better comparison between mine estimates and the actual mill grade through improved ore-waste selection procedure to reduce mis-classification errors of different classes of material at an operating mine. The developed BHK system is indeed an operational and practical tool which is implemented at one of the operating copper mines in Arizona.

Although the BHK system is a quite complex and integrated computer program package to perform some highly complex operations in grade control, this complexity is not visible to the users. The system is extremely simple to operate. For example, some complicated operations are involved in polygon clipping, graphics display and grade-tonnage calculation whenever an area is cut away (mine) or put back (unmine). But this kind of task can be done by a mouse quite easily without the user having to think about the actual processes. Another example is the database maintenance for both blast holes and rock type polygons. Some data structures are discussed in section 5.3, but the user do not need to think the problems at that level.
CHAPTER 6

CONCLUSIONS AND RECOMMENDATION

6.1 Conclusions

As mentioned in chapter 1, this study covers three major topics in computerized open pit mine design and planning. These topics are:

1. develop a new graph theory oriented algorithm for ultimate pit limit design,
2. investigate the current approaches to long range mine production scheduling,
3. discuss the problem solving techniques for the practical problems involved in development of a computerized interactive grade control system for ore-waste delineation in daily production.

With respect to the first topic, a graph theory oriented algorithm for ultimate pit limit design is developed and implemented in chapter 3. Based on the two theorems (i.e., optimality theorem and convergence theorem) introduced in this study, the new algorithm assures to generate a true optimal solution and maximize the total undiscounted net profit for a given 3-D mine model in a finite number of iterations. The performance of the new algorithm is much better than the well known Lerchs–Grossmann algorithm, in terms of reduction in computation time, reduction in computer memory requirements, and simplicity in implementation. The general procedures and the suggested specific steps of the algorithm, the mathematical proofs of its optimality and convergence, some experience and strategies in implementation of the algorithm, and two case study results are given in chapter 3.
Based on this author's experience, some strategies can be used to reduce the computation time considerably. These strategies are: (1) using the shell cone template to identify the restricting blocks to a particular ore block without performing any actual search, (2) using the model size reduction technique to reduce the physical size of the model to 20 to 40 percent of the original size, which still contains the optimum pit limit, (3) using the vertical bottom-up search procedure to locate the blocks between the maximum possible pit limit and the current surface to avoid unnecessary repetitions, and (4) optimization by layers. Two case study results presented in chapter 3 indicate that the solution time for real mine models applying the new algorithm developed in this study is well within the practically acceptable range. In addition to obtaining true optimal pit limit designs through the application of the new algorithm, more benefits can also be gained from solving long range mine planning problems, since the pit limit algorithm is one of the most important basic tools used in long range mine planning.

With respect to the second topic, both the proposed mathematical optimization approach and the popular trial and error "pushback" approach to long range mine planning are investigated. The emphasis is placed on the demonstration of the impossibility of the application of the proposed mathematical approaches to obtain a "true optimum" solution using today's computing facility and the available operations research techniques.

During the last two decades, the combination of the Lagrangian relaxation with the ultimate pit limit algorithm is considered as the possible direction to optimal production scheduling (Johnson, 1968; Davis-Williams, 1973; Dagdelen, 1985; and Elevli, et al., 1989). The concept of the Lagrangian relaxation technique is not new. Some new thoughts about the proposed approaches given in this dissertation show that it is impossible to generate a true optimum solution to the general, practical situation through such an approach.
The proposed mathematical approach consists of two parts; (1) removing the practical constraints, such as ore and waste tonnage constraints, through the Lagrangian relaxation method in order to simplify the scheduling problem into the form of the ultimate pit limit problem, and (2) solving the relaxed problem by an ultimate pit limit algorithm. For a given set of the Lagrange multipliers, an ultimate pit limit can be determined. Generally, the solution obtained is unlikely to satisfy the tonnage constraints. The problem then is to determine these particular Lagrange multipliers which will result in a solution satisfying those tonnage constraints. However, there are great difficulties to obtain such a set of Lagrange multipliers for the reasonably specified ore and waste tonnage constraints for the practical mine production scheduling problem. There are two major problems associated with the application of such an approach to mine production scheduling. The first problem is the non-convergence due to the existence of redundant optimal solutions. The second one is the non-convergence due to the requirement of advanced stripping. These two problems can not be tackled by the relaxation of error tolerances of the targeted ore and waste tonnage constraints nor improved methods to find the Lagrangian multipliers. These two problems are not caused by the method of Lagrangian multipliers per se, but the application of the ultimate pit limit algorithm to the relaxed mine production scheduling problem.

The ultimate pit limit algorithm does not have any capability to detect redundant optimal solutions. When there exist redundant optimal solutions, the proposed approach combining the Lagrangian relaxation with the ultimate pit limit algorithm can not identify any of them. Instead, the approach will provide a solution which is either the union of all the redundant optimal solutions or empty. Consequently, the solution will not converge in the case of existence of redundant optimal solutions, since no Lagrange multipliers can be found such that the solution will meet both the ore and waste tonnage requirements. The ultimate pit limit algorithm can not mine the waste blocks which are not the restricting blocks to the ore blocks. However, this situation is not true in the production scheduling problem.
Some waste blocks, which may not be the restricting blocks to the ore blocks to be scheduled in the current period, must be mined in the current period in order to expose enough ore material for subsequent periods. The above requirement of advanced stripping is employed in almost all production scheduling. The ultimate pit limit algorithm does not have any capability to handle the required advanced stripping. If advanced stripping is required, the approach combining the Lagrangian relaxation with the ultimate pit limit algorithm will not converge. The importance of the investigation about the application of the Lagrangian relaxation technique to mine production scheduling made by chapter 4 is not as much as to contribute to the solution techniques in mine production scheduling, but to clarify the concept of the approach combining the Lagrangian relaxation with the ultimate pit limit algorithm and to prove the impossibility of such a research direction.

With respect to the third topic, the practical problems which must be solved in the implementation of the computerized grade control, and the relevant problem solving techniques are discussed.

The planning boundaries, the surface contours, haul road access, and existing production constraints have to be specified. Point-in-polygon algorithms are the most basic technique used in computerized mine planning. Polygon area algorithms are the most important technique in accurately computing tonnages and grades in mine planning. Polygon clipping algorithms are necessary to determine the intersection, union and complement between polygons. As it can be seen, all the polygon manipulation algorithms play a key role in mine planning. For this purpose, the applied geometry concepts that underlies point-in-polygon algorithms, polygon area algorithms, and polygon clipping algorithms are developed.

Data handling is very important in daily grade control. This is because the large amount of data become available every day from various sources in bits and pieces. Coordinates and assay values must be merged and verified. A special data handling system which consists of a blast hole data editor, a blast hole database and
a rock type database is necessary in order to systematically and efficiently collect, edit, verify and store all the data. For this purpose, the practical aspect of data handling techniques, such as how to design the blast hole editor and how to set up the data structure for both blast hole and rock type information, is presented.

A program package called Blast Hole Kriging (BHK) system was implemented for the personal computer to perform the functions of blast hole handling, geology information handling, blast hole kriging, interactive graphics ore–waste delineations in daily production. The BHK system can be used to increase the ore tonnages and achieve a better comparison between mine estimates and the actual mill grade through improved ore-waste selection procedure to reduce the chance of mis-classification errors of ore and waste material at an operating mine.

6.2 Recommendations for Future Research

The optimum pit limit algorithm developed in this dissertation has made a great progress in obtaining the optimal pit limit design. It is the fastest optimum algorithm so far. The algorithm is able to provide a true optimal solution within the practically acceptable time for relatively large mine models. However, for some extremely large mine models with more than 200 columns x 200 rows x 100 benches, up to several days may be needed even on the SUN station computers to come up with the optimal solution. The recommendation for future research regarding pit limit design is to continue the search for a faster and simpler optimum pit design algorithm. It seems that such an algorithm can be only obtained from the adoption of graph theory.

The mine production scheduling is still the most important topic in mine planning. As discussed in chapter 4, it is not practical to obtain a truly optimal solution to mine production scheduling problem using today's operations research techniques and computing facilities. Moreover, it is very difficult to incorporate some practical constraints, such as minimum working space, minimum amount of
ore exposed at the end of each period, and minimum shovel moves, into the optimal scheduling model even in the future.

The approach of scheduling among pushback approaches is intuitively more likely to yield the maximum NPV since the phase order is chosen to be the next best, irrespective of changes in product prices and production costs. A multi-period scheduling problem among the pushbacks could even be formulated into a solvable 0-1 program problem. Although the decision variable of mine sequencing among the pushbacks can be defined as a whole bench, a better results can be obtained by dividing a whole bench into efficient working slices, defining the precedence relationship among these working slices, then carrying out pit sequencing among these working slices.

Just like any other computer system, the BHK system can be further enhanced in the future. The BHK system can not handle the estimation problem of the split benches at the moment. That is, one-half of a bench is ore and the other half bench is waste, due to sharp geologic contacts. Currently, only ordinary kriging method is available. For erratic deposits with high variability, the technique of indicator kriging may be a better choice since this method can reduce the effect of high grade outliers in addition to the capability to predict the probability of block grades.
APPENDIX A

Pit Limits Corresponding to Different Penalty Parameters
for a 2-D Cross Section Example
Figure A.1  Modified Block Values and the Associated Pit Limits

\[(\mu_o, \mu_w) = (0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6)\]
<table>
<thead>
<tr>
<th>$\mu_o = 1$</th>
<th>$-1$</th>
<th>$-1$</th>
<th>$-1$</th>
<th>$-1$</th>
<th>$-1$</th>
<th>$-1$</th>
<th>$-1$</th>
<th>$-1$</th>
<th>$-1$</th>
<th>$-1$</th>
<th>$-1$</th>
</tr>
</thead>
</table>
| $\mu_w = 0$ | $-1$ | $-1$ | $2$ | $5$ | $2$ | $-1$ | $-1$ | $4$ | $5$ | $3$ | $5$ | $4$ | $-1$
|-------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| $\mu_o = 1$ | $-2$ | $-2$ | $-2$ | $-2$ | $-2$ | $-2$ | $-2$ | $-2$ | $-2$ | $-2$ | $-2$ | $-2$
| $\mu_w = 1$ | $-2$ | $-2$ | $2$ | $5$ | $2$ | $-2$ | $-2$ | $4$ | $5$ | $3$ | $5$ | $4$ | $2$
|-------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| $\mu_o = 1$ | $-3$ | $-3$ | $-3$ | $-3$ | $-3$ | $-3$ | $-3$ | $-3$ | $-3$ | $-3$ | $-3$ | $-3$
| $\mu_w = 2$ | $-3$ | $-3$ | $2$ | $5$ | $2$ | $-3$ | $-3$ | $4$ | $5$ | $3$ | $5$ | $4$ | $3$
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_o = 1$</td>
<td>$-4$</td>
<td>$-4$</td>
<td>$-4$</td>
<td>$-4$</td>
<td>$-4$</td>
<td>$-4$</td>
<td>$-4$</td>
<td>$-4$</td>
<td>$-4$</td>
<td>$-4$</td>
<td>$-4$</td>
<td>$-4$</td>
</tr>
</tbody>
</table>
| $\mu_w = 3$ | $-4$ | $-4$ | $2$ | $5$ | $2$ | $-4$ | $-4$ | $4$ | $5$ | $3$ | $5$ | $4$ | $4$
|-------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| $\mu_o = 1$ | $-5$ | $-5$ | $-5$ | $-5$ | $-5$ | $-5$ | $-5$ | $-5$ | $-5$ | $-5$ | $-5$ | $-5$ | $-5$
| $\mu_w = 4$ | $-5$ | $-5$ | $2$ | $5$ | $2$ | $-5$ | $-5$ | $4$ | $5$ | $3$ | $5$ | $4$ | $5$
|-------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| $\mu_o = 1$ | $-6$ | $-6$ | $-6$ | $-6$ | $-6$ | $-6$ | $-6$ | $-6$ | $-6$ | $-6$ | $-6$ | $-6$ | $-6$
| $\mu_w = 5$ | $-6$ | $-6$ | $2$ | $5$ | $2$ | $-6$ | $-6$ | $4$ | $5$ | $3$ | $5$ | $4$ | $6$
|-------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| $\mu_o = 1$ | $-7$ | $-7$ | $-7$ | $-7$ | $-7$ | $-7$ | $-7$ | $-7$ | $-7$ | $-7$ | $-7$ | $-7$ | $-7$
| $\mu_w = 6$ | $-7$ | $-7$ | $2$ | $5$ | $2$ | $-7$ | $-7$ | $4$ | $5$ | $3$ | $5$ | $4$ | $7$
|-------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|

Figure A.2  Modified Block Values and the Associated Pit Limits

$(\mu_o, \mu_w) = (1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6)$
Figure A.3  Modified Block Values and the Associated Pit Limits

\((\mu_o, \mu_w) = (2, 0), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (2, 6)\)
Figure A.4  Modified Block Values and the Associated Pit Limits

\[(\mu_o, \mu_w) = (3, 0), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6)\]
<table>
<thead>
<tr>
<th>(\mu_o=4)</th>
<th>-1 -1 -1 -1 -1 -1 -1 -1 -1</th>
<th>(\mu_w=0)</th>
<th>-1 -1 -1 -1 -1 -1 -1 -1 -1 -1</th>
<th>(a)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-2 -2 -1 30 -2</td>
<td></td>
<td>-2 -2 -1 23 -3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\mu_o=4)</th>
<th>-2 -2 -2 -2 -2 -2 -2 -2 -2</th>
<th>(\mu_w=1)</th>
<th>-2 -2 -2 -2 -2 -2 -2 -2 -2</th>
<th>(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-3 -3 -2 30 -2</td>
<td></td>
<td>-3 -3 -2 23 -3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\mu_o=4)</th>
<th>-3 -3 -3 -3 -3 -3 -3 -3</th>
<th>(\mu_w=2)</th>
<th>-3 -3 -3 -3 -3 -3 -3 -3</th>
<th>(c)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-4 -4 -3 30 -1</td>
<td></td>
<td>-4 -4 -3 23 -1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\mu_o=4)</th>
<th>-4 -4 -4 -4 -4 -4 -4 -4</th>
<th>(\mu_w=3)</th>
<th>-4 -4 -1 -4 -4 -4 -4 -4</th>
<th>(d)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-5 -5 -1 30 -4</td>
<td></td>
<td>-5 -5 -1 23 -4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\mu_o=4)</th>
<th>-5 -5 -5 -5 -5 -5 -5 -5 -5</th>
<th>(\mu_w=4)</th>
<th>-5 -5 -5 -5 -5 -5 -5 -5 -5</th>
<th>(e)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-6 -6 -1 30 -2</td>
<td></td>
<td>-6 -6 -1 23 -2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\mu_o=4)</th>
<th>-6 -6 -6 -6 -6 -6 -6 -6 -6</th>
<th>(\mu_w=5)</th>
<th>-6 -6 -6 -6 -6 -6 -6 -6 -6</th>
<th>(f)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-7 -7 -7 30 -7</td>
<td></td>
<td>-7 -7 -7 23 -7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\mu_o=4)</th>
<th>-7 -7 -7 -7 -7 -7 -7 -7 -7</th>
<th>(\mu_w=6)</th>
<th>-7 -7 -7 -7 -7 -7 -7 -7 -7</th>
<th>(g)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-8 -8 -8 30 -8</td>
<td></td>
<td>-8 -8 -8 23 -8</td>
<td></td>
</tr>
</tbody>
</table>

Figure A.5  Modified Block Values and the Associated Pit Limits  
\((\mu_o, \mu_w) = (4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (4, 6)\)
<table>
<thead>
<tr>
<th>$\mu_0$</th>
<th>$\mu_w$</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
<td>-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1</td>
</tr>
<tr>
<td>1</td>
<td>-1 -1 -2 1 -2 -1 -1 -1 0 1 -1 1 0 -1</td>
<td></td>
</tr>
<tr>
<td>(a)</td>
<td>-2 -3 -2 2 2 -2 -2 -2 2 2 -2 -2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>-2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2</td>
</tr>
<tr>
<td>2</td>
<td>-2 -2 -2 1 -2 -2 -2 -2 0 1 -1 1 0 -2</td>
<td></td>
</tr>
<tr>
<td>(b)</td>
<td>-3 -3 -3 -3 0 -2 -2 -3 -3 3 2 -3 -3 -3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>-3 -3 -3 -3 -3 -3 -3 -3 -3 -3 -3 -3</td>
</tr>
<tr>
<td>3</td>
<td>-3 -3 -2 1 -2 -3 -3 0 1 -1 1 0 -3</td>
<td></td>
</tr>
<tr>
<td>(c)</td>
<td>-4 -4 -4 2 2 -4 -4 -4 0 1 -1 1 0 -4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>-5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5</td>
</tr>
<tr>
<td>3</td>
<td>-5 -5 -2 1 -2 -5 -5 0 1 -1 1 0 -5</td>
<td></td>
</tr>
<tr>
<td>(e)</td>
<td>-6 -6 -6 2 2 -6 -6 -6 2 2 -6 -6 -6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>-5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5</td>
</tr>
<tr>
<td>5</td>
<td>-5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5</td>
<td></td>
</tr>
<tr>
<td>(d)</td>
<td>-6 -6 -6 2 2 -6 -6 -6 2 2 -6 -6 -6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>-6 -6 -6 -6 -6 -6 -6 -6 -6 -6 -6 -6</td>
</tr>
<tr>
<td>6</td>
<td>-6 -6 -2 1 -2 -6 -6 0 1 -1 1 0 -6</td>
<td></td>
</tr>
<tr>
<td>(f)</td>
<td>-7 -3 -7 2 2 -7 -7 -7 2 2 -7 -7 -7</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>-7 -7 -7 -7 -7 -7 -7 -7 -7 -7 -7 -7</td>
</tr>
<tr>
<td>7</td>
<td>-7 -7 -7 -7 -7 -7 -7 -7 0 1 -1 1 0 -7</td>
<td></td>
</tr>
<tr>
<td>(g)</td>
<td>-8 -3 -8 2 2 -8 -8 -8 2 2 -8 -8 -8</td>
<td></td>
</tr>
</tbody>
</table>
Figure A.7  Modified Block Values and the Associated 
Pit Limits  

\[(\mu_0, \mu_w) = (6, 0), (6, 1), (6, 2), (6, 3), (6, 4), (6, 5), (6, 6)\]

<table>
<thead>
<tr>
<th>(\mu_0 = 6)</th>
<th>(\mu_w = 0)</th>
<th>(\mu_0 = 6)</th>
<th>(\mu_w = 1)</th>
<th>(\mu_0 = 6)</th>
<th>(\mu_w = 2)</th>
<th>(\mu_0 = 6)</th>
<th>(\mu_w = 3)</th>
<th>(\mu_0 = 6)</th>
<th>(\mu_w = 4)</th>
<th>(\mu_0 = 6)</th>
<th>(\mu_w = 5)</th>
<th>(\mu_0 = 6)</th>
<th>(\mu_w = 6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1</td>
<td>-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1</td>
<td>-2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2</td>
<td>-2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2</td>
<td>-3 -3 -3 -3 -3 -3 -3 -3 -3 -3 -3 -3 -3 -3 -3</td>
<td>-4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4</td>
<td>-5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5</td>
<td>-6 -6 -6 -6 -6 -6 -6 -6 -6 -6 -6 -6 -6 -6 -6</td>
<td>-7 -7 -7 -7 -7 -7 -7 -7 -7 -7 -7 -7 -7 -7 -7</td>
<td>-8 -8 -8 -8 -8 -8 -8 -8 -8 -8 -8 -8 -8 -8 -8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
![Image of modified block values and associated pit limits](image)

| $\mu_o=7$ | $-1$ | $-1$ | $-1$ | $-1$ | $-1$ | $-1$ | $-1$ | $-1$ | $-1$ | $-1$ | $-1$ | $-1$ |
| $\mu_w=0$ | $-1$ | $-1$ | $-4$ | $-1$ | $-4$ | $-1$ | $-1$ | $-2$ | $-1$ | $3$ | $-1$ | $-2$ |
|           | $-1$ | $-1$ | $-2$ | $-5$ | $-2$ | $-1$ | $-1$ | $-2$ | $-2$ | $0$ | $-2$ | $-2$ |
| (a)       | $-2$ | $-5$ | $-2$ | $27$ | $-2$ | $-2$ | $-4$ | $-2$ | $20$ | $-3$ | $-4$ | $-2$ |

| $\mu_o=7$ | $-2$ | $-2$ | $-2$ | $-2$ | $-2$ | $-2$ | $-2$ | $-2$ | $0$ | $-2$ | $-2$ | $-2$ |
| $\mu_w=1$ | $-2$ | $-2$ | $-4$ | $-1$ | $-4$ | $-2$ | $-2$ | $-2$ | $-1$ | $3$ | $-1$ | $-2$ |
|           | $-2$ | $-2$ | $-2$ | $-5$ | $-2$ | $-2$ | $-2$ | $-3$ | $-3$ | $0$ | $-3$ | $-3$ |
| (b)       | $-3$ | $-5$ | $-3$ | $27$ | $-3$ | $-3$ | $-4$ | $-3$ | $20$ | $-4$ | $-4$ | $-3$ |

| $\mu_o=7$ | $-3$ | $-3$ | $-3$ | $-3$ | $-3$ | $-3$ | $-3$ | $-3$ | $-3$ | $-3$ | $-3$ | $-3$ |
| $\mu_w=2$ | $-3$ | $-3$ | $-4$ | $-1$ | $-4$ | $-3$ | $-3$ | $-2$ | $-1$ | $3$ | $-1$ | $-2$ |
|           | $-3$ | $-3$ | $-2$ | $-5$ | $-2$ | $-3$ | $-3$ | $-4$ | $-4$ | $0$ | $-4$ | $-4$ |
| (c)       | $-4$ | $-5$ | $-4$ | $27$ | $-4$ | $-4$ | $-4$ | $-4$ | $20$ | $-5$ | $-4$ | $-5$ |

| $\mu_o=7$ | $-4$ | $-4$ | $-4$ | $-4$ | $-4$ | $-4$ | $-4$ | $-4$ | $-4$ | $-4$ | $-4$ | $-4$ |
| $\mu_w=3$ | $-4$ | $-4$ | $-4$ | $-1$ | $-4$ | $-4$ | $-4$ | $-4$ | $-2$ | $-1$ | $3$ | $-1$ |
|           | $-4$ | $-4$ | $-2$ | $-5$ | $-2$ | $-4$ | $-4$ | $-5$ | $-5$ | $0$ | $-5$ | $-5$ |
| (d)       | $-5$ | $-5$ | $-5$ | $27$ | $-5$ | $-5$ | $-4$ | $-5$ | $-5$ | $20$ | $-6$ | $-4$ |

| $\mu_o=7$ | $-5$ | $-5$ | $-5$ | $-5$ | $-5$ | $-5$ | $-5$ | $-5$ | $-5$ | $-5$ | $-5$ | $-5$ |
| $\mu_w=4$ | $-5$ | $-5$ | $-4$ | $-1$ | $-4$ | $-5$ | $-5$ | $-5$ | $-2$ | $-1$ | $3$ | $-1$ |
|           | $-5$ | $-5$ | $-2$ | $-5$ | $-2$ | $-5$ | $-5$ | $-6$ | $-6$ | $0$ | $-6$ | $-6$ |
| (e)       | $-6$ | $-5$ | $-6$ | $27$ | $-6$ | $-6$ | $-4$ | $-6$ | $-6$ | $20$ | $-7$ | $-4$ |

| $\mu_o=7$ | $-6$ | $-6$ | $-6$ | $-6$ | $-6$ | $-6$ | $-6$ | $-6$ | $-6$ | $-6$ | $-6$ | $-6$ |
| $\mu_w=5$ | $-6$ | $-6$ | $-4$ | $-1$ | $-4$ | $-6$ | $-6$ | $-6$ | $-2$ | $-1$ | $3$ | $-1$ |
|           | $-6$ | $-6$ | $-2$ | $-5$ | $-2$ | $-6$ | $-6$ | $-7$ | $-7$ | $0$ | $-7$ | $-7$ |
| (f)       | $-7$ | $-5$ | $-7$ | $27$ | $-7$ | $-7$ | $-4$ | $-7$ | $-7$ | $20$ | $-8$ | $-4$ |

| $\mu_o=7$ | $-7$ | $-7$ | $-7$ | $-7$ | $-7$ | $-7$ | $-7$ | $-7$ | $-7$ | $-7$ | $-7$ | $-7$ |
| $\mu_w=6$ | $-7$ | $-7$ | $-4$ | $-1$ | $-4$ | $-7$ | $-7$ | $-2$ | $-1$ | $3$ | $-1$ | $-2$ |
|           | $-7$ | $-7$ | $-2$ | $-5$ | $-2$ | $-7$ | $-7$ | $-8$ | $-8$ | $0$ | $-8$ | $-8$ |
| (g)       | $-8$ | $-5$ | $-8$ | $27$ | $-8$ | $-8$ | $-4$ | $-8$ | $-8$ | $20$ | $-9$ | $-4$ |

*Figure A.8  Modified Block Values and the Associated Pit Limits*

$$(\mu_o, \mu_w) = (7, 0), (7, 1), (7, 2), (7, 3), (7, 4), (7, 5), (7, 6)$$
APPENDIX B

Pushback Contour Maps of a Gold Deposit
Figure B.1  Initial Contour (1 inch = 600 feet)
Figure B.2  Contour after Pushback 1 (1 inch = 600 feet)
Figure B.3  Contour after Pushback 2 (1 inch = 600 feet)
Figure B.4  Contour after Pushback 3 (1 inch = 600 feet)
SELECTED BIBLIOGRAPHY


73. Myers, D. E., 1980, “Determination of \( \prod_{f_1, \ldots, f_n} \)”, Lecture Notes 3, the Department of Mathematics, the University of Arizona, Tucson, Arizona.


