A heuristic algorithm to optimise stope boundaries

Majid Ataee-Pour

University of Wollongong

Recommended Citation
NOTE

This online version of the thesis may have different page formatting and pagination from the paper copy held in the University of Wollongong Library.

UNIVERSITY OF WOLLONGONG

COPYRIGHT WARNING

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site. You are reminded of the following:

Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material. Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.
A HEURISTIC ALGORITHM TO OPTIMISE STOPE BOUNDARIES

A thesis submitted in fulfilment of the requirements for the award of the degree

DOCTOR OF PHILOSOPHY

from

UNIVERSITY OF WOLLONGONG

by

MAJID ATAAEE-POUR

B.Eng., M.Eng. (Hons.) Mining Engineering

Faculty of Engineering

March 2000
IN THE NAME OF GOD

This thesis is dedicated to

the soul of

my dear mother;

Sedigheh Alipour-Hejranian

for her love during her life.
DECLARATION

This thesis has been purposefully and originally undertaken by the author for a degree of Doctor of Philosophy in the department of Civil, Mining and Environmental Engineering at the University of Wollongong. This thesis contains no material submitted, in whole or in part, for a degree at this or any other institution. The following publications have been based on this thesis:


MAJID ATAEPOUR
ACKNOWLEDGEMENT

I am very grateful for the helpful contributions made by a number of people during the course of this study. I wish to express my sincere thanks to:

Associate Professor E Y Baafi, Faculty of Engineering, University of Wollongong, for his supervision of the thesis and his advice, encouragement, guidance and critical review during the entire course of this study;

Dr. P Standish, Elura mine, Cobar, NSW, for allowing the visit to the Elura mine at Cobar;

Mrs. K Draisma, Learning Development Centre, University of Wollongong, for her guidance in thesis writing and Ms. J Shaw for her proofreading of the thesis; and

All staff members in the Faculty of Engineering, University of Wollongong, for providing computer facilities, support and help during this work.

I wish to, gratefully, acknowledge the Ministry of Culture and Higher Education of the Islamic Republic of Iran, for the financial support and sponsorship through the period of this project.

Last, but not the least, I would like to express my sincere appreciation to my wife Zahra, my daughter Maedeh and all members of my family in Iran, for their patience and acceptance of hardship during the entire course of my study and for their understanding, which has been much more than I could explain.
ABSTRACT

Determination of the optimum mine layout is one of the important tasks in mine planning. In the case of open pit mines, a large number of algorithms using a range of techniques have been developed to generate the true optimum solution. Several commercial computer packages are available to assist mining engineers design open pits. In contrast, only a few algorithms, using limited techniques, have been developed to optimise the stope geometry in underground operations. Most of which fail to provide optimum 3D solutions.

A heuristic algorithm, termed the "Maximum Value Neighbourhood" (MVN) was developed in this thesis to optimise stope boundaries. The MVN algorithm benefits from its simplicity in both concept and implementation. It provides a 3D analysis and can be applied to any underground mining method, although it does not guarantee the true "optimum" stope layout. The MVN algorithm uses a 3D fixed economic block model to locate the best neighbourhood of a block, which guarantees the maximum net value. Neighbourhoods are restricted by the mine geometry constraints. The neighbourhood concept is based on the number of mining blocks equivalent to the minimum stope size. Since a variety of neighbourhoods are available for each block, the one that provides the maximum net value (the maximum value neighbourhood, MVN) is located for inclusion in the final stope.

In order to test the algorithm, the 3D version of the MVN algorithm was implemented on small sized examples, using the Visual Basic for Applications (VBA) modules supported by Microsoft Excel. The framework of the Excel worksheets was suitable to store block data and display the optimised stope.

A Fortran 90 program, the "Stope Limit Optimiser" (SLO), was developed to implement the 3D MVN algorithm on actual mine data. The SLO optimiser integrates
the Fortran 90 code of the algorithm with the Winteracter user interface features, to provide dialog boxes and user friendly menus. The SLO provides a Windows based interactive environment to define and edit the project parameters including the block model parameters, the stope geometry constraints and the economic factors.
TABLE OF CONTENTS

DECLARATION iii
ACKNOWLEDGEMENTS iv
ABSTRACT v
TABLE OF CONTENTS vii
LIST OF FIGURES xiii
LIST OF TABLES xix
LIST OF SYMBOLS AND ABBREVIATIONS xx

CHAPTER ONE: INTRODUCTION

1.1 GENERAL INTRODUCTION ................................................................. 1-1
1.2 STATEMENT OF THE PROBLEM ......................................................... 1-3
   1.2.1 Status of Pit Limit Optimisation .............................................. 1-4
   1.2.2 Status of Stope Geometry Optimisation ...................................... 1-6
1.3 SCOPE OF THE THESIS ................................................................... 1-9
1.4 OUTLINE OF THE THESIS .................................................................. 1-11

CHAPTER TWO: ULTIMATE MINE DESIGN METHODS

2.1 INTRODUCTION ............................................................................... 2-1
2.2 MINE GEOMETRY OPTIMISATION ....................................................... 2-1
   2.2.1 Optimisation Criteria ................................................................. 2-2
   2.2.2 Problem Formulation ................................................................. 2-3
   2.2.3 Necessity of Optimisation Algorithms ....................................... 2-4
2.3 ULTIMATE PIT OPTIMISATION ALGORITHMS ..................................... 2-5
   2.3.1 Moving Cone Algorithms ........................................................... 2-7
   2.3.2 Dynamic Programming Algorithms .......................................... 2-11
   2.3.3 Graph Theory Algorithms ......................................................... 2-16
CHAPTER THREE: MAXIMUM VALUE NEIGHBOURHOOD (MVN) ALGORITHM

3.1 INTRODUCTION .................................................................................. 3-1

3.2 BASIC SPECIFICATIONS OF STOPE GEOMETRY .............................. 3-1

3.3 FORMULATION OF THE PROBLEM .................................................. 3-4

3.4 STOPE GEOMETRY CONSTRAINTS ................................................. 3-6

3.5 THE NEIGHBOURHOOD CONCEPT ................................................. 3-8
  3.5.1 The Neighbourhood Set .......................................................... 3-10
  3.5.2 The Optimum Neighbourhood ............................................... 3-14
  3.5.3 Infeasible Neighbourhoods .................................................. 3-18

3.6 THE MVN ALGORITHM .................................................................... 3-18
  3.6.1 The Optimisation Procedure ................................................. 3-19
  3.6.2 Numerical Examples ............................................................. 3-25
CHAPTER FOUR: THE 2D AND 3D MAXIMUM VALUE NEIGHBOURHOOD ALGORITHMS

4.1 INTRODUCTION ................................................................. 4-1
4.2 TWO- AND THREE-DIMENSIONAL NEIGHBOURHOODS ...... 4-1
  4.2.1 The Order of Neighbourhood......................................... 4-6
  4.2.2 The Optimum Neighbourhood ....................................... 4-8
  4.2.3 Infeasible Neighbourhoods ......................................... 4-12
4.3 A 2D NUMERICAL EXAMPLE ............................................. 4-15
4.4 IMPLEMENTATION OF 3D MVN ALGORITHM USING VBA CODE .. 4-23
4.5 SUMMARY ........................................................................ 4-40

CHAPTER FIVE: IMPLEMENTATION OF THE MAXIMUM VALUE NEIGHBOURHOOD ALGORITHM

5.1 INTRODUCTION .................................................................. 5-1
5.2 THE SLO PROGRAM COMPONENTS ....................................... 5-1
5.3 THE WINTERACTER RESOURCE SCRIPT DESCRIPTION .......... 5-2
  5.3.1 The SLO Menus ............................................................ 5-3
  5.3.2 The SLO Dialogs ........................................................... 5-4
5.4 THE FORTRAN 90 SOURCE CODE DESCRIPTION ............... 5-5
  5.4.1 The SLO Input/Output Files ......................................... 5-5
  5.4.2 The SLO Main Program ................................................. 5-6
  5.4.3 The SLO Sub-programs ................................................. 5-8
5.5 THE SLO GENERAL PROCEDURE ........................................ 5-9
5.6 THE SLO PROJECTS .......................................................... 5-12
5.7 PROJECT DEFINITION FILES ............................................. 5-14
  5.7.1 Block Model Parameters ............................................... 5-15
CHAPTER SIX: PROGRAMMING OF THE MAXIMUM VALUE NEIGHBOURHOOD ALGORITHM IN FORTRAN 90

6.1 INTRODUCTION .................................................. 6-1
6.2 GENERAL VIEW ON THE OPTIMISATION STAGE .......... 6-1
6.3 DEFINING THE OPTIMISATION DATA .......................... 6-5
6.4 LOCATING THE MAXIMUM VALUE NEIGHBOURHOOD ...... 6-8
   6.4.1 Neighbourhood Identification .......................... 6-10
   6.4.2 Neighbourhood Determination .......................... 6-14
   6.4.3 Coding the Procedure ................................ 6-19
   6.4.4 Feasibility and Valuation of Neighbourhoods ...... 6-20
6.5 UPDATING THE STOPE .......................................... 6-23
6.6 SUMMARY ....................................................... 6-26

CHAPTER SEVEN: OUTPUT RESULTS OF THE STOPE LIMIT OPTIMISER

7.1 INTRODUCTION .................................................. 7-1
7.2 BLOCK FLAG DATA ............................................. 7-2
7.3 DISPLAY OF RESULTS .......................................... 7-2
   7.3.1 Designing the Borders of the Plan / Section ......... 7-7
   7.3.2 Designing the Body of the Plan / Section ............ 7-8
7.4 REPORTS ........................................................ 7-9
   7.4.1 Intermediate Results .................................. 7-9
   7.4.2 Neighbourhood Results ................................. 7-13
CHAPTER EIGHT: VALIDATION OF THE STOPE LIMIT OPTIMISER

8.1 INTRODUCTION ................................................................................. 8-1
8.2 OPTIMISER VALIDATION ................................................................. 8-1
8.3 THE VALIDATION PROCEDURE ....................................................... 8-2
  8.3.1 Manual Validation ..................................................................... 8-2
  8.3.2 An Example with a Special Pattern ............................................ 8-4
  8.3.3 Validation with Excel VBA Modules .......................................... 8-6
  8.3.4 Program Debugging ................................................................. 8-6

CHAPTER NINE: SUMMARY AND CONCLUSIONS

9.1 SUMMARY ...................................................................................... 9-1
9.2 CONCLUSIONS .............................................................................. 9-4
9.3 RECOMMENDATION AND FUTURE WORKS ................................. 9-6

REFERENCES

APPENDIX A: COMPUTATIONS OF BLOCK ECONOMIC VALUES

A.1 Block Valuation ............................................................................. A-1
A.2 Equivalent Grade ......................................................................... A-5

APPENDIX B: SLO DIALOGS AND SUBPROGRAMS

APPENDIX C: THE STOPE LIMIT OPTIMISER USER’S GUIDE

C.1 INTRODUCTION ............................................................................. C-1
C.2 INSTALLATION ................................................................................ C-1
C.3 STARTING SLO ................................................................................ C-2
C.4 PROJECT MANIPULATION
C.4.1 Project | New option
C.4.2 Project | Open option
C.4.3 Project | Close option
C.4.4 Project | Save option
C.4.5 Project | Save as... option
C.4.6 Project | Exit option

C.5 INPUT EDITION
C.5.1 Edit | Block Model option
C.5.2 Edit | Sub-regions option
C.5.3 Edit | Stope Constraints option
C.5.4 Edit | Economic Factors option
C.5.5 Edit | Main Data option

C.6 PRE OPTIMISATION
C.6.1 Preoptimisation | Data Preparation Option
C.6.2 Preoptimisation | Select Region Option
C.6.3 Preoptimisation | Import Block Data Option

C.7 OPTIMISATION
C.7.1 Run | Optimise Option

C.8 POST OPTIMISATION
C.8.1 Results | Export Flag Data Option
C.8.2 Results | Summary Report Option
C.8.3 Results | Intermediate Results Option
C.8.4 Results | Neighbourhood Results Option
C.8.5 Results | Test Results Option
C.8.6 Results | Plot Plans/Sections Option
C.8.7 Results | View Plots Option

C.9 HELP
C.9.1 Help | About SLO Option
LIST OF FIGURES

Figure | Page
-------|-------
1.3 A Schematic View of the Proposed Study | 1-10
2.1 Variation of the Total Value Compared to the Unit Value of a Specific Mine over the Mine Production (Wilke et al., 1984) | 2-3
2.2 The Moving Cone Technique | 2-9
2.3 Failure of the Moving Cone to Recognise the Maximum Value Pit | 2-10
2.4 Failure of the Moving Cone Algorithm to Recognise a Positive Value Pit (Wright, 1990) | 2-10
2.5 A Dynamic Programming Technique Applied to Pit Limit Optimisation | 2-13
2.6 Graph Theory Applied to Pit Limit Optimisation | 2-18
2.7 The Network Flow Technique Applied to Pit Limit Optimisation | 2-20
2.8 Comparison between Open Pit and Block-Caving (Riddle, 1977) | 2-26
2.9 Adjacent Blocks in Open Pit and Block-Caving | 2-27
2.10 Conditional Simulation of Lenses (Deraisme and de Fouquet, 1984) | 2-30
2.11 Geometrical Constraints for the Three Mining Methods at Ben Lomond Mine (Deraisme et al., 1984) | 2-30
2.12 Compared Outlines for the Cut-and-Fill Method on a Section (Deraisme et al., 1984) | 2-31
2.13 Successive Removal of Sub-Volumes | 2-33
2.14 An Example of Quadtree Division | 2-34
2.15 Octree Division | 2-34
2.16 An Example of Vein Surrounded by an Initial Mineable Volume, Resulting from the Object Manipulator (Cheimanoff et al., 1989) | 2-35
2.17 Successive Sub-Volume Evaluation Steps Performed by Shape Generator, on a Cut-and-Fill Method (Cheimanoff et al., 1989) | 2-37
2.18 Joint Consideration of Sub-Volumes | 2-37
2.19 The "Inner" and the "Outer" Envelopes for a Single Block | 2-39
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>The Mining Industry Share of the Australian Export Dollar 1986-1998</td>
<td>1-2</td>
</tr>
<tr>
<td></td>
<td>(Source: Australian Bureau of Statistics, 1998)</td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>The Significance of Metallic Minerals in the Australian Mineral Products</td>
<td>1-2</td>
</tr>
<tr>
<td></td>
<td>(Source: Australian Bureau of Statistics, 1998)</td>
<td></td>
</tr>
<tr>
<td>1.3</td>
<td>A Schematic View of the Proposed Study</td>
<td>1-10</td>
</tr>
<tr>
<td>2.1</td>
<td>Variation of the Total Value Compared to the Unit Value of a Specific Mine</td>
<td>2-3</td>
</tr>
<tr>
<td></td>
<td>over the Mine Production (Wilke et al., 1984)</td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td>The Moving Cone Technique</td>
<td>2-9</td>
</tr>
<tr>
<td>2.3</td>
<td>Failure of the Moving Cone Technique to Recognise the Maximum Value Pit</td>
<td>2-10</td>
</tr>
<tr>
<td>2.4</td>
<td>Failure of the Moving Cone Algorithm to Recognise a Positive Value Pit</td>
<td>2-10</td>
</tr>
<tr>
<td></td>
<td>(Wright, 1990)</td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td>A Dynamic Programming Technique Applied to Pit Limit Optimisation</td>
<td>2-13</td>
</tr>
<tr>
<td>2.6</td>
<td>Graph Theory Applied to Pit Limit Optimisation</td>
<td>2-18</td>
</tr>
<tr>
<td>2.7</td>
<td>The Network Flow Technique Applied to Pit Limit Optimisation</td>
<td>2-20</td>
</tr>
<tr>
<td>2.8</td>
<td>Comparison between Open Pit and Block-Caving (Riddle, 1977)</td>
<td>2-26</td>
</tr>
<tr>
<td>2.9</td>
<td>Adjacent Blocks in Open Pit and Block-Caving</td>
<td>2-27</td>
</tr>
<tr>
<td>2.10</td>
<td>Conditional Simulation of Lenses (Deraisme and de Fouquet, 1984)</td>
<td>2-30</td>
</tr>
<tr>
<td>2.11</td>
<td>Geometrical Constraints for the Three Mining Methods at Ben Lomond Mine</td>
<td>2-30</td>
</tr>
<tr>
<td></td>
<td>(Deraisme et al., 1984)</td>
<td></td>
</tr>
<tr>
<td>2.12</td>
<td>Compared Outlines for the Cut-and-Fill Method on a Section</td>
<td>2-31</td>
</tr>
<tr>
<td></td>
<td>(Deraisme et al., 1984)</td>
<td></td>
</tr>
<tr>
<td>2.13</td>
<td>Successive Removal of Sub-Volumes</td>
<td>2-33</td>
</tr>
<tr>
<td>2.14</td>
<td>An Example of Quadtree Division</td>
<td>2-34</td>
</tr>
<tr>
<td>2.15</td>
<td>Octree Division</td>
<td>2-34</td>
</tr>
<tr>
<td>2.16</td>
<td>An Example of Vein Surrounded by an Initial Mineable Volume, Resulting from</td>
<td>2-35</td>
</tr>
<tr>
<td></td>
<td>the Object Manipulator (Cheimanoff et al., 1989)</td>
<td></td>
</tr>
<tr>
<td>2.17</td>
<td>Successive Sub-Volume Evaluation Steps Performed by Shape Generator, on a</td>
<td>2-37</td>
</tr>
<tr>
<td></td>
<td>Cut-and-Fill Method (Cheimanoff et al., 1989)</td>
<td></td>
</tr>
<tr>
<td>2.18</td>
<td>Joint Consideration of Sub-Volumes</td>
<td>2-37</td>
</tr>
<tr>
<td>2.19</td>
<td>The &quot;Inner&quot; and the &quot;Outer&quot; Envelopes for a Single Block</td>
<td>2-39</td>
</tr>
</tbody>
</table>
2.20 Limitations of the Moving Cone and Floating Stope Methods 2-41
2.21 Typical Piecewise Linear Approximation Function 2-43
2.22 Cumulative Block Value Function Applied to a Row of Blocks 2-44
2.23 SOS2 Solution Example 2-47
2.24 The Ore-Body Surrounded by a large box 2-49
2.25 The Ore-Body Surrounded by the Physical Block Model 2-49
2.26 An Example of a Geological Unit Coded into a Block Model (Noble, 1992) 2-50
2.27 Modified Block Models Compared to a Fixed Block Model (Badiozamani and Roghani, 1988) 2-51
3.1 Geometric Constraints, Open Pit versus Underground Mines 3-2
3.2 Four Examples of Possible Choices (stopes) for Mining the Block, \( B_{ij} \) 3-3
3.3 Block Dependence, Open Pit versus Underground Mining 3-4
3.4 The Spatial Address of an Economic Block 3-5
3.5 Stope Constraints: ( a ) 1D, ( b ) 2D and ( c ) 3D Problems 3-7
3.6 Minimum Stope Size versus the Neighbourhood (NB) 3-9
3.7 Possible Neighbourhoods of the Block, \( B_4 \), for \( NB \) Orders of 2, 3 and 4 3-11
3.8 A Typical Row of Blocks with “\( n \)” Blocks 3-11
3.9 Locating the Maximum Value Neighbourhood 3-16
3.10 Infeasible Neighbourhoods 3-19
3.11 Generalised Flow-Chart for the Optimisation Procedure 3-22
3.12 A Simplified Pseudo Code for the \( MVN \) Algorithm 3-24
3.13 A Row of Blocks with 10 Columns taken from an Economic Block Model 3-25
3.14 The Optimised Example Manually using the \( MVN \) Algorithm 3-31
3.15 Optimising a Stope Section using the \( MVN \) Algorithm with a 1D (height) Constraint 3-32
4.1 Examples of 2D Possible Neighbourhoods for a Specific Block, \( B_{33} \) 4-3
4.2 A Typical Block Model illustrating 3D Stope Size Constraints 4-5
4.3 Examples of 3D Neighbourhoods 4-5
4.4 Possible 2D Neighbourhoods of the Block, \( B_{44} \), with \( [Onb]^{(2)} = (3, 3) \) 4-7
4.5 Possible 3D Neighbourhoods of the Block, \( B_{y_k} \), with \( [Onb]^{(3)} = (2, 2, 2) \) 4-8
5.10 Example of the Model Parameters File, ".mpr"
5.11 The Dialog Box for Defining the Order of Neighbourhood (entire model)
5.12 The Dialog Box to Select a Sub-Region for Defining it's Order of Neighbourhood
5.13 The Dialog Box for Defining the Order of Neighbourhood (sub-regions)
5.14 Example of the Stope Geometry Constraints File, ".cst"
5.15 The SLO Dialog for Defining Products of the Project
5.16 The SLO Dialog for Defining Prices and Price Units
5.17 The Dialog Box for Defining Various Costs
5.18 The Dialog Box for Defining Various Rates of Recovery for the Main Product
5.19 The Dialog Box for Defining Various Rates of Recovery of By-Products
5.20 The SLO Dialog Box for Defining Grade Units
5.21 The Dialog Box for Defining Ore Properties
5.22 Example of the Economic Factors File, ".eco"
5.23 The Dialog Box for Defining the (Main) Data File
5.24a Example of the Assay Value Data File, ".dat"
5.24b Example of the Economic Value Data File, ".dat"
5.25 Preparation of the Main Data File for the Optimisation Core
6.1 Major Modules Used to Perform the Optimisation Algorithm
6.2 The SLO Dialog Box which Specifies the Domain of the Optimisation
6.3 Numbering the Elements of a 2D Neighbourhood, \([\text{Onb}]^{(2)} = (4, 3)\)
6.4 Numbering the Elements of a General 2D Neighbourhood (numbers inside the cells indicate the sequential ID number of that element within the NB.)
6.5 Numbering the Elements of a 3D Neighbourhood, \([\text{Onb}]^{(3)} = (4, 2, 3)\)
6.6 The Identification of 2D Neighbourhoods
6.7 Two Neighbourhoods for the Block, \(B_{ijk}\), with the Order of (2, 2, 2)
6.8 A 2D Infeasible Neighbourhood (it is known that the element, BEV_{ijk}, is outside the model.)
6.9 The General Flow-Chart of the "SelectBlock" Subroutine
7.1 Example of the Output File, ".out" (the block model is \(4 \times 3 \times 3\))
7.2 SLO Dialog Box for Plotting Specified Plans or Sections
7.3 The Orientation of Plots in the SLO Program
7.4 Example of the Plans/Sections Plotted in the SLO Program
7.5 Main Subroutines Called to Plot any Plan or Section
7.6 Example of the Intermediate Optimisation Results Collected in the "*.res" Files
7.7 Example of the Neighbourhood Results Collected in the "*.nbv" Files
7.8 A Screen Summary Report
8.1 A Simple Block Model
8.2 Optimised Stope Example (using SLO)
8.3 Optimised Stope Example (manual technique)
8.4 Dollar Value Data of the Block Model Repeated in all Levels
8.5 The Optimised Stope Plot in all Levels (plans)
8.6 The Optimised Stope Section Obtained in the Excel Worksheets
8.7 Example of the Information Stored in "*.tst" Files Related to the Block Assay Information
8.8 Example of the Block Economic Data Stored in a "*.tst" File
8.9 Example of the Optimisation Process Results Stored in "*.tst" Files
8.10 Example of the Summary Results Stored in "*.tst" Files
A.1 Block value function
C.1 The Winteracter Based Welcome Page of the Stope Limit Optimiser (SLO)
C.2 Options Available for the Project Menu Item
C.3 The SLO Dialog to Specify New Projects
C.4 The SLO Dialog for Opening an Existing Project
C.5 The SLO Dialog for Saving the Current Project with a New Name/Directory
C.6 Options Available from the Edit Menu Item
C.7 Mode Selection for Block Model Input
C.8 The SLO Dialog which allows the Definition of the Block Model in the XYZ Mode
C.9 The SLO Dialog which allows the Definition of the Block Model in the IJK Mode
C.10 The Dialog Box for Choosing Options in the Block Model
C.11 The SLO dialog for Manipulating Sub-Regions
C.12 Defining/ Editing a Sub-Region within the Block Model
<table>
<thead>
<tr>
<th>C.13</th>
<th>The SLO Dialog for Defining the Stope Geometry Constraints (entire model)</th>
<th>C-14</th>
</tr>
</thead>
<tbody>
<tr>
<td>C.14</td>
<td>The SLO Dialog for Defining the Stope Geometry Constraints (sub-regions)</td>
<td>C-14</td>
</tr>
<tr>
<td>C.15</td>
<td>The Dialog for Defining the Products of the SLO Project</td>
<td>C-16</td>
</tr>
<tr>
<td>C.16</td>
<td>The Dialog for Defining Prices and Price Units in the SLO Program</td>
<td>C-16</td>
</tr>
<tr>
<td>C.17</td>
<td>The Dialog for Defining Costs in the SLO Program</td>
<td>C-17</td>
</tr>
<tr>
<td>C.18</td>
<td>The SLO Dialog for Defining the Rates of Recovery (main product)</td>
<td>C-18</td>
</tr>
<tr>
<td>C.19</td>
<td>The SLO Dialog for Defining the Rates of Recovery (by-products)</td>
<td>C-19</td>
</tr>
<tr>
<td>C.20</td>
<td>The SLO Dialog for Defining Grade Units</td>
<td>C-19</td>
</tr>
<tr>
<td>C.21</td>
<td>The SLO Dialog for Defining Ore Properties</td>
<td>C-20</td>
</tr>
<tr>
<td>C.22</td>
<td>The SLO Dialog for Defining the (main) Data File</td>
<td>C-21</td>
</tr>
<tr>
<td>C.23</td>
<td>Options Available for the Preoptimisation Menu Item</td>
<td>C-22</td>
</tr>
<tr>
<td>C.24</td>
<td>The SLO Dialog for Defining the Optimisation Domain</td>
<td>C-23</td>
</tr>
<tr>
<td>C.25</td>
<td>Options Available for the Results Menu Item</td>
<td>C-24</td>
</tr>
<tr>
<td>C.26</td>
<td>Example of the SLO Summary Report</td>
<td>C-25</td>
</tr>
<tr>
<td>C.27</td>
<td>The SLO Dialog for Plotting the Optimum Stope</td>
<td>C-26</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>1-3</td>
</tr>
<tr>
<td>1.2</td>
<td>1-7</td>
</tr>
<tr>
<td>1.3</td>
<td>1-8</td>
</tr>
<tr>
<td>3.1</td>
<td>3-13</td>
</tr>
<tr>
<td>3.2</td>
<td>3-32</td>
</tr>
<tr>
<td>4.1</td>
<td>4-38</td>
</tr>
<tr>
<td>5.1</td>
<td>5-6</td>
</tr>
<tr>
<td>9.1</td>
<td>9-5</td>
</tr>
<tr>
<td>A.1</td>
<td>A-7</td>
</tr>
<tr>
<td>A.2</td>
<td>A-8</td>
</tr>
<tr>
<td>A.3</td>
<td>A-8</td>
</tr>
<tr>
<td>B.1</td>
<td>B-2</td>
</tr>
<tr>
<td>B.2</td>
<td>B-5</td>
</tr>
<tr>
<td>B.3</td>
<td>B-12</td>
</tr>
</tbody>
</table>
LIST OF SYMBOLS AND ABBREVIATIONS

$\text{b}$: Billion dollars
$\text{m}$: Million dollars
$[NB]^{(2)}$: Two-dimensional neighbourhood
$[NB]^{(3)}$: Three-dimensional neighbourhood
$[NBS]^{(2)}$: Set of 2D neighbourhoods
$[NBS]^{(3)}$: Set of 3D neighbourhoods
$[O_{nb}]^{(2)}$: Two-dimensional order of neighbourhood
$[O_{nb}]^{(3)}$: Three-dimensional order of neighbourhood
1D: One-dimensional
2D: Two-dimensional
3D: Three-dimensional
$\text{AI}$: Artificial Intelligence
$\text{ANN}$: Artificial Neural Network
$\text{ASCII}$: American Standard for Communication Interchange
$\text{BEV}$: Block Economic Value
$B_{ijk}$: the block located at the $i^{th}$ row, $j^{th}$ column and $k^{th}$ section
$\text{BMC}$: Block Mining Cost
$\text{BRR}$: Block Revenue Ratio
$\text{CAD}$: Computer Aided Drafting
$\text{CM}_{M}$: unit cost of metal extraction from the ore
$\text{CM}_{ore}$: unit cost of ore (waste) extraction from the deposit
$\text{DP}$: Dynamic Programming
$\text{EF}$: Equivalence Factor
$F$: Flag of blocks
$g$: grade
$\text{GA}$: Genetic Algorithms
$g_{c}$: cut-off grade
$\text{GT}$: Graph Theory
GV: Gross Value

I: the number of blocks in the X direction of the block model

i: the block number in the X direction of the block model

J: the number of blocks in the Y direction of the block model

j: the block number in the Y direction of the block model

K: the number of blocks in the Z direction of the block model

k: the block number in the Z direction of the block model

kg: kilogram

LG: Lerchs-Grossmann

LP: Linear Programming

m: metre

m²: square metre

m³: cubic metre

Max: maximum

Min: minimum

MNBV: Maximum Neighbourhood Value

MPEG: Main Product Equivalence Grade

MVN: Maximum Value Neighbourhood

n(A): the size (number of elements) of a given set, A

NA: Not Applicable

NB: Neighbourhood

NBS: Set of Neighbourhoods

NBV: Neighbourhood Value

NBVS: Set of Neighbourhood Values

NSW: New South Wales

Onb: order of neighbourhood

oz: ounce

P: price

ppm: part per million

r: recovery

SBR: Stope Block Ratio

SEV: Stope Economic Value
SLO: Stope Limit Optimiser

SOS2: Type-Two Special Ordered Sets
   t: tonne
   U: universal set
   V: volume

VBA: Visual Basic for Applications
   \( \rho \): density
1.1 GENERAL INTRODUCTION

The mining industry is a significant contributor to Australia's export dollars. Figure 1.1 illustrates the mining industries contribution to Australia's total export dollars from 1986 to 1998. The current trend shows that mineral product exports have grown more slowly than that total exports. However, the mining sector is still supplying about one quarter of the country’s total exports.

Mineral products consist of three groups, which are metallic minerals, coal and oil and gas. During the year ended June 1997, the total value of minerals produced in Australia was $Aus 31.4 billion, of which the metallic minerals group was the major contributor (43%), followed by the coal industry (29%) (Australian Bureau of Statistics, 1998). Figure 1.2 shows each mineral products group share of the Australian export dollar for 1996-1997. The statistics from the previous years show a similar share for the metallic minerals group. The corresponding statistics for the last five years have been summarised in Table 1.1. This shows an average of approximately $Aus 12 billion per annum, and 42.9% of the total mineral production.
Figure 1.1: The Mining Industry Share of the Australian Export Dollar 1986-1998
(Source: Australian Bureau of Statistics, 1998a)

Figure 1.2: The Significance of Metallic Minerals in the Australian Mineral Products
(Source: Australian Bureau of Statistics, 1998b)
Table 1.1: Value of Australian Minerals Produced (1992-1997)*

<table>
<thead>
<tr>
<th>Year</th>
<th>Total value ($m)</th>
<th>Metallic minerals ($m)</th>
<th>Metallic minerals (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1992 - 93</td>
<td>26,721</td>
<td>10,920</td>
<td>40.9</td>
</tr>
<tr>
<td>1993 - 94</td>
<td>25,702</td>
<td>10,861</td>
<td>42.3</td>
</tr>
<tr>
<td>1994 - 95</td>
<td>26,738</td>
<td>11,715</td>
<td>43.8</td>
</tr>
<tr>
<td>1995 - 96</td>
<td>28,779</td>
<td>12,793</td>
<td>44.5</td>
</tr>
<tr>
<td>1996 - 97</td>
<td>31,358</td>
<td>13,422</td>
<td>42.8</td>
</tr>
<tr>
<td>Average</td>
<td>27,860</td>
<td>11,942</td>
<td>42.9</td>
</tr>
</tbody>
</table>

The efficient management of producing mineral products significantly influences the amount of Australia's income derived from exports. Hence, even minor improvements in the production system, or reduction in the cost per tonne of the minerals, would result in considerable profit to the industry. For example, if the enhancement of the production management results in an increase of only one percent of the production value, the increase in earnings will be a marginal value of $Aus 120 \times 10^6 annually.

1.2 STATEMENT OF THE PROBLEM

Mining is a process that involves a number of stages including: exploration; ore-body modelling; mine valuation and evaluation; mining method selection; ore extraction and transportation; ore treatment; and finally marketing. Optimisation at each stage is important to ensure efficient utilisation of natural resources and reduce production costs. In this regard, many mine designers are concerned with the optimisation of the mine geometry as it is one way of improving the efficiency of the overall mine production.

* Source: Australian Bureau of Statistics, 1998b
Due to the geo-technical and mining constraints, the extraction of a block of a high grade ore may entail the extraction of some blocks of waste as well. In other words, extra costs are incurred. For example, the extraction of an ore block in open pit mining, requires that all materials above the block, within the pit limit, be mined first. The shape of this pit is like an inverted cone, with the block as its base. In underground mining, the minimum sizes of the working space may require the mining of a waste block, alternatively, a block of ore may be left non-extracted because of the additional cost of extracting the waste blocks surrounding the ore. However, a selected combination of blocks may be found, as an optimum, which satisfies the exclusion of waste blocks from- and inclusion of ore blocks to- the mine layout. The selection of a block is based on the net economic value of the block and its neighbouring blocks. Optimisation of the mine geometry aims to maximise the total economic value of the mine by determining blocks for the final mine layout, subject to a number of mining constraints and economic parameters. This means that not only are more values of ore produced, but the cost per unit of production is decreased.

1.2.1 Status of Pit Limit Optimisation

A variety of algorithms have been proposed during the past 35 years to optimise ultimate pit limits. The early algorithms were soon followed by some modifications to allow those algorithms to be applied, with less restriction, and in a broader range of mining options. The major algorithms for the optimisation of pit limits include heuristic and mathematical approaches. Heuristic approaches include:

- Moving Cone technique (Pana, 1965; and Carlson et al., 1966) and
- Korobov’s Algorithm (Korobov, 1974; and Dowd and Onur, 1993)

Mathematical approaches include:

- Dynamic Programming technique (Lerchs and Grossmann, 1965; Johnson and Sharp, 1971; Wright, 1987; and Koenigsberg, 1982),
- Dynamic Cone method (Wilke, Mueller and Wright, 1984; and Yamatomi et
al., 1995),

- Graph Theory approach (Lerchs-Grossmann, 1965; Lipkewich and Borgman, 1969; Chen, 1977; and Zhao and Kim, 1992) and
- Network Flow approach (Johnson and Barnes, 1988; Yegulalp and Arias, 1992; and Jiang, 1995).

In addition, artificial intelligence approaches including:

- Genetic Algorithms, (GAs), (Denby and Schofield, 1994) and
- Artificial Neural Networks (ANN) approach (Achireko and Frimpong, 1996)

have been suggested recently. Other miscellaneous approaches include the:

- Parameterisation techniques (Lerchs and Grossmann, 1965; Whittle, 1988; Francois-Bongarcon and Guibal, 1982; Coleou, 1986; Coleou, 1989; and Wang and Sevim, 1992),
- Bounding techniques (Barnes and Johnson, 1982; and Caccetta, Giannini and Carras, 1986) and
- Transportation algorithm (Huttagosol and Cameron, 1992)

Most of these algorithms fail to result in true optimum pit limits and approximate an "optimum" value. However, it has been mathematically proven that the 3D Graph Theory algorithm, and the network flow algorithm, guarantee true optimisation of ultimate pit boundaries (Whittle and Rozman, 1991).

Mining engineers are well assisted, in pit limit optimisation, with a variety of commercial computer packages, including those that implement the well known moving cone and Lerchs-Grossmann algorithms. The commercial packages for the optimisation of open pit limits include:

- Minex-3D of Engineering Computer Services (ECS),
- Lynx Geosystems,
- Minesoft,
- Mintec,
• The Miner System,
• MineMap,
• Whittle Three-D, Whittle Four-D and
• L-TOPS.

In essence, the study of pit geometry optimisation has now reached saturation level. Many algorithms, using a vast range of techniques, have been developed. The true optimum solution is guaranteed and several computer packages are available to the industry.

1.2.2 Status of Stope Geometry Optimisation

Unlike open pit mining, there is a limited amount of available material for the optimisation of the stope layout. A summary of 62 publications, which contribute to mine geometry optimisation, is presented in Table 1.2. The major sources for the survey include: various proceedings of the international symposiums on the Application of Computers in the Minerals Industry (APCOM); transactions of the Institution of Mining and Metallurgy; and proceedings of the conferences on Mine Planning and Equipment Selection. The results of the 62 papers clearly show there is a dearth of techniques for optimising underground mine geometry.

Table 1.3 summarises the 62 papers on mine geometry optimisation, which are categorised by subject. The subject areas include: general mine layout optimisation; the introduction of algorithms; development of software for an algorithm; case study applications of the algorithms; and reviews of the existing algorithms.
Table 1.2: Publications which Contribute to Mine Geometry Optimisation
(open pit versus underground)

<table>
<thead>
<tr>
<th>#</th>
<th>Period (years)</th>
<th>Open pit limits optimisation (number of papers)</th>
<th>Stope geometry optimisation (number of papers)</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1970</td>
<td>4</td>
<td>-</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1971-75</td>
<td>1</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1976-80</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1981-85</td>
<td>6</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>1986-90</td>
<td>10</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>1991-95</td>
<td>21</td>
<td>4</td>
<td>25</td>
</tr>
<tr>
<td>7</td>
<td>1996-</td>
<td>7</td>
<td>-</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td><strong>52</strong></td>
<td><strong>10</strong></td>
<td><strong>62</strong></td>
</tr>
</tbody>
</table>

Apart from the quantity of research, the quality of the works carried out and the tools available for optimisation of underground metalliferous mines, are as expected, much lower than that for their open pit counterpart. Criteria, in this regard, include the status of the developed algorithms for both methods. The proposed algorithms for open pit limit optimisation are numerous and versatile, approximately two and a half times more than that of underground methods. There are approximately five different algorithms available for the optimisation of stope boundaries, including:

- Dynamic Programming Algorithm (Riddle, 1977),
- Downstream Geostatistical Approach (Deraisme et al., 1984),
- Octree Division Approach (Cheimanoff, Deliac and Mallet, 1989),
- Floating Stope Algorithm (Alford, 1995) and
- Branch and Bound Technique (Ovanic and Young, 1995).
Table 1.3: Reports on Mine Geometry Optimisation (by subject)

<table>
<thead>
<tr>
<th></th>
<th>Subject</th>
<th>Number of papers</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Open pit</td>
<td>Underground</td>
</tr>
<tr>
<td>1</td>
<td>General</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Introduction of an algorithm</td>
<td>26</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>Introduction of software</td>
<td>5</td>
<td>2*</td>
</tr>
<tr>
<td>4</td>
<td>Application of an algorithm</td>
<td>8</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>Review of algorithms</td>
<td>4</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>Others</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td><strong>52</strong></td>
<td><strong>10</strong></td>
</tr>
</tbody>
</table>

|   | Number of algorithms developed    | 12      | 5          | 15    |

There are various reasons for the lack of research in underground mine optimisation. These include generality, complexity and acceptability (Ovanic and Young, 1995).

- **Generality**: Unlike open pit mining, there are a variety of mining methods for underground mines. Each mining method has its own conditions and limitations. Therefore, it is difficult to develop a unique algorithm to optimise stope boundaries for the various mining methods.

- **Complexity**: Geological, geotechnical and economic data tend to be quite complex in underground mines. There are no simple mathematical formulations for many of the design problems in underground mines.

* The papers introduce both an algorithm and its software implementation.
- **Acceptability**: Although CAD systems have automated the steps, underground mine design practitioners are loyal to the traditional techniques of applying 'rules of thumb' to plans and sections.

The lack of comprehensive computer-based planning tools for underground mine planning, until the last few years, has influenced the situation in underground metalliferous mine planning (Alford, 1995). Software developers used their skills to tackle underground mine design, planning and production problems, after they had successfully applied computer techniques to open pit mining applications (Foley, 1992).

### 1.3 SCOPE OF THE THESIS

This thesis aims to develop and implement a 3D heuristic algorithm, termed the "Maximum Value Neighbourhood" (MVN), to optimise layouts of underground metalliferous mines. The MVN algorithm, developed in this thesis, is implemented on a fixed economic block model and provides a 3D analysis of the stope geometry optimisation problem. A schematic view of the whole system, on which the thesis is based, is illustrated in Figure 1.3. The whole study is performed in three stages, that is:

- development of the "neighbourhood" (NB) concept,
- development of the "Maximum Value Neighbourhood" (MVN) algorithm, on the basis of the neighbourhood concept, and
- development of a Fortran 90 based program, the Stope Limit Optimiser (SLO), to implement the MVN algorithm. The SLO was developed with a user interface software developer, Winteracter.

The *neighbourhood* (NB) concept is based on transforming the minimum allowable size of the stope, into discrete blocks in the three dimensions. Information concerning a block model, together with that of the stope geometry constraints, were used to develop the concept of the *neighbourhood*. Discrete blocks were defined in the block model. The geometric constraints defined the minimum stope size in the three directions. The *neighbourhood* concept, for the individual blocks, is then the basis for the whole
system. That is, the \textit{MVN} algorithm was developed on the basis of the \textit{neighbourhood} concept. Various possible neighbourhoods for a block are determined and evaluated to locate the \textit{Maximum Value Neighbourhood} of the block. The \textit{MVN} algorithm locates the best neighbourhood of each block, in a block model, and combines them to form the final optimum stope.

\textbf{Figure 1.3:} A Schematic View of the Proposed Study

Finally, an interactive system, the \textit{Stope Limit Optimiser (SLO)}, was developed to implement the \textit{MVN} algorithm. The \textit{Stope Limit Optimiser} is an application program, developed by integrating \textit{Fortran 90} codes and \textit{Winteracter} subroutines to provide a user friendly system. The features associated with this version of the high level language, \textit{Fortran 90}, have considerably improved the capabilities of the language. The
major improvements in Fortran 90 include the definition of allocatable (dynamic) arrays, pointers and targets, enabling the definition of derived types, case structure and use of modules. A user interface environment was developed by employing the features of Winteracter version 1.10. Winteracter and Interacter are two products of Interactive Software Services Ltd. (ISS), which supply software development tools for visual interface by the user. Interacter provides a DOS environment for the development of software while Winteracter is based on Windows environment. Winteracter provides menus and dialog boxes, through which the user can select options, input required data and retrieve outputs.

1.4 OUTLINE OF THE THESIS

The thesis has been designed in nine chapters. The first (current) chapter is to introduce the research. Chapter Two discusses the ultimate mine optimisation methods in general and reviews the major existing optimisation algorithms for both open pit limits and stope boundaries. It also reviews briefly the process of constructing an economic block model of an ore-body. Chapters Three and Four correspond to the development of the MVN algorithm. Chapter Three discusses the basics of the neighbourhood concept and the MVN algorithm in one dimension, while applying the algorithm on simple examples manually. Chapter Four reviews generalisation of the MVN algorithm for 2D and 3D neighbourhoods. Application of the 2D algorithm on a simple example is also provided. In addition, implementation of the algorithm on a small sized 3D example is introduced using Visual Basic for Applications (VBA) macros of Excel, to test the methodologies. Chapters Five, Six and Seven cover the development of the Stope Limit Optimiser (SLO). Chapter Five presents an overview of the SLO system and explains how the input of the optimiser is provided, while Chapter Six discusses the implementation of the optimisation algorithm and Chapter Seven discusses the outputs of the SLO. Chapter Eight discusses how the SLO was validated and Chapter Nine gives the conclusions and suggestions for future work.
CHAPTER TWO

ULTIMATE MINE DESIGN METHODS

2.1 INTRODUCTION

The stages of mining begin with exploration and end with the production of a commercial product. The intermediate stages of mining include: ore-body modelling; mine valuation; mine evaluation; mining method selection; ore extraction; ore transportation; ore treatment; and marketing. In general, attempts are made to balance each stage in the most optimum manner. A number of algorithms have been developed in the past for the optimal determination of the ultimate mine geometry for both the open pit and underground mining methods. These optimisation algorithms are mostly based on computerised methods and are performed on a geological model. When using these algorithms, care should be taken that it is the model that is optimised and not the real ore-body (Kim, 1978). That is, the accuracy of the optimisation depends on a reliable representation of the ore-body. This chapter reviews the existing algorithms for the optimisation of the open pit limits and underground stope boundaries. A review of computerised ore-body modelling, required for optimisation algorithms, is also presented in this chapter.
2.2 MINE GEOMETRY OPTIMISATION

The definition of the mine layout is one of the most important stages of mine planning for both surface and underground mines. Outlining the mineable ore assists in the determination of the amount of the reserve, as well as the mine life and production scheduling. There is a mutual relationship between mine layout definition and other mine planning stages, such as: equipment selection; haulage routes; and bench height. This means mine design is an interactive process with a paradox between mine layout definition and production scheduling. According to Whittle and Rozman (1991), prior to working out the mining schedule, the optimal layout of the mineable ore has to be defined. The optimal layout cannot be determined until the values of the blocks are known. However, the block values depend on factors, such as the commodity price and mining costs (blasting and transportation), which in turn depend on when the blocks are to be mined. Despite this fact, the mine layout optimisation and the production scheduling are usually addressed separately (Gill, Robey and Caelli, 1996), which leads to the use of certain assumptions and approximations. In this thesis, only, mine layout optimisation is covered.

2.2.1 Optimisation Criteria

Mine geometry optimisation requires using the formal procedures of Operations Research (OR). These include: the formulation of the problem; the definition of the objective function; and the formulation of constraints. The following objective function criteria have been used in mine geometry optimisation (Wright, 1990):

1. maximisation of the total mine economic value,
2. maximisation of value per tonne of the saleable product,
3. maximisation of the life of the mine, provided the value per tonne does not fall below a certain figure and
4. maximisation of the metal content within the mine.

The most frequently used criterion is the maximisation of the total mine economic
value. Wilke, Mueller and Wright (1984) have shown that the variation of the total value with the size of the pit (tonnage of the saleable product) is different from that of the unit value (Figure 2.1), for a specific mine.

\[
G = \text{total value ($)} \\
g = \text{specific value ($/t)}
\]

**Figure 2.1**: Variation of the Total Value Compared to the Unit Value of a Specific Mine over the Mine Production (Wilke, Mueller and Wright, 1984)

The total value and the unit value curves in Figure 2.1, show the value per tonne of the saleable product, reaches its maximum at a lower amount of production when compared with the total economic value of the mine. That is, the optimal mine layout is greatly influenced by the chosen optimisation criterion.

### 2.2.2 Problem Formulation

Taking the maximisation of the total mine economic value as the optimisation criterion, the problem is formulated to find the mine outline which has the maximum total economic value. In order to achieve this, an economic block model of the ore-body is required. The problem is reduced to selecting those blocks within the economic model that maximise the total value, while the selection of the blocks is constrained by the geometry of the mine (pit or stope). The problem is, then, simplified to find (Gill, Robey and Caelli, 1996):
Chapter Two: Ultimate Mine Design Methods

\[ \text{Max. } \sum_{(i,j,k) \in \gamma} BEV_{ijk} \]  \hspace{1cm} (2.1)

where

$BEV_{ijk}$: the block economic value and

$\gamma$: the set of blocks making up feasible mine geometry design.

The principal method of mining defines the optimisation constraint. In open pit cases, the mine geometry follows an inverted cone shape. The constraint is, therefore, the maximum allowable slope angle of this cone. However, the geometry constraint, in the optimisation of the stope boundaries follows the cubic shape of the stope. This can be formulated in terms of the minimum stope dimensions.

The optimum mine layout, if found, is unique. Since the optimal solution is the one with the highest dollar value, there is no single block, or combination of blocks, which can be added to, or removed from this mine layout that can produce an increase in the total value of the mine, within the specified layout. Assuming that there are two layouts with the same value implies that none of the layouts is optimal, since if taken together, the two layouts would produce an outline of higher value (Whittle, 1990).

2.2.3 Necessity of Optimisation Algorithms

The optimal mine layout is usually given in terms of a list of blocks, selected from an economic block model. It is necessary to determine whether or not every single block should be included in the optimal list of blocks. That is, there are two options for each block, selected or not selected. The number of possible alternatives for a combination of $n$ blocks within the model is equal to $2^n$, one of which is the optimal. A trial and error approach to find the unique optimal layout out of $2^n$ alternatives, even for a very small-sized model, can take millions of the time unit (Whittle, 1993). For example, a 2D section of $10 \times 10$ blocks requires a check of $2^{100}$ (approximately $10^{30}$) alternatives. Therefore, it is necessary to develop optimisation algorithms based on practical mining constraints. These algorithms must provide a search rule or a mathematical formulation,
which excludes most possible alternatives from the calculations. For example, a heuristic algorithm which searches ore blocks, can considerably reduce the possible alternatives. Considering that ore blocks may constitute 30% of the whole block model, the number of alternatives would decrease significantly from $2^{100}$ to $2^{30}$ alternatives (that is, from $10^{30}$ to $10^9$ alternatives).

2.3 ULTIMATE PIT OPTIMISATION ALGORITHMS

A variety of algorithms have been developed during the last decades to determine the shape of the ultimate pit. Lerchs and Grossmann (1965) and Pana (1965), who independently studied the same problem, can be considered the pioneers in this field.

Kim (1978) has categorised the ultimate open pit limit design methodologies into two main groups: rigorous and heuristic. The term rigorous was used to imply the availability of mathematical proof for the optimising technique while the term heuristic was used to describe an algorithm which works in nearly all cases but which lacks rigorous mathematical proof. Graph Theory, Network Flow, Linear Programming and Dynamic Programming algorithms fall into the rigorous category while the Moving Cone technique and the algorithm of Korobov were considered heuristic. More recently, Gill, Robey and Caelli (1996) have categorised the algorithms into historical approaches, including Moving Cone and Dynamic Programming and modern techniques, which included Graph Theory and Network Flow solutions.

Thomas (1996) has presented five parameters to categorise the existing algorithms. These parameters include:

1. "rigorous", a term applied to techniques which always find the optimum solution if given sufficient time (like the Dynamic Programming technique),
2. "heuristic", a term applied to any technique which is not rigorous and finds only an approximate solution, regardless of being close to the true solution or not (like the Moving Cone technique),
3. "stochastic", a term applied to techniques whose analyses are based on probabilistic sampling from the range of possible solutions (like the genetic algorithms),

4. "static", a term applied to analyses which ignore the effects of time on the valuation process (like the Lerchs-Grossmann algorithm) and

5. "dynamic", a term applied to analyses which take into account the effects of time (like Whittle Four-D).

Based on the above classification, the Graph Theory algorithm of Lerchs and Grossmann would be described as rigorous and static, whereas a genetic algorithm, which included financial discounting effects would be described as dynamic, stochastic and heuristic.

However, there is no global agreement on the classification of optimisation algorithms. The major existing algorithms, for optimisation of ultimate pit limits may be listed as:

- Moving Cone technique (simulation approach),
- Korobov’s Algorithm,
- Dynamic Programming technique,
- dynamic cone method,
- Graph Theory approach (Lerchs-Grossmann algorithm),
- Network Flow approach,
- Linear Programming technique,
- parameterisation technique,
- bounding technique,
- transportation algorithm,
- genetic algorithms (GAs) and
- artificial neural network (ANN) approach.

The most popular approaches used in open pit limit optimisation from the above list include the Moving Cone technique, Dynamic Programming algorithms and the Graph Theory algorithm. These three algorithms, together with their modified versions, are
briefly reviewed in the following sections.

2.3.1 Moving Cone Algorithms

The Moving Cone approach and the Korobov algorithm may be considered heuristic. The Moving Cone technique is the simplest and fastest algorithm developed for pit limit optimisation. The algorithm proposed by Korobov is a major improvement of the technique. The Dynamic (Moving) Cone algorithm, which is a combination of applying both Moving Cone and Dynamic Programming techniques, may be considered a heuristic algorithm as well.

The Moving Cone algorithm, as described by Carlson et al. (1966), is based on constructing an inverted cone above any ore block (the block with positive economic value). The cone includes all blocks that should be removed preceding the mining the ore block. The angle of this cone is controlled by the ultimate slope angle, which satisfies the stability of the pit and may vary in different directions. To be more precise, it is usually not a true cone, but a pyramidal approximation of an irregular cone (Gill, Robey and Caelli, 1996).

The Moving Cone algorithm uses a simulation approach to determine the optimal pit. The basic element to simulate the mining of the pit is the minimum removal cone (the inverted cone based on the block under consideration). Steps taken in the algorithm as stated by Wright (1990) are as follows:

1. Start from the surface and search for ore blocks (blocks with positive economic values).
2. Construct the minimum removal cones on such ore blocks.
3. If the cone value (the sum of block economic values, \(BEV\), of all blocks contained in the given cone, including the ore block in question) is positive, consider the cone removed (that is, mining is simulated).
4. Continue the search until all the ore blocks in the block model have been
5. The ultimate pit is formed by the shape left after removal of all positive valued cones.

Figure 2.2 illustrates the procedure used by the Moving Cone technique to the ultimate pit optimisation, where the maximum slope angle of the pit is 45°.

The Moving Cone algorithm benefits from its simplicity in both concept and computer implementation. Its main problem is how to deal with a group of blocks, which fall inside more than one cone, that is, the problem of overlapping cones. In the example shown in Figure 2.2c, it can be easily seen that after the algorithm is completed, the pit is not maximised. That is, the inverted cone based on the block, \( B_{27} \), which is valued at +1, can be added to the pit and make a pit valued at +3. Figure 2.3 illustrates the failure of the Moving Cone to recognise the maximum valued pit. This also shows that if the block, \( B_{44} \), is examined before the block, \( B_{27} \), the technique leads to a different pit value. In general, as stated by Lemieux (1979), the order of the search for examining ore blocks plays a very significant role in the Moving Cone algorithm.

In addition, two separate cones may be of negative value, yet if taken together, a positive pit is obtained. In these cases, the Moving Cone technique fails to recognise the positive valued pit. As illustrated in Figure 2.4, both cones based on ore blocks, \( B_{33} \) and \( B_{35} \), are of negative value, and the Moving Cone technique would conclude that there is no pit at all. However, by combining the two cones, a pit of value +1 can be obtained (Figure 2.4c).
A section example

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

\[C_{22} = -1\]

\[C_{23} = 1\]

\[
\begin{array}{ccccccc}
1 & & & & & & \\
1 & -1 & -1 & -1 & -1 & -1 & -1 \\
2 & -1 & 2 & 4 & 0 & 1 & 0 & 2 & -1 \\
\end{array}
\]

\[C_{25} = -1\]

\[C_{27} = -1\]

\[
\begin{array}{ccccccc}
1 & & & & & & \\
1 & & & & & & \\
2 & -1 & 2 & 0 & 1 & 0 & 2 & -1 \\
\end{array}
\]

\[C_{47} = 1\]

\[C_{48} = 1\]

\[
\begin{array}{ccccccc}
1 & & & & & & \\
1 & & & & & & \\
2 & -1 & 2 & 0 & 1 & 0 & 2 & -1 \\
\end{array}
\]

Note: \(C_{ij}\) is the cone economic value based on the block, \(B_{ij}\).

**Figure 2.2:** The Moving Cone Technique
Chapter Two: Ultimate Mine Design Methods

Figure 2.3: Failure of the Moving Cone to Recognise the Maximum Value Pit

Figure 2.4: Failure of the Moving Cone Algorithm to Recognise a Positive Value Pit (Wright, 1990)

Modifications, including Korobov (1974), have been suggested to overcome the
problem of overlapping cones. The Korobov algorithm was based on evaluating the positive net value blocks in levels, level by level, from the top to the bottom. Within each cone, the algorithm allocated the positive valued blocks against the negative valued blocks until all negative valued blocks can be combined with the positive ones. The algorithm required several passes to repeat the process. However, the earliest version of the Korobov algorithm, as described by David, Dowd and Korobov (1974), failed to overcome the problem of overlapping cones in all conditions. A major enhancement to the Korobov algorithm was suggested by Dowd and Onur (1992), in which they proposed the joint testing of cones before it was decided whether to remove a cone or not. The latest improvement in the Moving Cone technique, called Moving Cone II, may be found in Wright (1999).

2.3.2 Dynamic Programming Algorithms

Application of a Dynamic Programming (DP) technique to the optimisation of the ultimate pit limits was first introduced by Lerchs and Grossmann (1965). The first DP algorithm was simple and rigorous but suitable for only 2D cross sectional problems. Smoothing routines were then developed to overcome this shortcoming of the 2D DP algorithm. Other modifications contain the development of algorithms to directly apply 3D DP technique.

The basis of the 2D DP approach is illustrated in Figure 2.5. The block model is first divided into several slices or parallel vertical sections, which are arranged in \( I \) rows and \( J \) columns. The economic value of blocks \((BEV)\) are represented by the term \( m_{ij} \), shown inside each cell (Figure 2.5a). To determine the optimum contour of each section, three straight lines must be defined. That is, the bottom of the pit and two walls, at a slope angle of \( \alpha \), which is the maximum angle allowed to meet the slope stability requirement. For simplicity, the earliest algorithms assumed that \( \alpha \) is constant over the whole pit.

Width and height of blocks define the pit slope, \( \alpha \), that is:
\[
\frac{\text{Block Height}}{\text{Block Width}} = \tan(\alpha)
\]

In other words, the maximum pit slope constraint is satisfied by moving one block left (right) and one block up (down). In order to mine a block (in Figure 2.5a), all blocks above it must be removed prior to mining the block. The summation of the values of the blocks of the same column, from the surface to and including the block under consideration, defines the cumulative economic value of that block. This is termed \( M_{ij} \), that is:

\[
M_{ij} = \sum_{n=1}^{i} m_{nj} \quad \text{for } j = 1, 2, ..., J
\]  

(2.2)

where

\( m_{nj} \): the economic value of the block located in row \( n \) and column \( j \)

\( J \): the number of columns in the section and

\( M_{ij} \): the cumulative economic value of a column of blocks from the surface to and including the block, \( B_{ij} \).

The value \( M_{ij} \) is computed and assigned for each block (Figure 2.5b). The mining of the block, \( B_{ij} \), requires removing all blocks which fall within the minimum cone, that satisfies the slope constraint. That is, the block must be extracted jointly with one of its neighbouring blocks from the previous column, either from the previous row \( (B_{i-1,j-1}) \), same row \( (B_{i,j-1}) \), or subsequent row \( (B_{i+1,j-1}) \). The optimum contour requires that the one which provides the maximum pit value, be selected to be considered jointly with the block, \( B_{ij} \). Hence, another term, \( P_{ij} \), is used to represent the pit value when the pit is a cone based on the block, \( B_{ij} \). The pit value corresponding to each block is computed through a recursive formula (Equation 2.3).

\[
P_{ij} = M_{ij} + \max \left\{ P_{i-1,j-1}, P_{i,j-1}, P_{i+1,j-1} \right\}
\]  

(2.3)
where $P_j$ is the maximum possible contribution of columns 1 to $j$ to any feasible pit that contains the block, $B_{ij}$, on its contour.

**Figure 2.5:** A Dynamic Programming Technique Applied to Pit Limit Optimisation
The formula can be modified to a general form as expressed by Equation 2.4:

\[ P_y = M_y + \max \{ P_{i+r,j-1} \} \]  

(2.4)

where \( r \) is the range of rows of blocks, of the previous column, that should be included in the neighbouring blocks, to satisfy the maximum slope constraint.

Determining the optimum neighbour does not necessarily cause its inclusion in the optimum pit, but rather implies that if the examined block is a boundary block, it has to be mined jointly with its optimum neighbour. The algorithm starts with adding a new row on the top of the section (that is, \( i = 0 \)), as shown in Figure 2.5b. This artificial row contains air blocks with zero pit values (that is, \( P_{0j} = 0 \)), and helps in maximising the pit value of neighbouring blocks when processing blocks in the first row. The pit value, corresponding to each block, \( P_y \), is computed starting from the block, \( B_{00} \), as in Figure 2.5c. Blocks are examined row by row within each column then column by column within the section. After determining the optimum neighbour for each block, an arrow is drawn from the examined block pointing to its optimum neighbour (Figure 2.5c). After all blocks are examined, the optimum pit is obtained by maximising the pit value computed for blocks in the artificial row (that is, \( i = 0 \)), as expressed in Equation 2.5.

\[ P_{\text{max}} = \max P_{0,j} \quad \text{for } j = 1, 2, \ldots, J \]  

(2.5)

If the maximum value of \( P \) in the artificial row is positive, then the optimum contour is obtained by tracing the arrows from, and to the left of, the block containing this maximum. In cases, where the maximum value of \( P \) in the first row is negative, there is no pit containing a positive value. Figure 2.5c illustrates the determination of the optimum contour for the section by computing the \( P \) value and the optimum neighbour of each block.

The 2D DP algorithm provides the true optimum contour for each section. However, it is impractical in the real 3D pit, because when all the optimised contours of the vertical
sections are set to form the final pit, it appears that they do not fit together. In other words, while cross-sections have been optimised subject to maximum slope angle, $\alpha$, longitudinal sections do not necessarily satisfy this constraint.

Some modifications have been proposed for smoothing the pit contour so that the slope limit requirement be met. The algorithm of Johnson and Sharp (1971) was based on double optimisation, that is, once for cross sections and another time for right angles to the original sections. Their algorithm does not solve the problem completely and suits the elongated deposits. Another smoothing routine is the Dynamic Path approach suggested by Wright (1987). The Dynamic Path approach exploits two facts from the Moving Cone and 2D DP techniques. The first is that the 2D DP technique guarantees the true optimum contour for vertical sections and the second is that the 3D Moving Cone approach automatically supports the boundary smoothing. The Dynamic Path algorithm can solve the smoothing problem completely. It applies the same 2D recursive formula (Equation 2.4) in a forward pass while changing the direction of analysis when moving from one section to another one. The union of the minimum removal cones along the Dynamic Path defines the ultimate pit. Despite the fact that smoothing routines provide speed and small computer memory requirements, they lack the ability to guarantee the true optimal solution (Wright, 1990).

Other attempts have been made to directly apply the DP technique to optimise the 3D pit limits. These include the algorithm of Koenigsberg (1982) and the Dynamic Cone algorithm, which have been developed mostly by modifying the 3D neighbouring blocks. In the 2D DP algorithm a given block can be accompanied with only one neighbouring column in the backward direction, that is, the column which was previously examined. However, in a 3D problem, there are four neighbouring columns in the backward direction. Koenigsberg (1982) addressed these columns as:

$Si$ in column $(j-1, k)$: side of $i$

$BSi$ in column $(j-1, k-1)$: back of side of $i$

$SBSi$ in column $(j, k-1)$: side of back of side of $i$

$SSBSi$ in column $(j+1, k-1)$: side of side of back of side of $i$
The recursive formula was then modified accordingly, to cover maximisation among all the neighbouring blocks. Wilke, Mueller and Wright (1984) addressed the degeneration problem of the Koenigsberg’s algorithm and developed a new recursive formula, which was based on the use of 3D increments, in the form of minimum removal cones over each block. Such an approach uses the Moving Cone technique in the DP application and has been labelled as the Dynamic Cone algorithm. A recent modification of the dynamic cone algorithm, the selective extraction dynamic cone algorithm, was introduced by Yamatomi et al. (1995). The improvement is based on examining whether or not: any negatively valued blocks are included in a unit cone; it is possible to leave the negatively valued blocks unmined in the unit cone, without disturbing the geometric condition imposed on the slope angle of the cone.

2.3.3 Graph Theory Algorithms

The application of Graph Theory (GT) to the optimisation of the ultimate pit design was first introduced by Lerchs and Grossmann (1965) as a result of the impracticality of their 2D DP algorithm in three dimensions. The technique did not receive much attention for many years, mostly due to some computing time problems. With improvement in capabilities in computers, the application of the Graph Theory technique was again considered. The early algorithm was improved and commercial computer packages were developed (Lipkewich and Borgman, 1969; Chen, 1977; Zhao and Kim, 1992; Alford and Whittle, 1986). The 3D GT algorithm is widely known as the Lerchs-Grossmann (LG) algorithm. Alternative approaches to the LG Graph Theory algorithm have also been proposed. These include the mathematical programming solution suggested by Picard (1976) and the solution derived from a Dual Simplex viewpoint described by Underwood and Tolwinski (1996).

The LG algorithm is based on transforming the block model into a weighted directed graph and finding the maximum closure of the graph (Lerchs and Grossmann, 1965). A directed graph, \( G = (V, A) \), is defined by a set of elements, \( V = \{v_1, v_2, ..., v_n\} \), which are called vertices of \( G \), as well as a set of ordered pairs of elements, \( A = \{a_1, a_2, ..., a_m\} \).
where \( a_k = (v_i, v_j) \), which are called arcs of \( G \). In a physical interpretation, an arc is an arrow, which connects to vertices, pointing from the first vertex (predecessor) to the last vertex (successor). A closure of \( G \) is defined as a set of vertices \( Y \subset V \) such that if \( v_i \in Y \) and \( (v_i, v_j) \in A \), then \( v_j \in Y \). In other words, if \( v_i \) is in the closure, then all successors are in the closure as well. When a real number, \( m_n \), which can be negative, zero, or positive, is associated to each vertex, \( v_n \), of the graph, \( G \), which is called the mass of \( v_n \), then a maximum closure of \( G \) is defined as a closure of maximum mass. As stated by Picard (1976), the main application of the maximal closure of a graph is found in the ultimate pit optimisation.

The application of the GT to the optimum pit, as formulated by Lerchs and Grossmann (1965), follows. Let the entire pit be divided into a set of blocks, \( B_n \), regardless of being regular or irregular blocks. Each block is associated with a mass, \( m_n \), representing the net economic value of the block. The block, \( B_n \), is adjacent to \( B_j \), if it has at least one point in common with \( B_j \). (Figure 2.6) The block, \( B_n \), is also dependent on \( B_j \), if removing \( B_n \) is not permitted unless \( B_j \) is mined already. Transforming blocks, \( B \), to vertices, \( V \), an arc \( (v_b, v_j) \) is drawn if \( B_n \) is adjacent to and dependent on \( B_j \). Hence, a 3D directed graph \( G = (V, A) \) with a set of vertices, \( V \), and a set of arcs, \( A \), is obtained. Any feasible contour of the pit can then be represented by a closure of \( G \). Considering the net value of each block as the mass, \( m_n \), associated with its corresponding vertex, the problem is formulated in finding the maximal closure of \( G \). The LG algorithm, to find the maximum closure of a graph is based on a tree structure, starting with the construction of a tree, \( T^0 \), in \( G \). The Branches of trees, which are defined by arcs, represent the relationships between the blocks, that is, whether or not the ore blocks can support the attached waste blocks. Based on graph rules, the tree, \( T^0 \) is transformed into successive trees, \( T^1, T^2, \ldots, T^n \), until there is no possible transformation (Lerchs and Grossmann, 1965). The vertices of a set of well-identified branches of the final tree will then define the maximum closure of \( G \) and represent the pit with the maximum profit.

The LG algorithm is able to ensure the true optimality of pit design. It can also handle the pit optimisation of irregular block models. The implementation of the algorithm
requires too much computing time and memory. The connection information of a medium-sized model (150,000 blocks) requires over a gigabyte of storage space or virtual memory (Gill, Robey and Caelli, 1996).

\[ (a) \text{ A section of a block model} \]

\[ (b) \text{ A graph theoretical representation of the block model} \]

**Figure 2.6: Graph Theory Applied to Pit Limit Optimisation**

Lipkewich and Borgman (1969) suggested the evaluation of the LG model using subgraphs in separate levels, starting from the top and working to the bottom of the model. The LG algorithm was implemented for each level and the optimised sub-pit was removed before proceeding to the next level. Chen (1977) addressed the difficulty of the LG algorithm in calculating an optimum pit with variable wall slopes. He proposed a
search pattern for handling this problem.

Zhao and Kim (1992) developed a modified LG algorithm, which was simpler than the well-known LG algorithm to implement. The algorithm of Zhao and Kim was aimed at considerably reducing the number of generated arcs. In their algorithm, arcs were only generated between either ore to waste vertices, or waste to ore vertices. That is, no arcs were generated between two ore vertices, nor two waste vertices. Their algorithm was claimed to be faster than the LG algorithm, although the claim has been questioned (Gill, Robey and Caelli, 1996).

Alford and Whittle (1986) discussed the computer implementation of the LG algorithm in one package for designing open pit mines. Their package was later improved and is commercially marketed as Whittle Three-D. The latest version, Whittle Four X, is a highly sophisticated package, which allows for economic analysis and simulation of long-term mining projects.

2.3.4 Network Flow Analysis

The application of the Network Flow (NF) technique to the pit limit optimisation was first proposed by Johnson (1968). A mathematical support for the NF analysis can be found in Picard (1976). Johnson and Barnes (1988) applied the earliest maximal flow algorithms to design pits. The implementation of the latest algorithms has been reported by Yegulalp and Arias (1992) and Jiang (1995). Giannini et al. (1991) developed the pit design software, PITOPTIM, incorporating a high speed Network Flow technique.

A network consists of nodes, arcs and values associated with arcs, which are called capacities. Figure 2.7 illustrates a typical network model. In constructing such a network model the following steps are undertaken (Jiang, 1995).

1. Represent each block, within the block model, by a node in the network model.
2. Add two artificial nodes to the network model, that is, a source node and a sink (terminal) node.

3. Generate arcs from the source node to all nodes corresponding to the positive valued blocks. Each arc is assigned a maximum flow capacity that is equal to the net economic value of the corresponding block (BEV).

4. Generate arcs from all nodes corresponding to the negative, or zero valued blocks, to the sink node while assigning the corresponding BEVs as capacities of those arcs.

5. Generate arcs from all nodes with positive valued blocks to nodes which are negative or zero valued blocks. Non-positive valued blocks are to be mined if the positive valued block is mined.

![Network Flow Technique](image)

(a) A section example

![Network Representation](image)

(b) A network representation of the block model

**Figure 2.7:** The Network Flow Technique Applied to Pit Limit Optimisation
The objective of network analysis is to maximise the amount of flow from the source node to the sink node, knowing that all arcs are directed to the sink node. That is, the flow through the network is permitted only in the direction of the sink node. The flow through the network from the source to the sink represents the transfer of the economic value away from the ore blocks, as these are required to cover the cost of the removal of the related waste blocks (Dincer and Golosinski, 1993). When the maximum flow solution is obtained, arcs may or may not reach their maximum capacity (saturation). Saturated arcs show that the corresponding positive valued blocks are exhausted and cannot contribute to the final pit value. The unsaturated arcs from the source (those with excess capacity) can identify the optimum pit limits. The positive valued blocks corresponding to these arcs, together with all the negative or zero valued blocks on which these blocks are dependent, can be mined at a profit.

Several algorithms have been developed to maximise flow through the network. The earliest one was developed by Ford and Fulkerson (1956) and is called a labelling algorithm. Yegulalp and Arias (1992) have listed 18 algorithms for the maximisation of the flow through the network.

The Network Flow analysis benefits from a relatively simpler concept and algorithm (compared to Lerchs-Grossmann (LG) algorithm) and guarantee the true optimal solution to the pit limits optimisation. For large-sized block models, the Network Flow analysis is not suitable, although it may provide faster solutions for ore-bodies with a small number of blocks (Yegulalp and Arias, 1992).

### 2.3.5 Other Approaches

In addition to the above algorithms, the application of a wide range of techniques to pit geometry has been proposed. Some of these approaches have been devised for the joint consideration of ultimate pit limits determination and extraction scheduling. Some others aim to directly solve the ultimate pit limits problem. There are approaches which
aim to reduce the computing time associated with the existing algorithms.

2.3.5.1 Linear Programming Approach

The application of the Linear Programming (LP) techniques to pit optimisation was described by Meyer (1969). However, the most significant disadvantage of the approach was its excessive need for computer memory and computing time. The ultimate pit limit problem was formulated as a large scale transportation problem with an LP solution by Huttagosol and Cameron (1992). The adapted simplex algorithm of the LP is used to solve the dual system. The required computing time and memory are the main constraints of the method, especially for a large scale problem.

2.3.5.2 Parameterisation Techniques

Due to variation of economic parameters over time, the optimum pit limits would vary as well. The technique is, therefore, based on generating the nested pits, each of which is obtained by running the optimisation algorithm while changing the value of each block. In the latest improvement, Whittle (1988) defined an economic parameter as the ratio of metal price to the mining cost and developed the Whittle Four-D software, which generates incremental pits by increasing this economic parameter.

Some geostatisticians proposed to separate the economic parameters (costs and price) from the physical properties of the ore-body (for example, grade, tonnage and specific gravity) and the technical constraints, such as, the maximum allowable slope. The approach is called the "Technical Parameterisation" and the "Geostatistical Approach" and includes works by Francois-Bongarcon and Guibal (1982), Coleou (1986), Coleou (1989) and Wang and Sevim (1992). The main technical parameters that are used in defining the mining project are the tonnage of the total material mined, \( V \), the tonnage of the ore mined, \( T \), and the metal content, \( Q \). The ratio of \( V \) to \( Q \) and \( T \) to \( Q \) can then be defined and used in double parameterisation (Dagdelen and Francois-Bongarcon, 1982).
2.3.5.3 Bounding Techniques

"Bounding techniques" are modifications to the existing pit limit optimisation algorithms (Barnes and Johnson, 1982). Two problems associated with the existing algorithms are the computing core memory and time requirements. These requirements increase with the number of blocks in a block model, especially in the large open pit mines (Barnes and Johnson, 1982; Caccetta, Giannini and Carras, 1986). The objective here is to reduce the model size by removing the unnecessary blocks. For example, blocks located outside the largest possible cone, which satisfies the maximum slope constraint and fits to the block model, are removed from the process as they are out of the optimum pit. The software development for “bounding techniques” implementation has been reported in the PITOPTIM package (Giannini et al., 1991). With rapid advances in the capabilities of computer facilities, especially in the 1990s, the significance of the speed of the algorithms, and consequently the application of bounding techniques, has considerably decreased.

2.3.5.4 Application of Artificial Intelligence

The application of the Genetic Algorithms in open pit design was first proposed by Denby and Schofield (1994). They combined the concepts from stochastic optimisation with heuristic search and included financial discounting to develop a dynamic algorithm. The technique is based on a self-training genetic algorithm and provides simultaneously, a combined pit limit and extraction schedule that aims to maximise the net present value (NPV) of the production. The process starts with a population of the random solutions. The Genetic operators, together with the probability techniques, are used to progress from one population to the next. The genetic operators used in the process include “reproduction”, “crossover” and “mutation”. No details of the application of the algorithm using real data, have been reported.

More recently, Achireko and Frimpong (1996) applied the Artificial Neural Networks, (ANN) in combination with the “Conditional Simulation” technique, to optimise the pit limits. They addressed the random field properties associated with the ore grades and
reserves, which are ignored in the preceding algorithms, and modelled them using the "Modified Conditional Simulation" (MCS). This is based on the best linear unbiased estimation and the local average subdivision techniques. ANN are used to divide the blocks into classes based on their conditioned economic values. In order to optimise the pit limits, "error back propagation", a neural networks training algorithm, is used to minimise the desired and actual output errors. By implementing the algorithm on a medium size deposit, the results match that obtained from the LG algorithm. It is reported that the algorithm is fast since it mimics the operation of the brain in data processing and algorithmic computations (Frimpong and Achireko, 1997).

### 2.4 STOPE LIMIT OPTIMISATION ALGORITHMS

In comparison to open cut mining, not much has been done in the formulation and optimisation of design problems for underground mines. Limited algorithms are available for the optimisation of the ultimate stope boundaries. It should also be noted that until the last few years, comprehensive computer-based planning tools for underground mine planning did not exist (Alford, 1995). Software developers used their skills to tackle underground mine design, planning and production problems after they had successfully applied computer techniques to open pit mining applications (Foley, 1992). The main reasons for lack of research in underground mine optimisation include the versatility and complexity of underground mines (Ovanic and Young, 1995).

The available algorithms for the optimisation of stope boundaries are listed below:

- Dynamic Programming (DP) Algorithm,
- Downstream Geostatistical Approach,
- Octree Division Approach,
- Floating Stope Algorithm and
- Branch and Bound Technique.

From the above, the DP, Geostatistical and the Branch and Bound techniques may be
considered as mathematically based approaches.

2.4.1 Dynamic Programming Algorithm

The application of the DP technique to the optimisation of the ultimate underground mine limit has only been reported for designing the layout of block caving mines (Riddle, 1977). The approach is a modification of the algorithm of Johnson and Sharp (1971). The initial algorithm was developed for optimising the ultimate open pit limits. It assumes equal-size blocks and a $45^\circ$ allowable pit slope, and uses the typical formulae for pit limit optimisation by the DP approach:

\[
\begin{align*}
M_{ij} &= \sum_{q=1}^{i} m_{qj} \\
P_{ij} &= M_{ij} + \max \{ P_i + r_{j-1} \}
\end{align*}
\]

where

- $m_{qj}$: the economic value of the block is located in row $q$ and column $j$,
- $M_{ij}$: the cumulative economic value of a column of blocks from the surface to and including the block, $B_{ij}$,
- $r$: the range of rows of blocks of the previous column that should be included in the neighbouring blocks to satisfy the maximum slope constraint and
- $P_{ij}$: the maximum possible contribution of columns 1 to $j$ to any feasible pit that contains the block, $B_{ij}$, on its contour.

In order to modify the algorithm to suit the underground block-caving methods, the recursive formula was modified according to the specific mining constraints imposed by block caving.

There are basic differences when applying the Riddle's DP technique to open pit and block caving constraints. The open pit must be initiated from the surface in one-block steps. However, a block-caving mine may include a vertical boundary cutoff, so that mining may begin at any elevation within a fringe stope. This entry point may vary horizontally (at any column), that is, mining does not necessarily start from the first
drawpoint. The height mined is constrained by draw control rather than pit slope. Additionally, boundary cutoffs may be established at several locations across a section, under certain conditions, so as to leave areas void of values as footwall regions. Figure 2.8 illustrates the optimum layouts of the same section of a block model, optimised by both open pit and block-caving methods together with their profitability.

Two major stages necessary to develop the Riddle’s algorithm include:

1. developing the variation in “r” to allow draw-control in the recursive formula (Equation 2.6) and

2. providing consideration for entry and exit at any block elevation with the use of boundary cutoffs.

**Figure 2.8:** Comparison between Open Pit and Block-Caving (Riddle, 1977)
In open pit cases, assuming equal-size blocks and a 45° allowable pit slope, the integer \( r \) value is \(-1, 0 \text{ or } 1\), in the recursive formula. However, in the block-caving situation, the value of \( r \) is a function of the row index (elevation) and the percent draw control used at the mine. A percent draw control of 10% requires that the variation in draw between adjacent drawpoints must be no more than 10%. Therefore, if a block is examined in row 14 and the draw control is 20%, variation must be no more than \( \pm 2.8 \) blocks, which is rounded to 3. Figure 2.9 illustrates the adjacent blocks concept in both open pit and block-caving. Mathematically, the value of \( r \) is expressed as in Equation 2.6:

\[
\begin{align*}
    r &= \text{all integer values in the range of } r_{\text{min}} \text{ to } r_{\text{max}} \\
    r_{\text{min}} &= -\text{Max} \{1, (\% \text{ draw control}) \times i\} \\
    r_{\text{max}} &= +\text{Max} \{1, (\% \text{ draw control}) \times i\}
\end{align*}
\]  

(2.6)

where \( i \) represents the row number of the examined block.

---

Blocks: square  
Pit slope: 45 degrees  
r = \(-1, 0, 1\)

Blocks: square  
Percent draw point = 20%  
i = 14  
r = \(-3, -2, -1, 0, 1, 2, 3\)

(a) Open pit case  
(b) Block caving case

**Figure 2.9:** Adjacent Blocks in Open Pit and Block-Caving

In order to handle the horizontal variation of the starting block, an additional subscript is given to the \( P_y \) parameter to define the starting block in the block caving. The basic
DP recursive formula for block-caving mines is then expressed as:

\[
P_{ij0} = M_{ij} + \max\{P_{i+r_{j-1}}\}
\]  

(2.7)

where

- \(M_{ij}\): the cumulative net value of blocks (the same as in open pit)
- \(r\): the range indicating adjacent blocks
- \(P_{ij0}\): the profit achieved by mining through the block in row \(i\) of drawpoint \(j\) and starting at any level of drawpoint \(o\).

Assuming a 2D section of blocks with \(I\) rows and \(J\) columns, the algorithm could then be summarised in the following steps:

1. Calculate all \(P_{i,j,1}\) (that is, calculate \(P_{ij}\) for all blocks, given that the mine is starting at any level of drawpoint 1).

2. Eliminate column 1, calculate all \(P_{i,j,2}\) (that is, calculate \(P_{ij}\) for all blocks, except for the blocks in column 1, given that the mine is starting at any level of drawpoint 2).

3. Continue through column \(J\), where \(P_{i,j,J} = M_{i,j}\) (that is, repeat step 2 through to the last column, each time calculating \(P_{ij}\) for all blocks, except for blocks in the previous columns. At the last column, since all previous columns are ignored in calculating \(P_{ij}\), results equal the same cumulative net value of blocks).

4. Determine \(\max\{P_{k,m,n}\}\) for \(k = 1, 2, \ldots, I; m = 1, 2, \ldots, J; n = 1, 2, \ldots, j\).

Then the exit block is \(B_{km}\), and arrows may be traced back to the entry point in column \(n\) as the starting drawpoint.

The DP algorithm of Riddle is a multi-section 2D solution for 3D problems. The
method provides an optimum stope in 2D section. However, it fails to determine the true optimum stope in three dimensions. Although the sections are optimised, when they are put together, they may violate the stope constraints. As a solution, Riddle (1977) suggested applying the algorithm once in north-south sections and next in east-west sections. Additionally, the algorithm is restricted to block-caving mining method.

2.4.2 Geostatistical Approach

The application of geostatistics to determine the outline of the mineable ore in underground mines has been addressed by Deraisme, de Fouquet and Fraisse (1984). They introduced “downstream” geostatistics to build 2D sectional numerical models of the deposit and to define the outlines of the ore to be mined. Mathematical morphology was used to impose the stope geometry constraints, to the ore-waste images, for transformation of the images. As described by Deraisme et al. (1984), downstream geostatistics was introduced when the linear and non-linear geostatistics failed to estimate the mineable reserves, when the selection of ore is controlled by underground mining constraints.

Deraisme et al. (1984) combined conditional simulation with underground mining simulation to compare: selectivity; productivity; and profitability of three mining methods. Their suggested approach is based on, firstly, constructing a numerical model of the deposit and, secondly, defining the outlines of the mineable ore for the three mining methods. In order to ensure block selectivity, the conditional simulation model was constructed with small blocks of size 1 meter. Figure 2.10 shows one realisation of the geostatistical model. Figure 2.11 shows the geometrical constraints for the three mining methods of the Ben Lomond uranium mine used by Deraisme et al. (1984).
Figure 2.10: Conditional Simulation of Lenses (Deraisme and de Fouquet, 1984)

Figure 2.11: Geometrical Constraints for the Three Mining Methods at Ben Lomond Mine (Deraisme et al., 1984)
Comparative outlines of a section using the Deraisme method and a manual approach are shown in Figure 2.12.

Figure 2.12: Compared Outlines for the Cut-and-Fill Method on a Section
(Deraisme et al., 1984)

2.4.3 Octree Division Algorithm

Cheimanoff, Deliac and Mallet (1989) described a prototype production scheduling tool, BONANZA, as part of the GEOCAD package, for the development of geological resources to mining reserves. Features of CAD and artificial intelligence modules were used to develop a rule-based system to generate the shape of the mineable ore while imposing underground mining constraints. Their program performs the octree space division algorithm to remove non-desired mining blocks based on the minimum stope size.

The GEOCAD package was originally aimed at the development of a computer based system to assist both the geologist and the production or planning engineer. As stated by Cheimanoff et al. (1989), BONANZA is designed in three steps. The first step involves:
gathering data, including bore-hole data or geological underground sampling (hard data); results from the geostatistical analysis or interpreted contours supplied by the geologist (soft data); and intuition from the geoscientist or the mining engineer regarding the shapes of geological objects (expert data). These data are used to build a geometrical model. The second step is transforming the modelled geological resources into mineable reserves. In this step, different production factors are simulated to build possible workable volumes, using the available geological resource. In the final step, the geological resource is economically evaluated to determine workable reserves and mining sequences.

As described by Cheimanoff et al. (1989), the mine layout is determined using a rule-based programming method. The two main constraints are geometric constraints and economic constraints. Geometric constraints, which limit the dimensions of the working space, are defined on the basis of geomechanical behaviour of the ore-body, mining equipment used and the grade control policy. The economic constraints take into account the cut-off grade and the mining costs including, access cost and the cost of services, such as ventilation. These constraints are not fixed in time and are constantly updated.

Mineable volumes, well adapted to these constraints, are determined through two main modules, the "Object Manipulator" and the "Shape Generator". The "Object Manipulator" module determines geometrically mineralised areas, which later become mineable volumes. The principle is to gather veins in convex blocks, the shape of which is given by the second module (Shape Generator), and to evaluate briefly the payability of the block by examining its grade and volume returned from the Shape Generator module. In order to achieve this, the amalgamation process begins with veins sufficiently large and valuable to justify a mineable block. The process continues by merging close veins into a single block while separating veins, too far from one another, into two different blocks. It then proceeds with evaluating the small veins, which do not have adequate values, if considered alone, until the whole set of veins are examined.
The "Shape Generator" module effectively creates a mineable block from the vein amalgamation created by the Object Manipulator module. This is achieved by surrounding the amalgamation vein with an initial large mineable volume and the successive removal of the sub-volumes (Figure 2.13) using the syntactic pattern recognition technique.

![Figure 2.13: Successive Removal of Sub-Volumes](image)

The octree division is a general form of the binary-tree division technique. The binary-tree division is a 1D approximation of the position of an object, with respect to a defined minimum unit. When the technique is used for 2D problems, a rectangle is drawn surrounding the object and is halved in each dimension, to give an area of four equal parts. This technique is called the quadtree division technique and examines each quarter for further division. The search continues, limiting the selection into smaller and smaller sub-areas until there is no possible smaller sub-area (Figure 2.14). Generalising the problem for 3D approximation, the volume surrounding the object is considered. This volume is halved in each dimension, that is, eight equal sub-volumes (the octree
division) as shown in Figure 2.15.

![Minimum size of the feasible dividable area](image)

**Figure 2.14:** An Example of Quadtree Division

![Octree Division](image)

**Figure 2.15:** Octree Division

Unlike other approaches, the octree division algorithm is not implemented on the well-known block models. Instead, the vein amalgamation, created by the *Object Manipulator* module is represented by a set of (horizontal) sections in space,
surrounded by an initial large mineable volume (Figure 2.16). The operation then starts with a list, containing the initial surrounding volume, “V”. This initial volume is divided into eight sub-volumes, therefore, the volume “V” is replaced by the list of its eight sub-volumes.

![Diagram of vein surrounded by initial mineable volume](image)

**Figure 2.16:** An example of Vein Surrounded by an Initial Mineable Volume, Resulting from the *Object Manipulator* (Cheimanoff *et al.*, 1989)

Knowing that the geometric constraints are translated into the minimum allowable dimensions of the workable space, each sub-volume in the list is then evaluated on the following three conditions. A decision is then made on whether the sub-volume should be removed, divided or kept in the final mineable block.

1. A sub-volume, which contains no part of mineralised veins and satisfies the constraints, can be removed from the list. The new list would, then, be the remainder of the previous list.

2. The sub-volume lies completely within the mineralised veins or it has the minimum allowable dimensions, therefore, it has to be stored in the future
mineable block. The new list would again be the remainder of the previous list.

3. The sub-volume is not totally outside the mineralised veins nor completely included in them (that is, it is partly located inside the ore and partly outside the ore) and it does not have the minimum allowable dimensions. Therefore, it has to be divided into eight new equal sub-volumes. The new list would be the result of appending eight new sub-volumes to the remainder of the previous list.

The decision made for each sub-volume, updates the list of sub-volumes to a new list. The evaluation process is then repeated for the new list. The algorithm is ended when there is no sub-volume in the list, that is, all sub-volumes have been evaluated. Figure 2.17 illustrates the successive evaluation steps of the *Shape Generator*, implemented on a cut-and-fill mining example.

The octree division algorithm, embedded in BONANZA, attempts to solve the problem of moving from geological resources to mineable reserves and provides a 3D solution to the optimum stope geometry definition. The algorithm does not consider sub-volumes jointly, which causes more waste to be included in the final mine layout. For example, consider two adjacent sub-volumes, which have the minimal dimensions and are both stored in the final layout (Figure 2.18). The top sub-volume contains more waste than ore, nevertheless, it is included in the mine due to the minimal dimensions (satisfying the second condition in the evaluation process). However, if it is evaluated jointly with the bottom sub-volume, the amount of the included waste is reduced while still satisfying the minimum stope size. This problem occurs since the minimum unit allowed, in the optimisation algorithm, equals the minimum stope size (that is, partial stopes are not allowed in the optimisation). The minimum unit allowed, in other algorithms that use a block model, equals the block size (if these algorithms do not allow partial blocks in optimisation). Since the block size is usually several times smaller than the stope size, the block model based algorithms may represent smaller
variations in the optimal layout.

Figure 2.17: Successive Sub-Volume Evaluation Steps Performed by the Shape Generator, on a Cut-and-Fill Method (Cheimanoff et al., 1989)

Figure 2.18: Joint Consideration of Sub-Volumes
2.4.4 Floating Stope Algorithm

The Floating Stope algorithm is a tool developed, by Datamine, to define the optimal (boundary) limit for mineable ore, or stope envelope, that may be economically extracted by underground stoping methods. Alford (1995) outlined the general concept of the Floating Stope approach and described it as a heuristic approach. It is analogous to the Moving Cone method of open pit limit optimisation. Typical applications of the Floating Stope technique include: preliminary resource appraisal; selection of the stoping method; and detailed mine design.

The approach is implemented for a fixed block model of the ore-body. The geology and mineralisation are modelled from 2D sections into the regular (or sub-cell) block model. A cut-off grade is specified to discriminate between ore and waste blocks. A target head grade is also specified for the stope. The main constraint is the geometry of the stope, which is translated into the minimum stope dimension in three orthogonal directions. The problem is then to determine if any block, above the specified cut-off grade, can be included in a stope that meets a nominated head grade. In many cases, there are several alternative stopes, therefore, the block is taken in the stope with the highest head grade.

The term Floating Stope is derived from the technique of floating a stope shape, of the minimum stope dimensions, around any block to locate the stope position of the highest stope grade. The stope is forced to float around the block, relative to the origin, with specified stope float increments in the three orthogonal directions. A cross-sectional analysis can be obtained if the float increment is made equal to the section spacing in that direction. When the minimum stope size is not a multiple of the block size, fractions of the edge blocks are included in the stope. In these cases, use of the sub-cells in the block model allows the inclusion of the partial blocks. The process of floating the stope shape can lead to the definition of two separate envelopes, of which the first is a subset of the later. These include "inner" and "outer" envelopes, as described by Alford (1995):
1. An "inner envelope" may be defined from all of the blocks above the cut-off grade that can be mined. It is the union of the best grade stope shapes.

2. An "outer envelope" may also be defined from all of the blocks above the cut-off grade that can be mined. It is the union of all possible stope positions for each block.

Using a 2D example, consider a section of blocks with a $4 \times 3$ metre block size, in which minimum stope dimensions are $15 \times 10$ metres and the float increment is 5 metres. Figure 2.19 shows the "inner" as well as the "outer" envelopes for a single block, $B_y$. Clearly, the "outer" envelope suggests a greater percentage of the total tonnes. However, it is the "inner" envelope that provides higher average grade ore. The additional ore, with the lower grade, is taken in the "outer" stope envelope. This may result in final average falling below the nominated head grade. The final stope design will be decided by an experienced engineer taking the practical design into considerations. In practice, the final stope design should be as close to the "inner" envelope as practical and fall within the "outer" envelope.

![Figure 2.19: The "Inner" and the "Outer" Envelopes for a Single Block](image-url)
Both the Floating Stope technique and the Moving Cone method have different limitations. The Moving Cone technique examines cones independently. In fact, the mutual support between blocks are ignored in the Moving Cone method. As a result, two different cones may individually be uneconomical, but when they are jointly considered, the union of the two cones may be profitable. The Floating Stope approach has the opposite limitation. That is, two overlapping best stopes, when considered individually may be economical due to a (number of) shared high grade block(s). However, when they are considered jointly, the union of stopes may be uneconomical (Alford, 1995). Figure 2.20 illustrates the above comparison.

In general, the method for selection of the best stope position is dictated by the optimisation objective. Different optimisation objectives offered by the Floating Stope algorithm include: maximising ore tonnes (minimising below cut-off waste); maximising grade; maximising contained metal; and maximising the accumulated (dollar) value. The output of the Floating Stope process will be a block model, in which the cell sizes have been specified by the float increments. The output block model defines the volume of the stope envelope as well as the grade values of the best stope.

The most significant advantage of the Floating Stope algorithm is its simplicity. It provides a three-dimensional evaluation (unlike the previous 2D algorithms) over the optimisation and sensitivity analysis for mineable reserves and the stope geometry. The algorithm benefits from the generality, that is, it is not specialised for a certain mining method (for underground metalliferous mines). A commercial software package has been developed for the Floating Stope algorithm, but it is a heuristic approach and lacks rigorous mathematical support. Most importantly, the Floating Stope algorithm does not determine the optimum stope, regardless of being true or non-true optimum. The algorithm only restricts the optimal stope boundaries between two envelopes, that is, the "inner envelope" and the "outer envelope", without defining the exact location of the stope.
Chapter Two: Ultimate Mine Design Methods

The value of the combined cone = \( C_1 + C_2 - (-3) = 1 \)

Cone 1 and Cone 2 are, individually, uneconomic, but the combined cone is economic.

The value of the joint stope = \( S_1 + S_2 - (-1 + 2 + 5 + 4) = -1 \)

Stope 1 and stope 2 are, individually, economic, but the joint stope is uneconomic.

**Figure 2.20:** Limitations of the Moving Cone and Floating Stope Methods

### 2.4.5 Branch and Bound Technique

The application of the branch-and-bound approach to determine the stope layout was proposed by Ovanic and Young (1995). They developed the optimum stope boundary by optimising the starting and ending locations of mining, within each row of blocks. In order to determine these two locations, two piecewise linear cumulative functions were used for each row. They applied a mixed integer approach, known as "Type-Two
Special Ordered Sets”, for the optimisation of their stope boundary model.

The “Type-Two Special Ordered Sets” (SOS2) technique, referred to as separable programming, is used to optimise piecewise linear functions. An SOS2 is defined as an ordered set of special variables for which the solution allows at most two variables to be non-zero. If two special variables are allowed to be non-zero, they must be adjacent (Creegan and Monforte, 1990).

Consider a 2D non-linear function \( y = f(x) \), which is linear (or can be approximated to be linear) between two adjacent discrete points (that is, a piecewise linear function, as shown in Figure 2.21). The reference value (the value of “\( x \)” in this case) is monotonic. The solution, which is one of the points on the curve \( (x,f(x)) \), shown in Figure 2.21, can be expressed in terms of the influence of the discrete points (of the curve). If the solution is on the line between the two discrete points, the solution is then influenced by those two adjacent discrete points (other discrete points do not have any influence on the solution). Alternatively, if the solution is on a discrete point, the solution is influenced only by that point. In this regard, each discrete point on the curve, \( (x_i, f(x_i)) \), is assigned a special variable, \( \lambda_i \), which indicates the weight of that point in the solution. Therefore, there is a set of special variables, \( \lambda_i \), such that:

\[
\begin{align*}
0 & \leq \lambda_i \leq 1 \\
\text{and} \\
\sum \lambda_i & = 1
\end{align*}
\]

The two possibilities for the solution can then be described as:

1. The solution is one of the break (discrete) points, in which case the weight of that point, \( \lambda_i \), is equal to one and the weights assigned to other points are equal to zero.

2. The solution is a point (on the curve) between two adjacent break points. In
this case, the weights of these two points are non-zero and sum to one. Those of all other points are equal to zero. The exact location of the solution, therefore depends on the weights of the two adjacent points and can be represented by \((x_i \lambda_i + x_{i+1} \lambda_{i+1}, f(x_i \lambda_i + x_{i+1} \lambda_{i+1}))\).

![Typical Piecewise Linear Approximation Function](image)

**Figure 2.21:** Typical Piecewise Linear Approximation Function

The algorithm of Ovanic and Young (1995) is an attempt to incorporate the "Type-Two Special Ordered Sets" (SOS2) into a mathematical model for stope optimisation. The outline of the stope can be determined by optimising its starting and ending locations. Therefore, two separate sets are defined for the two end points of the stope. Assume a row of blocks with their associated block positions and block net values (Figure 2.22a). The objective is to find the optimum stope boundaries, which maximises the block attributes (economic values). The real position of blocks along the panel, is strictly increasing and may be considered as the reference value in the piecewise linear function. Block economic values are considered to be attribute values, however, the cumulative values of the blocks have to be used. The discrete points are located at the beginning or the ending points of the blocks. The number of discrete points, therefore equals the number of blocks, plus one. Figure 2.22b shows a typical cumulative block value function.
The properties of the "Type-Two Special Ordered Sets" (that is, special variables are bounded between zero and one; the sum of variables in each set is 1.0; no more than two variables from each set can appear in any solution; and if two appear, they must be adjacent) are used to ensure that the stope starting (ending) point is located on the discrete points of the cumulative block value function. When only one special variable appears in the solution, the solution lies at a breakpoint on the cumulative function (at the edge between two blocks). Alternatively, when two special variables appear in the solution, the solution occurs at a point between two breakpoints on the cumulative function within a block (Ovanic and Young, 1995).
Considering that the mining proceeds from left to right and starts somewhere within the panel, two separate $SOS2$ sets are generated. The first set defines the leading boundary (ending point) and is identified by the special variable "$L_i$", while the second set defines the trailing boundary (starting point) and is identified by the special variable "$T_i$". The objective function is then defined as the difference between the two cumulative functions and corresponds to the leading and trailing boundaries, as given by Equation 2.8.

\[ \text{Maximise } SV = \sum_{i=0}^{n} a_i L_i - \sum_{i=0}^{n} a_i T_i \quad (2.8) \]

where

$SV$: the difference between the cumulative values obtained for the leading and trailing boundaries (stope value),

$a_i$: the $i^{th}$ attribute value (cumulative block economic value),

$L_i$: the $i^{th}$ leading special variable, bounded between 0 and 1,

$T_i$: the $i^{th}$ trailing special variable, bounded between 0 and 1.

This maximisation is subject to a number of constraints. Firstly, a relationship between the special variables and the block position is required. This relationship is established by a special set of constraints, known as reference rows, as given by Equation 2.9.

\[ \begin{align*}
\sum_{i=0}^{n} x_i L_i - B_i &= 0.0 \\
\sum_{i=0}^{n} x_i T_i - B_i &= 0.0
\end{align*} \quad (2.9) \]

where

$x_i$: the $i^{th}$ block position (reference value)

$B_i$: the leading boundary location variable

$B_i$: the trailing boundary location variable

The interpretation is that for an optimal solution, $B_i$ represents the real position of the optimal leading boundary and $B_i$ represents the real position of the trailing boundary.
Another special type of constraint, called the convexity row, is still required for each SOS2 set. This constraint requires that the special variables in any set sum to one, as expressed in Equation 2.10.

\[
\sum_{i=0}^{n} L_i = 1.0 \\
\sum_{i=0}^{n} T_i = 1.0
\]  

(2.10)

The remaining constraints refer to the geometric limitations of the stope that is to be bounded between a minimum and a maximum size. For example, the ground control considerations may limit the maximum stope dimensions, while the size of the selected equipment may control the minimum stope dimensions. Consider the stope within the above row of blocks where the difference between the real position of the starting and ending of the stope (that is, the stope length) is limited, as stated in Equation 2.11.

\[
\text{MinimumValue} \leq B_i - B_j \leq \text{MaximumValue}
\]  

(2.11)

To apply the "Type-Two Special Ordered Sets" (SOS2), mixed integer programming, via a branch-and-bound algorithm, is used. When an SOS2 is selected for branching, the branching algorithm computes the sum of the special variables \((L_i, T_i)\) in the set, multiplied by their reference values \((x_i)\). The subsequent branching step is set up in consideration of the two subsets. The first subset refers to all the variables from the beginning of the set, up to the first variable whose reference is greater than the computed value. The second subset refers to the variables from the end of the set, down to the first variable whose reference is less than the computed value (Creegan and Monforte, 1990). A lower bound and an upper bound, which indicate the first and the last member of the set that may be non-zero, are established. The non-zero members are forced to be selected from the subset of interest. The branch-and-bound algorithm limits selection from a smaller and smaller subset until it forces an integer (discrete) solution.

In the above example, assume that the stope is constrained to a minimum length of 33
1/3 meters. Applying the branch-and-bound algorithm described above, the optimum solution for the stope optimisation is shown in Figure 2.23. There are two special variables corresponding to the leading boundary \((L_4 \text{ and } L_5)\), that is, one third of the block number 5 is included in the stope. In addition, there is only one special variable corresponding to the trailing boundary \((T_3)\). They clearly satisfy all the conditions required for the SOS2 set. The stope, therefore, begins 75 m from the origin \((x_3T_3 = 75)\), ends at the point 108.33 m \((x_4L_4 + x_5L_5 = 108.33)\) and contains 113.33 block value units. An application of the algorithm to an Iron ore mine has been reported by Ovanic and Young (1999).

![Stope boundary positions](image)

**Figure 2.23: SOS2 Solution Example**

The branch-and-bound algorithm removes some restrictions from the mine layout design, and in particular, the stope geometry optimisation. There is no restriction for the blocks to be treated as a whole, but rather partial blocks can be included in the optimal stope. Blocks are not limited to regular or uniform shapes. Their shape or size does not influence the optimisation, since the block cumulative value function is used in the modelling. This is especially helpful when the geological interpretations are overlaid on the block model. The blocks may be shaped to follow the geological variations and discontinuities interpreted by the geologists. The algorithm, however, has been developed to optimise the stope boundary along the panel of the blocks in one dimension. It is not clear if any smoothing is required for the adjacent panels in two and three dimensions. The algorithm works in special cases, where the geometry constraints can be reduced to the stope length. In addition, the algorithm does require the use of
specialised mathematical optimisation software (Thomas, 1996).

2.5 ORE-BODY BLOCK MODEL

There are a number of choices for the computer modelling of an ore-body. Parameters that have to be considered in selecting a model include: the type of deposit; the commodity being modelled; the value of interest; the complexity of the geological deposit; and the user's acceptance and familiarity with the various modelling techniques (Badiozamzni, 1992). However, the computerised algorithms for the optimisation of the mine geometry, in both open pit and underground mining, are usually implemented on a block model of the deposit. A block model is the collection of a three dimensional set of blocks of given X, Y and Z dimensions stacked on top of one another, which represent a model of a property to a given depth. Each block may be identified by the X, Y and Z co-ordinate at the centre of the block and contains the percentage values for each item of interest, such as the product grade, the waste, the density and so on. The two major alternatives for the block model include the cross-sectional model and the gridded model. Each of these models is used for specific conditions and specific mining operations. The block model is usually used for disseminated deposits such, as porphyry copper, uranium, gold and other non-stratabound deposits. The gridded model is normally used for bedded deposits such as coal, phosphate, sulphur and limestone. The cross-sectional model is commonly used for complex folded and faulted, or steeply dipping deposits (Badiozamzni, 1992).

The first step in the process of the computer modelling of an ore-body is to gather the geological data. This basic data is the minimum information required for the modelling of the ore-body and consists of a set of information that identifies the drill hole location, the geology and the assay or analytical data (Badiozamzni, 1992). Then a geological interpretation of the deposit is made. In order to construct a block model, a cubic block (box) is placed around that geological interpretation (Figure 2.24). This box should be large enough to encompass the area of interest. The large box is then sub-divided into a group of small regular (or irregular) 3D blocks along each dimension (Figure 2.25).
Finally, each block has to be assigned a set of estimated values which represent the corresponding geological unit. Mathematical methods are applied to estimate these values for each block using data collected from the diamond drill holes. Figure 2.26 (Noble, 1992) shows an example of the coding a geological unit into a block model.

**Figure 2.24:** The Ore-Body Surrounded by a Large Box

**Figure 2.25:** The Ore-Body Surrounded by the Physical Block Model
2.5.1 Types of Block Model

There are various types of block model used in mines, including: regular 3D fixed block model; 3D variable block model; 2D irregular block model; and 3D irregular block model. However, the regular 3D fixed block model is the most widely used in practice. This type of model was first published by Kennecott Copper Corporation in the early 1960s. It was designed for massive porphyry copper deposits and is best suited for that type of deposit, although it can be used for bedded deposits as well (Kim, 1978). In the regular 3D fixed block model, blocks are divided into ore and waste, based on the determined cut-off grade. An irregular block model is usually used for irregular and spotty mineralisation defined by the polygons on each level (Kim, 1978).

In cases where the separation of lithological units, as well as geological facies are necessary, the use of fixed block models is not efficient. Therefore, some variations have been suggested to modify the block model, including the "variable block model" and the "variable zone model". In a variable block model, the X and Y dimensions are fixed and the Z dimension is variable to better approximate the geological variability of the deposit (Banfield and Wolff, 1979). Although this modification allows the thickness
to vary from one block to another, blocks do contain equal thickness sub-zones and form rectangular blocks of different size (Figure 2.27b). The variable zone modelling approach was then suggested by Badiozamani and Roghani (1988). It allows for variation in the thickness of the sub-zones, which results in blocks of trapezohedron shape (Figure 2.27c).

Nevertheless, the fixed block model is still the most popular model used in the optimisation of mine geometry. In a regular 3D fixed block model, as seen in Figure 2.25, blocks are supposed to be contiguous in all directions, that is, there is no gap
between the blocks of a model. An exact number of blocks should fit into the model in all three dimensions. In other words, the extent of the model in each direction is a multiple of the block size in that direction. Otherwise some corrections are needed to adjust the extent of the model, or the block size, in the specified direction.

The final output of the mine layout optimisation algorithms, is normally a list of blocks that are selected by the algorithm for inclusion in the final pit or stope. It is necessary therefore, to identify each block for referencing in the optimisation formulae. Generally, the Cartesian system, with three dimensions, is used to identify the model blocks. That is, the X, Y and Z co-ordinates of the block centre. However, in a regular 3D fixed block model, blocks can also be counted along each dimension. It is often more convenient to address a block by its sequential order in the three orthogonal directions. Considering there are \( I \), \( J \) and \( K \) blocks in X, Y and Z directions, respectively, the block model can be represented by an \( I \times J \times K \) matrix with the typical entry, \( B_{ijk} \), where \( i \), \( j \) and \( k \) indicate the sequential order of the block in X, Y and Z directions, respectively.

### 2.5.2 Block Size

The selection of the appropriate block size is crucial in the generation of models, which have to represent reality. It is a balancing act between generating blocks that are not too large or too small. If the blocks are too large, they may lose the masking variations of the data. On the other hand, generating blocks that are too small increases the number of blocks. This, in turn, increases the computer time used and the resources needed (Badiozamani, 1992). However, with modern computers, it seems that there is no significant restriction in generating small blocks. It should be noted that it is not efficient to generate blocks smaller than that required.

In general, the optimum block size is primarily dependent on the mineralisation geometry and the primary reason for creating the model (Brew and Lee, 1988). The factors that must be considered in determining the optimum block size include: the size of the resulting model; drill-hole spacing; geological controls; and the mining method.
Smaller block sizes provide minimal improvement in the estimation, unless strong geological controls are present. Block size may be related to a proposed mining method. For example, the vertical height of the blocks is usually the same as the bench height of a deposit that will be mined by open pit methods (Noble, 1992).

Whittle (1989) has provided four categories of block sizes based on the purpose for the use of the block model. These include:

1. **Ore-body modelling**: Blocks have to be small enough to outline the ore-body clearly. This may lead to block models consisting of millions of blocks.

2. **Block valuation**: When the objective is to calculate the net economic value of blocks, the block size has to provide the minimum volume which can be selectively mined.

3. **Ultimate mine design**: If the final mine layout occupies a substantial part of the full (rectangular) model and if the final mine layout (not the ore-body) is reasonably regular, the optimisation of the full model of 100,000 blocks is usually sufficient for design work.

4. **Sensitivity studies**: For sensitivity analysis, a detailed design is not required as the comparison among the results of the different values of a certain parameter is important. In these cases, a model of 10,000 to 20,000 blocks is sufficient.

Badiozamani (1992) has suggested two rules of thumb for selecting the block size. The first is that the best results are obtained when each grid cell contains only one drill-hole. The second is to adjust the grid spacing to the type of usage. A large grid spacing is acceptable for a cursory evaluation, whereas a detailed mine plan requires dense grid spacing.
2.5.3 Estimation Process

Each block within the model contains estimates of the parameters such as: percentage values for each product; density; rock type; and other characteristics of the ore. For example, in a porphyry copper deposit, a block may contain the rock type and density of the block, as well as the percentage of copper, zinc, lead, waste and air. However, the one that is the most important for use in the algorithms for the optimisation of the ultimate mine is the grade(s) of the product(s). What is available for an ultimate mine designer is a set of irregularly spaced data distributed in drill-holes. This means that the estimation process involves the application of a series of mathematical algorithms to interpolate and extrapolate from this set of data to the centre of the block. The estimation techniques vary from a simple triangulation, to polygonal approximation - where an area of influence is assigned to each drill-hole - to much more complex algorithms such as the trend-surface analysis, the Fourier series and geostatistics (Badiozamani, 1992). The application of artificial neural networks, in combination with geostatistical methods, has also been proposed more recently (Cortez, Sousa and Durao, 1998).

The estimation techniques are generally divided into two categories: traditional geometric techniques and computer based interpolation techniques. Traditional geometric methods, such as area averaging and cross-sectional methods, are based on the geometric weighting of assays. These techniques are implemented manually on plan maps, or cross-sectional maps, that cut the deposit into sets of parallel slices. The geometric techniques have the advantage of simplicity and ease of implementation. However, they may imply more selective mining than may be achieved by the mining method. High-grade blocks usually include lower-grade material when they are mined, and low-grade blocks usually include some higher-grade material (Noble, 1992).

Moving average methods are the most widely used procedures for computer assisted resource estimation. They include inverse-distance weighting and kriging techniques. In both moving average techniques, the grade, or other geological parameters of each block, is estimated by searching the database for the samples around the block and
computing the weighted average of those samples (Noble, 1992). The weighted average is computed using Equation 2.12:

\[ g^* = \sum_{i=1}^{n} w_i \cdot g_i \quad i = 1, 2, 3, ..., n \]  

(2.12)

where

\( g^* \): \quad \text{the estimated grade},

\( g_i \): \quad \text{the grade of sample } i,

\( w_i \): \quad \text{the weight given to sample } i \text{ and}

\( n \): \quad \text{the number of samples selected.}

The inverse distance weighting technique is based on the fact that closer samples have more influence on the estimate than those further away. Therefore, the weight function may be defined to assign the weight of each sample, as expressed in Equation 2.13.

\[ w_i = \frac{1}{d_i^p} \quad \sum_{i=1}^{n} \frac{1}{d_i^p} \]  

(2.13)

where

\( d_i \): \quad \text{the distance between the location being estimated and sample } i \text{ and}

\( p \): \quad \text{the inverse distance weighting power.}

The linear inverse distance uses power one for the weight function, however, a higher degree for the weight function is used to decrease the influence of the furtherest samples at a faster rate. The inverse squared distance, that is, the weight function of power two, is widely used in grade estimation.

The most popular technique for grade estimation is kriging, which is developed to provide the best linear unbiased estimate. Kriging is a geostatistical method that is based on the least squares minimisation of the estimation error. Geostatistics, also, considers the different weights for samples in estimation, however, it formulates the distance parameter into a variogram function, which represents the variation of grade (or any
quality) over its distance. When there is a correlation between the sample values and their spatial position, which is the case in mineral deposits, geostatistics provide the best estimation for the unknown points. The mathematical solution to the ordinary kriging estimate problem is based on the following two important factors (Noble, 1992):

1. The average of the estimates should not be systematically higher or lower than the true value; this is established mathematically by setting the sum of the weights equal to one, as expressed by Equation 2.14. This requirement provides an unbiased estimate.

\[ \sum_{i=1}^{n} w_i = 1 \]  
(2.14)

2. The error of estimation, \( \sigma_k^2 \), known as variance \( (G - G^*) \), which is expressed by Equation 2.15, is minimised.

\[ \sigma_k^2 = \sigma_{B,D}^2 - 2 \sum_i w_i \sigma_{B,Xi} + \sum_i \sum_j w_j w_i \sigma_{X_i,X_j} \]  
(2.15)

where

- \( G \): the true grade,
- \( G^* \): the estimated grade,
- \( \sigma_{B,D}^2 \): the variance of the block size being estimated in the deposit,
- \( \sigma_{B,Xi} \): the covariances between each sample and the block being estimated and
- \( \sigma_{Xi,Xj} \): the covariances between the individual samples.

### 2.6 ECONOMIC BLOCK MODEL

What is available in a block model, is a set of blocks in three dimensions, each of which contains estimates of a set of data, most importantly assay values. Assay values are useful in discriminating between blocks of ore and waste and are based on a given cut-off grade. However, for mine layout optimisation, it is necessary to express the blocks
in economic terms so as to indicate their net worth, that is, their dollar values. The reason for this is that blocks with the same grade value may have a different net worth, that affects their mineablity and the optimum mine layout. Some factors that influence the net value of blocks include: the location of the blocks; when they are to be mined; and the applied mining method.

The relative location of a block may affect its net value as the haulage distance is influenced by the block location. The effect is not considerable for small differences in block locations. However, for blocks that are located far from the dump site or the crusher, it may be significant. In particular, the depth of the mine can be divided into different categories, each specifying a separate cost for haulage.

The block net value is also affected by when the block is to be mined. The revenue obtained from a block depends on the price of the recovered (metal) product contained in the block. The product price is usually considered to be the main economic uncertainty over time. The amount spent for the associated cost of blocks, including the payment for equipment, materials and wages vary with time. The inflation rate and the time value of money must be taken into account as the revenue and costs are discounted by a factor that increases over time. This means that the value of a dollar, obtained from mining a block earlier, is different from (higher than) the dollar value, which may be obtained from mining the same block later (Whittle, 1990).

Various mining methods may also influence the economic value of a block. The value of a block, when excavated using open pit mining methods, is not necessarily the same as when it is mined using underground methods. In addition, the block values may vary with different underground methods. For example, the mining costs for a block using a selective mining method, such as cut-and-fill, may be different from those for the block mined in the block-caving, or sub-level stoping, method. In the block-caving method, there is no cost for ore extraction (except occasional drilling for initiating the caving process) nor ore handling cost (since the ore falls down due to its gravity). However, development in the block-caving method is complicated and time consuming, so that it
may take years to complete the development (Hamrin, 1982). On the other hand, in
the cut-and-fill method, there is no cost for development and the selectivity of the
method provides good recovery. However, the cost of mining entails drilling, blasting,
filling and handling ore to the orepass within the stope.

In mine layout optimisation, it is common practise to use the Block Economic Values
(BEV) as attributes of the blocks. The corresponding model is called the economic
block model. An economic block model is a block model, where each block is assigned
an estimate of its net economic (dollar) value. The typical element of the economic
block model is denoted by $BEV_{ijk}$, which is a real scalar number and represents the
economic value of the block, $B_{ijk}$.

### 2.6.1 Rules of Thumb in Calculating Block Values

When calculating block values for optimisation purposes, basic rules must be followed.
Whittle (1989) has suggested three rules of thumb, in this regard, as presented below.

1. The value must be calculated based on the assumption that the block has *already*
   been uncovered. That is, the cost required to access the block must not be included
   in the block costs.

2. The value must be calculated based on the assumption that the block *will* be mined.
   A block, which contains more waste than ore is not going to be primarily chosen for
   the optimal layout. However, if it has to be mined to satisfy the mining constraints,
   the ore content will pay for some of the included waste.

3. When considering the cost of mining, or the cost of processing blocks, only those
costs that would stop if mining stopped are included. For example, fuel costs and
wages would stop if mining stopped and therefore, must be included in the
corresponding cost of mining, processing or refining. The reason is that the addition
of each extra block to the mine layout extends the life of the mine. Therefore, that
extra block should pay for the extra cost during the extra life of the mine (Whittle, 1990).

The assumption, made in the first rule, is true for open pit mining since the cost of accessing a block has in fact, been paid already when calculating the values of the preceding blocks. In other words, uncovering a block is equivalent to mining its preceding blocks. The block cannot be mined directly without mining its preceding blocks. This means that when a block is to be mined, it is already uncovered and no extra cost is required. However, in underground mines, accessing a block does not mean uncovering that block. That is, each block must contribute to the accessing cost, including costs required for shafts, inclines, underground roadways and so on.

2.6.2 Block Valuation

There are various formulae for calculating the economic value of a block. However, the approach used in this study, is based on Equation (2.16).

\[
BEV_{ijk} = BRR \ g_{ijk} - BMC
\]  

(2.16)

where

\(BEV_{ijk}\): the economic value of the block, \(B_{ijk}\), in $,

\(g_{ijk}\): the grade of the metal estimated for the block, \(B_{ijk}\), in "%" or "ppm",

\(BRR\): the block revenue ratio, a term obtained using Equation (2.18) and

\(BMC\): block mining costs, a term obtained using Equation (2.17)

\[
BRR = (P - C_M) \ r \ V \rho
\]

\[
BMC = C_{ore} \ V \rho
\]  

(2.17)

where

\(P\): the price of the (metal) product to be sold, in $/t of the metal,

\(r\): total proportion of the metal recovered form the ore, including mining,
processing and refining recovery,

\[ V: \text{the volume of the block, } B_{jk}, \text{ in } m^3, \]

\[ \rho: \text{the density of blocks, in } t/m^3, \]

\[ C_{ore}: \text{the ore-based costs (the cost of mining a tonne of rock (ore or waste), in } $/t \text{ of rock) and} \]

\[ C_M: \text{the metal-based costs (other costs required for processing a tonne of metal, refining it and preparing it for sale, in } $/\text{tonne of the metal).} \]

The above formula uses the grade of only one product, usually there is some by-products in the deposit as well. In multi-product deposits, the main product is set as the base and an equivalent grade is defined and calculated. This substitutes for grades of all existing products. The "main product equivalent grade" (MPEG) is then used in the above valuation formula. Assuming that there are "n" by-products besides the main product, the equivalent grade can be calculated through Equation 2.18.

\[ MPEG = g_o \left( 1 + \sum_{i=1}^{n} EF_i \right) \quad (2.18) \]

where

MPEG: the main product equivalent grade,

\[ g_o: \text{the grade of the main product and} \]

\[ EF_i: \text{the equivalence factor for the } i^{th} \text{ by-product.} \]

Details of the reasoning for both the block valuation and the equivalent grade formulae, together with a simple example, have been provided in Appendix A.

### 2.7 SUMMARY

Mine geometry optimisation is important in mining design as it greatly influences the mine economy and is the basis of production scheduling. There are many algorithms available for the optimisation of the ultimate pit limits, while few approaches have been developed to optimise stope geometry. Open pit algorithms are not applicable to underground mines because of the specific constraints associated with each of them.
The methods used for underground optimisation have been tailored to a specific mining method, lack rigorous mathematical proof or fail when implemented on 3D problems. None of the underground algorithms guarantee the true optimal solution for the stope layout and no comprehensive mathematical solution has been proposed.

Algorithms for the optimisation of the mine layout are usually implemented on a computer model of an ore-body. The modelling choices are extensive, however, the most widely used is the regular 3D fixed block model. The fixed block model is a 3D set of contiguous spatial blocks, which represent a model of a property to a given depth. Each block is identified by the 3D co-ordinates of its centre, or by its sequential order in the three directions. It contains an estimate for each item of interest, particularly the grade, measured for the real ore-body within the drill holes. The major types of block model were discussed, the process of constructing a 3D fixed block model was briefly described, parameters influencing the selection of the block size were reviewed and the estimation methods used to in valuing each parameter of the ore-body of a block were outlined.

The next chapter introduces a heuristic algorithm, which has been developed in this thesis, to determine the optimum stope geometry. The algorithm is implemented on a 3D fixed economic block model and benefits from its simplicity in both its concept and computer implementation. The algorithm provides a 3D analysis of the optimisation problem.
CHAPTER THREE

MAXIMUM VALUE NEIGHBOURHOOD (MVN) ALGORITHM

3.1 INTRODUCTION

This chapter introduces a heuristic algorithm, termed the "Maximum Value Neighbourhood" (MVN), which was developed in this thesis to optimise stope boundaries. The algorithm uses a fixed economic block model of an ore-body, and searches for the best combination of blocks to provide the maximum profit, while imposing certain geo-technical and mining constraints. The mining constraint, considered in this thesis, is the minimum stope dimensions in three principal directions. The MVN method uses the concept of the "order of neighbourhood" in each orthogonal direction. A set of sequential blocks, that should be mined to satisfy the minimum stope dimensions, defines the "neighbourhood" (NB) for a given block. The neighbourhood with the maximum net economic value, qualifies for inclusion in the final stope.

3.2 BASIC SPECIFICATIONS OF STOPE GEOMETRY

There are basic differences between the specifications of an open pit and those of a stope. These differences influence the optimisation of the mine geometry in both the
open pit and underground mines. The specifications may include: the shape; number of possible choices; and block dependence.

**Shape:** The shape of the mining space required to extract a given block, in open pit mines, is approximately an inverted cone. In the case of open pit mining, geo-technical factors, such as the type and strength of the rock, control the stability of the proposed pit. These factors will then impose the maximum slope angle allowed for the final pit. In order to extract block \( B_y \) (Figure 3.1a), all blocks inside the shaded inverted cone must be removed to satisfy the maximum slope angle constraint. However, in underground mining, the minimum acceptable size of the stope plays the most significant role in its optimisation. Underground mining of the same block, \( B_y \), (Figure 3.1b) requires that all blocks inside the shaded area be removed to satisfy the stope size constraints. This shaded area could be any rectangle containing block \( B_y \).

**Possible pits/stopes:** In open pit mining, the cone-shaped pit which is based on a given block is, \( B_y \), (Figure 3.1a) is unique. That is, there is only one possible set of blocks in a cone-shaped pit, above that block, which should be removed to allow the extraction of a given block. In underground mining, there is theoretically an infinite number of possible stopes around a block, that should be removed to allow the extraction of the given block. Figure 3.2 illustrates some of the possible choices for mining the block, \( B_y \). Consider that the stopes are restricted to starting at discrete points (locations of blocks), then the number of possible stopes will be reduced to a finite, fixed number of choices.
In Figure 3.2, there are 12 possible stopes for mining the block, $B_{ij}$, if the minimum stope contains four blocks in length and three blocks in height.

![Diagram showing four possible stopes for mining the block, $B_{ij}$](image)

**Figure 3.2:** Four Examples of Possible Choices (stopes) for Mining the Block, $B_{ij}$

**Block dependence:** In the open pit cases, the mining of each block depends on mining the blocks on the levels above that block, within the pit limit. In Figure 3.3a, the mining of each block is dependent on the mining of the three blocks immediately above the block. In underground mining, there is a mutual dependence between all blocks which form the minimum allowable stope size. The mining of each block is dependent on the mining of all the neighbouring blocks contained in the minimum stope size. In Figure 3.3b, the mining of the block, $B_{ij}$, depends on the mining of all 11 neighbouring blocks and vice versa. The two-way arrows represent the mutual dependence of the mining blocks on each other. In essence, no block can be mined unless all other blocks contained in the minimum stope are mined.
Chapter Three: The MVN Algorithm

(a) dependence of blocks in open pit mining

(b) dependence of blocks in underground mining

Figure 3.3: Block Dependence, Open Pit versus Underground Mining

### 3.3 FORMULATION OF THE PROBLEM

The MVN algorithm is implemented on a regular 3D fixed economic block model of an ore-body. Therefore, such a block model should be available, or otherwise constructed, before the algorithm is used. In an economic block model, each cell is assigned a value representing an estimated economic net value, that may be negative, zero or positive. The block economic value is typically denoted by $BEV_{ijk}$ where $i, j$ and $k$ are the spatial addresses of the block (Figure 3.4).
The input data of the MVN algorithm, are those block economic values which represent the profit earned, or loss made, if a block is mined. The most desirable situation would entail the extraction of profitable blocks only (positively valued blocks). However, there are practical mining restrictions in the extraction process which lead to the extraction of blocks with negative economic values. The problem of finding the best combination of desirable and non-desirable blocks that result in the maximisation of profit, may be expressed by Equation (3.1).

\[
\begin{align*}
\text{Objective function:} \\
\text{Maximise } SEV &= \sum_{jk} F_{ijk} BEV_{ijk} \\
\text{subject to:} \\
\text{stope geometry constraints}
\end{align*}
\] (3.1)

where

\(SEV\): total stope economic value,

\(BEV_{ijk}\): the economic value of the block, \(B_{ijk}\),

\(F_{ijk}\): an indicator function showing whether the block, \(B_{ijk}\), is mined or not. It is defined by Equation (3.2).
Chapter Three: The MVN Algorithm

$$F_{ijk} = \begin{cases} 1 & \text{if the block, } B_{ijk}, \text{ is selected,} \\ 0 & \text{otherwise.} \end{cases}$$ (3.2)

3.4 STOPE GEOMETRY CONSTRAINTS

The geometry of the working spaces in underground mines is restricted in each orthogonal direction by both a minimum and a maximum size (length, width or height). There are several factors that control the minimum dimensions of the stope. Physical parameters, such as the geo-mechanical properties of the ore-body and the surrounding rock; the dip; the depth; and the thickness of the ore-body, affect the proposed underground mining methods, which, in turn, impose some practical restrictions on the extraction of desirable blocks. The method of mining may require that adjacent waste blocks be mined with an ore block. For example, the "block caving" method imposes different constraints to the stope geometry from that imposed by a "cut-and-fill" method. In a "cut-and-fill" method, mining is more flexible so that it allows the extraction of the high-grade ore whilst leaving the low-grade material in the stope as fill. The stope, in this instance, may need a height of about four to seven meters (Hamrin, 1982). The height of a block caving stope is controlled by the distance between two consecutive levels, which may be approximately 100 meters. The horizontal cross section of the large block to be caved, that is, the combination of the minimum limits of the stope in X and Y directions, is usually a square with a cross sectional area of more than 1000 square meters (Hamrin, 1982).

The minimum stope dimensions may be imposed by the size of the equipment used for the mining process. The minimum size of the stope must be designed so that sufficient space is provided for the activities of drilling, blasting and loading, as well as the traffic of personnel.

The maximum limits of the stope dimensions are usually dictated by the geo-technical factors. If the size of the underground working space exceeds a certain limit, due to ground control considerations, the stope walls may collapse. In this thesis, the formulation of the stope geometry constraints considered only the minimum stope
The stope size restriction may be evaluated as a one-, two- and three-dimensional problem. When the stope size is restricted in only one dimension, that is, only the length, the width or the height of the stope is restricted to a minimum size, it is a one-dimensional constraint problem (Figure 3.5a). Similarly, when the stope size is limited in two directions, for example, width and height, it is a 2D constraint problem (Figure 3.5b). In practice, the length, width and height of the stope, are restricted to minimum sizes (Figure 3.5c).

Figure 3.5: Stope Constraints: (a) 1D, (b) 2D and (c) 3D Problems
3.5 THE NEIGHBOURHOOD CONCEPT

The minimum stope size, which is the critical restriction on the size of the underground working space, is expressed as a continuous real value in terms of a length unit. For example, the minimum stope length may be expressed as 12.7 meters, which indicates that if any point of the ore-body is to be mined, that point should be accompanied, in extraction, by the surrounding materials within 12.7 meters. Representation of the ore-body with a fixed block model, provides a convenient way to describe the ore-body with a discrete integer value, that is, in terms of the number of blocks. The minimum stope size may also be represented by a discrete integer value, which indicates the number of blocks of the fixed block model that should be mined together. This leads to the definition of the two basic terms of the MVN algorithm, the "neighbourhood" and the "order of neighbourhood".

When considering a one-dimensional stope constraint, a "neighbourhood" (NB) is the set of all the sequential blocks, including the block of interest in the specified dimension, which may be mined to satisfy the minimum stope size requirement. Figure 3.6 shows a row of blocks in seven blocks, that is, \( \{ B_j | j = 1, 2, ..., 7 \} \). If the minimum stope size is 20 m and the block size of the fixed block model is 5 m, then a possible neighbourhood for the block, \( B_4 \), may consist of the set of four sequential blocks, that is, \( \{ B_3, B_4, B_5, B_6 \} \), as shown in Figure 3.6. It is clear that the illustrated neighbourhood is not the only possible neighbourhood for the block, \( B_4 \). One can easily assume that the set of \( \{ B_2, B_3, B_4, B_5 \} \) could be another possible neighbourhood for the same block, \( B_4 \).

In order to formulate the minimum stope size constraint, in terms of the neighbourhood, two terms, the "stope block ratio" and the "order of neighbourhood", are introduced. Assuming that the stope geometry is restricted in length or height only, then the term "stope block ratio" (SBR) may be expressed by Equation (3.3).

\[
SBR = \frac{\text{minimum stope size (length or height)}}{\text{fixed block size (length or height)}} \quad (3.3)
\]
where $SBR$ is the stope block ratio. For example, the stope block ratio of 3.8 suggests that the minimum stope length (height) must be 3.8 times the length (height) of the model block in the same dimension.

![Diagram](image)

**Figure 3.6:** Minimum Stope Size versus the *Neighbourhood* ($NB$)

The stope block ratio is a continuous real function, which may imply the inclusion of the fractions of a block in the minimum stope size. The MVN approach does not allow partial blocks so, the value of $SBR$ must be rounded. Clearly, if the value of the stope block ratio ($SBR$) is rounded down, the minimum stope size constraint will be violated. Therefore, a positive integer value termed the "order of neighbourhood" ($O_{nb}$) is used to represent the rounding up value of the stope block ratio, as expressed in Equation (3.4).

$$O_{nb} = \text{Int} (SBR - 0.01) + 1$$  \hspace{1cm} (3.4)

where

- $O_{nb}$: order of neighbourhood,
- $\text{Int}$: a function that returns the integer part of a real number, and
- $SBR$: the stope block ratio.

In the above example, "$O_{nb}$" would be 4 for ($SBR = 3.8$). In other words, the "order of neighbourhood" represents the size of the neighbourhood set. That is, the total number of sequential blocks, including the block of interest, in the specified dimension which
may be mined to satisfy the minimum stope size requirement.

3.5.1 The Neighbourhood Set

The neighbourhood for any block, described as a set of blocks in a special pattern, is not unique. In other words, there are a number of combinations forming a neighbourhood of a block. These neighbourhoods may be defined precisely, collected and identified in a set. Figure 3.7 shows the possible neighbourhoods of the block, $B_4$, for the neighbourhood orders of 2, 3 and 4. When the order of the neighbourhood is equal to 2, there are two possible neighbourhoods for any block, each of which has two members (Figure 3.7a). For the $NB$ order of 3, there are three possible neighbourhoods, each of which contains three members (Figure 3.7b) while for the $NB$ order of 4, four neighbourhoods may be defined, so that each neighbourhood contains four members (Figure 3.7c).

These neighbourhoods may also be expressed in terms of sets of blocks. The example shown is the block, $B_4$, with the neighbourhood order of 4.

Possible set of NBs for $B_4$ with $O_{nb} = 4$:

$$\begin{align*}
\text{Possible set of NBs} & = \{NB_1, NB_2, NB_3, NB_4\} = \{\{B_4, B_5, B_6, B_7\}, \\
& \{B_3, B_4, B_5, B_6\}, \\
& \{B_2, B_3, B_4, B_5\}, \\
& \{B_1, B_2, B_3, B_4\}\}
\end{align*}$$

or in the general form:

$${\{B_{j+m-m}, B_{j+2-m}, B_{j+3-m}, B_{j+4-m}\}} \quad \text{for} \quad 1 \leq m \leq 4$$

Similarly, it may be induced that for the neighbourhood order of "$l"$, there is "$l"$ number of possible neighbourhoods for a given block, each of which contains "$l"$ members. The above specific neighbourhoods for the block, $B_4$, may be generalised for any block, $B_j$, while expressing neighbourhoods as sets of blocks. Consider a row of blocks with "$n"
columns, that is, \( \{B_j | j = 1, 2, \ldots, n\} \), as shown in Figure 3.8.

\[
\begin{array}{cccccccc}
B_1 & B_2 & B_3 & B_4 & B_5 & B_6 & B_7 & B_n \\
B_j & B_{j-1} & B_{j-2} & B_{j-3} & B_{j-4} & B_{j-5} & B_{j-6} & B_{j-7} \\
B_{j+1} & B_{j+2} & B_{j+3} & B_{j+4} & B_{j+5} & B_{j+6} & B_{j+7} & \ldots \\
\end{array}
\]

**Figure 3.8:** A Typical Row of Blocks with "n" Blocks

If the block model above is replaced by its corresponding economic model, the set of
economic values of all \( "n" \) blocks in the row may be defined as the universal set, termed \( U \). The universal set implies that all other neighbourhood sets would be subsets of \( U \). The universal set may be expressed by Equation 3.5.

\[
U = \{ BEV_j \in R \mid j = 1,2,\ldots,n \}
\]

(3.5)

where

- \( U \): the universal set for all \( "n" \) block economic values in the row,
- \( j \): the column number in the row of blocks,
- \( n \): the total number of columns in the row of blocks,
- \( BEV_j \): the economic value of the typical block, \( B_j \), and
- \( R \): the set of real numbers.

The possible neighbourhoods of any block may be collected in a set, as a subset of \( U \). Assuming that the order of the neighbourhood equals \( "l" \), this set of neighbourhoods has the size of \( "l" \), in which the elements (neighbourhoods) are numbered from one to \( "l" \). Generally speaking, the set of all possible neighbourhoods for any block, \( B_j \), with an order of \( "l" \) may be represented by the term \( NBS_{j,l} \) and expressed as:

\[
NBS_{j,l} = \{ NB_{m,j,l} \subset U \mid m = 1,2,\ldots,l \}
\]

(3.6)

where

- \( m \): the neighbourhood number,
- \( l \): the order of neighbourhood,
- \( NBS_{j,l} \): the set of all possible neighbourhoods for the \( j^{th} \) block with the order of \( "l" \) and \( NB_{m,j,l} \): the \( m^{th} \) neighbourhood for the \( j^{th} \) block with the order of \( "l" \), as defined below.

A summary of all the neighbourhoods for the block, \( B_j \), with different sizes for the order of neighbourhood, \( "O_{nb}" \), is presented in Table 3.1.
Table 3.1: Possible Neighbourhoods of the Block, $B_j$, for different $O_{nb}$

<table>
<thead>
<tr>
<th>$O_{nb}$</th>
<th>No. of NBs</th>
<th>Set of neighbourhoods for the block, $B_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>$NB_1$: ${B_{j}, B_{j+1}}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$NB_2$: ${B_{j-1}, B_{j}}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In general: ${B_{j+1-m}, B_{j+2-m}}$ for $1 \leq m \leq 2$</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>$NB_1$: ${B_{j}, B_{j+1}, B_{j+2}}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$NB_2$: ${B_{j}, B_{j}, B_{j+1}, B_{j+2}}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$NB_3$: ${B_{j}, B_{j}, B_{j}, B_{j+1}}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In general: ${B_{j+1-m}, B_{j+2-m}, B_{j+3-m}}$ for $1 \leq m \leq 3$</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>$NB_1$: ${B_{j}, B_{j+1}, B_{j+2}, B_{j+3}}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$NB_2$: ${B_{j}, B_{j+1}, B_{j+2}, B_{j+3}}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$NB_3$: ${B_{j}, B_{j}, B_{j}, B_{j+1}}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$NB_4$: ${B_{j-2}, B_{j-1}, B_{j}}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In general: ${B_{j+1-m}, B_{j+2-m}, B_{j+3-m}, B_{j+4-m}}$ for $1 \leq m \leq 4$</td>
</tr>
<tr>
<td>$l$</td>
<td>$l'$</td>
<td>$NB_1$: ${B_{j}, B_{j+1}, B_{j+2}, \ldots, B_{j+l-1}}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$NB_2$: ${B_{j}, B_{j+1}, B_{j+2}, \ldots, B_{j+l-1}}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$NB_m$: ${B_{j-(m-1)}, B_{j-m}, \ldots, B_{j+l-(m+1)}, B_{j+l-m}}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$NB_{l',1}$: ${B_{j+(l'-1)}, B_{j+l'}, \ldots, B_{j+l'}, B_{j+l}}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$NB_{l',1}$: ${B_{j+(l'-1)}, B_{j+l'}, \ldots, B_{j+l'}, B_{j+l}}$</td>
</tr>
</tbody>
</table>

As inferred from Table 3.1, the typical member of the set of neighbourhoods, $NB_m$, for any block, $B_j$, with any order of neighbourhood, "$l$", may be defined by the set of "$l'$" sequential blocks in ascending order, starting from the block, $B_{j-m+l}$, and ending with the block, $B_{j+l-m}$, where $1 \leq m \leq l$, as expressed by Equation 3.7:

$$NB_{m,j,l} = \{BEV_{\beta} \in U | \beta = j-m+1, j-m+2, \ldots, j-m+l\}$$  \hspace{1cm} (3.7)

where
\( \beta \): the column number, \( j \), used for blocks in the \( m \)th neighbourhood and

\( BEV_\beta \): the economic value of a typical block, located in the \( m \)th neighbourhood.

Therefore, definition of any neighbourhood requires the following three factors:

(i) the location of the block, for which the neighbourhood is defined (the \( j \) address of the block in a 1D constraint problem),

(ii) the order of neighbourhood, "\( O_{nb} \)",

(iii) the neighbourhood number (that is, which neighbourhood is required).

3.5.2 The Optimum Neighbourhood

In order to locate the optimum neighbourhood of a block, the economic value of each neighbourhood has to be calculated and compared to one another. The term neighbourhood value (NBV) represents the net value of the neighbourhood if all blocks are extracted as a set. As an example, considering \( NB_i \) in Figure 3.7c, the corresponding neighbourhood value, for the block, \( B_4 \), is defined as:

\[
NBV_i = BEV_4 + BEV_5 + BEV_6 + BEV_7
\]

In general, each neighbourhood, \( NB_{m,j,i} \), may be assigned a value, denoted by \( NBV_{m,j,i} \). After defining the starting and ending blocks of the neighbourhoods in Equation 3.7, any neighbourhood value (NBV) may be determined using Equation 3.8:

\[
NBV_{m,j,i} = \sum_{\beta = j-m+1}^{j-m+l} BEV_\beta
\]

where \( NBV_{m,j,i} \) is the net economic value of the \( m \)th neighbourhood for the block, \( B_j \), if the order of neighbourhood equals "\( l \)".

Each NBV is a real scalar value. Since there are "\( l \)" neighbourhoods for any block, there will also be "\( l \)" real scalar values NBVs. These neighbourhood net values are collected in a set, denoted by NBVS, with the size of "\( l \)". In general, the set of neighbourhood
values may be defined by Equation 3.9:

\[ NBVS_{j,l} = \{NBV_{m,j,l} \in R \mid m = 1,2,...,l \} \] (3.9)

where \( NBVS_{j,l} \) is the set of real neighbourhood values for the block, \( B_j \), if the order of neighbourhood equals “1”.

### 3.5.2.1 Maximum Neighbourhood Value (MNBV)

The set of the neighbourhood values (\( NBVS \)) implies that, for extracting a given block there is a set of possible blocks with various net values. However, the optimisation policy is based on maximising the stope value. Therefore, the subset with the maximum value among all members of the set (the maximum \( NBV \)) should be located. As a numerical example, consider the row of blocks shown in Figure 3.9. If the minimum stope length is 10 meters and each block is three meters long, the stope block ratio is 3.3, which results in an order of neighbourhood of 4. All neighbourhood values for the block, \( B_j \), with the order of 4 are collected in the set \( NBVS_{j,4} \). Then, the 4\(^{th}\) (last) neighbourhood value (\( NBV \)) provides the maximum neighbourhood value, termed \( MNBV \) for this example.

In general, considering any block, \( B_j \), with the neighbourhood order of “1”, the maximum neighbourhood value (\( MNBV \)), among all the elements of the set of neighbourhood values, may be calculated in the following way:

\[ MNBV_{j,l} = \text{Max.} \{NBV_{1,j,l}, NBV_{2,j,l}, ..., NBV_{m,j,l}, ..., NBV_{l,j,l}\} \]

where \( MNBV_{j,l} \) is the maximum neighbourhood value for the block, \( B_j \), with the neighbourhood order of “1”. The above relation may be reduced to:

\[ MNBV_{j,l} = \text{Max.} \ NBVS_{j,l} \]

\[ = \text{Max.} \ NBVS_{j,l} \]

\[ = \text{Max.} \ NBVS_{j,l} \mid m = 1,2,...,l \]
Chapter Three: The MVN Algorithm

Minimum stope length = 10 m  block length = 3 m

\[ SBR = 3.3; \quad O_{nb} = 4 \]

\[ NB_4 \quad NB_3 \quad NB_2 \quad NB_1 \]

\[ NB_V_1 = 6 \quad \begin{array}{cccc} 4 & -2 & 1 & 3 \\ \end{array} \]

\[ NB_V_2 = 2 \quad \begin{array}{cccc} -1 & 4 & -2 & 1 \\ \end{array} \]

\[ NB_V_3 = 4 \quad \begin{array}{cccc} 3 & -1 & 4 & -2 \\ \end{array} \]

\[ NB_V_4 = 7 \quad \begin{array}{cccc} 1 & 3 & -1 & 4 \end{array} \]

\[ NB_V = \{6, 2, 4, 7\} \quad \Rightarrow \quad MNB_V = 7 = NB_V_4 \quad \Rightarrow \quad MVN = NB_4 \]

\[ NB_4 = \{B_{j.3}, B_{j.2}, B_{j.1}, B_j\} \quad \text{Flagging: } [F_{j.3} = 1; F_{j.2} = 1; F_{j.1} = 1; F_j = 1] \]

Figure 3.9: Locating the Maximum Value Neighbourhood

Consider that the \( NB_V \) set is maximised at \( m = \mu \), hence,

\[ MNB_{V,j,l} = NBV_{\mu,j,l} \quad (3.10) \]

where

\( \mu \): the index (element number) of the \( NB \) in the corresponding \( NB_V \) set, which has the maximum value,

\( NBV_{\mu,j,l} \): the net value of the \( \mu^{th} \) neighbourhood for the block, \( B_j \), with the neighbourhood order of \( "l" \), which is the maximum one among the set of neighbourhood values.
3.5.2.2 The Maximum Value Neighbourhood (MVN)

The $MNBV$ obtained above is a scalar value that indicates the maximum net value the block may contribute to the stope. Next, it is required to locate and flag those blocks of the neighbourhood which make this contribution. The corresponding neighbourhood is then the set of blocks that qualify for inclusion in the final stope. This set of blocks is called the maximum value neighbourhood ($MVN$). Using Equation 3.7, the $MVN$ can be defined for each block, $B_j$, with any order of neighbourhood, "I", as below:

$$MVN_{j,I} = NB_{\mu,j,I} = \{BEV_\beta \in M \mid \beta \in I, \beta = j - \mu + 1, j - \mu + 2, \ldots, j - \mu + I\}$$

where

$NB_{\mu,j,I}$: the neighbourhood with the maximum value for the block, $B_j$, with the neighbourhood order of "I",

$MVN_{j,I}$: the maximum value neighbourhood for the block, $B_j$, with the neighbourhood order of "I",

$\beta$: the column number of blocks in the maximum value neighbourhood and

$BEV_\beta$: the economic value of the block, $B_\beta$, which is located in the maximum value neighbourhood.

After the $MVN$ is defined, all its elements are marked for inclusion in the final stope if they positively contribute to the stope value. In order to mark the selected blocks, an indicator (flag) function, $F$, is used. It is valued at "0" for all blocks except those which qualify for inclusion in the final stope, in which case the flag is set to "1".

3.5.3 Infeasible Neighbourhoods

For each block with the neighbourhood of order "I", there are "I" possible neighbourhoods. However, all of these neighbourhoods are not necessarily feasible. An infeasible neighbourhood occurs for blocks located inside the model boundaries if the neighbourhoods of these blocks contain members (blocks) outside the block model. In
cases, where one or more members of a neighbourhood (NB) is outside the block model, that NB is undetermined, therefore, infeasible. Infeasible neighbourhoods, in 1D constraint problems, occur for blocks at both ends of the block model.

Consider a row of blocks, where the order of neighbourhood equals 3, as shown in Figure 3.10. There will be three possible neighbourhoods for each block in the row. However, for the block, \( B_1 \), there is only one feasible neighbourhood and for the block, \( B_2 \), there are two feasible neighbourhoods, while all three possible neighbourhoods for the block, \( B_3 \), are feasible (Figure 3.10). Boundary blocks on the other side face the same problem as well. Therefore, the block, \( B_n \), has only one feasible neighbourhood and there are two feasible neighbourhoods for the block, \( B_n-1 \), while all three possible neighbourhoods of the block, \( B_n-2 \), are feasible. Other blocks have their three possible neighbourhoods all feasible. For the order of the neighbourhood of 3, only the first and the last two blocks of each row have infeasible neighbourhoods.

In general, for any \( O_{nb} = l \), the first and the last \( l-1 \) blocks of each row, with the same dimension as the minimum stope, are considered the boundary blocks and have infeasible neighbourhoods. The number of feasible neighbourhoods is one for the first block, \( B_1 \); it is 2 for the second block, \( B_2 \), and so on until the \( (l-1)^{th} \) block, \( B_{l-1} \). The same rule is applied for the last \( l-1 \) blocks in the row.

### 3.6 THE MVN ALGORITHM

The neighbourhood concept, described above, is based on locating the optimum neighbourhood for a single block. However, the application of this approach, to a given ore-body block model, requires a well-defined algorithm computer program.
3.6.1 The Optimisation Procedure

The model optimisation contains a number of processes that should be performed on each block. There are at least eight stages that should be performed during the optimisation process:

(i) for each block, consider its block economic value \((BEV)\);
(ii) determine the neighbourhoods of the block, based on the order of neighbourhood. That is, construct the set of possible neighbourhoods \((NBS)\);
(iii) evaluate the feasibility of each neighbourhood within the \(NBS\) set;
(iv) calculate the economic value for each neighbourhood, neighbourhood value \((NBV)\), and determine the set of neighbourhood values \((NBVS)\);
(v) locate the maximum neighbourhood value \((MNBV)\) within the \(NBVS\) set;
(vi) determine the maximum value neighbourhood \((MVN)\);
(vii) flag the blocks of the \(MVN\) and
(viii) update the stope economic value \((SEV)\).

The above eight stages may be implemented in two ways. The first is to perform each stage for all blocks before proceeding to the next stage. This could be termed a "stage-oriented algorithm". The other approach is to conduct all of the optimisation stages for a given block, before proceeding to examine the next block, an approach, which may be termed a "block-oriented algorithm".

In a stage-oriented algorithm, the relevant activity such as definition, calculation and comparison, is conducted for all blocks of the model. That is, the process does not proceed to the next stage until all blocks have been examined. After all stages have been performed, the optimisation is completed. However, the above stages may be reduced to the following three main steps: taking the block economic values \((BEV)\) of the blocks, stage (i); defining the maximum value neighbourhood \((MVN)\) for all blocks, stages (ii) to (vii); and calculating the stope value, stage (viii).

In contrast, the block-oriented algorithm performs all of the stages of the optimisation process for any block, before proceeding to the next block. After all blocks have been examined, the optimisation is completed and the optimum layout is obtained. This means that in this algorithm: blocks are taken one by one; for each block the set of possible neighbourhoods \((NBS)\) is defined; the feasibility of its elements is evaluated; the values of all feasible neighbourhoods are calculated; the \(MNBV\) is located; and the corresponding neighbourhood, \(MVN\), is determined. Next, the blocks of the \(MVN\) are flagged and the optimum stope and its value are updated.

In this thesis, the block-oriented algorithm has been used. It allows a number of check points to be made to filter the process for infeasible blocks at each stage. For example, there is no need to include those blocks in the final stope limit, which have a negative economic value.

The generalised flow-chart for the optimisation procedure in the \(MVN\) algorithm is illustrated in Figure3.11. The procedure is based on the block-oriented type whereby
blocks are taken into consideration one by one, in the order of rows (X direction), columns (Y direction) and finally sections (Z direction). If a block has a non-negative value and is not already flagged, the procedure will continue to construct its set of neighbourhoods, \((NBS)\), based on the order of neighbourhood “I”; calculate the values of neighbourhoods; and finally locate the maximum neighbourhood value \((MNBV)\) and the maximum value neighbourhood \((MVN)\) of the block. It is then determined whether or not the maximum neighbourhood value is non-negative, that is, contributes to the stope value. Ignoring the blocks with a negative \(MNBV\), the procedure continues for the non-negative ones.

The \(MVN\) of each block provides a marginal contribution to the stope value. This marginal value may be negative, zero or positive. Negative marginal values are ignored and the algorithm selects any block with an \(MVN\) that provides a non-negative marginal value. The elements of that \(MVN\) are then flagged and included in the stope boundaries. The above process will be repeated for all blocks within the block model. After all the blocks are examined, the final optimum stope boundaries are displayed.

Four checks are used in this algorithm to exclude unnecessary blocks. A block is ignored:

1. if the block is flagged already;
2. if the block has a negative value;
3. if the maximum neighbourhood value \((MNBV)\) of the block is negative; and
4. if the marginal value provided by the \(MVN\) is negative.

The first check is to review the flag of the block. A block may have been flagged and included in the final stope already. This indicates that it was an element of the \(MVN\) of a different block that was processed earlier.
Chapter Three: The MVN Algorithm

**START**

Define $SBR$ and $O_{nb}$. Take the first block.

Yes

$BEV < 0$ ?

or

$F = 1$ ?

No

Construct the set of neighbourhoods ($NBS$) of order "$l$" and the set of neighbourhood values ($NBVS$).

Locate the Maximum neighbourhood Value ($MNBV$) among the $NBVS$ elements.

Yes

$MNBV < 0$ ?

No

Obtain the marginal value of the maximum value neighbourhood ($MVN$).

Yes

Marginal value $< 0$ ?

No

Update the stope boundary.

Is it the last block ?

No

Yes

Output the optimum stope layout.

**END**

Figure 3.11: Generalised Flow-Chart for the Optimisation Procedure
Chapter Three: The MVN Algorithm

Since zero valued blocks do not contribute to profit, they should be ignored. On the other hand, they contribute to mine productivity with no loss, so they could be included to allow more ore to be utilised. The MVN algorithm excludes only negative value blocks and performs the optimisation process for zero and positive value blocks.

The next check is undertaken after the calculation of the maximum neighbourhood values (MNBV) of the blocks. Here the MNBV is obtained and reviewed to determine whether or not it is negative. If the maximum neighbourhood value is negative, it means that, although the block itself is valuable, its surrounding blocks cost more than the profit obtained by the block. In other words, the block cannot pay for its neighbours and it is not necessary to continue the process for this block. The MVN algorithm treats zero value maximum neighbourhood values the same as positive value ones and only rejects blocks with a negative MNBV from the process.

The last check may be undertaken after obtaining the marginal value of the maximum value neighbourhood (MVN). A test should be made to ensure that the MVN of the block contributes to the final stope value, that is, its marginal value is not negative. Although the maximum neighbourhood value of a block is positive, sometimes inclusion of its MVN may decrease the total value of a stope. Since some elements (blocks) are common among the MVNs of two or more blocks, the determined MVN of the current block may include some elements that are flagged already. These are to be prevented from being considered again, for contribution to the stope value.

The real difference that the inclusion of the MVN will make in the stope value, is in the marginal value. The marginal value of an MVN is defined by the total value of those elements of the MVN, that are new to the final stope, and contribute to the stope value when considering the current block. Negative marginal values, which cause a decrease in the stope value, may occur when valuable elements of an MVN have been flagged earlier and the costly elements are new to the stope. In these cases, the algorithm rejects the block from further stages of the optimisation process. Again, blocks providing zero or positive marginal value, proceed to the next step of the algorithm.
The algorithm, which was summarised in the flowchart (shown in Figure 3.11), may be coded using high level procedural programming languages, such as FORTRAN, C++ and BASIC. However, a generalised pseudo code for the optimisation procedure is presented so as to match with any language code. This is illustrated in Figure 3.12.

<table>
<thead>
<tr>
<th><strong>NI, NJ, NK:</strong></th>
<th>Maximum No. of blocks in 3D model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>F:</strong></td>
<td>Flag function of blocks</td>
</tr>
<tr>
<td><strong>SEV:</strong></td>
<td>Stope Economic Value.</td>
</tr>
<tr>
<td><strong>MVN_{ijk}</strong>:</td>
<td>Maximum Value Neighbourhood (a set)</td>
</tr>
<tr>
<td></td>
<td>μ: the index of the NB with maximum value</td>
</tr>
<tr>
<td></td>
<td>FF: An indicator function for blocks so that FF = 1 if F = 0, FF = 0 otherwise</td>
</tr>
</tbody>
</table>

**Start**

**Step 1:** Initialise variables

Set \( F_{ijk} = 0 \), \( m = 1,2,\ldots,NI \); \( j = 1,2,\ldots,NJ \); \( k = 1,2,\ldots,NK \)

Set \( FF_m = 0 \), \( m = 1,2,\ldots,1 \); Set SEV = 0

**Step 2:** Examine the first block

Read BEV and F of the first block

**Step 3:** Check the negativity and flag of the block

If \( BEV_{ijk} < 0 \) or \( F_{ijk} = 1 \), Go to Step 9

**Step 4:** Locate the maximum neighbourhood value (MNBV)

Construct the set of neighbourhoods of order "1" (NBS_{i,j,k-1})

Check the feasibility of neighbourhoods

Construct the set of neighbourhood values (NBV_{i,j,k-1})

Set \( MNBV_{ijk,1} = \text{Max.} \ NBV_{i,j,k-1} \)

**Step 5:** Check the negativity of MNBV

If \( MNBV < 0 \), Go to Step 9

**Step 6:** Calculate the marginal value

Given \( MNBV_{ijk,1} = NBV_{i,j,k-1} \) then set \( MVN_{ijk,1} = \text{NBV}_{\mu,j,k} \) \( (\mu = 1,2,\ldots,1) \);

set \( FF_{MN} = -(FF_{MN} - 1) \)

Set Marginal value = \( \{BEV_{MN}\}_{ij} \cdot \{FF_{MN}\}_{ij} \)

**Step 7:** Check the negativity of the marginal value

If Marginal value < 0, Go to Step 9

**Step 8:** Update the stope

Set \( SEV = SEV + \text{Marginal value} \)

Set \( F_{MN} = 1 \)

**Step 9:** Check the end of the model

If no more blocks, Go to Step 11

**Step 10:** Examine the next block

Read BEV and F of the next block

Go to Step 3

**Step 11:** Output the results

List the elements of the final stope

**Stop**

**Figure 3.12:** A Simplified Pseudo Code for the MVN Algorithm
3.6.2 Numerical Examples

Consider a row of blocks with ten columns, that is, \( \{ B_j \mid j = 1, 2, ..., 10 \} \), as shown in Figure 3.13. The blocks are 10 m long, while the stope length is restricted to a minimum of 25 m. The stope block ratio is then 2.5, which results in the order of neighbourhood of three, in the same direction that the row of blocks is set. The number inside each cell represents the economic value, in terms of dollars, for the corresponding block. The distribution of the block values, \( BEV_j \), follows a function of grade distribution among the block model. However, for simplicity, the block values are assigned by integer numbers, from the list \([-7, 7]\), inclusive.

The following is a step by step explanation of the manual application of the MVN algorithm, so as to optimise this row of blocks. The example has been designed so that all three checks are used.

START

Step 1: Initialise variables.

Block length = 10 m; Minimum stope length = 25
SBR = 2.5; \( O_{nb} \) = 3
SEV = 0; \( F \) = 0; \( FF \) = 0

Step 2: Examine the first block.

\( B_1 \): \( BEV_1 = -3 \); \( F_1 = 0 \)

Step 3: Check the negativity and the flag of the block.

\( BEV_1 < 0 \quad \Rightarrow \quad \) Block \( B_1 \) is exempted from further process.
Go to Step 9.

Step 9: Check the end of the model.

B_1 is not the last block. Continue.

Step 10: Examine the next block.

B_2: BEV_2 = 1; F_2 = 0
Go to Step 3

Step 3: Check the negativity and the flag of the block.

BEV_2 ≥ 0 and F_2 ≠ 1 → Continue.

Step 4: Locate the maximum neighbourhood value (MNBV).

\[
\begin{align*}
NB_1 &= \{1, 0, -4\} \rightarrow NBV_1 = 1 + 0 - 4 = -3 \\
NB_2 &= \{-3, 1, 0\} \rightarrow NBV_2 = -3 + 1 + 0 = -2 \\
NB_3 &= \text{Not feasible} \\
\Rightarrow \quad MNBV &= -2; \ MVN = NB_2
\end{align*}
\]

Step 5: Check the negativity of the MNBV.

MNBV < 0 → Block B_2 is exempted from further process.
Go to Step 9.

Step 9: Check the end of the model.

B_2 is not the last block. → Continue.

Step 10: Examine the next block.

B_3: BEV_3 = 0; F_3 = 0
Go to Step 3

Step 3: Check the negativity and the flag of the block.

BEV_3 ≥ 0 and F_3 ≠ 1 → Continue.

Step 4: Locate the maximum neighbourhood value (MNBV).

\[
\begin{align*}
NB_1 &= \{0, -4, 3\} \rightarrow NBV_1 = 0 - 4 + 3 = -1 \\
NB_2 &= \{1, 0, -4\} \rightarrow NBV_2 = 1 + 0 - 4 = -3 \\
NB_3 &= \{-3, 1, 0\} \rightarrow NBV_3 = -3 + 1 + 0 = -2
\end{align*}
\]
Step 5: Check the negativity of the $MNBV$.

\[ MNBV < 0 \rightarrow \text{Block } B_3 \text{ is exempted from further process.} \]

Go to Step 9.

Step 9: Check the end of the model.

\[ B_3 \text{ is not the last block. } \rightarrow \text{Continue.} \]

Step 10: Examine the next block.

\[ B_4: \ BEV_4 = -4; \ F_4 = 0 \]

Go to Step 3

Step 3: Check the negativity and the flag of the block.

\[ BEV_4 < 0 \rightarrow \text{Block } B_4 \text{ is exempted from further process.} \]

Go to Step 9.

Step 9: Check the end of the model.

\[ B_4 \text{ is not the last block. } \rightarrow \text{Continue.} \]

Step 10: Examine the next block.

\[ B_5: \ BEV_5 = 3; \ F_5 = 0 \]

Go to Step 3

Step 3: Check the negativity and the flag of the block.

\[ BEV_5 \geq 0 \text{ and } F_5 \neq 1 \rightarrow \text{Continue.} \]

Step 4: Locate the maximum neighbourhood value ($MNBV$).

\[
\begin{align*}
\text{NB}_1 &= \{3, -2, 4\} \rightarrow \text{NBV}_1 = 3 - 2 + 4 = 5 \\
\text{NB}_2 &= \{-4, 3, -2\} \rightarrow \text{NBV}_2 = -4 + 3 - 2 = -3 \\
\text{NB}_3 &= \{0, -4, 3\} \rightarrow \text{NBV}_3 = 0 - 4 + 3 = -1 \\
\end{align*}
\]

\[ \rightarrow \ MNBV = 5; \quad \text{MVN} = \text{NB}_1 \]

Step 5: Check the negativity of the $MNBV$.

\[ MNBV \geq 0 \rightarrow \text{Continue.} \]

Step 6: Calculate the marginal value.
Chapter Three: The MVN Algorithm

MVN = NB_1 = {BEV_5, BEV_6, BEV_7}

F_5 = 0 \rightarrow FF_5 = 1; F_6 = 0 \rightarrow FF_6 = 1; F_7 = 0 \rightarrow FF_7 = 1

Marginal Value = BEV_5 \cdot FF_5 + BEV_6 \cdot FF_6 + BEV_7 \cdot FF_7

= (3 \times 1) + (-2 \times 1) + (4 \times 1) = 5

Step 7: Check the negativity of the marginal value.

Marginal Value \geq 0 \rightarrow Continue.

Step 8: Update the stope.

SEV = SEV + Margin Value = 0 + 5 = 5 \rightarrow SEV = 5

F_5 = 1; F_6 = 1; F_7 = 1

Step 9: Check the end of the model.

B_5 is not the last block. \rightarrow Continue.

Step 10: Examine the next block.

B_6: BEV_6 = -2; F_6 = 1

Go to Step 3

Step 3: Check the negativity and the flag of the block.

BEV_6 < 0 \rightarrow Block B_6 is exempted from further process.

Go to Step 9.

Step 9: Check the end of the model.

B_6 is not the last block. \rightarrow Continue.

Step 10: Examine the next block.

B_7: BEV_7 = 4; F_7 = 1

Go to Step 3

Step 3: Check the negativity and the flag of the block.

F_7 = 1 \rightarrow Block B_7 is exempted from further process.

Go to Step 9.

Step 9: Check the end of the model.

B_7 is not the last block. \rightarrow Continue.
Chapter Three: The MVN Algorithm

Step 10: Examine the next block.

\[ B_8: \quad BEV_8 = 0; \quad F_8 = 0 \]

Go to Step 3

Step 3: Check the negativity and the flag of the block.

\[ BEV_8 \geq 0 \quad \text{and} \quad F_8 \neq 1 \Rightarrow \text{Continue.} \]

Step 4: Locate the maximum neighbourhood value (MNBV).

\[
\begin{align*}
NB_1 &= \{0, -1, 2\} \quad \Rightarrow \quad NBV_1 = 0 - 1 + 2 = 1 \\
NB_2 &= \{4, 0, -1\} \quad \Rightarrow \quad NBV_2 = 4 + 0 - 1 = 3 \\
NB_3 &= \{-2, 4, 0\} \quad \Rightarrow \quad NBV_3 = -2 + 4 + 0 = 2
\end{align*}
\]

\[ MNBV = 3; \quad MVN = NB_2 \]

Step 5: Check the negativity of the MNBV.

\[ MNBV \geq 0 \Rightarrow \text{Continue.} \]

Step 6: Calculate the marginal value.

\[
\begin{align*}
MVN &= NB_2 = \{BEV_7, \ BEV_8, \ BEV_9\} \\
F_7 &= 1 \Rightarrow FF_7 = 0; \quad F_8 = 0 \Rightarrow FF_8 = 1; \quad F_9 = 0 \Rightarrow FF_9 = 1 \\
\text{Marginal Value} &= BEV_7 \cdot FF_7 + BEV_8 \cdot FF_8 + BEV_9 \cdot FF_9 \\
&= (4 \times 0) + (0 \times 1) + (-1 \times 1) = -1
\end{align*}
\]

Step 7: Check the negativity of the marginal value.

\[ \text{Marginal Value} < 0 \Rightarrow \text{Block } B_8 \text{ is exempted from further process.} \]

Go to Step 9.

Step 9: Check the end of the model.

\[ B_8 \text{ is not the last block.} \Rightarrow \text{Continue.} \]

Step 10: Examine the next block.

\[ B_9: \quad BEV_9 = -1; \quad F_9 = 0 \]

Go to Step 3

Step 3: Check the negativity and the flag of the block.
BEV_9 < 0 \implies \text{Block B}_9 \text{ is exempted from further process.}

Go to Step 9.

Step 9: Check the end of the model.

B_9 \text{ is not the last block.} \implies \text{Continue.}

Step 10: Examine the next block.

B_{10}: \quad BEV_{10} = 2; \quad F_{10} = 0

Go to Step 3.

Step 3: Check the negativity and the flag of the block.

BEV_{10} \geq 0 \text{ and } F_{10} \neq 1 \implies \text{Continue.}

Step 4: Locate the maximum neighbourhood value (MNBV).

NB_1 = \text{Not feasible}

NB_2 = \text{Not feasible}

NB_3 = \{0, -1, 2\} \implies \text{MNBV} = \text{NB}_3 = 0 - 1 + 2 = 1

\implies \text{MNBV} = 1; \quad \text{MVN} = \text{NB}_3

Step 5: Check the negativity of the MNBV.

MNBV \geq 0 \implies \text{Continue.}

Step 6: Calculate the marginal value.

MVN = \text{NB}_3 = \{BEV_8, BEV_9, BEV_{10}\}

\begin{align*}
F_8 &= 0 \implies FF_8 = 1; \quad F_9 = 0 \implies FF_9 = 1; \quad F_{10} = 0 \implies FF_{10} = 1 \\
\text{Marginal Value} &= \text{BEV}_8 \cdot FF_8 + \text{BEV}_9 \cdot FF_9 + \text{BEV}_{10} \cdot FF_{10} \\
&= (0 \times 1) + (-1 \times 1) + (2 \times 1) = 1
\end{align*}

Step 7: Check the negativity of the marginal value.

Marginal Value \geq 0 \implies \text{Continue.}

Step 8: Update the stope.

SEV = SEV + Marginal Value = 5 + 1 = 6 \implies SEV = 6

F_8 = 1; \quad F_9 = 1; \quad F_{10} = 1

Step 9: Check the end of the model.
Bi0 is the last block. \( \Rightarrow \) Go to Step 11.

Step 11: Output the results.

\[
\text{Stope} = \{B_5, B_6, B_7, B_8, B_9, B_{10}\}; \quad \text{Stope Value} = 6
\]

STOP

END

Figure 3.14 illustrates the optimised row of blocks, in which the flagged blocks are shaded. The first four blocks, including one positive and one zero value block, are excluded from the final stope as they do not meet the requirements. In addition, two negative blocks have been included in the optimum stope to insure the constraints are satisfied.

![Figure 3.14: The Optimised Example Manually using the MVN Algorithm](image)

The optimisation would be more meaningful when used in a section of 3D grouped blocks, in the form of a block model. The following example (Figure3.15a) shows a section of six rows with six columns of blocks, that has been manually optimised using the algorithm. It was assumed that the order of neighbourhood equals 3 in the vertical direction, that is, the stope has a minimum height restriction of three blocks. The optimised section has been shown in Figure3.15b, where the flagged blocks have been shaded. Table 4.1 has summarised the intermediate results of applying the algorithm to the third column only. The summary shows the step by step application of the MVN algorithm to the optimisation of column 3 of the section.
Figure 3.15: Optimising a Stope Section using the MVN Algorithm with a 1D (height) Constraint

Table 3.2: Summary of the Algorithm Applied for Column 3 (Figure 3.15)
The first column in Table 3.2 shows the row and column addresses of the blocks under consideration. The corresponding block economic values \((BEV)\) are shown in the second column. The algorithm takes the block in the first row, \(B_{13}\), which has an economic value of 3, into the process. The block is then checked for a negative value or flagging. The block, \(B_{13}\), is not negatively valued and is not flagged. These responses are noted in columns 3 and 4 respectively, and the procedure continues for the block. Possible neighbourhoods for the block are constructed and shown in the fifth column. Since the order of neighbourhood is 3, there are three possible neighbourhoods for any block, each of which consists of three elements (blocks). However, for the block, \(B_{13}\), there is only one feasible neighbourhood, \(\{B_{13}, B_{23}, B_{33}\}\) or \(\{3, 2, 6\}\), which is valued at 11. Therefore, the maximum neighbourhood value \((MNBV)\) for the block is 11 and the maximum value neighbourhood \((MVN)\) is the first neighbourhood, \(\{B_{13}, B_{23}, B_{33}\}\), whose elements are shaded. The previous flags of blocks are shown in the sixth column, which are zero for the elements of the \(MVN\) of the current block. To update the stope, the \(MVN\) of the current block should be added to the previous stope, excluding any common elements. A new parameter, \(FF\), is introduced to exclude the intersection of the previous stope and the current \(MVN\) from the addition process, such that:

\[
FF = \begin{cases} 
1 & \text{if } F = 0, \\
0 & \text{if } F = 1. 
\end{cases}
\]

The \(FF\) parameter is set to "1" for all elements of the \(MVN\) of the block, \(B_{13}\), as the \(F\) parameter is "0" for all of them. The summation of the products of \(FF \times BEV\) for, all elements of the current \(MVN\), shows the contribution of that \(MVN\) to the stope. This contribution is added to the previous stope value \((SEV)\) to determine the new \(SEV\). Finally, the elements of the \(MVN\) are flagged (their \(F\) parameter is set to "1"). The new flags are shown in the last column of the Table.

The algorithm then proceeds to take the block in the second row, \(B_{23}\), with an economic value of 2. The block is not negatively valued, but it is already flagged \((F_{23} = 1)\), therefore, the algorithm rejects the block from further process. The same situation is
applied for block, $B_{33}$, and it is taken out of the process.

The next block, $B_{43}$, with a net value of 3 is neither negatively valued or flagged, so the algorithm is continued for that block. The set of neighbourhoods consists of \{$B_{43}$, $B_{33}$, $B_{63}$\}, \{$B_{33}$, $B_{43}$, $B_{53}$\} and \{$B_{23}$, $B_{33}$, $B_{43}$\} or, if stated by the net values of the blocks, \{2, 6, 3\}, \{6, 3, 3\} and \{3, 3, -2\}. The second neighbourhood, whose elements are shaded, provides the maximum value, $MNBV$, at 12 units. One member of the $MVN$ is already flagged ($F_{33} = 1$), so the corresponding $FF$ is set to "0" ($FF_{3,3} = 0$). The real contribution of the $MVN$ of the block is 6 units. This is added to the previous stope value to provide a new $SEV$ at 17 units. Finally, two new blocks of the $MVN$ (that is, blocks, $B_{43}$ and $B_{53}$) are flagged.

The algorithm continues to take the block, $B_{53}$, with a net value of 3, into the process. The block has a non-negative value, but it is flagged already ($F_{53} = 1$), hence, the algorithm rejects the block from further process and continues to take the next block, $B_{63}$. Since the block, $B_{63}$, is negatively valued, there is no need to proceed.

### 3.6.3 Algorithm Capability to Recover the Stope

The block model contains blocks with positive, zero and negative values. The summation of the total blocks, with positive values, represents the total worth embedded in the model. However, this worth is not completely recoverable, when the mining constraints are imposed. Various optimisation algorithms are applied to maximise the recovery of this worth. The capability of an algorithm to optimise a block model may be assessed through its capability to recover the model worth and to exclude non-desirable blocks from the suggested final stope.

Some indexes may be defined to evaluate this capability of the algorithm. The most representative index may be defined as: the ratio of the total value of the optimised stope ($SEV$) to the total possible worth of the model, and it is called the stope recovery by the algorithm, $R$. 
The term $SEV$ includes all positive, zero and negative value blocks in the stope. In order to indicate the recovery of positive value blocks, two indexes may be defined. The first one is $R_n$, which shows the number of positive value blocks recovered.

$$R_n = \frac{\text{number of positive blocks included in the stope}}{\text{number of positive blocks in the model}}$$

However, if the net values of these positive blocks are considered, the recovery index may be modified to $R_v$ which indicates the ratio of positive value blocks recovered by value.

$$R_v = \frac{\text{total value of positive blocks included in the stope}}{\text{total value of positive blocks in the model}}$$

The negative blocks, which are included in the optimised stope, and the positive blocks, which are excluded from the optimised stope, may both be obtained by their numbers and values.

$$NI_n = \text{the number of negative blocks included in the stope}$$

$$NI_v = \text{the value of negative blocks included in the stope}$$

$$PN_n = \text{the number of positive blocks not - included in the stope}$$

$$PN_v = \text{the value of positive blocks not - included in the stope}$$

Based on the above definitions, the one-dimensional $MVN$ algorithm, applied to the example shown in Figure 3.15, provides a stope recovery of 87%. The example model consists of 36 blocks with the total value of $66$, including 25 positive blocks with the value of $87$ and 11 negative blocks with the value of $-21$. The total value of the optimised stope equals $76$. The performance of the optimisation may be summarised as follows:
\[ NL_n = 6 \quad NL_v = 11 \]
\[ PN_n = 0 \quad PN_v = 0 \]
\[ R_n = 100\% \quad R_v = 100\% \quad \text{Stope recovery:} \quad R = 87\% \]

3.7 SUMMARY

The "maximum value neighbourough" concept, developed during the current study, for the optimisation of stope layout was described in this chapter. General assumptions were made to employ a fixed block model, of an ore-body, using the MVN method. After expressing the optimisation objective function, the imposed stope geometry constraints were formulated in terms of "neighbourhood" and "order of neighbourhood". The process of the neighbourhood definition; determination of \( NB \) elements; set of possible neighbourhoods and set of neighbourhood values, \( NBVS \); locating the maximum neighbourhood value, \( MNBV \); and flagging elements of the maximum value neighbourhood, \( MVN \); was described with an aid of numerical examples to explain the mathematical notation. For an individual block, the MVN approach suggested a small island of mineable blocks within the whole ore-body.

A block-oriented optimisation procedure (the MVN algorithm) was developed to apply the neighbourhood concept to all blocks of the ore-body. The algorithm aims to form optimum stope boundaries by developing, as much as possible, the small island (collection) of blocks suggested by the MVN approach. The MVN algorithm considers blocks one by one, in the order of rows, columns and sections. For each block, the feasible neighbourhoods are determined, the maximum \( NBV \) is calculated and elements of the \( MVN \) are flagged. Some checkpoints are set to suspend the optimisation in stages where it is not needed. Step by step manual employment of the algorithm was described using numerical examples.

The MVN algorithm, introduced in this chapter, covers the basics of the one-dimensional stope geometry constraint approach. In reality, the stope geometry is restricted in three dimensions. The following chapter discusses the extension of the MVN algorithm to cover 2D cases and generalisations for 3D neighbourhood problems.
CHAPTER FOUR

THE 2D AND 3D MAXIMUM VALUE NEIGHBOURHOOD ALGORITHMS

4.1 INTRODUCTION

The neighbourhood concept, introduced in the previous chapter, was based on one-dimensional neighbourhoods. In practice, neighbourhoods are considered in three dimensions. The MVN algorithm for 1D neighbourhoods has to be modified to meet the requirements of 2D and 3D cases. These include: the stope block ratio; the order of neighbourhood; the identification rule for referencing the elements and the neighbourhoods; the determination of any neighbourhood for any block; the calculation of the neighbourhood values; and the feasibility of the neighbourhoods. This chapter discusses the main specifications of the 2D and 3D neighbourhoods and the application of the MVN algorithm to the three dimensions. A simple two-dimensional numerical example is included to illustrate the implementation of the algorithm.

4.2 TWO- AND THREE-DIMENSIONAL NEIGHBOURHOODS

A 2D neighbourhood occurs when the minimum stope size is restricted in two dimensions only. The general form of a neighbourhood is a 3D neighbourhood in which the minimum size of the stope is restricted in all of the three principal dimensions (that
is, X, Y and Z directions). The terms “stope block ratio” (SBR) and “order of neighbourhood” \((O_{nb})\) must be redefined for two and three dimensions. While the set of blocks forming a 1D neighbourhood layout is a line, that of a 2D neighbourhood layout is a rectangle, and in a 3D neighbourhood, blocks are arranged in a cubic pattern.

A two-dimensional neighbourhood is any set of all blocks, including the block of interest, which are sequentially arranged in a rectangular pattern, in either of the two specified directions, that may be mined to satisfy the minimum stope geometric requirements. Consider a section of blocks consisting of six rows and six columns, that is,

\[
\{B_{ij} | i = 1, 2, ..., 6; j = 1, 2, ..., 6\}
\]

where blocks are 15 m long and 10 m high, as shown in Figure 4.1. Given that the stope is restricted to a minimum length of 25 m and a minimum height of 15 m, the result will be two different stope block ratios, and orders of neighbourhood, for the given dimensions. That is,

\[
\begin{align*}
SBR_i &= \frac{15}{10} = 1.50 \quad \Rightarrow \quad (O_{nb})_i = 2 \\
SBR_j &= \frac{25}{15} = 1.67 \quad \Rightarrow \quad (O_{nb})_j = 2
\end{align*}
\]

where

- \(SBR_i\): the stope block ratio in the X direction,
- \(SBR_j\): the stope block ratio in the Y direction,
- \((O_{nb})_i\): the order of neighbourhood in the X direction,
- \((O_{nb})_j\): the order of neighbourhood in the Y direction.
Chapter Four: The 2D and 3D MVN Algorithm

Minimum stope height 15 m

1 2 3 4 5 6

Minimum stope length

A 2D minimum stope size

(a) A typical section of block model with 6 rows and 6 columns

(b) Examples of 2D neighbourhoods for B\text{33}

Figure 4.1: Examples of 2D Possible Neighbourhoods for a Specific Block, B\text{33}

Any neighbourhood for a given block, in the above example, consists of four blocks, which are arranged in a rectangular pattern of two blocks in the X direction and two blocks in the Y direction. The possible 2D neighbourhoods for the block, B\text{33}, are shown in Figure 4.1b.

The above concept may be generalised to define a three-dimensional neighbourhood. A 3D neighbourhood is defined as, a set of all those blocks, including the block of interest,
which are sequentially arranged in a cubic pattern in each of the three directions that may be mined to satisfy the minimum stope geometric requirements. Consider a complete block model consisting of five sections, each of which consists of five rows and five columns. That is,

\[ \{B_{ijk} \mid i = 1, 2, ..., 5; j = 1, 2, ..., 5; k = 1, 2, ..., 5\} \]

where each block is 15 m long, 10 m wide and 10 m high, as shown in Figure 4.2. Assume that the minimum stope dimensions are 25 m in length, 16 m in width and 18 m in height. The three respective stope block ratios and orders of neighbourhood are:

\[
\begin{align*}
SBR_i &= \frac{16}{10} = 1.60 \quad \Rightarrow \quad (O_{nb})_i = 2 \\
SBR_j &= \frac{25}{15} = 1.67 \quad \Rightarrow \quad (O_{nb})_j = 2 \\
SBR_k &= \frac{18}{10} = 1.80 \quad \Rightarrow \quad (O_{nb})_k = 2
\end{align*}
\]

where

- \( SBR_i \): the stope block ratio in the X direction,
- \( SBR_j \): the stope block ratio in the Y direction,
- \( SBR_k \): the stope block ratio in the Z direction and
- \( (O_{nb})_i \): the order of neighbourhood in the X direction and
- \( (O_{nb})_j \): the order of neighbourhood in the Y direction.
- \( (O_{nb})_k \): the order of neighbourhood in the Z direction.

The above example implies that any neighbourhood, for a given block, \( B_{ijk} \), consists of eight blocks forming a cubic shape with two blocks in each direction; 3D possible neighbourhoods are shown in Figure 4.3.
Chapter Four: The 2D and 3D MVN Algorithm

A 3D minimum stope size

Figure 4.2: A Typical Block Model illustrating 3D Stope Size Constraints

Two neighbourhoods whose common member is only the block of interest ($B_{ijk}$)

Figure 4.3: Examples of 3D Neighbourhoods
4.2.1 The Order of Neighbourhood

The order of neighbourhood in 1D cases was represented by a scalar integer value. However, in 2D cases, two scalar integers are required to represent the order of neighbourhood in two dimensions. These two scalars may be collected in an ordered set, termed \([O_{nb}]^{(2)}\), that is:

\[
[O_{nb}]^{(2)} = \{(O_{nb})_i, (O_{nb})_j\}
\]

where \([O_{nb}]^{(2)}\) represents the 2D order of neighbourhood. Similarly, in a general form, the 3D order of neighbourhood is expressed in terms of an ordered set with three members. The 3D order of neighbourhood is termed \([O_{nb}]^{(3)}\) and is described as:

\[
[O_{nb}]^{(3)} = \{(O_{nb})_i, (O_{nb})_j, (O_{nb})_k\}
\]

Similar to 1D neighbourhoods, the total number of elements in a 2D (or 3D) neighbourhood (size of the \(NB\)) and the total number of possible corresponding neighbourhoods of a block (size of the set of \(NBS\)) are equal. This size is obtained by taking the product of the elements of the corresponding set, that is:

\[
\begin{align*}
n([NB]^{(2)}) &= n([NBS]^{(2)}) = (O_{nb})_i (O_{nb})_j \quad (4.1) \\
n([NB]^{(3)}) &= n([NBS]^{(3)}) = (O_{nb})_i (O_{nb})_j (O_{nb})_k
\end{align*}
\]

where

- \(n(A)\): the size (number of elements) of a given set, \(A\),
- \([NB]^{(2)}\): any possible 2D neighbourhood of a block,
- \([NBS]^{(2)}\): the set of all possible 2D neighbourhoods for a block,
- \([NB]^{(3)}\): any possible 3D neighbourhood of a block and
- \([NBS]^{(3)}\): the set of all possible 3D neighbourhoods for a block.

Therefore, a 2D order of neighbourhood of \((2, 2)\) results in four possible neighbourhoods, each in a square pattern of \(2 \times 2\) blocks. Similarly, the neighbourhood order of \((3, 3)\) leads to nine possible neighbourhoods, each consisting of a set of nine
blocks arranged in a square pattern of $3 \times 3$ blocks. Figure 4.4 illustrates all the nine possible 2D neighbourhoods for the block, $B_{44}$, assuming that $[O_{nb}]^{(2)} = (3, 3)$. In general, for a 2D order of neighbourhood of $[O_{nb}]^{(2)} = (l_1, l_2)$, assuming that $l_i \cdot l_j = L^{(2)}$, there are $L^{(2)}$ number of possible neighbourhoods for a given block, each of which contains $L^{(2)}$ number of blocks arranged in an $l_1 \times l_2$ matrix pattern.

Consider a minimum stope size of two blocks in length, two blocks in width, and two blocks in height. The order of neighbourhood may be shown by the set, $[O_{nb}]^{(3)} = (2, 2, 2)$, which results in eight possible neighbourhoods, each in a cubic pattern of $2 \times 2 \times 2$ blocks. Figure 4.5 illustrates all of the eight possible neighbourhoods for any block, $B_{ijk}$, assuming that $[O_{nb}]^{(3)} = (2, 2, 2)$. Figure 4.6 shows the collection of all blocks contributing to the set of 3D neighbourhoods. Similarly, a neighbourhood order of $(4, 3, 3)$ leads to 36 possible neighbourhoods, each of which consists of a set of 36 elements.
in a cubic pattern of $4 \times 3 \times 3$ blocks. In general, for a 3D order of neighbourhood of $[O_{nb}]^{(3)} = (l_b, l_j, l_k)$, assuming that $l_i \cdot l_j \cdot l_k = L^{(3)}$, there are $L^{(3)}$ number of possible neighbourhoods for a given block, each of which contains $L^{(3)}$ number of members in a form of an $l_i \times l_j \times l_k$ matrix.

**Figure 4.5:** Possible 3D Neighbourhoods for the Block, $B_{ijk}$, with $[O_{nb}]^{(3)} = (2, 2, 2)$

### 4.2.2 The Optimum Neighbourhood

In order to make a comparison among the neighbourhoods of a block and to locate the optimum one, all the neighbourhoods of the block are determined. For 2D neighbourhoods, there are $L^{(2)}$ ($L^{(2)} = l_i l_j$) number of possible 2D neighbourhoods for a
given block. These may be numbered sequentially, that is:

\[ [NBS]^{(2)} = \{NB_1, NB_2, ..., NB_{L(2)} \} \]

Figure 4.6: Overlaid 3D Neighbourhoods for the Block, \( B_{ijk} \), with the Order of \((2, 2, 2)\)

The same approach is used to reference 3D neighbourhoods, where there are \(L^{(3)}(L^{(3)} = l_i l_j l_k)\) number of possible 3D neighbourhoods for a given block. That is:

\[ [NBS]^{(3)} = \{NB_1, NB_2, ..., NB_{L^{(3)}} \} \]

The referencing of multi-dimensional neighbourhoods may be reduced to one-dimensional (See Chapter Six). Consider a two-dimensional order of neighbourhood of \([O_{nb}]^{(2)} = (l_i, l_j)\). Any neighbourhood within the set of these 2D neighbourhoods consists of \(l_i\) rows and \(l_j\) columns with \(L^{(2)}\) number of blocks. Figure 4.7 shows a typical (the \(m^{th}\)) neighbourhood of the block, \( B_{ij} \), in which the block, \( B_{ij} \), is located in the \(r^{th}\) row and the \(c^{th}\) column. The general formula to define any 2D neighbourhood \( (NB_m) \) with any order, \((l_i, l_j)\), for any block, \( B_{ij} \), may be expressed by Equation 4.2:
\[
NB_{m,i,j,l} = \left\{ BEV_{\alpha\beta} \in U \mid \begin{align*}
\alpha &= i-r+1, i-r+2, \ldots, i-r+l_i \\
\beta &= j-c+1, j-c+2, \ldots, j-c+l_j
\end{align*} \right\}
\] (4.2)

where

- \( NB_{m,i,j,l} \): the \( m^{th} \) neighbourhood of the block, \( B_{ij} \), with a 2D order of neighbourhood of \((l_i, l_j)\)
- \( \alpha \): the row number of blocks within the model,
- \( \beta \): the column number of blocks within the model,
- \( BEV_{\alpha\beta} \): the economic value of blocks within the \( m^{th} \) neighbourhood of the block, \( B_{ij} \),
- \( U \): the universal set, consisting of all block economic values in the model,
- \( r \): the row number of blocks within the 2D neighbourhood and
- \( c \): the column number of blocks within the 2D neighbourhood.

Using a similar approach, a general formula may be defined to determine any 3D neighbourhood. Consider a general three-dimensional order of neighbourhood, that is, \([O_{nb}]^{(3)} = (l_i, l_j, l_k)\). There are \( L^{(3)} \) number of possible neighbourhood for a given block, \( B_{ijk} \). Each of these neighbourhoods consists of \( l_i \) rows, \( l_j \) columns and \( l_k \) sections with \( L^{(3)} \) number of blocks. It is assumed that the \( m^{th} \) neighbourhood is the one, in which the
block, $B_{ijk}$, is located in the $r^{th}$ row, the $c^{th}$ column and the $s^{th}$ section of the neighbourhood. Therefore, the general formula to define any 3D neighbourhood ($NB_m$) with any order, $(l_i, l_j, l_k)$, for any block, $B_{ijk}$, may be expressed by Equation 4.5:

$$NB_{m,ijk,l_i,l_j,l_k} = \left\{ BEV_{\alpha\beta\gamma} \in U \begin{array}{c}
\alpha = i - r + 1, i - r + 2, \ldots, i - r + l_i \\
\beta = j - c + 1, j - c + 2, \ldots, j - c + l_j \\
\gamma = k - s + 1, k - s + 2, \ldots, k - s + l_k
\end{array} \right\}$$ (4.5)

where

- $NB_{m,ijk,l_i,l_j,l_k}$: the $m^{th}$ neighbourhood of the block, $B_{ijk}$, with a 3D order of neighbourhood of $(l_i, l_j, l_k)$
- $\gamma$: the section number of blocks within the model
- $BEV_{\alpha\beta\gamma}$: the economic value of blocks within the $m^{th}$ neighbourhood of the block, $B_{y}$, and
- $s$: the section number of blocks within the 3D neighbourhood.

The net values of the above neighbourhoods are scalar values. These may be obtained by the summation of all the elements of those neighbourhoods. The neighbourhood value ($NBV$) for a 2D neighbourhood may be calculated using Equation 4.4:

$$NBV_{m,ij,l_i,l_j} = \sum_{\beta = j - c + 1}^{j - c + l_j} \sum_{\alpha = i - r + 1}^{i - r + l_i} BEV_{\alpha\beta\gamma}$$ (4.4)

where

- $NBV_{m,ij,l_i,l_j}$: the net value of the $m^{th}$ neighbourhood for the block, $B_{y}$, where the 2D order of neighbourhood equals $(l_i, l_j)$,
- $r$: the row index of blocks within the 2D neighbourhood and
- $c$: the column index of blocks within the 2D neighbourhood

Similarly, the net value of a 3D neighbourhood is a scalar value obtained by the summation of all the elements of that neighbourhood. It may be calculated using Equation 4.5:
Chapter Four: The 2D and 3D MVN Algorithm 4-12

\[ NBV_{m,ijk,l_1,l_2,l_3} = \sum_{y=k-r+1}^{k+s+1} \sum_{z=c-r+1}^{c+s+1} \sum_{\alpha=i-r+1}^{i+s+1} BEV_{\alpha yz} \]  

(4.5)

where

- \( NBV_{m,ijk,l_1,l_2,l_3} \): the net value of the \( m^{th} \) neighbourhood for the block, \( B_{ijk} \), where the 3D order of neighbourhood equals \( (l_1, l_2, l_3) \),
- \( r \): the row index of the blocks within the 3D neighbourhood,
- \( c \): the column index of the blocks within the 3D neighbourhood and
- \( s \): the section index of the blocks within the 3D neighbourhood.

The above neighbourhood values are collected in 1D sets, \([NBVS]^{(2)}\) for 2D neighbourhoods and \([NBVS]^{(3)}\) for 3D neighbourhoods. These sets are maximised to locate the maximum neighbourhood value \((MNBV)\) and the corresponding neighbourhood, that is, the maximum value neighbourhood \((MVN)\). When the problem of the 2D and the 3D neighbourhoods are reduced to a 1D problem, the 1D process may then be used to complete the \( MVN \) algorithm (See Chapter Three).

4.2.3 Infeasible Neighbourhoods

The feasibility of 2D and 3D neighbourhoods is more restricted when compared with 1D cases. In 1D neighbourhoods, boundary blocks are only considered in one direction at both ends of the block model. However, in 2D neighbourhoods, blocks located at both ends of the two directions (for example, the X and Y axes) are considered boundary blocks. These boundary blocks may present infeasible neighbourhoods. Consider a block model section as shown in Figure. 4.8, where \([O_{nb}]^{(2)} = (2, 2)\). There are four possible neighbourhoods for each block. However, the number of feasible neighbourhoods for blocks located in all four sides of the model is reduced to two neighbourhoods. Blocks located at the four corners of the section produce only one feasible neighbourhood. It should be noted that the feasible neighbourhoods of the blocks located at one side (edge), or corner, of the block model are different from those of blocks located on other sides or corners. For example, the feasible neighbourhoods of boundary blocks located at the top edge are \( NB_1 \) and \( NB_3 \), while the feasible neighbourhoods of blocks located at the bottom edge are \( NB_2 \) and \( NB_4 \). Similarly, the
feasible neighbourhoods of the blocks in the left side are $NB_1$ and $NB_2$, while those of the blocks in the right side of the ore-body model are $NB_3$ and $NB_4$. Figure 4.8 illustrates the feasible neighbourhoods for the boundary blocks of a section where the 2D order of neighbourhood is $(2, 2)$.

![Figure 4.8](image)

**Figure 4.8:** Feasibility of 2D Neighbourhoods for Boundary Blocks where $[Onb]^{(2)} = (2, 2)$

The feasibility of the neighbourhoods in 3D show more restrictions, as blocks located in all six faces of the block model are considered boundary blocks. Any block, located at either of the eight corners, produces only one feasible neighbourhood regardless of its $O_{nb}$. However, the number of non-feasible neighbourhoods for other boundary blocks depends on the order of the neighbourhood. Assuming that $[O_{nb}]^{(3)} = (2, 2, 2)$, blocks in each face of the block model have only four feasible neighbourhoods out of a possible eight. The blocks at the intersection of the faces (the edges) lose two more neighbourhoods. Figure 4.9 illustrates the number of feasible neighbourhoods, for different locations, within the boundary blocks of the ore-body model, for a $(2, 2, 2)$ order of neighbourhood.
Figure 4.9: Feasibility of 3D Neighbourhoods for Boundary Blocks

where \([O_{nb}]^{(3)} = (2, 2, 2)\)

In general, it may be said that the \(m^{th}\) possible neighbourhood \((NB_m)\) of a block with the order of \([O_{nb}]^{(3)} = (l_i, l_j, l_k)\), is infeasible if and only if

\[
\exists BEV_{ijk} \in NB_m \begin{cases} 
  i \leq 0 & \text{or} & i > I \\
  j \leq 0 & \text{or} & j > J \\
  k \leq 0 & \text{or} & k > K
\end{cases}
\]  \hspace{1cm} (4.6)

where

- \(I\): the number of blocks of the model in the X axis,
- \(J\): the number of blocks of the model in the Y axis and
- \(K\): the number of blocks of the model in the Z axis.
4.3 A 2D NUMERICAL EXAMPLE

Consider a section of the economic block model with six rows and ten columns, shown in Figure 4.10. That is,

\[ \{B_{ij} \mid i = 1, 2, \ldots, 6; \quad j = 1, 2, \ldots, 10\} \]

Figure 4.10: A Section of an Economic Block Model with 6 Rows and 10 Columns

Assume that the 2D order of neighbourhood imposes a minimum of two blocks, both in X and Y directions, that is, \( [O_{nb}]^2 = (2, 2) \). The general procedure for optimising the stope section is the same as that explained for the 1D neighbourhood, in Chapter Three. There are 11 major steps in the MVN algorithm as described in Figure 3.12. A step by step display of implementing the MVN algorithm on a 2D neighbourhood problem, for the first two columns of the section example, follows.

START

Step 1: Initialise variables.

Block height = 10 m; \quad \text{Block length} = 15 \text{ m}
Chapter Four: The 2D and 3D MVN Algorithm

Minimum stope height = 15; Minimum stope length = 25
SBR_i = 1.5; SBR_j = 1.67
(O_nb)_i = 2; (O_nb)_j = 2
SEV = 0; F = 0; FF = 0

Step 2: Examine the first block.

B_{11}: BEV_{11} = -3; F_{11} = 0

Step 3: Check the negativity and the flag of the block.

BEV_{11} < 0 \Rightarrow Block B_{11} is exempted from further process.
Go to Step 9.

Step 9: Check the end of the model.

B_{11} is not the last block. \Rightarrow Continue.

Step 10: Examine the next block.

B_{21}: BEV_{21} = -1; F_{21} = 0
Go to Step 3

Step 3: Check the negativity and the flag of the block.

BEV_{21} < 0 \Rightarrow Block B_{21} is exempted from further process.
Go to Step 9.

Step 9: Check the end of the model.

B_{21} is not the last block. \Rightarrow Continue.

Step 10: Examine the next block.

B_{31}: BEV_{31} = 2; F_{31} = 0
Go to Step 3

Step 3: Check the negativity and the flag of the block.

BEV_{31} \geq 0 and F_{31} \neq 1 \Rightarrow Continue.

Step 4: Locate the maximum neighbourhood value (MNBV).

NB_1 = \{2, 3, 2, 0\} \Rightarrow NBV_1 = 2 + 3 + 2 + 0 = 7
NB_2 = \{-1, 2, 4, 2\} \Rightarrow NBV_2 = -1 + 2 + 4 + 2 = 7
NB_3 = Not feasible
Chapter Four: The 2D and 3D MVN Algorithm

NB₄ = Not feasible

⇒ MNBV = 7; MVN = NB₁

Step 5: Check the negativity of the MNBV.

MNBV ≥ 0 ⇒ Continue.

Step 6: Calculate the marginal value.

MVN = NB₁ = {BEV₃₁, BEV₄₁, BEV₃₂, BEV₄₂}

F₃₁ = 0 ⇒ FF₃₁ = 1; F₃₂ = 0 ⇒ FF₃₂ = 1;
F₄₁ = 0 ⇒ FF₄₁ = 1; F₄₂ = 0 ⇒ FF₄₂ = 1;

Marginal Value = BEV₃₁ . FF₃₁ + BEV₃₂ . FF₃₂
+ BEV₄₁ . FF₄₁ + BEV₄₂ . FF₄₂
= (2 x 1) + (3 x 1) + (2 x 1) + (0 x 1) = 7

Step 7: Check the negativity of the marginal value.

Marginal Value ≥ 0 ⇒ Continue.

Step 8: Update the stope.

SEV = SEV + Marginal Value = 0 + 7 = 7 ⇒ SEV = 7
F₃₁ = 1; F₄₁ = 1; F₃₂ = 1; F₄₂ = 1

Step 9: Check the end of the model.

B₃₁ is not the last block. ⇒ Continue.

Step 10: Examine the next block.

B₄₁: BEV₄₁ = 3; F₄₁ = 1

Go to Step 3

Step 3: Check the negativity and the flag of the block.

F₄₁ = 1 ⇒ Block B₄₁ is exempted from further process.

Go to Step 9.

Step 9: Check the end of the model.

B₄₁ is not the last block. ⇒ Continue.

Step 10: Examine the next block.
B_{51}: \ BEV_{51} = -1; \ F_{51} = 0

Go to Step 3

Step 3: Check the negativity and the flag of the block.

BEV_{51} < 0 \Rightarrow \ Block \ B_{51} \ is \ exempted \ from \ further \ process.

Go to Step 9.

Step 9: Check the end of the model.

B_{51} \ is \ not \ the \ last \ block. \Rightarrow \ Continue.

Step 10: Examine the next block.

B_{61}: \ BEV_{61} = 1; \ F_{61} = 0

Go to Step 3

Step 3: Check the negativity and the flag of the block.

BEV_{61} \geq 0 \ and \ F_{31} \neq 1 \Rightarrow \ Continue.

Step 4: Locate the maximum neighbourhood value (MNBV).

NB_{1} = \ Not \ feasible

NB_{2} = \{-1, 1, 1, 4\} \Rightarrow \ NBV_{2} = -1 + 1 + 1 + 4 = 5

NB_{3} = \ Not \ feasible

NB_{4} = \ Not \ feasible

\Rightarrow \ MNBV = 5; \ MVN = NB_{2}

Step 5: Check the negativity of the MNBV.

MNBV \geq 0 \Rightarrow \ Continue.

Step 6: Calculate the marginal value.

MVN = NB_{2} = \{BEV_{51}, \ BEV_{61}, \ BEV_{52}, \ BEV_{62}\}

F_{51} = 0 \Rightarrow \ FF_{51} = 1; \ F_{52} = 0 \Rightarrow \ FF_{52} = 1;

F_{61} = 0 \Rightarrow \ FF_{61} = 1; \ F_{62} = 0 \Rightarrow \ FF_{62} = 1;

Marginal \ Value = BEV_{51} \cdot FF_{51} + BEV_{52} \cdot FF_{52}

+ BEV_{61} \cdot FF_{61} + BEV_{62} \cdot FF_{62}

= (-1 \times 1) + (1 \times 1) + (1 \times 1) + (4 \times 1) = 5

Step 7: Check the negativity of the marginal value.
Marginal Value ≥ 0 → Continue.

Step 8: Update the stope.

\[ SEV = SEV + \text{Marginal Value} = 7 + 5 = 12 \rightarrow SEV = 12 \]

\[ F_{51} = 1; \quad F_{61} = 1; \quad F_{52} = 1; \quad F_{62} = 1 \]

Step 9: Check the end of the model.

B_{61} is not the last block. → Continue.

Step 10: Examine the next block.

B_{12}: BEV_{12} = 3; \quad F_{12} = 0

Go to Step 3

Step 3: Check the negativity and the flag of the block.

BEV_{12} ≥ 0 and F_{31} ≠ 1 → Continue.

Step 4: Locate the maximum neighbourhood value (MNBV).

\[ NB_1 = \{1, 4, 0, 2\} \rightarrow NBV_1 = 1 + 4 + 0 + 2 = 7 \]

NB_2 = Not feasible

\[ NB_3 = \{-3, -1, 1, 4\} \rightarrow NBV_3 = -3 - 1 + 1 + 4 = 1 \]

NB_4 = Not feasible

\[ \rightarrow MNBV = 7; \quad MVN = NB_1 \]

Step 5: Check the negativity of the MNBV.

MNBV ≥ 0 → Continue.

Step 6: Calculate the marginal value.

\[ MVN = NB_1 = \{BEV_{12}, BEV_{22}, BEV_{13}, BEV_{23}\} \]

\[ F_{12} = 0 \rightarrow FF_{12} = 1; \quad F_{13} = 0 \rightarrow FF_{13} = 1; \]

\[ F_{22} = 0 \rightarrow FF_{22} = 1; \quad F_{23} = 0 \rightarrow FF_{23} = 1; \]

Marginal Value = BEV_{12} \cdot FF_{12} + BEV_{13} \cdot FF_{13} + BEV_{22} \cdot FF_{22} + BEV_{23} \cdot FF_{23}

\[ = (1 \times 1) + (0 \times 1) + (4 \times 1) + (2 \times 1) = 7 \]

Step 7: Check the negativity of the marginal value.
Marginal Value $\geq 0 \rightarrow$ Continue.

Step 8: Update the stope.

$$\text{SEV} = \text{SEV} + \text{Marginal Value} = 12 + 7 = 19 \rightarrow \text{SEV} = 19$$

$$F_{12} = 1; \quad F_{22} = 1; \quad F_{13} = 1; \quad F_{23} = 1$$

Step 9: Check the end of the model.

B_{12} is not the last block. $\rightarrow$ Continue.

Step 10: Examine the next block.

B_{22}: \quad BEV_{22} = 4; \quad F_{22} = 1

Go to Step 3

Step 3: Check the negativity and the flag of the block.

$$F_{22} = 1 \rightarrow \text{Block B}_{22} \text{ is exempted from further process.}$$

Go to Step 9.

Step 9: Check the end of the model.

B_{22} is not the last block. $\rightarrow$ Continue.

Step 10: Examine the next block.

B_{32}: \quad BEV_{32} = 2; \quad F_{32} = 1

Go to Step 3

Step 3: Check the negativity and the flag of the block.

$$F_{32} = 1 \rightarrow \text{Block B}_{32} \text{ is exempted from further process.}$$

Go to Step 9.

Step 9: Check the end of the model.

B_{32} is not the last block. $\rightarrow$ Continue.

Step 10: Examine the next block.

B_{42}: \quad BEV_{42} = 0; \quad F_{42} = 1

Go to Step 3

Step 3: Check the negativity and the flag of the block.

$$F_{42} = 1 \rightarrow \text{Block B}_{42} \text{ is exempted from further process.}$$

Go to Step 9.
Step 9: Check the end of the model.

B_{42} is not the last block. \Rightarrow \text{ Continue.}

Step 10: Examine the next block.

B_{52}: \quad \text{BEV}_{52} = 1; \quad F_{52} = 1

Go to Step 3

Step 3: Check the negativity and the flag of the block.

F_{52} = 1 \Rightarrow \text{ Block B}_{52} \text{ is exempted from further process.}

Go to Step 9.

Step 9: Check the end of the model.

B_{52} is not the last block. \Rightarrow \text{ Continue.}

Step 10: Examine the next block.

B_{62}: \quad \text{BEV}_{62} = 4; \quad F_{62} = 1

Go to Step 3

Step 3: Check the negativity and the flag of the block.

F_{62} = 1 \Rightarrow \text{ Block B}_{62} \text{ is exempted from further process.}

Go to Step 9.

Step 9: Check the end of the model.

B_{62} is not the last block. \Rightarrow \text{ Continue.}

The optimisation results of the MVN algorithm, applied to the first two columns of the section example, are shown in Figure 4.11. The last block examined is B_{62}, which is highlighted by bold borders. The complete implementation of the algorithm on the section has resulted in the selection of 51 out of 60 blocks. This is shown in Figure 4.12. In both figures, shaded blocks are flagged to define the stope geometry.
Figure 4.11: Results of Applying the 2D MVN Algorithm to the First Two Columns of the Given Example Section

Figure 4.12: The Section Optimised with the 2D MVN Algorithm
The capability of the algorithm to recover the worth contained in the stope may be assessed using some of the indices discussed in the previous chapter. Based on the definitions, the 2D MVN algorithm, applied to the example shown in Figure 4.10, provides a stope recovery of 71%. The section consists of 60 blocks, with the total value of $49, including 44 positive value blocks with the (maximum recoverable) value of $87 and 16 negative valued blocks with the cost of -$38. The total value of the optimised stope (SEV) equals $62. These results may be summarised as follows.

\[
\begin{align*}
N_{I_n} &= 9 & N_{I_v} &= -$21 \\
PN_n &= 2 & PN_v &= $4 \\
R_n &= 95 \% & R_v &= 95 \% \\
\end{align*}
\]

\[R = 71 \%\]

where

- \(N_{I_n}\): the number of negative value blocks included in the stope,
- \(N_{I_v}\): the value of negative value blocks included in the stope,
- \(PN_n\): the number of positive value blocks included in the stope,
- \(PN_v\): the value of positive value blocks included in the stope,
- \(R_n\): the ratio of positive value blocks recovered (by number),
- \(R_v\): the ratio of positive value blocks recovered (by value) and
- \(R\): the ratio of the stope value to the maximum recoverable value of the model (total value of positive blocks).

### 4.4 Implementation of the 3D MVN Algorithm Using the VBA Code

The 3D MVN algorithm was coded using the Visual Basic for Applications (VBA) modules (Walkenbach, 1994). The advantage of the VBA software is its capability to provide easy access to the object model in Excel. The structure of the spreadsheet used in Excel is highly suitable to the storage and the display of the economic block model. The different cross-sections of the economic model may be defined by the Excel worksheets. Each block within a section of the model may be represented by a cell in the corresponding worksheet. The cell values define the respective block economic value. The system of cell addressing, in the Excel worksheets, is the same as that used in this thesis. That is, in each section (worksheet) the rows of blocks (cells) are arranged
Figure 4.13: Flow-Chart of the Main Program Written in VBA
When the model specification button is clicked, the user is provided with a window to define the specifications of the block model, such as the 3D co-ordinates of the origin of the model, plus the size of the model and the blocks. Figure 4.15 shows the dialog box for entering the parameters of the model.

As an example, consider a very small economic model as shown in Figure 4.16, which consists of six rows, eight columns and six sections. That is,

\[ \{B_{ijk} \mid i = 1, 2, ..., 6; \quad j = 1, 2, ..., 8; \quad k = 1, 2, ..., 6\} \]
The number of blocks in X, Y and Z directions are 6, 8 and 6, respectively. These are input, directly, to the corresponding fields of the dialog box, and saved in three integer variables. The VBA code to read the model’s specifications, entered by the user, is as follows:

```vba
ThisWorkbook.DialogSheets("Dialog3").Show
NI = DialogSheets("Dialog3").EditBoxes("NBXEdit").Text
NJ = DialogSheets("Dialog3").EditBoxes("NBYEdit").Text
NK = DialogSheets("Dialog3").EditBoxes("NBZEdit").Text
```
The next step input requirement is the definition of the mining constraints, that is, the minimum stope geometry. Selecting the second option from the main menu (clicking the "Define Mining Constraints" button) provides a new window that supports a dialog box for entering the mining constraint data, as shown in Figure 4.17. It is possible to define the dimensions of the minimum stope, and that of the fixed blocks, to calculate the "stope block ratio" (SBR) and the "order of neighbourhood" (Onb), or directly enter the order of neighbourhood in terms of the minimum allowable number of blocks in the stope, in the X, Y and Z directions. As an example, assume that the three-dimensional order of neighbourhood imposes a minimum of two blocks in either of X, Y and Z directions, that is, \([O_{nb}]^3(3) = (2, 2, 2)\), as entered directly to the corresponding fields in Figure 4.17. The related VBA code to read the order of neighbourhood is as follows:
Selecting the third option from the main menu (clicking on the "Read Economic (Assay) Data" button) opens another window which allows the block data to be read, as shown in Figure 4.18. The program can read the economic values of the blocks in an ASCII data file, or Excel worksheet. It is also possible to enter these data manually. The following lines of VBA code are used to read data from the Excel worksheets:
For \( k = 1 \) To \( NK \)
For \( j = 1 \) To \( NJ \)
    For \( i = 1 \) To \( Nl \)
        \[ F(i, j, k) = 0 \]
        \[ BEV(i, j, k) = \text{Workbooks} (\text{EFName}).\text{Worksheets} (k).\text{Cells}(i, j) \]
    Next \( i \)
Next \( j \)
Next \( k \)

where \( \text{EFName} \) is the name of the Excel file, in which the economic block model data are stored.

Figure 4.18: The Dialog Box Providing Options for Reading Economic Block Data

Selecting the fourth option from the main menu (clicking on the "Perform Optimisation" button), provides another window which allows the user to define a file name for writing the results and to start the optimisation. The first subroutine that is
called by the main program is "MaxNbor". This is the most critical subroutine of the system. The main objective of which is to examine a block and define the maximum value neighbourhood (MVN) of that block. Figure 4.19 illustrates the flow-chart of the "MaxNbor" subroutine.

![Flow-Chart of the Subroutine "MaxNbor"](image)

**Figure 4.19:** Flow-Chart of the Subroutine "MaxNbor"
The "MaxNbor" subroutine, firstly, assigns a large negative value to the $MNBV$ variable. It then determines the size of the neighbourhood set, based on the 3D order of neighbourhood. The size of the neighbourhood set defines the number of repetitions of the process that evaluates each neighbourhood. The subroutine determines the relative location of each neighbourhood which is represented by a set of blocks. The first and the last blocks of the set (blocks with the lowest $i, j, k$ address and the highest one within a neighbourhood, respectively) determine the relative location of the neighbourhood. The following VBA code is used to locate the address of the first block in the $m^{th}$ neighbourhood.

\[
\begin{align*}
a &= \text{Int}(m / (ONBi * ONBj)) \\
b &= \text{Int}(bc / ONBi) \\
c &= bc \mod ONBi \\
If \ c \neq 0 \ Then \\
& \quad FBI = i - b: \\
& \quad FBJ = j - (c - 1): \\
& \quad FBK = k - a \\
ElseIf \ b \neq 0 \ Then \\
& \quad FBI = i - (b - 1): \\
& \quad FBJ = j - (nm - 1): \\
& \quad FBK = k - a \\
Else \\
& \quad FBI = i - (nn - 1): \\
& \quad FBJ = j - (nm - 1): \\
& \quad FBK = k - (a - 1)
\end{align*}
\]

where $FBI$, $FBJ$ and $FBK$ are the $i$, $j$ and $k$ addresses of the first block of the neighbourhood, respectively. The VBA code to locate the last block is as follows:

\[
\begin{align*}
LBI &= FBI + mm - 1 \\
LBJ &= FBJ + nn - 1 \\
LBK &= FBK + pp - 1 \\
\end{align*}
\]

where $LBI$, $LBJ$ and $LBK$ are the $i$, $j$ and $k$ addresses of the last block of the neighbourhood, respectively. Each neighbourhood is examined to check if it lies within the block model limits, that is, whether or not the neighbourhood is feasible. The
feasibility of the neighbourhoods is studied using the following lines of the VBA code:

\[
\text{If } \text{FBI} < 1 \text{ Or } \text{LBI} > \text{NI} \text{ Then } \text{NB}(m) = "-"
\]
\[
\text{If } \text{FBJ} < 1 \text{ Or } \text{LBJ} > \text{NJ} \text{ Then } \text{NB}(m) = "-"
\]
\[
\text{If } \text{FBK} < 1 \text{ Or } \text{LBK} > \text{NK} \text{ Then } \text{NB}(m) = "-"
\]

The total net values of the neighbourhoods (neighbourhood value, \(NBV\)) are then obtained for the feasible neighbourhoods while the non-feasible neighbourhoods are ignored. This is determined by the summation of the block economic values inside the neighbourhood. Below is the VBA code used to accomplish this task.

\[
\text{NB}(m) = 0
\]
\[
\text{For Section = FBK To LBK}
\]
\[
\text{For Column = FBJ To LBJ}
\]
\[
\text{For Row = FBI To LBI}
\]
\[
\text{NB}(m) = \text{NB}(m) + \text{BEV}(\text{Row}, \text{Column}, \text{Section})
\]

\[
\text{Next Row}
\]
\[
\text{Next Column}
\]
\[
\text{Next Section}
\]

A comparison is made and the \(MNBV\) (the maximum neighbourhood value) variable is updated to the maximum of the previous \(MNBV\) and the obtained neighbourhood value, \(NBV_m\), as follows:

\[
\text{MNBV} = \text{Application.Max}(\text{MNBV}, \text{NB}(m))
\]

The process is repeated for all the neighbourhoods of the block and finally the value of the \(MNBV\) variable is returned to the main program.

If the obtained \(MNBV\) is not negative, another subroutine, "SelectBlock", is called to determine which of those blocks of the maximum value neighbourhood, \(MVN\), are new to the stope and contribute effectively to the final stope and stope value. The following lines of the VBA code provides this facility:
For c = FBK To LBK
  For b = FBJ To LBJ
    For a = FBI To LBI
      If F(a, b, c) = 1 Then FF(a, b, c) = 0
      If F(a, b, c) = 0 Then F(a, b, c) = 1; FF(a, b, c) = 1
      Next a
    Next b
  Next c

The blocks with an FF value of "1" are new to the stope and provide a marginal value if the obtained MVN is included in the stope. Next subroutine, "StopeVal", is then used to update the net value of the stope with the following lines of the VBA code:

For c = FBK To LBK
  For b = FBJ To LBJ
    For a = FBI To LBI
      SEV = SEV + FF(a, b, c) * BEV(a, b, c)
    Next a
  Next b
Next c

The summary of the optimisation is provided in Figure 4.20. As the optimisation progresses, the (i, j, k) address of the block under consideration is shown instantly, and the stope value variation is instantly displayed with the stope value variation. When the optimisation is completed, the address of the last block within the economic model, B686, is displayed as well as the optimum stope value ($390).

Finally, the "Flag" subroutine is executed. The subroutine examines the flag indicator, F, of each block to check whether the flag is "1" or "0". If a block has an F value of "1", the corresponding cell in the Excel worksheets is filled with yellow colour, the font colour of the cell value, the BEV, is made red and a border is drawn around the cell. The stope value is also recalculated for double-checking. The corresponding VBA code to accomplish this task is provided below:
The Block under Process Is 6 8 6
The Stope Economic Value Is $390
Calculations Completed.
Now, It is Flagging the Selected Blocks.

<table>
<thead>
<tr>
<th>No.</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Included Negative Blocks:</td>
<td>60 -84</td>
</tr>
<tr>
<td>Excluded Non-Negative Blocks:</td>
<td></td>
</tr>
</tbody>
</table>

Flagging Completed.
Optimisation Completed. To View the Results, Back to Main Menu.

Figure 4.20: The "Perform Optimisation" Window

\[
SValue = 0
\]

For \( k = 1 \) To \( NK \)
  For \( j = 1 \) To \( NJ \)
    For \( i = 1 \) To \( NL \)
      Workbooks(MyResults).Worksheets(k).Cells(i,j)=BEV(i,j,k)
      Select Case \( F(i, j, k) \)
        Case 1
          Workbooks(MyResults).Worksheets(k).Cells(i,j).Name = "Flagged"
          Workbooks(MyResults).Worksheets(k).Range("Flagged").Interior.ColorIndex = 6
          Workbooks(MyResults).Worksheets(k).Range("Flagged").Borders.ColorIndex = 3
          SValue = SValue + BEV(i,j,k)
        
Next i
Next j
Next k

The number of negative value blocks included in the final stope, along with their associated value (cost), plus the number of non-negative value blocks excluded from the
final stope along with their associated value, are obtained. As the "Perform Optimisation" window shows, a total of 60 negative value blocks, with a total cost of $84, have been included in the optimised stope. No positive value block is missed from the final stope. The optimised stope includes 210 out of a possible 288 blocks of the block model.

The optimisation results may be displayed by selecting the "View the Results" button from the main menu. Figure 4.21 illustrates the second section (k = 2) of the final (optimised) stope, with the selected blocks shaded. Table 4.1 summarises the intermediate results of the optimisation algorithm for the model example.

The capability of the algorithm to recover the stope worth may be assessed through the following indices, defined in Chapter Three. Based on those definitions, the 3D MVN algorithm, applied to the example shown in Figure 4.16, provides a total stope recovery of 82%. The example model consists of 288 blocks, including 150 non-negative value blocks, with the maximum recoverable value of $474, plus 138 negative value blocks with the cost of $414. The capabilities of the algorithm are summarised as below.

\[
\begin{align*}
NI_n &= 60 \\
PN_n &= 0 \\
R_n &= 100 \% \\
NI_v &= -$84 \\
PN_v &= $0 \\
R_v &= 100 \% \\
\text{Stope Recovery: } R &= 82 \%
\end{align*}
\]
Changing the order of neighbourhood, for the same stope example, results in different ultimate stope boundaries being obtained. If a neighbourhood of order \((3, 3, 3)\), which makes 27 possible neighbourhoods for each block is imposed, then a significant decrease in the size and value of the optimum stope occurs. That is, the optimised stope contains 156 blocks out of a possible 288 with a total stope value of $228 (Figure 4.22a). However, the neighbourhood order of \((2, 3, 4)\) with 24 possible neighbourhoods, causes the optimum stope to include 192 blocks with a total value of $222 (Figure 4.22b) whereas the stope example optimised, due to a neighbourhood order of \((3, 2, 3)\) with 18 possible neighbourhoods, contains 180 blocks with a total value of $240 (Figure 4.22c).
Table 4.1: Results* Summary for Blocks in Section 2 of the Example

<table>
<thead>
<tr>
<th>Block address (i, j, k)</th>
<th>BEV ($\text{$})$</th>
<th>No. of feasible neighbourhoods</th>
<th>ID of NB with $MVN$</th>
<th>$MVN$ ($\text{$})$</th>
<th>Stope Value ($\text{$})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 1, 2</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>28</td>
<td>144</td>
</tr>
<tr>
<td>2, 1, 2</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>28</td>
<td>144</td>
</tr>
<tr>
<td>3, 1, 2</td>
<td>-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4, 1, 2</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>147</td>
</tr>
<tr>
<td>5, 1, 2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>148</td>
</tr>
<tr>
<td>6, 1, 2</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1, 2, 2</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>28</td>
<td>148</td>
</tr>
<tr>
<td>2, 2, 2</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3, 2, 2</td>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4, 2, 2</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5, 2, 2</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6, 2, 2</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>16</td>
<td>155</td>
</tr>
<tr>
<td>1, 3, 2</td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2, 3, 2</td>
<td>-7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3, 3, 2</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>18</td>
<td>164</td>
</tr>
<tr>
<td>4, 3, 2</td>
<td>3</td>
<td>8</td>
<td>1</td>
<td>18</td>
<td>164</td>
</tr>
<tr>
<td>5, 3, 2</td>
<td>3</td>
<td>8</td>
<td>3</td>
<td>20</td>
<td>Negative margin</td>
</tr>
<tr>
<td>6, 3, 2</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>16</td>
<td>Negative margin</td>
</tr>
<tr>
<td>1, 4, 2</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>18</td>
<td>174</td>
</tr>
<tr>
<td>2, 4, 2</td>
<td>2</td>
<td>8</td>
<td>3</td>
<td>18</td>
<td>174</td>
</tr>
<tr>
<td>3, 4, 2</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4, 4, 2</td>
<td>5</td>
<td>8</td>
<td>3</td>
<td>22</td>
<td>181</td>
</tr>
<tr>
<td>5, 4, 2</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6, 4, 2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>181</td>
</tr>
<tr>
<td>1, 5, 2</td>
<td>7</td>
<td>4</td>
<td>2</td>
<td>18</td>
<td>181</td>
</tr>
<tr>
<td>2, 5, 2</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3, 5, 2</td>
<td>6</td>
<td>8</td>
<td>1</td>
<td>28</td>
<td>188</td>
</tr>
<tr>
<td>4, 5, 2</td>
<td>1</td>
<td>8</td>
<td>3</td>
<td>28</td>
<td>188</td>
</tr>
<tr>
<td>5, 5, 2</td>
<td>4</td>
<td>8</td>
<td>4</td>
<td>18</td>
<td>192</td>
</tr>
<tr>
<td>6, 5, 2</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1, 6, 2</td>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2, 6, 2</td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3, 6, 2</td>
<td>3</td>
<td>8</td>
<td>2</td>
<td>28</td>
<td>192</td>
</tr>
<tr>
<td>4, 6, 2</td>
<td>4</td>
<td>8</td>
<td>2</td>
<td>28</td>
<td>192</td>
</tr>
<tr>
<td>5, 6, 2</td>
<td>-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6, 6, 2</td>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1, 7, 2</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>193</td>
</tr>
<tr>
<td>2, 7, 2</td>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3, 7, 2</td>
<td>-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4, 7, 2</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5, 7, 2</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6, 7, 2</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>195</td>
</tr>
<tr>
<td>1, 8, 2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>195</td>
</tr>
<tr>
<td>2, 8, 2</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>195</td>
</tr>
<tr>
<td>3, 8, 2</td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4, 8, 2</td>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5, 8, 2</td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6, 8, 2</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>195</td>
</tr>
</tbody>
</table>

* The block model is $6 \times 8 \times 6$ and the order of neighbourhood is $(2, 2, 2)$. 
### Figure 4.22a: Optimum Stope Example, \([k = 2, O_{nb} = (3, 3, 3)]\)

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

### Figure 4.22b: Optimum Stope Example, \([k = 2, O_{nb} = (2, 3, 4)]\)

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

### Figure 4.22c: Optimum Stope Example, \([k = 2, O_{nb} = (3, 2, 3)]\)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>-2</td>
<td>1</td>
<td>7</td>
<td>-5</td>
</tr>
<tr>
<td>4</td>
<td>-1</td>
<td>-7</td>
<td>2</td>
<td>-1</td>
<td>-2</td>
</tr>
<tr>
<td>-6</td>
<td>-5</td>
<td>2</td>
<td>-1</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>-1</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>-1</td>
<td>3</td>
<td>-1</td>
<td>4</td>
<td>-6</td>
</tr>
<tr>
<td>-1</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>-3</td>
<td>-5</td>
</tr>
</tbody>
</table>
4.5 SUMMARY

In this chapter, the basics of the 1D neighbourhood concept were extended, to suit 2D and 3D neighbourhoods. The stope block ratio and the order of neighbourhood were redefined to cover all dimensions and were collected in an ordered set, rather than a single scalar. A general formula was presented to determine any neighbourhood, of any block, with any order of neighbourhood. The required formula to calculate the neighbourhood value was also adjusted for 2D and 3D neighbourhoods. More restrictions are imposed on the feasibility of the neighbourhoods of the blocks located at the model boundaries, in 2D and 3D cases. These were illustrated and a general rule for identifying infeasible neighbourhoods was presented.

The block-oriented optimisation procedure (the $MVN$ algorithm), described in the previous chapter, was implemented on numerical examples. A step by step manual employment of the algorithm to optimisation, of a 2D neighbourhood example, was presented in detail. The implementation of the 3D $MVN$ algorithm, using $VBA$ modules in Excel, was described. The $VBA$ code of the algorithm, which benefits from the framework of the Excel worksheets is suitable for small sized examples to test the methodology and demonstrate how the algorithm works. However, in order to be able to apply the $MVN$ algorithm on real mine data, it is necessary to code the algorithm using a general programming language such as $Fortran$, as presented in the following chapter.
CHAPTER FIVE

IMPLEMENTATION OF THE MAXIMUM VALUE NEIGHBOURHOOD ALGORITHM

5.1 INTRODUCTION

This chapter provides an overview of the Stope Limit Optimiser (SLO) system. The Visual Basic for Applications (VBA) code, as described in the previous chapter, is not suitable for large sized block models. In order to implement the MVN algorithm to real sized data, it is necessary that the algorithm be coded using a high level programming language. The Fortran 90 programming language was used to code the MVN algorithm. The computerised program, developed for the algorithm, should be user friendly and as such, features of Fortran 90 - Winteracter were used to develop a Windows based user interface system.

5.2 THE SLO PROGRAM COMPONENTS

The application program developed in this thesis, the SLO system, was integrated with: the latest version of Fortran 90 (Lahey Computer Systems, Inc., 1997a and Lahey Computer Systems, Inc., 1997b), a programming language traditionally used in engineering fields; and Winteracter (Interactive Software Services Ltd., 1998), a user interface developer, which provided the necessary menus and dialog boxes. The features
associated with Fortran 90 have considerably improved the capabilities of the language. Those pertinent to the study include: the definition of allocatable (dynamic) arrays; case structure; and use of modules. Winteracter supports the user interface features of the SLO system. Winteracter provides a Microsoft Windows environment for the development of software by providing menus and dialog boxes, through which the user can select options, input required data and retrieve outputs.

Any Winteracter based program consists of two parts, a source code and a resource script. The source code contains all Fortran 90 statements, including: control; specification; program structure; and assignment statements, arranged in the required sequential order. The source code is saved as a file with the extension ".f", ".for" or ".f90". The Fortran 90 compiler is used to compile the source code to produce the executable file. The resource script is an ASCII based format that describes facilities used to provide options to the user. These include dialogs, menus, icons and bitmaps. These scripts are saved in a file with the ".rc" extension and compiled using the resource compiler supplied with the Fortran 90 compiler. The resulting file is then linked into an executable file using a linker. The resource data is available to Winteracter at a runtime.

The source code contains a large number of "CALL" Fortran 90 statements. These call for the Winteracter subroutines, used for managing windows, messages, menus and dialogs. For example, a number of dialog-handling subroutines are called to load, select, display, hide and unload a dialog box. Separate Winteracter subroutines are also called for assigning data to, or retrieving data from, real, integer or string fields.

5.3 THE WINTERACTER RESOURCE SCRIPT DESCRIPTION

Winteracter version 1.15 was used to supply user-interface capabilities for the SLO software. There are two visual development tools in Winteracter, which allows menus and dialogs to be created interactively. These are the MenuEd and the DialogEd tools. When using these tools, there is no need to edit the resource file directly as the script
will be created and maintained automatically.

5.3.1 The SLO Menus

MenuEd is the Winteracter menu resource editor, which allows the user to create menu resource scripts for use with any Microsoft RC compatible resource compiler. Each menu item is identified by an item ID and has the opportunity to be: a popup menu; checked; greyed; or a separator. The assignment of the accelerator key(s) for a menu item is provided. The SLO program was designed to have only one menu, with the following seven main items, Project, Edit, Preoptimisation, Run, Results, Tools and Help.

- The Project menu item consists of six options: “New”; “Open”; “Save”; “Save as”; “Close”; and “Exit”.
- The Edit menu item includes options for editing: “Block Model”; “Sub-regions”; “Stope Geometry Constraints”; “Economic Factors; and the “Assay Data”.
- The Preoptimisation menu item contains options for “Data Preparation”, “Select Region” and “Import Block Data”. The "Data Preparation" item is a popup menu with two options, "Block Grade Values" and "Block Economic Values".
- The Run menu item contains two options: “Optimise”; and “Second Pass”.
- The Results menu item includes: “Export Flag Data”; "Summary Report"; "Intermediate Results”; “Neighbourhood Results”; “Test Results”; Plot Plans/Sections”; and “View Plots” options.
- The Tools menu item contains options for the conversion of “XZY co-ordinates into block IJK”, block IJK into XYZ co-ordinates”, “Grade into Dollar Values”, “block IJK into block ID” and “block ID into block IJK”.
- The Help menu item consists of two options: “Contents”; and “About SLO".
5.3.2 The SLO Dialogs

DialogEd is the Winteracter dialog resource editor, which allows the user to create dialog resource scripts for use with any Microsoft RC compatible resource compiler. Dialogs are the main Windows method of obtaining information from the user. This is achieved via supplying several types of fields to assign and retrieve data by the user. The dialog fields used in the SLO include:

- **Labels**: Label fields are usually used to label other fields and display fixed text.

- **Pictures**: Picture/Frame fields are rectangular regions in dialogs, used to display a bitmap, icon or frame. The logo or icon of the program can be displayed in a picture/frame field.

- **Strings**: String fields are ordinary enterable text fields.

- **Numeric**: Numeric fields are used to enter or display integer or real values.

- **Push Buttons**: Push button fields are buttons, which can be pressed by clicking them with the mouse. They are usually used to close a dialog and/or select further options, such as, executing a subroutine.

- **Check Boxes**: Check box fields are labelled boxes, which may be either empty or checked. They are generally used for yes/no choices.

- **Radio Buttons**: Radio button fields are generally used for choosing between a small number of options, which are mutually exclusives.

- **Menu Fields**: A list of options may be listed and displayed in a menu field to allow the user to choose one. In the SLO program, drop down menus have been used in several dialogs, for example, to choose a price unit for a product.

- **Tab Controls**: A tab dialog is used to display multiple sub-dialogs within a dialog. Tab controls are not used for data entry. They are treated as dialog fields by DialogEd.

Each field must be identified by an ID, which is used to refer to the field in the Fortran 90 program. Each field is defined by its X and Y position as well as its width and height.
A field may have an initial state or value, be disabled or be grouped. It is also possible, depending on the field type, to define the alignment of the field, supply horizontal and vertical scrollbars, make the field read-only or multi-line, define a password or add a spinner.

In developing the SLO program, a total of 25 dialogs were defined. Some of these dialogs are sub-dialogs accessed through tab controls. The description of these dialogs is provided in Appendix B (See Table B.1). The resource scripts used to develop the SLO program, which contains the above defined menu and dialogs, are saved in the “resource.rc” file.

5.4 THE FORTRAN 90 SOURCE CODE DESCRIPTION

In order to code the SLO system, Fortran 90 version 4.5 was used. The main features of the language used in the SLO source code include:

- definition and use of modules,
- declaration and use of dynamic arrays,
- input/output management,
- the main program and a number of sub-programs and
- arithmetic calculations and logical decisions.

5.4.1 The SLO Input/Output Files

Fortran 90 files are either formatted (stored as Character data) or unformatted (stored as binary data). Either of these are sequential or direct access. Various statements in Fortran 90 were used to perform input/output of the system. These include statements to connect (OPEN) or disconnect (CLOSE) files; transfer data (PRINT, READ and WRITE), establish the position within a file (BACKSPACE and REWIND) and inquire about a file or its connection (INQUIRE).
A total of eleven input/output files were used in the SLO program (six files for inputs and five files for outputs). The name of these files depend on the project that is optimised by the SLO, however, these files are specified by their extensions. Table 5.1 summarises input/output files used in the SLO program.

### Table 5.1: Description of Input/Output Files used in the SLO Program

<table>
<thead>
<tr>
<th>#</th>
<th>File</th>
<th>Description</th>
<th>#</th>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>*.mpr</td>
<td>An ASCII formatted file to store block model parameters</td>
<td>1</td>
<td>*.out</td>
<td>An ASCII formatted file to store the block flag data</td>
</tr>
<tr>
<td>2</td>
<td>*.cst</td>
<td>An ASCII formatted file to store stope geometry constraints</td>
<td>2</td>
<td>*.res</td>
<td>An ASCII formatted file to store the intermediate results</td>
</tr>
<tr>
<td>3</td>
<td>*.eco</td>
<td>An ASCII formatted file to store economic factors</td>
<td>3</td>
<td>*.nbv</td>
<td>An ASCII formatted file to store neighbourhood results</td>
</tr>
<tr>
<td>4</td>
<td>*.dat</td>
<td>An ASCII formatted file to store block assay (economic) data</td>
<td>4</td>
<td>*.plt</td>
<td>An ASCII formatted file to store the plotted plans/sections</td>
</tr>
<tr>
<td>5</td>
<td>*.asy</td>
<td>A binary formatted file to store block assay data</td>
<td>5</td>
<td>*.tst</td>
<td>An ASCII formatted file to store optimisation transactions</td>
</tr>
<tr>
<td>6</td>
<td>*.bev</td>
<td>A binary formatted file to store block economic values data</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

All the above input/output files are formatted sequential files, except for "*.asy" and "*.bev". These are unformatted direct access files and intermediate files which may be considered both input and output files.

### 5.4.2 The SLO Main Program

The main program, in the SLO program, creates the SLO window and the related menu. It responds to any Windows message, entered by the user. The major steps taken by the main program to manage the entire system are as follows:

1. The main program calls the required user defined Fortran modules.
2. The required variables and their types are declared.
3. Winteracter is initialised.
4. Specifications of the SLO window is defined to contain the system menu on the title bar, the minimise button, the maximise button and the status bar.
5. The position of the SLO window on the screen is specified to be centred at 80% of the screen size.
6. The program identifies the menu, defined already in the resource script, to be attached to the SLO window. The initial window title is specified as well.
7. The SLO window is opened and displayed on the screen.
8. A welcome dialog is displayed. The dialog contains a symbolic picture of a miner (as the SLO logo) together with a piece of information about the development of the SLO optimiser. The welcome dialog disappears after three seconds.
9. A subroutine is called to initialise the variables of the different categories collected in various modules.
10. The system is enabled to report if the user changes the cursor to a new dialog field, or changes to a new sub-dialog.
11. An end less loop is provided to control the user input through the keyboard or the mouse. The SLO system waits for the "messages" input by the user and calls corresponding subroutines provided in the program, accordingly. Exit from the loop occurs when close is requested.
12. Finally, if the control exits from the loop, the window is closed and the program is terminated.

In Windows terminology, a "message" is any form of user input, including menu selection, key presses, and mouse clicks. The endless loop of the program provides five categories of Windows messages to respond appropriately, via supplied sub-routines. These include:

- **Menu Selection:** The subroutine "MenuSelectOptions" collects all menu item options (defined in the resource script) and directs the program control to perform the related sub-programs.
• **Pressing Push Buttons**: Each push button, defined in various SLO dialogs has been designed to perform a separate activity. The subroutine “*PushButtonOptions*” collects all these options to direct the control to the required sub-program.

• **Changing Dialog Fields**: On some occasions, it is necessary to perform calculations and update some variables as the user moves from, or to, a dialog field. The subroutine “*FieldChangedOptions*” controls the program in these cases.

• **Changing Tabbed Dialogs**: In the case of tabbed dialog, various sub-dialogs are activated and displayed when their tab control is selected. The subroutine “*TabChangedOptions*” performs all the required actions in this regard.

• **Requesting Window Close**: When a window message requests closure of the window, the program control exits the endless loop (no extra subroutine is required). Window closure may be requested via the system menu, the exit icon on the right top corner of the window, or by the Alt+F4 key.

Figure 5.1 shows the main parts of the main program in the source code, which manages the above steps.

### 5.4.3 The SLO Sub-programs

During the development of the SLO program, a total of 97 subroutines/functions have been defined in the source code. These may be called by the main program or other subroutines. Each of these sub-programs may, in turn, call other user defined functions/subroutines, the *Winteracter* subroutines and/or the *Fortran 90* intrinsic procedures, including numeric, character and array functions. The list and description of all the subroutines and functions defined in the SLO program are provided in *Appendix B* (See Tables B.2 and B.3).
PROGRAM Stope
  
  CALL WInitialise(' ')
  Window%Flags = SysMenuOn + MinButton + MaxButton + StatusBar
  Window%X = -1
  Window%Y = -1
  Window%Width = 0
  Window%Height = 0
  Window%MenuID = IDR_MENU1
  Window%Title = "ST O P E L I M I T S O P T I M I S E R"
  
  CALL WindowOpen(Window)
call WMenuRoot(0)
  CALL WDialogLoad(IDD_Welcome)
  CALL WDialogShow(-l, -1, 0, 2)
  
  CALL WDialogPutString(IDF_STRING1, TRIM(TextBuf))
  CALL IOSWait(300) ! Wait 3 seconds
  CALL WDialogUnLoad
  Titel = "Stope Limits Optimiser - [No Projects ]"
  CALL WindowTitle(Titel)
  CALL WMenuRoot(IDR_MENU1)
  CALL InitialiseSLO()
  
  ! Main message loop
  CALL WMessageEnable(FieldChanged, Enabled)
  CALL WMessageEnable(TabChanged, Enabled)
  DO
    CALL WMessage(IType, Message)
    SELECT CASE (IType)
      CASE (MenuSelect)
        CALL MenuSelectOptions(Message, Finish)
      CASE (PushButton)
        CALL PushButtonOptions(Message)
      CASE (FieldChanged)
        CALL FieldChangedOptions(Message)
      CASE (TabChanged)
        CALL TabChangedOptions(Message)
      CASE (CloseRequest)
        EXIT
    END SELECT
  END DO
  CALL WindowClose()
STOP
END PROGRAM Stope

Figure 5.1: Example Source Code of the Main Program of the SLO Program

5.5 THE SLO GENERAL PROCEDURE

The Stope Limit Optimiser (SLO) considers jobs for optimisation as projects. A project is a collection of input files that specify the block model parameters, stope geometry
constraints and economic factors. The SLO provides an interactive environment for
the user to define projects and import block data for the optimisation process.
Optimisation may be performed on the whole, or a sub-region of, the block model.
Figure 5.2 shows a simplified flow-chart of the SLO program. The general performance
of the program can be divided into three stages. These are the input stage, the
optimisation stage and the output stage.

During the input stage, all input data, including the block model definition, stope
constraints, economic factors and the block data are edited and prepared for use by the
optimisation stage. Each input data type is saved in a separate file for further use. In
cases where block data contain only block grade values, block economic values (BEV)
may be calculated from grade values by the SLO. The final product of the input stage
however, is an economic block model (a 3D array containing the economic values of the
blocks) together with the 3D order of neighbourhood.

The optimisation stage is the core of the whole program. At this stage, a sub-region is
specified for optimisation, which is performed on the block model in accordance with
the order of neighbourhood and based on the previously described MVN algorithm. The
optimisation stage receives the three-dimensional array of block economic values, as
well as the order of neighbourhood, and produces a three-dimensional array of block
flag data.

The output stage includes all processes concerning the visualisation of the optimisation
results. At this stage, the SLO receives the output of the optimisation stage, that is, the
3D array of block flags, arranges them in a plan or section order, wraps them in a table
format with suitable annotations and finally displays the optimised stope layout in an
ASCII formatted file. Alternatively, the SLO exports the flag data directly into an ASCII
formatted file, accessible to other computer packages, which have been developed for
2D and 3D display of results. Minex and Datamine are examples of such packages. The
output stage also includes the display of all the reports and intermediate results collected
throughout the optimisation stage.
Chapter Five: Implementation of the MVN Algorithm

Figure 5.2: Generalised Flow-Chart of the Stope Limit Optimiser (SLO)
5.6 THE SLO PROJECTS

There are two types of data used in the SLO program. These are processed data and auxiliary data. Processed data is the main (block) data processed by the optimisation core to produce the flag data, and is specified with the ".dat" extension. Auxiliary data is that which is not processed during optimisation but rather used to define the environment in which the main data is processed. This includes that data required to define the structure of the block model, the stope geometry constraints and the economic factors applied to the ore-body.

Auxiliary data for each job is defined in three different files, which are collected in a project by the SLO. A Project is a collection of:

```
"*.MPR" file (the Model PaRameter file),
"*.CST" file (the geometry ConSTraint file), and
"*.ECO" file (the ECOnomic factors file).
```

All information concerning the definition of a block model and its parameters, such as the model origin and dimensions, are collected and saved in a file with the "mpr" extension. All information defining stope geometry constraints, such as the minimum stope size and the order of neighbourhood in each dimension, is saved in a "cst" file. All information related to economic factors which applies to the ore-body, including the price of the product(s) and the costs of mining is saved in a file with the extension of "eco". A project itself is not a file and does not appear in file directories, however, it specifies the name of the above set of files. A project is created, saved, closed or opened, that is, its corresponding files ("mpr", "cst" and "eco") are created, saved, closed or opened. In fact, throughout this thesis, whenever a project is referenced, the above three files are implied. Figure 5.3 shows the "Project" vertical menu options from the main menu in the SLO program.
Projects are titled without any extensions. Any file name with a maximum of 15 characters is acceptable. The project name is defined by the user, however, the names of the corresponding files are automatically created by the SLO. This means that the SLO assigns the name of the project to all of the attached three files with their specified extensions. For example, if the user titles a project "MyProject", the model parameters file is called: "MyProject.mpr", the mining constraints file is called "MyProject.cst" and the economic factors file is called "MyProject.eco".

The user may select an existing project to open, edit, perform optimisation and/or save it as a new project. Figure 5.4 illustrates the SLO dialog for opening an existing project. In order to open a project, the directory containing that project must be specified by the user. By default, the current directory is shown in the specified field. All projects, defined previously, are listed and displayed in a drop down menu to help the user select the correct project name. The SLO does not allow the opening of multiple projects, that is, only one project may be opened at a time. When another project is opened, the current project is closed, the system is initialised and then the new project is opened.

Figure 5.3: The “Project” Option from the Main Menu in the SLO Program
The three project files are saved in ASCII format, hence it is possible to use a text editor to define and modify the required information, and save it as the specified project files. This is as long as the format that is used by the SLO, to read from the project files, is respected. Alternatively, the SLO provides a friendly environment for the user to define and modify, all required parameters, interactively. The SLO program assigns the input data to relevant variables, and writes the values of these variables to the project files (respecting the same format that is used to read them), when saving a project for further use.

5.7 PROJECT DEFINITION FILES

The input data that the optimisation core receives includes: that data entered by the user; that data calculated by the SLO program; and that transformed data provided during data preparation. There are a total of six input files used by the SLO system, including the ".mpr", ".cst", ".eco", ".dat", ".asy", and ".bev" files, as described in Table 5.1. The last two files are produced by the SLO system. The data saved in the first three files (project files) are partly input by the user and partly calculated by the SLO system. However, data saved in the ".dat" file is completely provided by the user.
5.7.1 Block Model Parameters

The model parameters files (".mpr") contain all of the information about the specifications of the ore-body block model. These specifications include the coordinates of the origin and the maximum limit of the block model, the extension and the number of the blocks in X, Y and Z directions, as well as the definition of possible sub-regions within the whole model. Some of the above information should be entered by the user, while the rest may be obtained by the SLO system. For example, the user may provide block model co-ordinates, in which case the number of blocks in X, Y and Z directions should be calculated. Conversely, the number of blocks in X, Y and Z directions may be available, in which case the co-ordinates of the block model is calculated. The SLO provides the processes for both of these options. Figure 5.5 shows the dialog box for the selection of either of the two modes.

![Figure 5.5: Mode Selection for Block Model Input](image-url)
In the first mode, the fixed block dimensions are entered by the user. The block volume, the number of blocks in each direction and the total number of blocks within the model, are automatically calculated. Since this option is based on the input block extensions in the three orthogonal directions, it is called the "XYZ mode". The second option requires that the user enter the number of blocks as well as the block’s extensions in X, Y and Z dimensions. The program SLO then calculates the minimum and maximum co-ordinates of the block model. This option is called the "IJK mode" since it receives the number of blocks in I, J and K dimensions. In both cases however, the block’s extensions in the three orthogonal directions have to be provided. Figure 5.6 shows the SLO dialog box for inputting the block model data in the “XYZ mode”.

![SLO - Block Model Definition](image)

**Figure 5.6:** The “XYZ Mode” Dialog Box for Defining the Block Model

The program SLO has an option to define sub-regions within the whole model. It may be necessary to divide the block model due to geo-technical factors, different rock types
and mining methods. These impose different stope constraints to different parts of the block model. If this were to happen, various sub-regions may be defined to handle a variety of stope constraints and orders of neighbourhood within the model. When a deposit is too large and separate zones of mineralisation can be distinguished within the entire deposit, the definition of various sub-regions may be helpful. In addition, the block model may contain either block grade values or block economic values. Figure 5.7 shows the dialog box, through which the user can state whether or not there are any sub-regions within the block model and specify the block data type.

![SLO - Block Model Definition](image)

**Figure 5.7:** The Dialog Box for Defining Sub-Regions in the Block Model

It is possible, in the SLO program, to define sub-regions, or modify and delete an existing one. The total number of sub-regions is firstly determined by the user. After defining each sub-region, its name is added to a list, which is available through a drop down menu for further editing or deletion, as shown in Figure 5.8. For each sub-region,
the name and the co-ordinates of its origin and maximum limit should be defined in
the X, Y and Z directions. Obviously, each sub-region must fall completely inside the
entire model, that is, sub-regions must be subsets of the whole model. However, they
may overlap, and there is no restriction for them to fill the whole block model. Figure
5.9 is an example of the dialog box for defining sub-regions in the block model.

All the above information, including any input by the user and/or that obtained by the
SLO program, is stored in a number of variables for use throughout the data preparation
and optimisation stages. They are also saved in the "mpr" file and are available for
future use. Figure 5.10 is an example of such a file.
Chapter Five: Implementation of the MVN Algorithm

Figure 5.9: Defining Sub-Regions within the Block Model

The description of the contents of the model parameters file is as follows:

Line 1: the name of the project,
Line 2: a blank line,
Line 3: the 3D co-ordinates of the origin of the block model, in terms of meters (The three columns used represent the X, Y and Z co-ordinates, respectively.),
Line 4: the X, Y and Z co-ordinates of the maximum limit of the block model in terms of meters,
Line 2: a blank line,
Line 6: the extensions of the fixed block in the X, Y and Z directions, in terms of meters,
Line 7: the volume of the block, calculated by the SLO system, in cubic meters,
Line 8: a blank line,
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>the number of blocks contained in the model in the X, Y and Z directions,</td>
</tr>
<tr>
<td></td>
<td>respectively,</td>
</tr>
<tr>
<td>10</td>
<td>the total number of blocks contained in the model,</td>
</tr>
<tr>
<td>11</td>
<td>a blank line,</td>
</tr>
<tr>
<td>12</td>
<td>a blank line,</td>
</tr>
<tr>
<td>13</td>
<td>an indicator number to determine whether or not there are any sub-regions</td>
</tr>
<tr>
<td></td>
<td>in the block model (The indicator is set to “one” for “yes” and “zero” for</td>
</tr>
<tr>
<td></td>
<td>“no”. If the indicator is “zero”, this line is the last line of the file,</td>
</tr>
<tr>
<td></td>
<td>otherwise the file contains the following lines as well.),</td>
</tr>
</tbody>
</table>
Line 14: the number of sub-regions contained in the block model,
Line 15: a blank line,
Line 16: a blank line,
Line 17: the number and name of the sub-region,
Line 18: a blank line,
Line 19: the X, Y and Z co-ordinates of the origin of the sub-region, in terms of meters,
Line 20: the X, Y and Z co-ordinates of the maximum limit of the sub-region, in terms of meters.

The remaining lines are repetitions of Lines 15 to 20, inclusive, for the rest of the sub-regions. The number of repetitions is determined in Line 14.

5.7.2 Stope Geometry Constraints

The ".cst" files contain all the information about the stope geometry constraints. The user enters some of this information and the program calculates the remainder. The user should enter the minimum stope size, in terms of meters, for each of the three orthogonal directions. The stope block ratio (SBR) and the order of neighbourhood (Onb), in each direction, is then calculated by the SLO program, which is based on the block size and the minimum stope size. The order of neighbourhood is finally expressed in terms of the integer numbers, that is, the number of blocks inside the minimum stope size in the X, Y and Z directions, respectively. The product of these three numbers indicates the total number of blocks within the minimum stope. This product also indicates the total number of possible neighbourhoods for each block related to the imposed stope constraints. Figure 5.11 shows the SLO dialog box for entering the stope geometry constraints.

If there is no sub-region in the block model (as in Figure 5.11), the related fields for the sub-regions are deactivated and the minimum stope size is defined only for the whole model. The existence of sub-regions often indicates that there is no consistent stope
geometry constraint within the block model, therefore, the constraints should be defined for all the sub-regions and not for the entire model. In these cases, the related fields for the whole region are deactivated and those of the sub-regions are activated (See Figure 5.12). If the division of the model into sub-regions is not due to various stope constraints (for example, it is because of a large deposit), then the stope constraints may be the same for all the sub-regions as well as for the entire model. This means that defining the sub-regions is not useful, unless the user is interested in performing the optimisation algorithm for a specified zone, instead of the whole model.

![Figure 5.11: The Dialog Box for Defining the Order of Neighbourhood (entire model)](image)

The list of the sub-regions is available through the drop down menu provided in Figure 5.12. The user may select a sub-region from the list and define, or modify, the stope geometry constraints for that sub-region. The process should be repeated for all sub-regions of the model. Figure 5.13 shows the dialog box for defining stope geometry constraints for the sub-regions.
Figure 5.12: The Dialog Box to Select a Sub-Region for Defining its Order of Neighbourhood

Figure 5.13: The Dialog Box for Defining the Order of Neighbourhood (sub-regions)
All of the above information, including any input by the user and/or calculated by the SLO program, is needed during the data preparation and optimisation stages. Figure 5.14 is an example of such a stope constraints file.

```
Project Name: Test1

1

WestTop
20.00 20.00 20.00
2 2 2
8

WestDown
30.00 30.00 30.00
3 3 3
27

EastTop
40.00 40.00 40.00
4 4 4
64

EastDown
20.00 30.00 40.00
2 3 4
24
```

**Figure 5.14:** Example of the Stope Geometry Constraints File, ".cst"

The description of the contents of the stope geometry constraints file is as follows:

Line 1:  the name of the project,
Line 2:  a blank line,
Line 3:  an indicator number to determine whether or not there are any sub-regions in the block model (The indicator is set to “one” for “yes”, and to “zero” for “no”. In this example, the indicator number is “one”. However, if the
indicator is "zero", the information corresponding to the whole model is stored in the same way as below for each sub-region.),

Line 4: the number of the sub-regions contained in the block model,
Line 5: a blank line,
Line 6: a blank line,
Line 7: the number and name of the sub-region,
Line 8: a blank line,
Line 9: the extensions of the minimum allowable stope in the X, Y and Z directions, in terms of meters,
Line 10: a blank line,
Line 11: the order of neighbourhood in the X, Y and Z directions, respectively,
Line 12: the total number of possible neighbourhoods for a block.

The remaining lines are repetitions of Lines 5 to 12, inclusive, for the rest of the sub-regions. The number of repetitions is determined in Line 4.

5.7.3 Economic Factors

The economic parameters applicable to the mining of the deposit are also required as input to the SLO program. This information is not directly used by the optimisation core but rather, is employed to help transform assay data into the dollar value of the blocks. The required economic parameters include information about the products of mining: their prices; grades and price units; costs of mining/processing of the products; rates of recovery applied to products; and the densities of the ore/waste. Figure 5.15 shows the economic parameters dialog box, by which products of the deposit may be defined.

The price and the price unit of each product should be included. The user may enter the price directly in a real field in the supplied dialog box. However, for price units, the SLO program provides a list of three options for the user to select. These options are available via a drop down menu and include: dollar per tonne; dollar per ounce; and cents per kilo. Figure 5.16 shows the corresponding dialog box in the SLO in which
prices and price units of the products may be defined.

![Image of SLO Economic Factors Dialog]

**Figure 5.15:** The SLO Dialog for Defining Products of the Project

Cost information must be defined for all projects. There are two categories of costings applied in the optimisation of stope boundaries, and these are, ore-based costs and metal-based costs. Ore-based costs consist of all expenditures necessary for the extraction of the rock from the mine, regardless of whether it is ore or waste. In order to obtain ore-based costs, all mining processes, including preparation, drilling, blasting or haulage, should be considered. The average cost for mining one tonne of rock (ore or waste) forms the ore-based costs.
Metal-based costs include all expenditures necessary to recover the metal product from the mined ore. This costing is determined by calculating the average cost for one tonne of the main product rather than the ore. The metal-based costs may be broken down into a number of components, such as, the cost of smelting or processing, the cost of refining, the cost of administration and other miscellaneous costs. The user has to include all the components of the metal-based costs. The total metal-based cost is calculated, automatically, as the user exits the field provided for that cost component. Figure 5.17 shows the SLO dialog, through which the user may enter cost information.
The product cannot be recovered one hundred percent, therefore, one of the parameters that needs to be taken into account, is the rate of recovery of the metal contents. The recovery should be defined in the various stages of the processing. There are three stages of the processing including the mining stage, the processing (smelting) stage and the refining stage. The rate of recovery for each stage is entered by the user, in terms of percentage, and the total rate of recovery is returned by the SLO. However, the user may enter directly the total rate of recovery, if available, and ignore the sub recoveries. Figure 5.18 shows the dialog box for entering the various rates of recovery when processing.

**Figure 5.17**: The Dialog Box for Defining Various Costs
Figure 5.18: The Dialog Box for Defining Various Rates of Recovery for the Main Product

If there are any by-products, the same rates should be defined for each by-product. A by-product may be selected from the list of by-products, available via a drop down menu (Figure 5.18), and the recovery rates defined or modified. This is achieved by using the "Edit" button and another dialog box which is shown in Figure 5.19. For all products, the user enters the rates only in terms of percentages (%).

Another economic parameter is the product’s grade unit. Although the grade values are entered via the assay data file, the units of grades should be defined. The definition of the grade units should be completed for each of the by-products as well as the main product. A list of two items is available for the selection of grade units. These are the percentage (%) and grams per tonne (ppm). Figure 5.20 shows the SLO dialog box used to define the unit of block grade values for each product.
Figure 5.19: The Dialog Box for Defining Various Rates of Recovery of By-Products

Figure 5.20: The SLO Dialog Box for Defining Grade Units
Ore properties are the last economic factor described in the project file. In order to obtain the weight of a block, the specific gravity of the rock is needed. The SLO program currently supports two distinct categories of density, that is, the density of the ore and the density of the waste. A cut-off grade should be defined for the main product so as to discriminate between the ore and the waste. Figure 5.21 shows the SLO dialog box for defining the cut-off grade of the main product, and the specific gravity of waste, as well as for ore.

![Figure 5.21: The Dialog Box for Defining Ore Properties](image)

All the above information is written into a file with the ".eco" extension for further use. An example of such an economic parameters file is shown in Figure 5.22.
### Figure 5.22: Example of the Economic Factors File, "eco"

The description of the contents of the economic parameter file is as follows:
Chapter Five: Implementation of the MVN Algorithm

Line 1: the name of the project,

Line 2: a blank line,

Line 3: a blank line,

Line 4: an indicator number to determine whether or not there are any by-products in the deposit (The indicator is set to "one" for "yes" and "zero" for "no". In this example, the indicator number is "one". However, if the indicator is "zero", those following lines, which correspond to by-products, are not applicable.),

Line 5: the number of by-products in the deposit,

Line 6: a blank line,

Line 7: the name of the main product and by-products (if any),

Line 8: a blank line,

Line 9: a blank line,

Line 10: the price of the main product and by-products (if any),

Line 11: codes of the price units of the main product and by-products, if any (Codes show the unit numbers within the list of available units, stored in Line 13.),

Line 12: a blank line,

Line 13: the list of three price units available in the SLO,

Line 14: a blank line,

Line 15: a blank line,

Line 16: the ore-based cost, in terms of $/tonne of rock,

Line 17: a blank line,

Line 18: the list of metal-based costs, in terms of $/tonne of the (main product) metal, including processing costs, refining costs, administration costs and miscellaneous costs, respectively,

Line 19: the total metal-based costs, in terms of $/tonne of the (main product) metal,

Line 20: a blank line,

Line 21: a blank line,

Line 22: an indicator number to determine whether the total recovery of the main product, or its sub-recoveries at three stages, is used (The indicator is set to "one" if the total recovery is used and "zero" otherwise. If the total recovery is used, the value of the last column in the next two lines are used and those of other columns are ignored.).
Line 23: percentage of the main product recovered during the mining stage, the
processing stage, the refining stage and the total recovery, respectively,

Line 24: proportion of the main product recovered during the mining stage, the
processing stage, the refining stage and the total recovery, respectively,

Line 25: a blank line,

Lines 22 to 25 are repeated for all by-products.

Line 42: a blank line,

Line 43: codes for defining the grade units of the main product and all the by-products,
if any (Codes show the unit numbers within the list of available units, stored
in Line 45.),

Line 44: a blank line,

Line 45: the list of two grade units supported in the SLO,

Line 46: a blank line,

Line 47: a blank line,

Line 48: the cut-off grade of the main product, in terms of its grade unit as defined in
Line 43 and its absolute proportion, respectively,

Line 49: a blank line,

Line 50: the density of the ore and

Line 51: the density of the waste.

5.8 BLOCK DATA FILE

There are two types of input files in the SLO program. These are processed data and auxiliary data. The project files described so far refer to auxiliary data. However, processed block data is the main input to the optimiser which is saved in an ASCII formatted file called "data file". This file is specified by the extension ".dat".

The main block inputs to the optimiser may be the economic value of blocks. In cases, where the economic value is not available, the assay data of blocks may be entered. However, assay values should firstly be converted to dollar values. In either case, the data file contents include some information for the blocks of the model. The data for each block is given in a separate line and consists of the block address and value. The
block address is defined in terms of the three integer numbers, denoted by $i, j$ and $k$. This indicates the sequential position of the block in the three orthogonal dimensions. If the data file is economic block data, then the value of the block is a real number that represents the estimate of the economic value of the block. Where the input is assay data, the block value consists of a group of, up to five, real numbers. This group contains the estimated grade of the main product, as well as that of a maximum of four possible by-products for the block.

The SLO program uses a default Fortran format for the data file. This is "$3I3, X, 5F10.2$" for the assay data file with a main product and four by-products, and "$3I3, X, F10.2$" for economic data file. However, the user may define a customised Fortran format for the block data file. Figure 5.23 shows the SLO dialog which allows the definition of the name and type of the block data file, and the format used to read data from the file.

![SLO - Data file](image)

**Figure 5.23:** The Dialog Box for Defining the (Main) Data File

There are no restrictions in the name of the data file, although it is more convenient that the name of the project files match that of the data file. The SLO program, also provides the possibility for the user to view the block "$\text{.dat}$" file before importing the file to the optimiser. Figures 5.24a and 5.24b are examples of data files for block assay values and
block dollar values, respectively. The first three integer numbers of Figure 5.24a represent the $i, j$ and $k$ addresses of the block, respectively. The remaining column of each line represent the grade values of products, estimated for the block or the estimated dollar value of the block, respectively.

<table>
<thead>
<tr>
<th></th>
<th>9</th>
<th>3</th>
<th>8</th>
<th>0.01</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>9</td>
<td>3</td>
<td>10</td>
<td>0.42</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>3</td>
<td>11</td>
<td>0.87</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>3</td>
<td>12</td>
<td>0.92</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>3</td>
<td>13</td>
<td>0.87</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>4</td>
<td>9</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>4</td>
<td>10</td>
<td>0.22</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>4</td>
<td>11</td>
<td>0.88</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>4</td>
<td>12</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>4</td>
<td>13</td>
<td>1.18</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>5</td>
<td>9</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>5</td>
<td>10</td>
<td>0.29</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>5</td>
<td>11</td>
<td>0.94</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>5</td>
<td>12</td>
<td>1.02</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>5</td>
<td>13</td>
<td>1.17</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>6</td>
<td>10</td>
<td>0.35</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>6</td>
<td>13</td>
<td>1.18</td>
</tr>
</tbody>
</table>

Figure 5.24a: Example of the Assay Value Data File, "dat"

<table>
<thead>
<tr>
<th></th>
<th>6</th>
<th>8</th>
<th>8</th>
<th>449960.69</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>657715.56</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>657715.56</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>8</td>
<td>8</td>
<td>665470.38</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>8</td>
<td>8</td>
<td>740817.56</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>8</td>
<td>8</td>
<td>200654.86</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>8</td>
<td>8</td>
<td>1613387.88</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>8</td>
<td>8</td>
<td>2859917.00</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>8</td>
<td>8</td>
<td>1613387.88</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>8</td>
<td>8</td>
<td>2361305.50</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>3358528.75</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>8</td>
<td>8</td>
<td>1799591.88</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>9</td>
<td>8</td>
<td>-7100.00</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>-7100.00</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>9</td>
<td>8</td>
<td>283756.81</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>9</td>
<td>8</td>
<td>1156327.25</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>9</td>
<td>8</td>
<td>5186771.5</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>9</td>
<td>8</td>
<td>3192324.75</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>9</td>
<td>8</td>
<td>4605058.00</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>9</td>
<td>8</td>
<td>1073225.25</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>9</td>
<td>8</td>
<td>948572.38</td>
</tr>
</tbody>
</table>

Figure 5.24b: Example of the Economic Value Data File, "dat"

The final input for the optimisation core is the block economic value data in a binary formatted file. In cases where the ASCII data file, "dat", contains the economic block
data, the only required preparation is to change the format of the data into a binary format and store in a file with the ".bev" extension. However, if the data file contains the assay data, the data should be transferred to the economic data after being changed to binary form. This is shown in Figure 5.25.

<table>
<thead>
<tr>
<th>Main data (processed)</th>
<th>Grade values:</th>
<th>Economic values:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ASCII</td>
<td>Binary</td>
</tr>
<tr>
<td></td>
<td>*.DAT</td>
<td>*.ASY</td>
</tr>
</tbody>
</table>

**Figure 5.25:** Preparation of the Main Data File for the Optimisation Core

### 5.9 DATA PREPARATION

The formatted sequential file is the slowest file type in Fortran 90, for to access a record all its previous records must first be accessed. The unformatted direct access file is the fastest file type. Grade (or economic) values in the ".dat" files are stored as character data, that is, in ASCII format. In order to speed up the access to the file, the assay (economic) data is stored in an unformatted direct access file in which the data is stored as binary data.

In the direct access files, each record is attributed by its number (order). In the conversion of the sequential assay data into the direct access file, the three-dimensional \((i, j, k)\) address of the blocks should be modified into a one-dimensional \((ID)\) address. This will determine the number of the record, in ascending order, to the corresponding binary file. It is assumed that blocks in a model with the subscripts of \(i, j\) and \(k\) are numbered in ascending order by their address. It means that the subscript of the leftmost dimension changes rapidly as one goes from the first block (record), to the last in the
block order.

Based on this assumption, a user defined function is to enter the subscripts of a block, \((i, j, k)\), and return the block order \((ID)\). This function is used to write the value of the block, \(B_{ijk}\), in the \(ID^{th}\) record (line) in the binary file. The inverse of the previous function is to invert the block order \((ID)\) into its previous subscripts. This function can be used when assigning the value of the \textit{Fortran} variable, \(BEV(i, j, k)\), from the \(ID^{th}\) record (line) of the binary file.

In contrast to the sequential file, there is no need to define the block address in each record of the direct access files, but rather the order of the record indicates the address of the block automatically. Therefore, the length of each record in the direct access file is shorter than that of the sequential files, as the record contains only the value(s) assigned to the block. In addition, the direct access file, which contains the assay data in binary format, is saved as a file with the "asy" extension. However, the name of both files is restricted to the name of the original block data file.

5.9.1 Equivalent Grade

The objective function for the optimisation of the stope boundaries is maximising the profit (value) obtained from the stope. The stope value is obtained by the summation of the potential profit/loss of a collection of selected blocks. The final inputs to the optimiser should be expressed in dollar units, the economic values of blocks \((BEV)\), as the profit is expressed in terms of dollar.

If there is more than one product in the mine for each block, the assay data file contains more than one grade value. In these cases, the grade values should be replaced by a single equivalent grade value, based on the main product grade. In order to calculate the equivalent grade value, the prices, grades and rates of recovery for every single product should be taken into consideration. The process of calculating the equivalent grade value
and the related formulae are provided in Appendix A.

After the grade values of all the blocks have been converted into dollar values, the input stage is completed. The output of data preparation, which is stored in the ".bev" files, is then ready for use by the program optimisation core as the final data input.

5.10 SUMMARY

The stope limit optimiser (SLO), developed during this study, benefits from the Winteracter interfacing features and the Fortran programming language. The system consists of a Fortran 90 source code and a Winteracter resource script. A main menu and a number of vertical menus, together with 25 dialogs defined in the resource script, plus a main program and approximately 100 subroutines/functions defined in the source code, are used to provide a windows based interactive environment for the user to edit and optimise a stope project and display the results. A complete user guide manual of the SLO program is provided in Appendix C.

The implementation of the MVN algorithm, through the SLO program, consists of the input, optimisation and output stages. During the input stage, the user may define the specifications of the block model, the stope constraints and the economic factors. This data is then collected in separate files as a stope project. This process involves using an interactive environment with a number of menus and dialogs. A converter may receive the assay data of the blocks to change them into dollar values which enables them to be entered into the optimiser. The following chapter introduces the implementation of the maximum value neighbourhood (MVN) algorithm, which corresponds to the optimisation stage of the SLO program.
CHAPTER SIX

PROGRAMMING OF THE MAXIMUM VALUE NEIGHBOURHOOD ALGORITHM IN FORTRAN 90

6.1 INTRODUCTION

The major program modules developed in Fortran 90 are presented in this chapter to illustrate how the optimisation algorithm was implemented. The optimisation stage is the core of the SLO program (See Figure 5.2). It manages all of the required relationships between the SLO components. For example, the input files, output files and the process algorithm.

6.2 GENERAL VIEW ON THE OPTIMISATION STAGE

The optimiser core transforms all of the block dollar values into a three-dimensional array variable. The domain (region) of the optimisation is then determined. An option is provided for the user to select a region of interest for the optimisation. The user may decide if the optimisation is to be carried out on a sub-region as defined in the project file, (*.mpr), or performed simply on the whole block model.

During the optimisation, the information stored in the project files is used to produce the flag data of the blocks. The blocks of the selected region are orderly selected into the
process by the optimisation core. The economic values of these blocks are processed one at a time using the $MVN$ algorithm described in Chapter Three. The constraint file is used to determine: the order of neighbourhood in each dimension; the number of elements within a neighbourhood ($NB$); the total number of neighbourhoods for a block; and the location of each neighbourhood. The (*.mpr) file is also used to provide: the information required for determining the region (domain) on which the optimisation is performed; the sequence of the blocks to be taken into the process; and the feasibility of the neighbourhoods for a given block during the optimisation process.

The last step in the optimisation stage is writing the flags of the blocks. The $SLO$ defines a three-dimensional array with zeros and ones. When the program initialises the variables, all of the elements of this array are set to "zero", which indicates that no block of the stope is flagged (selected). As the optimisation progresses, the values of the elements of the flag array are updated, that is, the flags of the selected elements are set to "one". After the optimisation is completed, the array values are written to an output file, which stores the flag data of the blocks. The output file is saved in the ASCII format with the extension (*.out). The name of the (*.out) file is the same as that of the input data file, (*.dat).

Figure 6.1 shows the major Fortran modules used in the optimisation stage to implement the $MVN$ algorithm. The subroutines "ReadDollarData" and "SelectSubs" define the data used by the $MVN$ algorithm. The "ReadDollarData" subroutine reads the block economic values from the (*.bev) file into the 3D array variable. The subroutine "SelectSubs" defines the (sub) region for the optimisation. The three sub­programs ("MaxValueNB", "SelectBlock" and "MarginValue") implement the $MVN$ algorithm on each block. The "MaxValueNB" function determines the value and location of the maximum value neighbourhood of a block. The "SelectBlock" subroutine defines which elements, of the maximum value neighbourhood ($MVN$), have not been already flagged. The "MarginValue" subroutine calculates the marginal contribution of the $MVN$ for blocks defined by the "SelectBlock" subroutine. After performing the optimisation, for all the blocks of the defined region, final optimisation
results are written to the (*.out) file using the "ExportFlag" subroutine. The function, "IDNumber", and the subroutine, "IDInverse", are peripheral facilities in the SLO system. These sub-programs are particularly used when shifting between the 3D (i, j, k) address of a block (in the 3D array variables) and the one-dimensional (ID) address of the block (in the direct access files).

![Diagram of major modules used to perform the optimisation algorithm](image)

**Figure 6.1:** Major Modules Used to Perform the Optimisation Algorithm

The "MaxValueNB" function uses a series of six other functions (subroutines) to locate
the MVN of a block. The subroutines "NBStartPosition" and "NBEndPosition" determine the \((i, j, k)\) address of the first element (block) and that of the last element, in a given neighbourhood of a given block, respectively. "NBTrue" is a logical function that determines, whether or not, the given neighbourhood is feasible. The "NBValue" function then calculates the total net economic value of a given feasible neighbourhood. Sub-programs "MAXLOC" and "MAXVAL" are Fortran 90 intrinsic functions that return the location and the value of the maximum value element of an array, respectively. These two functions are used to locate the maximum value neighbourhood among the elements of the set of the neighbourhood values \((NBVS)\). The neighbourhood locating subroutines ("NBStartPosition" and "NBEndPosition") are also called by the "SelectBlock" subroutine. In addition, the "SelectSubs" subroutine calls the "XYZtoIJK" subroutine during its process.

"Optimise" is the subroutine which controls most of the optimisation processes. It receives the limits of the sub-region to be optimised with the mining constraint and performs the optimisation procedure on the specified blocks. Arguments of the "Optimise" subroutine are defined as below:

\[
\text{SUBROUTINE Optimise(IMn, JMn, KMn, IMx, JMx, KMx, Order)}
\]

\[
\text{END SUBROUTINE Optimise}
\]

where

IMn, JMn and KMn: the \((i, j, k)\) address of the block specifying the \textit{start} of the (sub) region to be optimised,

IMx, JMx and KMx: the \((i, j, k)\) address of the block specifying the \textit{maximum limits} of the (sub) region to be optimised and

Order: a 1D array with three elements, specifying the order of neighbourhood in three orthogonal directions for the specified (sub) region.
6.3 DEFINING THE OPTIMISATION DATA

Defining the data for optimisation requires two steps. Firstly a (sub) region is selected for performing the optimisation, then the block economic values of the specified (sub) region are assigned to a 3D array variable, BEV(i, j, k). Each element of the BEV variable is then examined during the optimisation procedure.

Optimisation may be performed on the whole block model or on any sub-region of the block model. Figure 6.2 shows the SLO dialog box which allows selection of the optimisation domain (region). A sub-region may be selected from the list of the sub-regions available through a drop down menu, while the entire block model is chosen by the provided radio button.

![SLO Dialog Box](image)

**Figure 6.2**: The SLO Dialog Box which Specifies the Domain of the Optimisation

Each domain is defined by its first and last block, that is, blocks with the lowest address \((i, j, k)\) and the highest address \((i, j, k)\), respectively. In cases where the whole block model is selected for optimisation, the lower and upper bounds of the domain are simply
(1, 1, 1) and (NI, NJ, NK), respectively, where NI, NJ and NK are the number of blocks within the model in the X, Y and Z directions, respectively. The orders of neighbourhood in the three directions for the entire model, which are stored in ONBI, ONBJ and ONBK variables, are assigned to the three element 1D variable, Order, as shown below:

\[
\begin{align*}
\text{IMn} &= 1; \quad \text{IMx} = NI \\
\text{JMn} &= 1; \quad \text{JMx} = NJ \\
\text{KMn} &= 1; \quad \text{KMx} = NK \\
\text{Order}(1) &= \text{ONBI}(\text{NSubs} + 1) \\
\text{Order}(2) &= \text{ONBJ}(\text{NSubs} + 1) \\
\text{Order}(3) &= \text{ONBK}(\text{NSubs} + 1)
\end{align*}
\]

Alternatively, if a sub-region is selected from the drop down menu, the corresponding limits and order of neighbourhood of the sub-region are assigned to the appropriate variables of the "Optimise" subroutine. The number associated with each item of the list of the sub-regions, in the drop down menu, controls this link as shown below:

\[
\begin{align*}
\text{CALL WDialogGetMenu}(\text{IDF}_\text{SubsMenu}, \text{SubNum}) \\
\text{CALL XYZtoIJK}(\text{XMin}(\text{SubNum}), \text{YMin}(\text{SubNum}), \text{ZMin}(\text{SubNum}), \text{IMn}, \text{JMn}, \text{KMn}) \\
\text{CALL XYZtoIJK}(\text{XMax}(\text{SubNum}), \text{YMax}(\text{SubNum}), \text{ZMax}(\text{SubNum}), \text{IMx}, \text{JMx}, \text{KMx}) \\
\text{Order}(1) &= \text{ONBI}(\text{SubNum}) \\
\text{Order}(2) &= \text{ONBJ}(\text{SubNum}) \\
\text{Order}(3) &= \text{ONBK}(\text{SubNum})
\end{align*}
\]

Since the limits of the sub-regions are defined in terms of meters, a subroutine, "XYZtoIJK", is called to provide the origin and the maximum limits of the sub-region, in terms of the block number within the model. The "XYZtoIJK" subroutine uses the 3D co-ordinates of a point and returns the \((i, j, k)\) address of the block. The relationship between the X co-ordinate of a point and the \(i\) address of the corresponding block may be expressed by Equation 6.1.

\[
i = \text{Int}\left(\frac{x - 0.01}{L}\right) + 1 \quad (6.1)
\]

where
Chapter Six: Programming of the MVN Algorithm in Fortran 90

Chapter Six: Programming of the MVN Algorithm in Fortran 90

x: the location of the point in the X direction,
L: the size (extension) of the block in the X direction,
Int: a function that returns the integer part of a real number, and
i: the block number in the X direction.

Similar relationships may be defined to convert the Y and Z co-ordinates of a point, to the j and k addresses of the corresponding block.

After the optimisation domain has been defined, variables are initialised. The main variables used in the optimisation include BEV, F, FF and NBV, which represent the block economic value, the block flag, the flag of new blocks to the stope and the economic value of neighbourhoods, respectively. The first three variables can store values of a 3D-array, while NBV contains 1D-array values. These arrays are defined as Fortran 90 dynamic arrays, therefore, their shape and size are changeable during the program. These variables are initialised as follows:

```
DEALLOCATE (BEV, F, FF, NBV)
ALLOCATE (BEV(imn:imx, jmn:jmx, kmn:kmx))
ALLOCATE (F(imn:imx, jmn:jmx, kmn:kmx))
ALLOCATE (FF(Order(1), Order(2), Order(3)))
MaxNumNB = Order(1) * Order(2) * Order(3)
ALLOCATE (NBV(MaxNumNB))
MarginVal = 0
F = 0
FF = 0
SEV = 0
```

In the above Fortran 90 statements, the space allocation of variables BEV, F, FF and NBV are removed (de-allocated), then a new space is allocated for them. The space needed for BEV and F is the same as the space, in which optimisation is performed. That is, the start and the end of the selected (sub) region, in each direction, define the limits of the array. However, variables FF and NBV contain only those values corresponding to blocks in a neighbourhood. For a 3D FF variable, the array size in each dimension is controlled by the order of neighbourhood in that dimension. The NBV variable has the same size as the FF but with a different shape (1D array). Therefore, the size of the NBV array is the product of the orders of neighbourhood in the three directions. This is represented by the variable, "MaxNumNB", indicating the total number of possible
neighbourhoods for a block. During initialisation, all the elements of the flag data are set to zero, that is, $F(imn:imx, jmn:jmx, kmn:kmx)$, to ensure that no block is selected for the stope before optimisation starts.

The input data for each optimisation is the economic value of the blocks located within the selected (sub) region, which are stored in the direct access file, (*.bev). This data is read from the file and saved in the appropriate elements of the $BEV$ variable. The "IDNumber" function is used to locate the corresponding record number (line) in the (*.bev) file. The following Fortran 90 code assigns input data to the $BEV$ variable:

```
DO k = kmn, kmx
  DO j = jmn, jmx
    DO i = imn, imx
      ID = IDNumber(i, j, k, NI, NJ)
      READ (5, rec = ID) BEV(i, j, k)
    END DO
  END DO
END DO
```

### 6.4 LOCATING THE MAXIMUM VALUE NEIGHBOURHOOD

After the data has been processed, the optimisation locates the maximum value neighbourhood ($MVN$) of each block and updates the optimum stope blocks. These steps are repeated for all the blocks of the selected region. The Fortran 90 code for processing each block is as follows:

```
IF (BEV(i, j, k) < 0) GOTO 20
MNBV = MaxValueNB(i, j, k)
IF (MNBV < 0) GOTO 30
CALL SelectBlock(i, j, k)
CALL MarginValue(FBI, FBJ, FBK, LBI, LBJ, LBK, MarginVal)
IF (MarginVal < 0) GOTO 40
SEV = SEV + MarginVal
```

20  . . .  Go to the next block
30  . . .  Go to the next block
40  . . .  Go to the next block

The process of locating the maximum value neighbourhood for each block is carried out through the "MaxValueNB" function. This value is assigned to the $MNBV$ variable,
which represents the maximum neighbourhood value. A Fortran 90 loop is used to calculate the economic value of each neighbourhood and construct the set of neighbourhood values, using the 1D array variable, NBV. The loop, which is replicated as many times as the maximum number of possible neighbourhoods, firstly, determines the location of the specified neighbourhood and checks the feasibility of the neighbourhood. If the neighbourhood is not feasible, a large negative value is assigned as the neighbourhood value, otherwise a subroutine, "NBValue", is called to calculate the neighbourhood value. When the loop is completed, the maximum neighbourhood value is picked up from the array of neighbourhood values. The following is the corresponding Fortran 90 code to implement the above steps.

```
MNV = 0
DO NBNumber = 1, MaxNumNB
   CALL NBStartPosition(a, b, c, NBNumber, FBI, FBJ, FBK)  
   CALL NBEndPosition(FBI, FBJ, FBK, LBI, LBJ, LBK)  
   TrueNB = NBTrue(FBI, FBJ, FBK, LBI, LBJ, LBK)  
   SELECT CASE (TrueNB)
      CASE (.FALSE.)  
         NBV(NBNumber) = -99999
         GOTO 40
      CASE (.TRUE.)  
         TotNumNB = TotNumNB + 1
         NBV(NBNumber) = NBValue (FBI,FBJ,FBK,LBI,LBJ,LBK)
   END SELECT
40
END DO
MaxValueNB = MAXVAL(NBV)
MVNBNo = MAXLOC(NBV)
```

In order to locate the neighbourhood with the maximum value, all the neighbourhoods should first be located. The location of any neighbourhood depends on the following three parameters:

1. the location of the block, for which the neighbourhood is defined (the $i,j$ and $k$ addresses of the block);
2. the order of neighbourhood, ($O_{nb}$);
3. the neighbourhood number within the set of neighbourhoods.

This means that it is necessary to define a numbering rule to identify the neighbourhoods and provide a general formula to define any neighbourhood, of any block, with any order of neighbourhood. The numbering rule, described below, uses a
similar approach to identify a neighbourhood within a set of neighbourhoods, and a block within a neighbourhood. The rule reduces 2D and 3D neighbourhoods to 1D sets.

### 6.4.1 Neighbourhood Identification

For a given 2D neighbourhood, there are $L^{(2)} (L^{(2)} = l_i l_j)$ number of blocks which may be numbered sequentially as:

$$\{B_1, B_2, B_3, ..., B_m, ..., B_{L^{(2)}}\}$$

In this thesis, the numbering is via the positive X direction, followed by the positive Y direction, and then the positive Z direction. See Figure 6.3 for a 2D neighbourhood with the order of $(4, 3)$.

![Diagram of 2D Neighbourhood](image)

*Figure 6.3: Numbering the Elements of a 2D Neighbourhood, $[O_{nb}]^{(2)} = (4, 3)$*

Generally, when assuming a 2D neighbourhood with the order of $[O_{nb}]^{(2)} = (l_i, l_j)$, the neighbourhood forms a section of blocks with $l_i$ rows and $l_j$ columns, for which the element numbering is shown in Figure 6.4. The relationship between the sequential ID
number of elements and their \((i, j)\) address may be expressed by Equation 6.2.

\[
ID = (c-l) \, l_i + r
\]  

(6.2)

where

- **ID**: the sequential (one-dimensional) number, used to reference members of the 2D neighbourhood,
- **\(l_i\)**: the order of neighbourhood in the X direction,
- **\(c\)**: the column number of the member within the 2D neighbourhood and
- **\(r\)**: the row number of the block within the 2D neighbourhood.

**Figure 6.4**: Numbering the Elements of a General 2D Neighbourhood (numbers inside the cells indicate the sequential ID number of that element within the \(NB\).

Similarly, in a general 3D neighbourhood that contains \(L^{(3)} (L^{(3)} = l_i, l_j, l_k)\) number of blocks, the elements are numbered in a sequential order from one to \(L^{(3)}\), that is:

\[
\{B_1, B_2, B_3, ..., B_{m}, ..., B_{L^{(3)}} \}
\]

The elements of 3D neighbourhoods are numbered methodically in the X, then the Y and finally the Z directions. This indicates that the subscript of the leftmost dimension
changes more rapidly when IDs of members are sorted in ascending/descending order. Figure 6.5 is an example of block referencing within a 3D neighbourhood, with the order of (4, 2, 3).

![Diagram of block referencing within a 3D neighbourhood]

In general, in a 3D neighbourhood with the order of \([O_{nb}]^3 = (l_l, l_j, l_h)\), the ID number of members may be obtained through Equation 6.3.

\[
ID = (s-1) (l_l, l_j) + (c-1) l_l + r
\]  

(6.3)

where

- \(ID\): the sequential (one-dimensional) number to reference a member of the 3D neighbourhood,
- \(l_l\): the order of neighbourhood in the X direction,
- \(l_j\): the order of neighbourhood in the Y direction,
- \(s\): the section No. of the member within the 3D neighbourhood,
- \(c\): the column No. of the member within the 3D neighbourhood and
- \(r\): the row No. of the block within the 3D neighbourhood.
By assigning a sequential ID to the members of 2D or 3D neighbourhoods, the numbering rule described above, transforms neighbourhoods from a multi-dimensional set to a 1D set, which is more convenient for further calculations and programming. All possible neighbourhoods of a block (collected in the NBS set) are numbered in a way similar to that used to identify elements of a single neighbourhood. There are \( L^{(2)} \) elements in the set of the 2D neighbourhoods which means the elements (2D neighbourhoods) may be numbered sequentially from one to \( L^{(2)} \), that is:

\[
\{NB_1, NB_2, NB_3, ..., NB_m, ..., NB_L^{(2)} \}
\]

Similarly, elements of a set of 3D neighbourhoods may be numbered sequentially from one to \( L^{(3)} \), that is:

\[
\{NB_1, NB_2, NB_3, ..., NB_m, ..., NB_L^{(3)} \}
\]

A relationship should be defined so as to link the ID of a neighbourhood and the ID of the neighbourhood elements. This helps in determining the location of any neighbourhood. In general, the relationship may be defined as:

- **The** \( m^{th} \) **neighbourhood** \((NB_m)\) **for any block**, \( B_{ij} \), **is the neighbourhood, in which the block under consideration**, \( B_{ij} \), **is the** \( m^{th} \) **element (block).**

Hence, the first neighbourhood \((NB_1)\) for the block, \( B_y \), is the neighbourhood in which the block, \( B_y \), is the first element \((B_1)\). The second neighbourhood \((NB_2)\) is the one in which the block, \( B_y \), is the second element \((B_2)\) and so on. As an example, consider a 2D neighbourhood with the order of \([O_{nb}]^{(2)} = (2, 2)\). Figure 6.6 shows four possible neighbourhoods for the block, \( B_y \), illustrating the ID number of each neighbourhood, as well as the ID number of the elements of each neighbourhood.
The same relationship is applied to define the 3D neighbourhoods for any block, $B_{ijk}$. Consider a 3D neighbourhood with the order of $[O_{nb}]^{(3)} = (2, 2, 2)$. The first neighbourhood, $NB_1$, can be defined by the set of eight blocks in a $2 \times 2 \times 2$ matrix pattern such that the block, $B_{ijk}$, is the first element ($B_1$). Similarly, the second neighbourhood, $NB_2$, is the neighbourhood, in which the block, $B_{ijk}$, is the second element ($B_2$) and so on. Figure 6.7 illustrates the first and second (out of eight) 3D neighbourhoods for any block, $B_{ijk}$, with the order of $(2, 2, 2)$. (See Figure 4.5 for all eight possible 3D neighbourhoods).

6.4.2 Neighbourhood Determination

Any 3D neighbourhood ($NB$) may be determined by its starting block (first element) and ending block (last element), or by the starting block and the extension of the neighbourhood in each dimension. In general, the determination of any neighbourhood,
for a given block, $B_{ijk}$, requires three parameters. These are:

1. the location of the block, $B_{ijk}$;
2. the distance between the block, $B_{ijk}$, and the first element of the $NB$; and
3. the order of neighbourhood, $(O_{nb})$.

Figure 6.7: Two Neighbourhoods for the Block, $B_{ijk}$, with the Order of (2, 2, 2)

The first two parameters are required when defining the starting block of a neighbourhood, while the third parameter is used to define the ending block of the neighbourhood. The location of the block of interest may easily be defined by the $(i, j, k)$ address. The second parameter, that is, the distance (or relative location) between the block of interest, $B_{ijk}$, and the starting block, is provided by the ID of the neighbourhood. This may be obtained using the general definition of the $m^{th}$ neighbourhood.

The starting element (block) in a 2D neighbourhood ($NB$) is the block with the lowest $i$ and $j$ addresses among the $NB$ elements. This is located at the top left corner of the neighbourhood section. The address of this block (the first element of the $NB$) may be obtained if the address of the block of interest, $B_{ij}$, and the relative address between the two blocks, are known. The relative address is expressed in terms of the difference
between the rows and columns of the two blocks. The starting block is the first element of any neighbourhood and both its row and column numbers are “one”. The row and column addresses of the block, $B_{ij}$, may be obtained from $m$ (the neighbourhood ID described above) knowing that the block, $B_{ij}$, is the $m^{th}$ element within the $m^{th}$ neighbourhood. The row and column addresses of any block, $B_{ij}$, within its $m^{th}$ neighbourhood are obtained using Equation 6.4.

$$\begin{align*}
    r &= ID - l_i (c - 1) \\
    c &= \text{Int} \left( \frac{ID - 1}{l_i} \right) + 1
\end{align*}$$

(6.4)

where

- $ID$: the sequential identification of the neighbourhood,
- $r$: the row address of the block within the neighbourhood,
- $c$: the column address of the block within the neighbourhood,
- $l_i$: the order of neighbourhood in the X direction and
- $\text{Int}$: a mathematical function that returns the integer part of a number.

The relative addresses of the first element with respect to the block, $B_{ij}$, are, therefore, the $(r-1)$ and $(c-1)$ blocks in the X and Y directions, respectively. This means the absolute $(i, j)$ address of the first element may be determined using Equation 6.5:

$$\begin{align*}
    I_{fb} &= i - (r - 1) \\
    J_{fb} &= j - (c - 1)
\end{align*} \quad \text{or} \quad \begin{align*}
    I_{fb} &= i - r + 1 \\
    J_{fb} &= j - c + 1
\end{align*}$$

(6.5)

where

- $I_{fb}$: the $i$ address of the first block (element) within the $m^{th}$ NB and
- $J_{fb}$: the $j$ address of the first block (element) within the $m^{th}$ NB.

Given the order of neighbourhood in the X and Y directions, $(l_i, l_j)$, the location of the last element within the NB, which has the highest $i$ and $j$ addresses, is then obtained using Equation 6.6:
\[
\begin{align*}
I_{lb} &= I_{fb} + l_i - 1 \\
J_{lb} &= J_{fb} + l_j - 1
\end{align*}
\]
or
\[
\begin{align*}
I_{lb} &= i - r + l_i \\
J_{lb} &= j - c + l_j
\end{align*}
\]  
(6.6)

where

\(I_{lb}\): the \(i\) address of the last block (element) within the \(m^{th}\) \(NB\) and

\(J_{lb}\): the \(j\) address of the last block (element) within the \(m^{th}\) \(NB\).

Finally, the general formula to define any 2D neighbourhood \((NB_m)\) with any order, \([O_{nb}]^{(2)}\), for any block, \(B_{ij}\), can be expressed by Equation 6.7:

\[
NB_{m,ij,l_i, l_j} = \left\{ BEV_{\alpha\beta} \in U \left| \begin{array}{l}
\alpha = i - r + 1, i - r + 2, \ldots, i - r + l_i \\
\beta = j - c + 1, j - c + 2, \ldots, j - c + l_j \\
\end{array} \right. \right\}
\]  
(6.7)

given that

\[
\begin{align*}
\alpha &= r = m - l_i (c - 1) \\
\beta &= c = \text{Int} \left( \frac{m - 1}{l_i} \right) + 1
\end{align*}
\]

where

\(\alpha\): the row number of the blocks within the model,

\(\beta\): the column number of the blocks within the model,

\(BEV_{\alpha\beta}\): the economic value of the blocks within the \(m^{th}\) neighbourhood,

\(U\): the universal set of the blocks within the economic model,

\(l_i\): the order of neighbourhood in the Y direction and

\(NB_{m,ij,l_i, l_j}\): the \(m^{th}\) neighbourhood of the block, \(B_{ij}\), with the order of \((l_i, l_j)\).

A similar approach is used to define a general formula for any 3D neighbourhood. The row, column and section addresses of the block, \(B_{ijk}\), may be derived from its sequential address, as described by Equation 6.8:
$$r = ID - (s-1)(l_i l_j) - (c-1)l_i$$
$$c = \text{Int} \left( \frac{ID - 1 - (s-1)(l_i l_j)}{l_i} \right) + 1$$
$$s = \text{Int} \left( \frac{ID - 1}{l_i l_j} \right) + 1$$  \hspace{1cm} (6.8)

where

$s$: the section address of the block within the neighbourhood,

$l_i$: the order of neighbourhood in the X direction and

$l_j$: the order of neighbourhood in the Y direction.

Knowing that the (row, column, section) address of the first element in the NB is $(1, 1, 1)$, the relative address of this element, with respect to the block of interest, $B_{ijk}$, would be $(r-1)$, $(c-1)$ and $(s-1)$ rows, columns and sections, respectively. This will define the absolute $(i, j, k)$ address of the first element. The location of the last element within the NB, that is, the element with the highest $i$, $j$ and $k$ addresses can also be determined using the location of the first element and the order of neighbourhood in the X, Y and Z directions, $(l_b, l_j, l_k)$. The locations of the first and last elements of the neighbourhood are calculated as expressed by Equation 6.9:

$$I_{fb} = i - r + 1$$
$$J_{fb} = j - c + 1$$
$$K_{fb} = k - s + 1$$

$$I_{lb} = i - r + l_i$$
$$J_{lb} = j - c + l_j$$
$$K_{lb} = k - s + l_k$$  \hspace{1cm} (6.9)

where

$K_{fb}$: the $k$ address of the first block (element) within the $m^{th}$ NB and

$K_{lb}$: the $k$ address of the last element within the $m^{th}$ NB.

Using three given parameters, a general formula for any 3D neighbourhood ($NB_m$) with any order, $[O_{nb}]^{(3)}$, for any block, $B_{ijk}$, is expressed by Equation 6.10:
Chapter Six: Programming of the MVN Algorithm in Fortran 90

\[ NB_{m,ijk,l_1,l_2,l_3} = \left\{ \begin{array}{c} \alpha = i - r + 1, i - r + 2, \ldots , i - r + l_i \\ \beta = j - c + 1, j - c + 2, \ldots , j - c + l_j \\ \gamma = k - s + 1, k - s + 2, \ldots , k - s + l_k \end{array} \right\} \ \\
\]

\[ \begin{aligned} \mathbf{BEV}_{\alpha \beta \gamma} & \in U \\
\end{aligned} \]

given that

\[ r = m - (s - 1)(l_i l_j) - (c - 1)l_i + 1 \]

\[ c = \text{Int} \left( \frac{m - 1 - (s - 1)(l_i l_j)}{l_i} \right) + 1 \]

\[ s = \text{Int} \left( \frac{m - 1}{l_i l_j} \right) + 1 \]

where

\gamma: \quad \text{the section number of blocks within the model,} \\
BEV_{\alpha \beta \gamma}: \quad \text{the economic value of blocks within the } m^{th} \text{ neighbourhood,} \\
l_k: \quad \text{the order of neighbourhood in the Z direction and} \\
NB_{m,ijk,l_1,l_2,l_3}: \quad \text{the } m^{th} \text{ neighbourhood of the block, } B_{ijk}, \text{ with the order of } (l_i, l_j, l_k). \\

\subsection{6.4.3 Coding the Procedure}

The subroutines "NBStartPosition" and "NBEndPosition" which respectively determine the \((i, j, k)\) address of the starting and ending blocks of any neighbourhood, use the above formula. The main part of the Fortran 90 code for determining the starting block of a specified neighbourhood follows:
Chapter Six: Programming of the MVN Algorithm in Fortran 90

When the first block is determined, the address of the last block may be obtained, knowing that the number of blocks in each direction equals the order of neighbourhood in that direction. The corresponding Fortran 90 code is as follows:

\[
\begin{align*}
\text{WholeSections} &= \text{INT}\left(\frac{\text{NBNo}}{(\text{NBOrder}(1) \times \text{NBOrder}(2))}\right) \\
\text{FractSection} &= \text{MOD}\left(\text{NBNo}, (\text{NBOrder}(1) \times \text{NBOrder}(2))\right) \\
\text{WholeColumns} &= \text{INT}\left(\frac{\text{FractSection}}{\text{NBOrder}(1)}\right) \\
\text{FractColumn} &= \text{MOD}\left(\text{FractSection}, \text{NBOrder}(1)\right)
\end{align*}
\]

\[
\begin{align*}
\text{IF} \ (\text{FractColumn} /= 0) \ \text{THEN} \\
\ &\ \ F\text{I} = \text{ia} - (\text{FractColumn} - 1) \\
\ &\ \ F\text{J} = \text{ib} - \text{WholeColumns} \\
\ &\ \ F\text{K} = \text{ic} - \text{WholeSections} \\
\text{ELSEIF} \ (\text{WholeColumns} /= 0) \ \text{THEN} \\
\ &\ \ F\text{I} = \text{ia} - (\text{NBOrder}(1) - 1) \\
\ &\ \ F\text{J} = \text{ib} - (\text{WholeColumns} - 1) \\
\ &\ \ F\text{K} = \text{ic} - \text{WholeSections} \\
\text{ELSE} \\
\ &\ \ F\text{I} = \text{ia} - (\text{NBOrder}(1) - 1) \\
\ &\ \ F\text{J} = \text{ib} - (\text{NBOrder}(2) - 1) \\
\ &\ \ F\text{K} = \text{ic} - (\text{WholeSections} - 1)
\end{align*}
\]

\[
\text{ELSE IF} \ (\text{WholeColumns} /= 0) \ \text{THEN} \\
\ &\ \ F\text{I} = \text{ia} - (\text{NBOrder}(1) - 1) \\
\ &\ \ F\text{J} = \text{ib} - (\text{WholeColumns} - 1) \\
\ &\ \ F\text{K} = \text{ic} - (\text{WholeSections} - 1)
\]

\[
\text{END IF}
\]

When the first block is determined, the address of the last block may be obtained, knowing that the number of blocks in each direction equals the order of neighbourhood in that direction. The corresponding Fortran 90 code is as follows:

\[
\begin{align*}
\text{LI} &= F\text{I} + \text{NBOrder}(1) - 1 \\
\text{LJ} &= F\text{J} + \text{NBOrder}(2) - 1 \\
\text{LK} &= F\text{K} + \text{NBOrder}(3) - 1
\end{align*}
\]

6.4.4 Feasibility and Valuation of Neighbourhoods

A logical function, \textit{"NBTrue"}, checks the feasibility of each neighbourhood. The function receives the boundaries of a neighbourhood and returns a logical value, true or false. The neighbourhood is feasible if the function value is true and infeasible if the function value is returned as false. A neighbourhood is infeasible if, and only if, one of the member (blocks) is located outside of the block model (or the selected sub-region) on which the optimisation is performed. If this does occur, it means that at least one of the 3D addresses of the block is less than, or greater than, those of the block model. That is,

\[
\text{BEV}_{\alpha \beta \gamma} \text{ is outside the model } \Leftrightarrow \begin{cases} 
\alpha < 1 \quad \text{or} \quad \alpha > \text{NI} & \text{or} \\
\beta < 1 \quad \text{or} \quad \beta > \text{NI} & \text{or} \\
\gamma < 1 \quad \text{or} \quad \gamma > \text{NI}
\end{cases}
\]
Figure 6.8, shows a 2D neighbourhood, in which the element, $BEV_{a\beta}$, is known to be outside the block model.

![Diagram of a 2D neighbourhood with indices and block model]

**Figure 6.8:** A 2D Infeasible Neighbourhood (it is known that the element, $BEV_{a\beta}$, is outside the model.)

However, the first element has the lowest $(i, j, k)$ address and the last element has the highest one. Therefore, the feasibility of a neighbourhood may be determined using the first and last elements as follows.

\[
\begin{align*}
FI < \alpha \text{ and } LI > NI & : \quad \begin{cases} 
\text{if } \alpha < 1 & \Rightarrow FI < 1 \\
\text{if } \alpha > NI & \Rightarrow FI > NI 
\end{cases} \\
FJ < \beta \text{ and } LJ > NJ & : \quad \begin{cases} 
\text{if } \beta < 1 & \Rightarrow FI < 1 \\
\text{if } \beta > NJ & \Rightarrow FI > NJ 
\end{cases} \\
FK < \gamma \text{ and } LK > NK & : \quad \begin{cases} 
\text{if } \gamma < 1 & \Rightarrow FI < 1 \\
\text{if } \gamma > NK & \Rightarrow FI > NK 
\end{cases}
\end{align*}
\]

(6.11)

Hence, the feasibility of any neighbourhood can be expressed by Equation 6.12:
A neighbourhood is infeasible \(\Leftrightarrow\) 
\[
\begin{align*}
FI < IMn & \quad \text{or} \quad LI > IMx \quad \text{or} \\
FJ < JMn & \quad \text{or} \quad FJ > JMx \quad \text{or} \\
FK < KMn & \quad \text{or} \quad FK > KMx
\end{align*}
\] (6.12)

where

- \(FI, FJ, FK\): the \(i, j, k\) addresses of the first element within the neighbourhood,
- \(LI, LJ, LK\): the \(i, j, k\) addresses of the last element within the neighbourhood,
- \(IMn, JMn, KMn\): the \(i, j, k\) addresses of the first element within the block model,
- \(IMx, JMx, KMx\): the \(i, j, k\) addresses of the last element within the block model.

Below Fortran 90 code checks the feasibility of a neighbourhood:

```fortran
NBTrue = .True.
IF (FI .LT. Imin) NBTrue = .False.
IF (FJ .LT. Jmin) NBTrue = .False.
IF (FK .LT. Kmin) NBTrue = .False.
IF (LI .GT. Imax) NBTrue = .False.
IF (LJ .GT. Jmax) NBTrue = .False.
IF (LK .GT. Kmax) NBTrue = .False.
```

A real function, "\(NBValue\)" , is used to calculate the total economic value of the feasible neighbourhoods. The function uses the boundaries of the neighbourhood (the \(i, j, k\) addresses of the first and the last elements) and returns the net economic value contained in the neighbourhood. The function variable, "\(NBValue\)" , is initially set to "zero". Then, it is updated by adding the value of each element. Sections of the Fortran 90 code is provided below:
After the value of each neighbourhood is calculated, it is assigned to the corresponding element within the set of neighbourhood values. This is the NBV array.

6.5 UPDATING THE STOPE

Two features of the stope, which should be updated after the MVN of a block is located, are the stope blocks and the contained economic value. Updating the stope blocks is performed by:

(i) determining all members (blocks) of the maximum value neighbourhood (MVN);

(ii) distinguishing the MVN blocks which have already been included in the stope; and

(iii) marking the additional blocks if the marginal value provided by the MVN is not negative.

The "SelectBlock" subroutine performs the first two steps. Figure 6.9 shows the general flow-chart of the "SelectBlock" subroutine. The subroutine uses the neighbourhood number of the MVN (provided by the "MAXLOC" function) and determines the elements of the MVN by calling the "NBStartPosition" and "NBEndPosition" subroutines. The FF variable discriminates between the blocks that are already flagged and those that are not. Three successive loops are then used to examine each element of the MVN.
Chapter Six: Programming of the MVN Algorithm in Fortran 90

START

Input the element number of the maximum value neighbourhood (MVN)

NBStartPosition subroutine

Determine the elements of the MVN

NBEndPosition subroutine

Examine the first element

F = 1 ?

Set: FF = 1

Last element?

No

Yes

Examine the next element

Set: FF = 0

Figure 6.9: The General Flow-Chart of the "SelectBlock" Subroutine

The following lines of the Fortran 90 program are the major processes carried out in the "SelectBlock" subroutine:
\[ \text{NBNNumber} = \text{MVNBNo}(1) \]
\[ \text{CALL NBStartPosition}(a, b, c, \text{NBNNumber}, FBI, FBJ, FBK) \]
\[ \text{CALL NBEndPosition}(FBI, FBJ, FBK, LBI, LBJ, LBK) \]
\[ \text{DEALLOCATE (FF)} \]
\[ \text{ALLOCATE (FF(FBI:LBI, FBJ:LBJ, FBK:LBK))} \]
\[ \text{DO Section} = \text{FBK}, \text{LBK} \]
\[ \quad \text{DO Column} = \text{FBJ}, \text{LBJ} \]
\[ \quad \quad \text{DO Row} = \text{FBI}, \text{LBI} \]
\[ \quad \quad \quad \text{SELECT CASE} (F(\text{Row}, \text{Column}, \text{Section})) \]
\[ \quad \quad \quad \quad \text{CASE} (1) \]
\[ \quad \quad \quad \quad \quad \text{FF(\text{Row}, \text{Column}, \text{Section})} = 0 \]
\[ \quad \quad \quad \quad \text{CASE} (0) \]
\[ \quad \quad \quad \quad \quad \text{FF(\text{Row}, \text{Column}, \text{Section})} = 1 \]
\[ \quad \quad \quad \text{END SELECT} \]
\[ \quad \text{END DO} \]
\[ \text{END DO} \]
\[ \text{END DO} \]

where \text{NBNNumber} represents the neighbourhood number in the 1D set of neighbourhoods.

The marginal value of the MVN may be defined by the total economic value of the blocks whose \text{FF} indicator value is equal to 1. The marginal value, as a scalar value, may be obtained by multiplying the vector of the MVN elements (economic values of blocks contained in the MVN) by the elements of the vector of \text{FF} as in Equation 6.13.

\[
\text{Marginal Value} = \begin{bmatrix}
\text{BEV}_1 & \text{BEV}_2 & \ldots & \text{BEV}_{\text{MaxNumNB}} \\
\text{FF}_1 \\
\text{FF}_2 \\
\vdots \\
\text{FF}_{\text{MaxNumNB}}
\end{bmatrix}
\]  

(6.13)

where \text{MaxNumNB} represents the total number of elements in the neighbourhood.

The above multiplication is carried out by the "\text{MarginValue}" subroutine. The subroutine uses the boundaries of the MVN (lower and upper bounds of \text{i}, \text{j} and \text{k} subscripts) and uses the \text{FF} values of the MVN elements to obtain the marginal value.

When the obtained marginal value is negative, there is a loss if the neighbourhood is included in the stope, although the MVN is non-negatively valued. The process is then
stopped without updating the stope. However, if the marginal value is non-negative, the stope elements and the stope value are updated. The stope value is updated by adding the marginal value to the previous stope value. The stope elements are updated by assigning 1s to the corresponding $F$ elements of the $MVN$.

After the optimisation algorithm is performed for all blocks, the optimised stope is exported in the form of block flag data. The subroutine, "ExportFlag", reviews the block values and their flags. It double checks the stope value by multiplying the row vector $BEV$ by the column vector $F$ as expressed below.

$$SEV = \begin{bmatrix} BEV_1 & BEV_2 & \cdots & BEV_{NBlocks} \\ \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ \vdots \\ F_{NBlocks} \end{bmatrix}$$

(6.14)

where $NBlocks$ represents the number of blocks in the block model (or the selected sub-region). The block flag data are then exported to the output file, (*.out).

### 6.6 SUMMARY

The process of the computer implementation of the $MVN$ optimisation algorithm, using the Fortran 90 code, has been discussed in this chapter. The structure of the major modules, used to implement the optimisation stage of the $SLO$ and their relationships, were reviewed. "SelectSubs", "ReadDollarData", "MaxValueNB", "SelectBlock", "MarginValue" and "ExportFlag" are sub-programs used to perform this algorithm. A numbering rule was defined to identify neighbourhoods within the sets and blocks of the neighbourhood. This allowed the 2D and 3D neighbourhoods to be reduced to 1D sets. The next chapter discusses the output from the $SLO$ program.
CHAPTER SEVEN

OUTPUT RESULTS OF THE STOPE LIMIT OPTIMISER

7.1 INTRODUCTION

The Stope Limit Optimiser (SLO) provides an interactive environment for plotting plans and sections of the optimised stope. An interface with a text editor is provided to view the plot, while the user is in the SLO environment. This chapter discusses the SLO output files. There are three types of outputs produced in the SLO system. These are: the block flag data; the stope plan/section plots; and various reports. The main output of the optimisation stage is the block flag data, which is the source for the visualisation of the optimised stope, either by the SLO itself, or by other visualisation software. The block flag data identifies which blocks define the optimum stope geometry. The second type of the SLO output consists of plots, in the form of plans or sections, of the optimised stope which are displayed by the SLO system. In addition, the SLO system produces some output files which contain intermediate and final results.

There are a total of four files, which collect the SLO outputs. These include:

1. the flag data of blocks: “*.out” files,
2. the plots of the stope plans or sections: “*.plt” files,
3. the intermediate results: “*.res” files and
4. the neighbourhood results: "*.nbv" files.

7.2 BLOCK FLAG DATA

The flag data of the blocks is the main output of the stope limit optimiser (SLO) which is stored in a three-dimensional array. After the optimisation is completed, the array values are written to an ASCII output file, "*.out", using the "ExportFlag" subroutine. The name of the "*.out" file is the same as that of the input data file. It means that the name of the input data file, "*.dat" is kept during the process and only the extension is changed in the different steps. That is, the name of the files with the extensions "*.dat", "*.asy", "*.bev" and "*.out" are identical. The "*.out" file may be imported into other mine planning packages to display the end results in 2D and 3D views.

For each block of the (sub) region, two types of data are provided. The first being the block address, in terms of the block position number, in the block model. The block address contains a group of three integer values, which represent the i, j and k addresses of the block. The second being the economic value and the flag information of each block. The format of the "*.out" file is simply "3I5, F15.2, I5", which is fixed in the program. Figure 7.1 is an example of such an output file.

7.3 DISPLAY OF RESULTS

Although the "*.out" file may be used by other applications to display the optimised stope, the SLO provides utilities for the user to produce and view the plots of the ultimate stope boundaries. However, the SLO uses the existing three-dimensional array of flag data, rather than the "*.out" file, to plot the two-dimensional plans and sections of the final stope.
<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>k</th>
<th>BEV</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2336676.25</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3158662.75</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>-35500.00</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>-35500.00</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>-35500.00</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>-35500.00</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>5023157.50</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>-35500.00</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>-35500.00</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>-35500.00</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>-35500.00</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>3595499.00</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>-35500.00</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>-35500.00</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>4503357.50</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>-35500.00</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>-35500.00</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3525102.50</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>-35500.00</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>6613739.50</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2444607.00</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>-35500.00</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>-35500.00</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>-35500.00</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>-35500.00</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>-35500.00</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>-35500.00</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>6062757.00</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2022957.88</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>-35500.00</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>-35500.00</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
<td>-35500.00</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>-35500.00</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3988670.00</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>-35500.00</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
<td>5583853.00</td>
<td>1</td>
</tr>
</tbody>
</table>

**Figure 7.1:** Example of the Output File, ".out" (the block model is $4 \times 3 \times 3$)

The user may select the plotting of any of the 2D views of the stope, including X-Y plans, X-Z sections and Y-Z sections, or all of the plots. It is also possible to specify a certain plan, section, or a range of plans or sections of the optimised stope for plotting, by specifying the minimum and maximum number of the plans (sections) in the range. Figure 7.2 shows the SLO dialog box for selecting plans, or sections, of the optimised stope to be plotted.
Chapter Seven: Output results from SLO

The subroutine "Kplan" is called to plot a set of plans. The minimum and maximum range is used to make a loop for replication of the subroutine. It should be noted that the direction of the axes used for plotting in the SLO system, is different from that used in the Excel implementation, although both of them follow the standard Cartesian coordinate system. Figure 7.3 identifies the principal directions used by the SLO in 2D plots.

**Figure 7.2:** SLO Dialog Box for Plotting Specified Plans or Sections

**Figure 7.3:** The Orientation of Plots in the SLO Program
Each X-Y plan is named as the "k" number and contains the flag data of the relevant blocks, which are arranged in a rectangular matrix form. The matrix is then surrounded by a border that contains the required annotations. The "i" and "j" addresses of the blocks are used to annotate the matrix of the flag values in the X and Y axes, respectively. Figure 7.4 is a plan view example of an optimised stope.

**Figure 7.4:** Example of the Plans/Sections Plotted in the SLO Program

Figure 7.5 shows the general structure of the main Fortran modules, used by SLO to plot any 2D plan, or section of the optimised stope. Arrows are drawn from the caller subroutines to the called subroutines. The body of a plot contains the optimisation
results (flag data), while the borders contain the annotations and addresses of the blocks. Three separate subroutines have been developed to plot the 2D sections along the three principal directions. These are called "Kplan", "Isection" and "Jsection". The plotting of the top and bottom borders is a common task that should be carried out by all of these subroutines. However, to plot the flagged blocks of each 2D section, either the "Isection" or "Jsection" subroutine is used.

Figure 7.5: Main Subroutines Called to Plot any Plan or Section
7.3.1 Designing the Borders of the Plan / Section

The top border of the plan consists of four lines, as seen in Figure 7.4.

- The first line contains the name of the block address in the horizontal axis, that is, the "I" in the X-Y plans. The character variable, "Axis", is used to store the content "I" for the plans. The character is then written in the middle of the horizontal axis; three character spaces are provided for each block.
- The second line is blank.
- The third line contains the column number of the plot. This number is the block's i address in the X-Y plans. Since there are three character spaces allocated for each block, up to two-digit addresses may easily be displayed with one space between them.
- The fourth line contains a simple border. For each block, three consecutive asterisks are provided.

The following lines in Fortran 90 are used to display the top border of a plot.

```
! Top annotation
Hmid = INT((Hmax - Hmin) / 2) + Hmin
WRITE (9, '/(5X)', ADVANCE = 'NO')
DO ih = Hmin, Hmid
   WRITE (9, '(3X)', ADVANCE = 'NO')
END DO
WRITE (9, '(A)') Axis
! Top axis units
WRITE (9, '/(T7)', ADVANCE = 'NO')
DO ih = Hmin, Hmax
   WRITE (9, '(13)', ADVANCE = 'NO') ih
END DO
! Top horizontal border
WRITE (9, '/(T6, A)', ADVANCE = 'NO') "***"
DO ih = Hmin, Hmax
   WRITE (9, '(A)', ADVANCE = 'NO') "***"
END DO
WRITE (9, '(A)', ADVANCE = 'NO') "***"
```

Similarly, the following Fortran 90 codes are used to display the bottom border of a plot:
7.3.2 Designing the Body of the Plan / Section

The body of the plan/section consists of two different formats. The format of the middle row of blocks is different from that of the other rows. The middle row, uniquely contains:

1. the name of the vertical axis, that is, "J" in the X-Y plans,
2. the block address in the Y direction, which is the \( j \) address of the blocks in the X-Y plans,
3. an asterisk (as the border),
4. the list of block flags (three character spaces are allocated to output the flag of each block),
5. an asterisk,
6. the row number of the plot and
7. the name of the vertical axis.

The subroutine used to display the main body plots the upper, middle and lower rows. The corresponding *Fortran 90* code for plotting the main body of the X-Y plans is provided below:
Chapter Seven: Output results from SLO

\[ \text{Jmid} = \text{INT}((\text{Jmax} - \text{Jmin}) / 2) + \text{Jmin} \]

\[
\begin{align*}
\text{DO ja} &= \text{Jmax, Jmid + 1, -1} \\
\text{WRITE} (9, '(/, T3, I2, X, A)', \text{ADVANCE='NO'}) & \text{ ja, "*"} \\
\text{DO ia} &= \text{Imin, Imax} \\
\text{Results} & \\
\text{WRITE} (9, '(I3)', \text{ADVANCE='NO'}) & \text{ F(ia, ja, ka)} \\
\text{END DO} & \\
\text{WRITE} (9, '(X, A, X, I2)', \text{ADVANCE='NO'}) & \text{"*", ja} \\
\text{END DO} & \\
\text{Mid row} & \\
\text{WRITE} (9, '(/, A, X, I2, X, A)', \text{ADVANCE='NO'}) & \text{"J", ja, "*"} \\
\text{DO ia} &= \text{Imin, Imax} \\
\text{Results} & \\
\text{WRITE} (9, '(I3)', \text{ADVANCE='NO'}) & \text{ F(ia, ja, ka)} \\
\text{END DO} & \\
\text{WRITE} (9, '(X, A, X, I2, X, A)', \text{ADVANCE='NO'}) & \text{"*", ja, "J"} \\
\text{First half} & \\
\text{DO ja} &= \text{Jmid - 1, Jmin, -1} \\
\text{WRITE} (9, '(/, T3, I2, X, A)', \text{ADVANCE='NO'}) & \text{ ja, "*"} \\
\text{DO ia} &= \text{Imin, Imax} \\
\text{Results} & \\
\text{WRITE} (9, '(I3)', \text{ADVANCE='NO'}) & \text{ F(ia, ja, ka)} \\
\text{END DO} & \\
\text{WRITE} (9, '(X, A, X, I2)', \text{ADVANCE='NO'}) & \text{"*", ja} \\
\text{END DO} &
\]

All plans and sections are saved in an ASCII file with an extension "**.plt". The names of the plot files are identical to that of the data file, "**.dat".

7.4 REPORTS

The SLO program reports the information obtained for the optimised blocks at each stage, in addition to the "**.out" and "**.plt" files. This information is updated as the optimisation progresses and is finally saved in the report files. These include reports on the intermediate results, the neighbourhood results as well as a final on the screen summary report.

7.4.1 Intermediate Results

All intermediate results obtained during the optimisation are saved in a separate file with the extension "**.res". The name of the file is the same as that of the data file "**.dat". This will help the user to see the variation of the stope after the optimiser
examines each block. Results obtained for each block are written on separate lines, in the order that the blocks are counted in the model, and processed by the optimiser.

There are three groups of information which are collected for each block in the "*.res" files. These include:

1. the block addresses,
2. the block values and
3. the optimisation results obtained for the block.

In the first group both addresses of the blocks used in the optimisation, are reported in the file. These include the $(i, j, k)$ address and the one-dimensional sequential order of the block (the ID address). The block values include either the assay values or the economic value of the block. In cases where by-products are available, extra columns are required to write the grade value of the by-products. The real intermediate results (the third group) contain: the neighbourhood $(NB)$ number within the set of neighbourhoods of the block, which provides the maximum value; the maximum neighbourhood value (the $MNBV$); the marginal value obtained from the maximum value neighbourhood $(MVN)$ of the block; and finally the updated stope value.

The above information is produced by the SLO system as the optimisation progresses over the blocks. Headings for the table of results in the file are defined before initiating the optimisation loop for the blocks. Figure 7.6 shows typical intermediate results generated by SLO. The headings of the file are specified using the following lines of Fortran 90 code:

```
WRITE (7, '(/, 4A5, A10, A15, A7, 3A15/)') &
  "i", "j", "k", "ID", &
  "Grade(%)", "BEV ( $ )", &
  "MVNBNO", "MNBV", "MarginVal", "SEV"
```
<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>k</th>
<th>ID</th>
<th>Grade($)</th>
<th>BEV($)</th>
<th>MVNBNO</th>
<th>MNBV</th>
<th>MarginValue</th>
<th>SEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.10</td>
<td>570438.38</td>
<td>1</td>
<td>3576668.00</td>
<td>3576671.25</td>
<td>3576671.25</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4.30</td>
<td>2232477.25</td>
<td>1</td>
<td>3836361.00</td>
<td>827469.44</td>
<td>4404140.50</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>0.90</td>
<td>466560.94</td>
<td>4</td>
<td>3836361.00</td>
<td>0.00</td>
<td>4404140.50</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>9</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>10</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>11</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>12</td>
<td>1.20</td>
<td>622377.13</td>
<td>4</td>
<td>2179516.00</td>
<td>1713846.50</td>
<td>6117987.00</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>13</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>14</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>15</td>
<td>0.70</td>
<td>362683.50</td>
<td>6</td>
<td>3836361.00</td>
<td>0.00</td>
<td>6117987.00</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>16</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>17</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>18</td>
<td>1.50</td>
<td>778193.25</td>
<td>7</td>
<td>3836361.00</td>
<td>0.00</td>
<td>6117987.00</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>19</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>20</td>
<td>2.10</td>
<td>1095019.38</td>
<td>6</td>
<td>2179516.00</td>
<td>0.00</td>
<td>6117987.00</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>21</td>
<td>0.30</td>
<td>154928.66</td>
<td>3</td>
<td>2288588.00</td>
<td>1511285.50</td>
<td>7629272.50</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>22</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>23</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>24</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>25</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>26</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>27</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>28</td>
<td>1.30</td>
<td>674315.75</td>
<td>6</td>
<td>2127577.00</td>
<td>670765.75</td>
<td>8300038.00</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>29</td>
<td>2.02</td>
<td>1048274.56</td>
<td>5</td>
<td>2288588.00</td>
<td>0.00</td>
<td>8300038.00</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>30</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
<td>31</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>32</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>33</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>34</td>
<td>0.60</td>
<td>310744.81</td>
<td>8</td>
<td>2288588.00</td>
<td>0.00</td>
<td>8300038.00</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>35</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>36</td>
<td>0.00</td>
<td>-887.50</td>
<td>0.00</td>
<td>-887.50</td>
<td>-887.50</td>
<td>-887.50</td>
</tr>
</tbody>
</table>

**Figure 7.6:** Example of the Intermediate Optimisation Results Collected in the "*.res" Files
After examining each block, SLO outputs the corresponding results to the file, line by line. If the block has a negative value, then there is no processing result for the block, hence, only addresses and values of the block are recorded. To do so, the IDNumber function is called to obtain the ID address of the block and to read the assay value of the block from the IDth record in the “*.asy” file. The corresponding Fortran 90 code is provided below:

```fortran
ID = IDNumber(i, j, k, imx, jmx)
READ (4, rec = ID) Grade
WRITE (7, '(4I5, 5F10.2, F15.2)') 
   i, j, k, ID, Grade, BEV(i, j, k)
```

For the non-negative valued blocks, the process continues to find the maximum value neighbourhood. If the maximum neighbourhood value (MNBV) is negative, only two items of the intermediate results are recorded as the process is not continued for the block. The recorded items, which refer to the MVN information of the block, include the number and value of the maximum value neighbourhood. Below is the code for recording the intermediate results of blocks with a negative MNBV in Fortran 90.

```fortran
ID = IDNumber(i, j, k, imx, jmx)
READ (4, rec = ID) Grade
WRITE (7, '(4I5, 5F10.2, F15.2, 17, F15.2)') 
   i, j, k, ID, Grade, BEV(i, j, k), 
   MVNBNo, MVN
```

Continuing the procedure for the blocks with a non-negative MNBV (if the marginal value obtained for the block is negative) then the third item is also recorded in the file. However, the fourth item, still, cannot be recorded since the process is not completed for that block. The corresponding grade value is read from the “*.asy” file using the ID of the block and the information is written to the “*.res” file. If the marginal value provided by the MVN of the block is not negative, all the information regarding the intermediate results, including the updated stope value, is recorded in the “*.res” file, because the block passes the whole optimisation procedure. The intermediate results file may be updated by following the Fortran 90 codes below.
The "*.res" file is saved in the ASCII format.

**7.4.2 Neighbourhood Results**

The information collected in the "*.res" files does not refer to the details of the results of the optimisation. For example, it is useful to have the information concerning the results obtained for the neighbourhoods of the blocks. These are not referred to in the intermediate results file. Therefore, another report file keeps the results obtained for the neighbourhoods of each block.

The available information for each block is provided on a separate line. The following intermediate information is saved for each block:

1. both the \((i, j, k)\) and \(ID\) addresses of the block (containing four fields),
2. the net block value, \(BEV\) (containing one field), and
3. the set of neighbourhood values for the block.

In cases where the order of neighbourhood is \((2, 2, 2)\), the set of neighbourhoods \((NBS)\) consists of eight neighbourhoods. This means that there could be a maximum of eight neighbourhood values for each block, which necessitates the need for 13 fields to record the neighbourhood results for the block. The neighbourhood results are saved in an ASCII formatted file with the extension of "*.nbv" with the same file name as that of the data file "*.dat".

In cases where the block economic value \((BEV)\) is negative, no neighbourhood value is reported as the optimisation procedure is not continued for the block. This means that only the addresses and the value of the block are reported. If the block is not negatively valued, all of the neighbourhood values are reported, regardless of whether the
maximum neighbourhood value ($MNBV$) or the marginal value, is negative. However, the neighbourhood values ($NBV$) for infeasible neighbourhoods have sufficiently large negative values as these values are so assigned during the process, in the “MaxValueNB” subroutine.

After the optimisation is completed, the user may use Notepad to view the neighbourhood results for each block. Figure 7.7 is an example of such an "*.nbv" file.

7.4.3 Summary Report

After the optimisation is completed, a summary report of the optimisation results is provided and appended to the end of the intermediate results file, "*.res". The summary report includes information such as: the total number of blocks within the region; number of negative and non-negative valued blocks in the region; number of negative valued blocks included in the optimised stope; and their total values; number of non-negative valued blocks excluded from the optimised stope; and their total values; total stope value; and the percentage of the block values included in the ultimate stope. The user may access this report by selecting an option from the menu bar, which leads to an on screen summary report provided in a dialog box. Figure 7.8 is a typical on screen summary report.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>BEV($)</th>
<th>NBV($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4563507.00</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td>1</td>
<td>19</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>21</td>
<td>1</td>
<td>3</td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td>3</td>
<td>2</td>
<td>22</td>
</tr>
<tr>
<td>23</td>
<td>3</td>
<td>2</td>
<td>23</td>
</tr>
<tr>
<td>24</td>
<td>3</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>3</td>
<td>25</td>
</tr>
<tr>
<td>26</td>
<td>2</td>
<td>1</td>
<td>26</td>
</tr>
<tr>
<td>27</td>
<td>3</td>
<td>1</td>
<td>27</td>
</tr>
<tr>
<td>28</td>
<td>4</td>
<td>1</td>
<td>28</td>
</tr>
<tr>
<td>29</td>
<td>1</td>
<td>3</td>
<td>29</td>
</tr>
<tr>
<td>30</td>
<td>2</td>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>31</td>
<td>2</td>
<td>2</td>
<td>31</td>
</tr>
<tr>
<td>32</td>
<td>2</td>
<td>2</td>
<td>32</td>
</tr>
<tr>
<td>33</td>
<td>3</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>34</td>
<td>3</td>
<td>1</td>
<td>34</td>
</tr>
<tr>
<td>35</td>
<td>3</td>
<td>3</td>
<td>35</td>
</tr>
<tr>
<td>36</td>
<td>3</td>
<td>3</td>
<td>36</td>
</tr>
</tbody>
</table>

**Figure 7.7:** Example of the Neighbourhood Results Collected in the "*.nbv" Files
**Figure 7.8:** A Screen Summary Report
CHAPTER EIGHT

VALIDATION OF THE STOPE LIMIT OPTIMISER

8.1 INTRODUCTION

The stope limit optimiser, SLO, was validated to determine how accurately it defines the stope boundaries. This validation relies basically on the logic of the MVN concept and how the algorithm was implemented to outline the optimum stope. This chapter discusses the procedures used to validate the SLO program.

8.2 OPTIMISER VALIDATION

In validating the stope optimiser, the method of optimisation and the output results from the system should be checked for their accuracy. One way to verify the results of the optimiser is to compare them with the “true” value. Unfortunately, there was no mine data available for comparison. What a heuristic stope optimiser really attempts to do is to outline a proposed stope as close to the optimum geometry as possible. This means that the SLO program output cannot be validated with existing true mine data. What can be validated is the ability of the stope optimiser in recovering the optimum ore blocks for each deposit.
8.3 THE VALIDATION PROCEDURE

In this thesis, the validation procedure has four stages. In the first stage, the results of a simple block model example from the SLO program were manually checked. The block model size was further increased and the performance of the optimiser was double-checked. The third stage compared the results of the SLO program and those of the algorithm coded in Excel modules. Finally, the output of the SLO program to a test file was manually checked to ensure that the optimisation process was working properly.

8.3.1 Manual Validation

A block model of $3 \times 3 \times 3$ blocks shown in Figure 8.1, was optimised by the SLO system and then checked manually. The input data is given in terms of block economic values, which are selected randomly from a range of integer values located in $[-5, 5]$. The three-dimensional order of neighbourhood is assumed in the above example with a minimum stope size equivalent to two blocks, that is, $[O_{nb}]^{(3)} = (2, 2, 2)$.

Figure 8.2 shows the sections of the optimised stope, plotted in the SLO, while Figure 8.3 illustrates the optimised stope obtained by the manual method. The results obtained by optimisation of the block model in the manual method matches exactly those obtained by the SLO. Please note that the type of the co-ordinate system in the manual method is different from that used by the SLO optimiser. However, both types follow the standard Cartesian system.
Figure 8.1: A Simple Block Model

Figure 8.2: Optimised Stope Example (using SLO)
8.3.2 An Example with a Special Pattern

A block model of $6 \times 8 \times 6$ blocks was used in the next example. The blocks economic values, shown in Figure 8.4, were duplicated for each of the six levels. The order of neighbourhood is again assumed to be as simple as $(2, 2, 2)$. The SLO optimiser produced an identical pattern of the flagged blocks, shown in Figure 8.5, on each of the six levels as one would expect.
Figure 8.4: Dollar Value Data of the Block Model Repeated in all Levels

Figure 8.5: The Optimised Stope Plot in all Levels (plans)
8.3.3 Validation with Excel VBA Modules

In Chapter Four, a computer program was developed to implement the MVN algorithm using the Visual Basic for Applications (VBA) in Excel. The results from the VBA optimiser, for a small three-dimensional ore-body, were compared with those generated by the Fortran 90 stope limit optimiser (SLO). In this example, the block model was the same as that used in the second stage (Figure 8.4). Figure 8.6 gives the results obtained from the VBA optimiser. Identical results were generated by the Fortran 90 SLO (as shown in Figure 8.5). It should be noted that the type of the co-ordinate system used by the VBA optimiser is different from that used by the SLO program, as the structure of the spreadsheets in the Excel workbooks is more compatible with the type used by the VBA modules. However, both types follow the standard Cartesian system.

<table>
<thead>
<tr>
<th>i \ j</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-3</td>
<td>1</td>
<td>0</td>
<td>-4</td>
<td>3</td>
<td>-2</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>-1</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>-1</td>
<td>-5</td>
<td>3</td>
<td>-1</td>
<td>-2</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>-4</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>-5</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 8.6: The Optimised Stope Section Obtained in the Excel Worksheets

8.3.4 Program Debugging

In order to check the logic of all the steps used to develop the SLO program, the values of all the variables used by the program were output to a test file. The file content includes appropriate messages, variable names and their respective values. The file is automatically assigned the same name as the project name. The four major sections of
the test file are:

1. Block assay information;
2. Block economic values;
3. Logic of the optimisation; and
4. Optimum stope geometry.

8.3.4.1 Block Assay Information

Figure 8.7 is an example of a "*.tst" file showing block assay data.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0.70</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.010000</td>
<td>0.000001</td>
<td>0.000001</td>
<td>0.010000</td>
<td>0.010000</td>
</tr>
<tr>
<td>2300.00</td>
<td>500.00</td>
<td>90.00</td>
<td>500.00</td>
<td>1000.00</td>
</tr>
<tr>
<td>1</td>
<td>35242</td>
<td>35242</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0.90000</td>
<td>0.80000</td>
<td>0.80000</td>
<td>0.80000</td>
<td>0.80000</td>
</tr>
<tr>
<td>14.49</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.01</td>
<td>35500.00</td>
<td>5112000.00</td>
<td>284.00</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 8.7:** Example of the Information Stored in "*.tst" Files Related to the Block Assay Information

In Figure 8.7,

Line 1: specifies the one-dimensional (ID) address of the block.

Line 2: contains the grade value of the products, that is, the main product plus four possible by-products.

Line 3: contains the grade factor of the products, which indicate the proportion of the product in the ore.

Line 4: contains the price value of the products.

Line 5: contains the price factors, which represent the factors for converting the prices of the by-products into the price of the main product.

Line 6: includes the total recoveries of the products in terms of their proportions.

Line 7: includes the gross values obtained from the products.
Chapter Eight: Validation of the SLO Optimiser

8.3.4.2 Block Economic Values

The second section of information collected in the test file includes the block economic values read by the optimiser. This information is arranged in the form of a table which contains the final BEV of each block. This is used in the optimisation process. Each line of the table includes the block addresses, the block assay values and the block economic value. The block addresses contain both the three-dimensional (i, j, k) address and the one-dimensional ID address of each block. The assay values of the block include the grade of the main product as well as the grades of up to four by-products. In cases where the economic values are entered directly, the assay values are not displayed. The last column is the BEV of the block. Figure 8.8 illustrates an example of this information, for a number of blocks.

8.3.4.3 Optimisation Process

The third section of the "*.tst" file contains all the information created during the optimisation process. It should contain the results of all the calculations at each stage of the optimisation process; this information is useful in tracing the logic of the algorithm. Thus it is necessary to report suitable messages when a branch is selected at decision-making points. For example: when the block under consideration has a negative net value, a message reading: "The block ... is negative." is reported to show that the
process is halted for that block; or when a neighbourhood value is recorded as "-99999.00", it is known that the neighbourhood is not feasible and will not be included in the process of locating the maximum NBV.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>Grades</th>
<th>BEV ( $ )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>i</td>
<td>1.10</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>j</td>
<td>4.30</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>k</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
<td>ID</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>5</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>6</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>7</td>
<td></td>
<td>0.90</td>
<td>0.00</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>8</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>9</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>10</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**Figure 8.8:** Example of the Block Economic Data Stored in a "*.tst" File

The collected information in this section also contains the address of the block, its BEV, NBVS, MNBV, marginal value and the SEV. If the BEV of the block is negative, a message is displayed, otherwise, the set of the neighbourhoods of the block is constructed and all the neighbourhood values are stored. In cases where the NB is infeasible, a message is displayed. After locating the maximum NBV, the NBV is stored together with its element identification within the set of the NBVs. In addition, the obtained marginal value and the stope value are written to the test file. If the maximum NBV or the marginal value is negative, appropriate messages are displayed to show that the process is halted for the block. Figure 8.9 illustrates an example of the information collected in the "*.tst" file.

**8.3.4.4 Optimum Stope Geometry**

After the optimisation process is completed, a summary of the intermediate results of the process is provided and written to the test file. A summary information for each block appears on a separate line. For each block of the block model, its three-dimensional \((i, j, k)\) address is given, followed by the dollar value of the block. Next, the
flag variable $F$ of the block is written to indicate whether or not the block is selected (1 or 0) for the optimum stope. The stope value, which is updated after considering the \( BEV \) and $F$ variables of the block, is shown as the last column. Figure 8.10 illustrates an example of the above information for a number of blocks.

\[
\begin{align*}
BEV(222) &= 41180.00 \\
NBV(1) &= -176648.00 \\
NBV(2) &= -58049.60 \\
NBV(3) &= -171536.00 \\
NBV(4) &= -104057.60 \\
NBV(5) &= -161312.00 \\
NBV(6) &= -191984.00 \\
NBV(7) &= 94288.00 \\
NBV(8) &= 68728.00 \\
MVNBNo &= 7 \\
MaxValueNB &= 94288.00 \\
MarginValue &= 0.00 \\
StopeValue &= 148520.00
\end{align*}
\]

\[
\begin{align*}
BEV(322) &= -35500.00 \\
\text{Dollar value for Block (322) is negative.}
\end{align*}
\]

\[
\begin{align*}
BEV(422) &= 72363.19 \\
NBV(1) &= -99999.00 \\
NBV(2) &= -79008.81 \\
NBV(3) &= -99999.00 \\
NBV(4) &= -73896.82 \\
NBV(5) &= -99999.00 \\
NBV(6) &= -68784.81 \\
NBV(7) &= -99999.00 \\
NBV(8) &= -94344.81 \\
MVNBNo &= 6 \\
MaxValueNB &= -68784.81
\end{align*}
\]

Block (422) cannot pay for its neighbours.

\textbf{Figure 8.9:} Example of the Optimisation Process Results Stored in "*.tst" Files
<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>k</th>
<th>BEV</th>
<th>F</th>
<th>SEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>20732.00</td>
<td>1</td>
<td>20732.00</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>184316.00</td>
<td>1</td>
<td>205048.00</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>-35500.00</td>
<td>1</td>
<td>169548.00</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>-35500.00</td>
<td>0</td>
<td>169548.00</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>-35500.00</td>
<td>1</td>
<td>134048.00</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>-35500.00</td>
<td>1</td>
<td>98548.00</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>10508.00</td>
<td>1</td>
<td>109056.00</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>-35500.00</td>
<td>0</td>
<td>109056.00</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>-35500.00</td>
<td>0</td>
<td>109056.00</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>-35500.00</td>
<td>0</td>
<td>109056.00</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>-35500.00</td>
<td>0</td>
<td>109056.00</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>25844.00</td>
<td>0</td>
<td>109056.00</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>-35500.00</td>
<td>1</td>
<td>73556.00</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>-35500.00</td>
<td>1</td>
<td>38056.00</td>
</tr>
</tbody>
</table>

**Figure 8.10:** Example of the Summary Results Stored in the "*.tst" Files
CHAPTER NINE

SUMMARY AND CONCLUSIONS

9.1 SUMMARY

Mine geometry optimisation is important in mining design as it has substantial influence on the mine economy and is the basis of production scheduling. In the case of open pit mines, many algorithms have been developed during the past 35 years to optimise the pit limits, yet few algorithms are available for the optimisation of underground stope geometry. The major algorithms for open pit mines include the Moving Cone technique, the Dynamic Programming algorithm, the Graph Theory algorithm and the Network Flow technique.

The open pit algorithms are not applicable to underground mines because of the specific constraints associated with each of them. Those methods, used for underground optimisation, have been tailored for a specific mining method, and lack rigorous mathematical proof, or fail to be implemented on 3D problems. Algorithms developed for the optimisation of stope geometry include: the 2D Dynamic Programming algorithm for the block caving method; the downstream geostatistical approach, developed for the cut-and-fill and sub-level stoping methods used in a uranium mine; the Octree Division algorithm, the Floating Stope algorithm of Datamine; and the application of the Branch and Bound technique. None of these algorithms guarantee the true optimal solution for the stope layout, and no comprehensive mathematical solution
has been proposed for stope geometry optimisation.

A heuristic algorithm, termed the "Maximum Value Neighbourhood" (MVN), was developed in this thesis for the optimisation of the stope boundaries. Basic differences, between open pit and underground mines, were taken into account when developing the MVN algorithm as the differences influence the mine geometry constraints. These differences include: the pit/stope shape; a variety of possible pits/stopes based on a block; and the mutual dependency among blocks within a pit/stope. The neighbourhood concept, introduced in this thesis, is based on the minimum stope size as the critical constraint in the stope geometry. The minimum stope size is defined in terms of the block size of a given block model. In a 1D case, a "neighbourhood" (NB) is the set of all sequential blocks, including the block of interest, which could be mined to satisfy the minimum stope size requirement. The size of this set is called the "order of neighbourhood" ($O_{nb}$) and the total value of the set defines the neighbourhood value ($NBV$). Since a variety of neighbourhoods are available for each block, the one that provides the maximum net value (the maximum value neighbourhood, MVN) defines the final stope geometry. The blocks located at the boundaries of a block model will find some of their neighbourhoods non-feasible, since some of the neighbourhood elements may be located outside the block model.

The neighbourhood concept suggests a small island of mineable blocks for an individual block. The MVN algorithm applies the neighbourhood concept to all blocks of the ore-body so as to form a union of blocks to define the stope outline. The MVN algorithm considers all blocks in the block model. For each non-negative valued block, the feasible neighbourhoods are determined, the maximum $NBV$ is calculated and elements of the MVN are flagged. Some checkpoints are provided in the algorithm to suspend the optimisation in some cases, especially when the maximum neighbourhood value ($MNBV$) is negative.

The basics of the neighbourhood concept for 1D cases may be extended to cover 2D and 3D neighbourhoods. The minimum number of blocks, required to satisfy the minimum
stope size, in each direction, determines the order of neighbourhood in that direction. That is, while the order of neighbourhood in a 1D case is a scalar integer number, in 2D and 3D cases, it is expressed as an ordered set of integer numbers with two and three elements, respectively. The size of the neighbourhood is then, the product of its elements.

In order to validate the methodology, the MVN algorithm was applied manually to small examples with 1D and 2D neighbourhoods. In addition, the 3D MVN algorithm was implemented on an example block model using the Visual Basic for Applications (VBA) modules, supported by Microsoft Excel. The VBA code of the algorithm benefits from the general structure of the Excel worksheets for storing block data and displaying the optimised stope. In order to apply the MVN algorithm, on real-sized geological data, the algorithm was coded using a general programming language, such as Fortran 90.

The "Stope Limit Optimiser" (SLO) was developed during this thesis to implement the 3D MVN algorithm. The SLO program integrates the Fortran 90 programming with the Winteracter user interface features. This provides a Windows based interactive environment for the user to: define a project; view and edit the project; perform the MVN algorithm to optimise layouts of the selected stope; export results and plots; and display the plans/sections of the optimised stope. Input to the SLO program may be either block assay values, or block economic values. The data preparation stage of the SLO converts the assay data values into economic values using the project economic factors. The SLO accepts up to four by-products as well as the main product, and uses the equivalent grade of all products to calculate the economic value of the block.

The optimum stope blocks are flagged with indicator values of 1. The flagged data are written to a file that can be imported to other application programs. Each block flag is assigned 1, if the block is included in the stope, and 0 otherwise. The SLO provides an interactive environment for plotting any plan or section of the optimised stope. The user may select any plan/section of the optimised stope for plotting. The SLO arranges the flag values of blocks, in a plan or section order, and wraps them in a table format with
suitable annotations indicating the addresses of each block so as to display the plots in a 2D view. The user can use Notepad to view the plots while in the SLO environment.

Validation of the SLO program was carried out in four stages. In the first stage, a very simple example was run both manually and by the SLO program and the results were compared. In the second stage, a relatively larger example with the level block economic values duplicated on six different levels was used. The SLO program provided six identical plans. In the third stage, another example was optimised by both the SLO program and the Excel VBA based program and their results were compared. Finally, a test file was created to store the values of all the variables used by the optimiser. The test results were to validate the logical flow of the optimiser.

9.2 CONCLUSIONS

The MVN algorithm, developed during this thesis, provided optimum stope geometry for a given block model. The developed system relies on a fixed economic block model. It uses a heuristic approach for optimising ultimate limits of underground metalliferous mines. The MVN algorithm benefits from its generality and its simplicity, in both the concept and computer implementation, as well as providing a 3D analysis of the optimum solution. The computer implementation of the algorithm, the SLO, is simple. The tool defines block models, stope geometry constraints and economic parameters, it performs the MVN algorithm and plots the optimised stope in 2D view plans and/or sections.

The MVN algorithm can be applied to any underground metalliferous mines, regardless of their mining methods. However, it is restricted to a 3D fixed block model as most of the algorithms, developed for optimisation of the mine layout, have the same restriction. Table 9.1 summarises a comparison among the existing stope geometry optimisation algorithms.
Table 9.1: Capabilities and Restrictions of Stope Geometry Optimisation Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Model type</th>
<th>Mining method</th>
<th>Dimension</th>
<th>Mathematical formulation</th>
<th>Partial blocks</th>
<th>True optimality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Programming</td>
<td>Fixed blocks</td>
<td>Block-caving</td>
<td>2D</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Geostatistical Approach</td>
<td>Cross sections</td>
<td>Block-caving, Cut-and-fill</td>
<td>2D</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Octree Division</td>
<td>NA*</td>
<td>All</td>
<td>3D</td>
<td>No</td>
<td>NA*</td>
<td>No</td>
</tr>
<tr>
<td>Floating Stope</td>
<td>Fixed blocks</td>
<td>All</td>
<td>3D</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Branch and Bound</td>
<td>(Ir)regular blocks</td>
<td>All</td>
<td>1D</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MVN</td>
<td>Fixed blocks</td>
<td>All</td>
<td>3D</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

NA*: Not Applicable

The problem of true optimality has not yet been solved in stope limit optimisation, and none of the existing algorithms can guarantee the true optimum solution in three dimensions. The MVN algorithm fails to recognise the true optimal stope layout, as it uses a heuristic approach and lacks rigorous mathematical proof. The MVN algorithm guarantees the optimum (maximum) value neighbourhood for each block, however, the problem is how to merge these optimum neighbourhoods to produce the optimum layout. In this regard, the MVN algorithm searches for the best neighbourhood of blocks in one special order (that is, from the first block in the order of rows, columns and sections). After completion of the search, it may be observed that some valuable blocks are excluded from-, or some waste blocks are included in, the final stope. In other words, the order of the search in the heuristic approach affects the results.

The Stope Limit Optimiser (SLO) is a Windows based user interface application program, developed during the thesis, which may be used to interactively optimise stope boundaries. It is a computer implementation of the MVN algorithm. The major options available in the SLO system include:

- XYZ and IJK modes,
- defining sub-regions,
- assay value and economic value modes and
- accepting multi-product deposits (up to four by-products).
By using the SLO optimiser, it is easy to perform a parameterisation and produce nested stopes. By changing any economic parameter, such as: costs; recoveries; cut-off grades; and most importantly prices, various optimal stope layouts may be defined. These are useful in any decision making about the ore deposit, including the feasibility study, preliminary mine evaluation and the mine closure. The variation in stope geometry constraints may produce nested stopes, which may be used in mining method selection. Different mining methods impose different stope geometry constraints and consequently different orders of neighbourhood, therefore, executing the SLO program for alternative mining methods will assist in the selection of the method with the highest stope net value.

9.3 RECOMMENDATION AND FUTURE WORKS

The following areas have been identified for future research:

1. Rigorous mathematical programming techniques must be used to provide a 3D optimum solution to the stope geometry design.

2. Application of genetic algorithms to define the optimum stope boundaries may be useful, since its application to production scheduling in underground mines has already been reported (Denby and Schofield, 1995).

The MVN algorithm and the SLO optimiser, developed during the current thesis, require the following enhancements:

3. The presented algorithm should perform a second pass when searching the “best” blocks as the algorithm is sensitive to the order of blocks during the search. This second search could possibly be undertaken in the reverse direction, to check if included waste blocks may be removed from the stope and if the excluded valuable blocks may be included in the stope.

4. The possibility of the inclusion of partial blocks in the optimisation algorithm, without increasing the number of blocks in the model, should be studied.
5. The time value of money should be included, in the economic parameters, for determining the present worth of blocks during the optimisation.

6. Other practical constraints, such as the maximum sizes of the stope geometry, may be considered in the mathematical formulation of the *MVN* algorithm.
APPENDIX A

COMPUTATIONS OF BLOCK ECONOMIC VALUES

A.1 Block Valuation

Various formulae have been suggested to calculate the economic value of a block (Camus, 1992: Whittle, 1993). In this thesis, the economic value of a block \( BEV \) is equal to the revenue earned from selling the recovered metal (product) content of the block less all costs encountered for mining that block, processing the metal (product) from the ore and refining it to be prepared for sale. The basic relation can be expressed as:

\[
BEV = \text{Revenue} - \text{Costs}
\]

The revenue of a block is directly related to the metal content of the block and the market price of the product as described in the following relations:

\[
\text{Block revenue} = \text{Price} \times \text{Product} \\
= \text{Price} \times \text{Recovery} \times \text{Metal} \\
= \text{Price} \times \text{Recovery} \times \text{Grade} \times \text{Ore} \\
= \text{Price} \times \text{Recovery} \times \text{Grade} \times \text{Volume} \times \text{Density}
\]

and simply expressed by Equations (A.1).
Appendix A: Computation of Block Economic Values

\[ \text{Block Revenue} = P r g V \rho \]  \hspace{1cm} (A.1)

where

- \( P \): the price of the product (metal) to be sold, in \$/t of the metal,
- \( r \): total proportion of the metal recovered from the ore, including mining, processing and refining recovery,
- \( g \): grade of the metal estimated for the block, in "\%" or "ppm",
- \( V \): the volume of the block, \( B_{ijk} \), in m\(^3\) and
- \( \rho \): the density of blocks, in t/m\(^3\).

Costs, on the other hand, can be divided into two categories, that is, "ore-based" costs and "metal-based" costs.

\[ \text{Costs} = \text{Ore-based costs} + \text{Metal-based costs} \]

The first category contains those costs, which relate to the mining of a block from the (surface or underground) deposit, and delivering it to either the processing plant (ore block), or to the dump site, (waste block). "Ore-based" costs are calculated for each tonne of rock (ore or waste) contained in the block as described by:

\[ \text{Ore-based costs} = \text{Unit production cost} \times \text{Tonnage (Rock)} \]
\[ = \text{Unit production cost} \times \text{Volume} \times \text{Density} \]

and expressed by Equation A.2:

\[ \text{Ore-based costs} = C_{\text{ore}} V \rho \]  \hspace{1cm} (A.2)

where \( C_{\text{ore}} \) is the cost of mining a tonne of (ore or waste), in \$/t of rock.

The second category refers to those costs, which are necessary to extract the metal content of the ore through concentrating, processing, refining and preparing the product for sale. "Metal-based" costs are calculated for each tonne of the metal contained in the
and expressed by Equation (A.3):

\[
\text{Metal-based costs} = C_M \cdot r \cdot g \cdot V \cdot \rho
\]  

(A.3)

where \( C_M \) represents those costs required for processing a tonne of metal, refining it and preparing it for sale, in $/t of the metal. Substituting the revenue and costs in the basic equation for calculating the block value, the equation can be reduced to Equation A.4:

\[
\text{BEV} = P \cdot r \cdot g \cdot V \cdot \rho - (C_{\text{ore}} \cdot V \cdot \rho + C_M \cdot r \cdot g \cdot V \cdot \rho) = (P - C_M) \cdot r \cdot g \cdot V \cdot \rho - C_{\text{ore}} \cdot V \cdot \rho
\]

or simply:

\[
\text{BEV} = V \cdot \rho \cdot [(P - C_M) \cdot r \cdot g - C_{\text{ore}}]
\]  

(A.4)

In general, considering different densities for ore and waste blocks, the formula for calculating the net value of a typical block, \( B_{ijk} \), can be obtained through Equation A.5.

\[
\text{BEV}_{ijk} = \begin{cases} 
V \cdot \rho_o \cdot [(P - C_M) \cdot r \cdot g_{ijk} - C_{\text{ore}}] & \text{if } g_{ijk} \geq g_c \\
V \cdot \rho_w \cdot [(P - C_M) \cdot r \cdot g_{ijk} - C_{\text{ore}}] & \text{if } g_{ijk} < g_c
\end{cases}
\]

(A.5)

where

- \( \text{BEV}_{ijk} \): the economic value of the block, \( B_{ijk} \), in $,
- \rho_o \): the density of ore blocks, in t/m³,
- \rho_w \): the density of waste blocks, in t/m³,
- \( g_{ijk} \): the grade of the metal estimated for the block, \( B_{ijk} \), in "%" or "ppm" and
- \( g_c \): the cut-off grade.
Among the above parameters and for blocks of the same cost estimation category, only the grade value is variable from block to block, while the other parameters are constant. Therefore, Equation A.5 can be modified to a linear function \( y = ax + b \), in which the block economic value is a function of the block grade (provided that the unit costs are constant), as expressed by Equation A.6.

\[
BEV_{ijk} = BRR \ g_{ijk} - BMC \\
given: \\
BRR = (P - CM) r V \rho \\
BMC = C_{ore} V \rho
\]

where

- \( BRR \): the "block revenue ratio", as the multiplier in the formula and
- \( BMC \): the "block mining cost", as the constant of the formula.

When a block is barren, that is, the grade is zero, there is a cost required for mining the block. This is called the "block mining cost" \((BMC)\) and is the same for all blocks. Therefore, the value of barren blocks would be negative, is equal to this base cost and is the minimum block value. The metal content of mineable blocks will pay for all, or part of the base cost \((BMC)\) which is linearly related to the grade value of the block. However, the grade value compensates for the cost with a ratio (its multiplier, \([P-C_M] r V \rho\), in Equation A.6), which is called the "block revenue ratio" \((BRR)\). At a certain grade value, the block revenue can pay for the total block mining cost, in which the block net value is zero. For blocks with higher-grade values, the block economic value would be positive. Figure A.1 shows linear variation of block values \((BEV)\) as a function of the grade, \(g\), of blocks.
A.2 Equivalent Grade

In many cases there is more than one (metal) product in the deposit. The block value formula uses the grade value of only one metal. Therefore, it is required to determine an equivalent grade that substitutes grade values of all products and is used in the block valuation formula. Consider that there is one main product and "n" by-products in the deposit, of which the grade, recovery and price are known. The gross value obtained from the metal content may be calculated using the equation:

\[ \text{Gross value} = \text{Grade} \times \text{Recovery} \times \text{Price} \]

and is expressed for each product within the deposit by Equation A.7.

\[ GV_i = g_i r_i P_i \quad \text{for} \quad i = 0, 1, ..., n \]  \hspace{1cm} (A.7)

where

- \( GV_i \): the gross value of the \( i^{th} \) product,
- \( g_i \): the grade of the \( i^{th} \) product,
- \( r_i \): total recovery of the \( i^{th} \) product,
- \( P_i \): the unit price of the \( i^{th} \) product and
- \( n \): the total number of by-products (for the main product, \( n = 0 \)).
Considering one of the products as the base, a factor may be defined for each of the other products in order to obtain the base product equivalent grade. In practice, the main product is usually set as the base, and the grade of each by-product is converted to its "main product equivalent grade" (MPEG). The equivalence factor (EF) for each by-product is defined as the ratio of its gross value to the gross value of the main product.

\[
EF = \frac{\text{Gross value of the by-product}}{\text{Gross value of the main product}}
\]

As a result, the equivalence factor for the main-product would be equal to 1. Equation A.8 denotes the EF formula.

\[
EF_i = \frac{GV_i}{GV_0} = \frac{g_i r_i P_i}{g_0 r_0 P_0}, \quad i = 0,1,\ldots,n
\]  (A.8)

The equivalence factor for a by-product is the factor that has to be multiplied by the grade of the main-product to produce the MPEG of that by-product. The main product equivalent grade is obtained using Equation A.9.

\[
MPEG_i = EF_i(g_0) = \frac{g_i r_i P_i}{g_0 r_0 P_0} (g_0), \quad i = 0,1,\ldots,n
\]  (A.9)

Finally, the total equivalent grade of the main-product is obtained through the summation of the MPEGs of all products as shown below.

\[
MPEG_{Total} = \sum_{i=0}^{n} MPEG_i = \sum_{i=0}^{n} EF_i(g_0)
\]

\[
= EF_0(g_0) + \sum_{i=1}^{n} EF_i(g_0)
\]

Recalling that the equivalence factor of the main (base) product equals to 1 (\(EF_0 = 1\)),
Equation A.9 reduces to Equation A.10.

\[ MPEG_{Total} = \left( g_0 \right) \left( 1 + \sum_{i=1}^{n} EF_i \right) \]  
(A.10)

As a simple example, consider a deposit containing a main-product and two by-products. Knowing the grade, recovery and price of each (metal) product, then the gross values, equivalence factors and MPEGs are calculated based on Equation A.10 and shown in Table A.1.

**Table A.1: Equivalent Grades Calculated for a Deposit with Two By-Products**

<table>
<thead>
<tr>
<th></th>
<th>Main-product</th>
<th>by-product_1</th>
<th>by-product_2</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grade (%)</td>
<td>20</td>
<td>5</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Total recovery (%)</td>
<td>90</td>
<td>80</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>Price($/t)</td>
<td>100</td>
<td>150</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>Gross Value ($)</td>
<td>18</td>
<td>6</td>
<td>3.84</td>
<td>27.84</td>
</tr>
<tr>
<td>EF</td>
<td>1</td>
<td>0.333</td>
<td>0.213</td>
<td>1.547</td>
</tr>
<tr>
<td>MPEG (%)</td>
<td>20</td>
<td>6.667</td>
<td>4.267</td>
<td>30.933</td>
</tr>
</tbody>
</table>

In order to check the results, the grade of the main-product must be substituted with the total MPEG to obtain the equivalent gross value. The result should match the total gross value obtained earlier.

\[ Equivalent \text{ gross value} = Total \text{ MPEG} \times r_0 \times P_0 \]

\[ = 0.309 \times 0.9 \times 100 = 27.84 \]

Products of the deposit may have different units for their prices or grades. Grades and prices are usually expressed in various units, which require additional factors to produce equivalent price and grade units. Two major units for grades include "percent" (%) and "gram per tonne" (ppm). Three major units for price values are "dollar per tonne" ($/t), "cents per kilo" (c/kilo) and "dollar per ounce" ($/oz). Tables A.2 and A.3
show the grade factors and price factors used in the MPEG formulae, respectively.

**Table A.2:** Grade Factors Applied for Corrections in MPEG Formulae

<table>
<thead>
<tr>
<th>#</th>
<th>Grade unit</th>
<th>Grade factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>percent (%)</td>
<td>0.01</td>
</tr>
<tr>
<td>2</td>
<td>gram per tonne</td>
<td>0.000001</td>
</tr>
</tbody>
</table>

**Table A.3:** Price Factors Applied for Corrections in MPEG Formulae

<table>
<thead>
<tr>
<th>#</th>
<th>Price unit</th>
<th>Price factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Dollar per tonne ($/tonne)</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Cents per kilo (c/kilo)</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>Dollar per ounce ($/oz)</td>
<td>35,242</td>
</tr>
</tbody>
</table>

Applying the above factors, the relation for gross value is modified to:

\[
\text{Gross value} = (\text{Grade} \times \text{Grade factor}) \times \text{Recovery} \times (\text{Price} \times \text{Price factor})
\]

Which is expressed by Equation A.11.

\[
GV_i = (g_i \times GF_i) \times (P_i \times PF_i) \quad i = 0, 1, ..., n
\]  

(A.11)

where

\(GF_i\): the grade unit factor for the \(i^{th}\) product and

\(PF_i\): the price unit factor for the \(i^{th}\) product.
APPENDIX B

SLO DIALOGS AND SUBPROGRAMS

This appendix contains the list and description of the Winteracter dialog boxes and the Fortran 90 user defined functions and subroutines, which were used to develop the Stope Limit Optimiser (SLO) program.
### Table B.1: Dialogs Used in the SLO Program

<table>
<thead>
<tr>
<th>#</th>
<th>Dialog Identifier</th>
<th>Dialog Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>General Dialogs</strong></td>
</tr>
<tr>
<td>1</td>
<td>IDD_WELCOME</td>
<td>The dialog contains a picture as the logo of the SLO program together with information about the SLO.</td>
</tr>
<tr>
<td>2</td>
<td>IDD_NewProj</td>
<td>Allows definition of a new project in the SLO by providing a name for the project.</td>
</tr>
<tr>
<td>3</td>
<td>IDD_Projects</td>
<td>Allows specification of a directory and opens a project from the specified directory or saves a project in that directory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Block Model Dialogs</strong></td>
</tr>
<tr>
<td>4</td>
<td>IDD_BlockModel</td>
<td>A parent dialog with four tab controls, which hosts sub-dialogs used to define a block model.</td>
</tr>
<tr>
<td>5</td>
<td>IDD_SelectMode</td>
<td>A tabbed-dialog, which allows selection of either XYZ mode or IJK mode, to define the block model.</td>
</tr>
<tr>
<td>6</td>
<td>IDD_XYZMode</td>
<td>A tabbed-dialog, which allows definition of a block model in XYZ mode (using co-ordinates of the block model).</td>
</tr>
<tr>
<td>7</td>
<td>IDD_IJKMode</td>
<td>A tabbed-dialog, which allows definition of a block model in IJK mode (using block numbers of the block model).</td>
</tr>
<tr>
<td>8</td>
<td>IDD_AskRegion</td>
<td>A tabbed-dialog, which allows the user to define whether or not the block model contains sub-regions and assay or economic data.</td>
</tr>
<tr>
<td>9</td>
<td>IDD_DefineRegions</td>
<td>Provides definition of the total number of sub-regions in the block model and the selection of a sub-region for editing or deleting.</td>
</tr>
<tr>
<td>10</td>
<td>IDD_EditRegionsXYZ</td>
<td>Provides for the definition and edition of a sub-region in the XYZ mode (using co-ordinates of the sub-region).</td>
</tr>
<tr>
<td>11</td>
<td>IDD_EditRegionsIJK</td>
<td>Provides for the definition and edition of a sub-region in the IJK mode (using the block number of the sub-region).</td>
</tr>
</tbody>
</table>
## Appendix B: SLO dialogs and Subprograms

### Constraint Dialogs

<table>
<thead>
<tr>
<th>No.</th>
<th>Dialog Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>IDD_Constraints</td>
<td>Allows for the definition of the stope geometry constraints of the whole block model, or the selection of a sub-region for defining its constraints.</td>
</tr>
<tr>
<td>13</td>
<td>IDD_EditConstraints</td>
<td>Allows for the definition of the stope geometry constraints of the selected sub-region.</td>
</tr>
</tbody>
</table>

### Economic Factor Dialogs

<table>
<thead>
<tr>
<th>No.</th>
<th>Dialog Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>IDD_Economic</td>
<td>A parent dialog with six tab controls, which hosts sub-dialogs used to define the economic factors of the project.</td>
</tr>
<tr>
<td>15</td>
<td>IDD_Product</td>
<td>A tabbed-dialog, which allows the specification of the name of the main product, as well as up to four by-products.</td>
</tr>
<tr>
<td>16</td>
<td>IDD_Price</td>
<td>A tabbed-dialog, which allows for the definition of the price and the price unit, of the main product, as well as up to four by-products.</td>
</tr>
<tr>
<td>17</td>
<td>IDD_Cost</td>
<td>A tabbed-dialog which allows definition of all costs required to extract the ore and process and refine the products.</td>
</tr>
<tr>
<td>18</td>
<td>IDD_Recovery</td>
<td>A tabbed-dialog which allows for the: definition of the rate of recovery in three various stages; the total rate of recovery for the main product; as well as selection of a by-product for defining its rates of recovery.</td>
</tr>
<tr>
<td>19</td>
<td>IDD_EditRecovery</td>
<td>Allows for the definition/edition of the rate of recovery in three various stages, or the total rate of recovery for the selected by-product.</td>
</tr>
<tr>
<td>20</td>
<td>IDD_Grade</td>
<td>A tabbed-dialog which allows for the definition of the unit of the grade values of the main product and by-products, provided in the block data file.</td>
</tr>
<tr>
<td>21</td>
<td>IDD_Density</td>
<td>A tabbed-dialog which allows for the definition of the density of both the ore and waste materials, as well as the cut-off grade of the main product.</td>
</tr>
</tbody>
</table>

### Other Dialogs

<table>
<thead>
<tr>
<th>No.</th>
<th>Dialog Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>IDD_DataFile</td>
<td>Allows for the specification of the name of the block model data file and its data format as well as viewing the data file (employing Notepad).</td>
</tr>
<tr>
<td></td>
<td>IDD_SelectSubs</td>
<td>Allows for the selection of the whole block model, or a sub-region, for performing the MVN optimisation algorithm.</td>
</tr>
<tr>
<td>---</td>
<td>----------------</td>
<td>------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>24</td>
<td>IDD_Summary</td>
<td>Provides a summary report on the status of the selected region before and after applying the algorithm.</td>
</tr>
<tr>
<td>25</td>
<td>IDD_Plot</td>
<td>Provides a number of choices for plotting the optimised stope, including a range of plans/sections.</td>
</tr>
</tbody>
</table>
### Table B.2: List of Subroutines Defined in the SLO Source Code

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UnloadDeallocateSLO</td>
<td>Unloads the loaded dialogs and deallocates allocatable variables</td>
</tr>
<tr>
<td>2</td>
<td>InitialiseSLO</td>
<td>Initialises variables, allocates an initial value for allocatable variables and loads required dialogs</td>
</tr>
<tr>
<td>3</td>
<td>MenuSelectOptions</td>
<td>Branches the program control to perform the subroutine, required for the selected menu item</td>
</tr>
<tr>
<td>4</td>
<td>PushButtonOptions</td>
<td>Branches the program control to perform the subroutine, required for the pressed push button</td>
</tr>
<tr>
<td>5</td>
<td>FieldChangedOptions</td>
<td>Branches the program control to perform the subroutine, required for moving from or to dialog fields</td>
</tr>
<tr>
<td>6</td>
<td>TabChangedOptions</td>
<td>Branches the program control to perform the subroutine, required for moving to the new tab/sub-dialog</td>
</tr>
<tr>
<td>7</td>
<td>ReloadDialogs</td>
<td>Reloads specified dialogs already unloaded</td>
</tr>
<tr>
<td>8</td>
<td>DisplayModelParams</td>
<td>Displays the value of block model parameters on corresponding fields of the block model dialogs</td>
</tr>
<tr>
<td>9</td>
<td>ReAllocateRegions</td>
<td>Deallocates the variables specifying sub-regions and allocates a new space for them</td>
</tr>
<tr>
<td>10</td>
<td>UpdateIJK</td>
<td>Receives the co-ordinates of the block model and calculates the IJK address of the model</td>
</tr>
<tr>
<td>11</td>
<td>UpdateXYZ</td>
<td>Receives the IJK address of the block model and calculates the co-ordinates of the model</td>
</tr>
<tr>
<td>12</td>
<td>ResetXYZIJKData</td>
<td>Sets all variables concerning the co-ordinates and the IJK address of the block model to their initial values</td>
</tr>
<tr>
<td>13</td>
<td>XYZtoIJK</td>
<td>Receives the co-ordinates of a block and calculates its IJK address</td>
</tr>
<tr>
<td>No.</td>
<td>Subprogram</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>14</td>
<td>IJKtoXYZ</td>
<td>Receives the IJK address of a block and calculates its coordinates</td>
</tr>
<tr>
<td>15</td>
<td>SubsYes</td>
<td>Makes all changes in the status of fields, menu items and variables to allow the definition of sub-regions</td>
</tr>
<tr>
<td>16</td>
<td>SubsNo</td>
<td>Makes all changes in the status of fields, menu items and variables to prevent the definition of sub-regions</td>
</tr>
<tr>
<td>17</td>
<td>AddSubregions</td>
<td>Provides for the definition of a new sub-region within the block model</td>
</tr>
<tr>
<td>18</td>
<td>EditSubregions</td>
<td>Provides for the editing of parameters already defined for a sub-region</td>
</tr>
<tr>
<td>19</td>
<td>DeleteSubregions</td>
<td>Deletes a sub-region from the list of sub-regions within a block model</td>
</tr>
<tr>
<td>20</td>
<td>DisplaySubParams</td>
<td>Displays the values of the parameters defining a sub-region on the corresponding dialog</td>
</tr>
<tr>
<td>21</td>
<td>UpdateXYZSubs</td>
<td>Receives the co-ordinates of a sub-region and calculates its IJK address</td>
</tr>
<tr>
<td>22</td>
<td>UpdateIJKSubs</td>
<td>Receives the IJK address of a sub-region and calculates its co-ordinates</td>
</tr>
<tr>
<td>23</td>
<td>ResetXYZIJKSubs</td>
<td>Sets all the variables, concerning the co-ordinates and the IJK address of a sub-region, to their initial values</td>
</tr>
<tr>
<td>24</td>
<td>EditSubregionsOK</td>
<td>Saves the changes made to the parameters of the sub-region</td>
</tr>
<tr>
<td>25</td>
<td>EditSubregionsCancel</td>
<td>Closes the sub-region dialog without saving the changes made to its parameters</td>
</tr>
<tr>
<td>26</td>
<td>DisplayConstraints</td>
<td>Displays the values of the variables specifying the stope geometry constraints on the corresponding dialog</td>
</tr>
<tr>
<td>27</td>
<td>UpdateONB</td>
<td>Receives the minimum stope sizes and calculates the orders of neighbourhood</td>
</tr>
<tr>
<td>28</td>
<td>ResetConst</td>
<td>Sets all the variables, concerning the minimum stope sizes and the orders of neighbourhood, to their initial values</td>
</tr>
<tr>
<td>29</td>
<td>DisplayEcoFactors</td>
<td>Displays the values of variables, specifying the economic factors on the corresponding sub-dialogs</td>
</tr>
<tr>
<td></td>
<td>ByProductsNo</td>
<td>Makes all changes, in the status of fields, menu items, variables and so on, to prevent the definition of the sub-regions</td>
</tr>
<tr>
<td>---</td>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>31</td>
<td>ByProductsYes</td>
<td>Makes all changes, in the status of fields, menu items, variables and so on, to allow for the definition of the sub-regions</td>
</tr>
<tr>
<td>32</td>
<td>CheckTotal</td>
<td>Receives the value of the specified checkbox and allows definition of the total recovery for the specified product</td>
</tr>
<tr>
<td>33</td>
<td>UncheckTotal</td>
<td>Receives the value of the specified checkbox and allows for the definition of the rates of recovery in various stages</td>
</tr>
<tr>
<td>34</td>
<td>NumberByProducts</td>
<td>Receives the number of by-products in the deposit and allows for the definition of their names</td>
</tr>
<tr>
<td>35</td>
<td>ReAllocateEconomies</td>
<td>Deallocates the variables related to the products of the deposit and allocates new space for them</td>
</tr>
<tr>
<td>36</td>
<td>AcceptProducts</td>
<td>Saves the names of by-products and makes any changes required to define their properties</td>
</tr>
<tr>
<td>37</td>
<td>AcceptPrices</td>
<td>Saves the prices and price units of all products of the deposit</td>
</tr>
<tr>
<td>38</td>
<td>UpdateMetalCost</td>
<td>Receives the components of the metal-based costs and calculates its total</td>
</tr>
<tr>
<td>39</td>
<td>AcceptCosts</td>
<td>Saves all costs including the ore-based and metal-based costs</td>
</tr>
<tr>
<td>40</td>
<td>UpdateRecovery</td>
<td>Receives rates of recoveries of the product in various stages, or the total rate and calculates its proportion</td>
</tr>
<tr>
<td>41</td>
<td>EditByProductRecovery</td>
<td>Allows the editing of the variables specifying various rates of recovery, or the total recovery, of a by-product</td>
</tr>
<tr>
<td>42</td>
<td>AcceptByProductRecovery</td>
<td>Saves the rates of recovery, and their proportion, for a by-product of the deposit</td>
</tr>
<tr>
<td>43</td>
<td>CloseByProductRecovery</td>
<td>Closes the dialog specified for editing the rates of recovery of a by-product without saving changes</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---</td>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>44</td>
<td>AcceptGrades</td>
<td>Saves the grade units of all products of the deposit</td>
</tr>
<tr>
<td>45</td>
<td>UpdateCutOff</td>
<td>Receives the cut-off grade and converts it to a proportion</td>
</tr>
<tr>
<td>46</td>
<td>AcceptDensity</td>
<td>Saves the different densities for the ore and waste</td>
</tr>
<tr>
<td>47</td>
<td>FileView</td>
<td>Interfaces with Notepad to open and view some SLO input/output files</td>
</tr>
<tr>
<td>48</td>
<td>AssayAstoBi</td>
<td>Reads the grade values in the ASCII formatted (<em>.dat) file and writes them in a binary formatted (</em>.asy) file</td>
</tr>
<tr>
<td>49</td>
<td>AssayToDollar</td>
<td>Reads the grade values from the (<em>.asy) file and writes their corresponding economic values in the (</em>.bev) file</td>
</tr>
<tr>
<td>50</td>
<td>GetGradeFactor</td>
<td>Receives the grade unit of a product and returns the corresponding factor to be used in the economic calculation</td>
</tr>
<tr>
<td>51</td>
<td>GetPriceFactor</td>
<td>Receives the price unit of a product and returns the corresponding factor to be used in the economic calculation</td>
</tr>
<tr>
<td>52</td>
<td>EquivalentGrade</td>
<td>Calculates the equivalent grade in the multi-product deposits to be used in the economic calculation</td>
</tr>
<tr>
<td>53</td>
<td></td>
<td>See functions table</td>
</tr>
<tr>
<td>54</td>
<td></td>
<td>See functions table</td>
</tr>
<tr>
<td>55</td>
<td>IDInverse</td>
<td>Calculates the i, j and k addresses of a block, based on its sequential identification number</td>
</tr>
<tr>
<td>56</td>
<td>BEVsAstoBi</td>
<td>Converts the format of the block economic data from ASCII (<em>.dat file) into binary (</em>.bev file)</td>
</tr>
<tr>
<td>57</td>
<td>SelectSubs</td>
<td>Defines the required information, such as, the lower and upper bounds of optimisation, regarding the selected region</td>
</tr>
<tr>
<td>58</td>
<td>ImportDollarData</td>
<td>Reads the block economic values of the selected region and assigns them into corresponding variables</td>
</tr>
<tr>
<td>No.</td>
<td>Subprogram</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>59</td>
<td>CheckInputData</td>
<td>Writes the input data into a file to check if conversions have been done properly</td>
</tr>
<tr>
<td>60</td>
<td>Optimise</td>
<td>The main sub-routine, which controls all transactions required to perform the MVN algorithm</td>
</tr>
<tr>
<td>61</td>
<td></td>
<td>See functions table</td>
</tr>
<tr>
<td>62</td>
<td>SelectBlock</td>
<td>Determines those blocks of the maximum value neighbourhood, which are new to the stope</td>
</tr>
<tr>
<td>63</td>
<td>MarginValue</td>
<td>Calculates the total value of new blocks to the stope</td>
</tr>
<tr>
<td>64</td>
<td>NBStartPosition</td>
<td>Determines the location of the first element of a neighbourhood</td>
</tr>
<tr>
<td>65</td>
<td>NBEndPosition</td>
<td>Determines the location of the last element of a neighbourhood</td>
</tr>
<tr>
<td>66</td>
<td></td>
<td>See functions table</td>
</tr>
<tr>
<td>67</td>
<td></td>
<td>See functions table</td>
</tr>
<tr>
<td>68</td>
<td>ExportFlag</td>
<td>Writes the flag value of the blocks to an output file (*,out)</td>
</tr>
<tr>
<td>69</td>
<td>ReportSummary</td>
<td>Provides an on screen summary report concerning the results of optimisation</td>
</tr>
<tr>
<td>70</td>
<td>EnableKPlans</td>
<td>Enables the options used to select plans of the optimised stope for plotting</td>
</tr>
<tr>
<td>71</td>
<td>EnableAllSomeK</td>
<td>Enables/disables the options to select all/some plans of the optimised stope for plotting</td>
</tr>
<tr>
<td>72</td>
<td>EnableJSections</td>
<td>Enables the options used to select cross sections of the optimised stope for plotting</td>
</tr>
<tr>
<td>73</td>
<td>EnableAllSomeJ</td>
<td>Enables/disables the options to select all/some cross sections of the optimised stope for plotting</td>
</tr>
<tr>
<td>74</td>
<td>EnableISections</td>
<td>Enables the options used to select longitudinal sections of the optimised stope for plotting</td>
</tr>
<tr>
<td>75</td>
<td>EnableAllSomeI</td>
<td>Enables/disables the options to select all/some longitudinal sections of the optimised stope for plotting</td>
</tr>
<tr>
<td></td>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>76</td>
<td>PlotRange</td>
<td>Determines the range of plans/sections for plotting</td>
</tr>
<tr>
<td>77</td>
<td>DoXYPlot</td>
<td>Controls plotting the plans for the specified range</td>
</tr>
<tr>
<td>78</td>
<td>DoXZPlot</td>
<td>Controls plotting the cross sections for the specified range</td>
</tr>
<tr>
<td>79</td>
<td>DoYZPlot</td>
<td>Controls plotting the longitudinal sections for the specified range</td>
</tr>
<tr>
<td>80</td>
<td>KPlan</td>
<td>Controls the plotting of an individual plan</td>
</tr>
<tr>
<td>81</td>
<td>JSection</td>
<td>Controls the plotting of an individual cross section</td>
</tr>
<tr>
<td>82</td>
<td>ISection</td>
<td>Controls the plotting of an individual longitudinal section</td>
</tr>
<tr>
<td>83</td>
<td>TopBorder</td>
<td>Plots the top annotation of any plan/section</td>
</tr>
<tr>
<td>84</td>
<td>BottomBorder</td>
<td>Plots the bottom annotation of any plan/section</td>
</tr>
<tr>
<td>85</td>
<td>PlanBody</td>
<td>Plots the main body of a plan</td>
</tr>
<tr>
<td>86</td>
<td>JsecBody</td>
<td>Plots the main body of a cross section</td>
</tr>
<tr>
<td>87</td>
<td>ISecBody</td>
<td>Plots the main body of a longitudinal section</td>
</tr>
<tr>
<td>88</td>
<td>StartNewProject</td>
<td>Provides the new title and the new input/output files based on the name specified by the user, to define the new project</td>
</tr>
<tr>
<td>89</td>
<td>OpenFiles</td>
<td>Opens the input/output files</td>
</tr>
<tr>
<td></td>
<td><strong>ListProjects</strong></td>
<td>Provides the list of existing projects in the specified directory</td>
</tr>
<tr>
<td>---</td>
<td>------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td><strong>OpenProjectDialog</strong></td>
<td>Displays the dialog specified for opening the projects in the SLO</td>
</tr>
<tr>
<td></td>
<td><strong>OpenProject</strong></td>
<td>Reads information from project files and assigns them in relevant variables</td>
</tr>
<tr>
<td></td>
<td><strong>CloseProject</strong></td>
<td>Closes all files of a project</td>
</tr>
<tr>
<td></td>
<td><strong>SaveProject</strong></td>
<td>Writes the values of all relevant variables of a project to the corresponding project files</td>
</tr>
<tr>
<td></td>
<td><strong>SaveProjectAs</strong></td>
<td>Allows the definition of a new name for saving the current project</td>
</tr>
<tr>
<td></td>
<td><strong>DoSaveAs</strong></td>
<td>Writes the values of all relevant variables of the current project, to the new specified series of the project files</td>
</tr>
<tr>
<td></td>
<td><strong>MessageOut</strong></td>
<td>Displays various information or errors messages as required</td>
</tr>
</tbody>
</table>
Table B.3: List of Functions Defined in the SLO Source Code

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>53</td>
<td>IDNumber</td>
<td>Integer</td>
<td>Calculates the sequential identification number of a block, based on its i, j and k addresses</td>
</tr>
<tr>
<td>54</td>
<td>BDV</td>
<td>Real</td>
<td>Receives the equivalent grade value of a single block and calculates its dollar value</td>
</tr>
<tr>
<td>61</td>
<td>MaxValueNB</td>
<td>Real</td>
<td>Calculates the maximum neighbourhood value</td>
</tr>
<tr>
<td>66</td>
<td>NBTrue</td>
<td>Logical</td>
<td>Determines whether, or not, a neighbourhood is feasible</td>
</tr>
<tr>
<td>67</td>
<td>NBValue</td>
<td>Real</td>
<td>Calculates the total value of a neighbourhood</td>
</tr>
</tbody>
</table>
APPENDIX C

THE STOPE LIMIT OPTIMISER USER’S GUIDE

C.1 INTRODUCTION

The Stope Limit Optimiser (SLO), is a user interface application program developed to implement the MVN algorithm, both of which were developed in this thesis. The SLO program is based on the Fortran 90 and Winteracter programs. A copy of the application program, together with a number of project files containing four examples, have been provided in the attached diskette. This appendix assists the user in the application of the program.

C.2 INSTALLATION

All required files have been zipped and saved as “SLO.zip”. These may be unzipped the file into a working directory. A total of 17 files will be imported into the specified directory. These include the SLO application file as well as four series of project files and their data files:
Appendix C: The SLO User's Guide

There are three files describing each project example including: the model parameters file (*.mpr), the stope geometry constraints file (*.cst) and the economic factors file (*.eco). A data file (*.dat) is supplied for each project example.

C.3 STARTING SLO

In the specified directory, double click on the application program file (SLO.exe) to open the SLO window. A welcome dialog is displayed in the centre of the SLO window. The dialog contains a symbolic picture of a miner (as the SLO logo) together with a piece of information about the development of the SLO program, as shown in Figure C.1. After a few seconds, the welcome dialog disappears\(^1\) and the main menu will be available. The optimiser is then ready for use. The following steps should be taken when using the SLO program:

1. Specify a project for optimisation (by creating a new project or using an existing one).
2. Edit the parameters of the specified project.
3. Prepare the data for optimisation
4. Perform the optimisation algorithm.
5. Collect the results.

Any violation of the above order may result in message errors or wrong outputs.

\(^1\) To see the logo and information again, use the Help | About SLO option.
Appendix C: The SLO User’s Guide

C.4 PROJECT MANIPULATION

A project in the SLO program, is a collection of the three files (*.mpr), (*.est) and (*.eco). When a project is specified, these three files are involved. The user may specify a new project, open an existing one, save or close a project or save the current project with a new name. Figure C.2 shows the options of the Project menu item.
C.4.1 *Project* | *New option*

This option provides a dialog box (Figure C.3) to specify a name\(^2\) for the project. The required parameters of this project may be defined later, using the *Edit* menu item. By entering the name of the project, *SLO* program confirms the name and creates all input/output files accordingly. The current project (if any) is then closed with all the loaded dialogs being unloaded and all the variables set to their initial values.

---

\(^2\) A maximum of 15 characters is allowed for the project name.
C.4.2  *Project* | *Open* option

This option provides a dialog (Figure C.4) with a list of existing projects and allows the user to select a project to be opened. The name of the directory, in which the project is located, should be specified in the directory field. By default, the path of the current directory is displayed in the field. Click on the *Browse* button to locate the desired directory. The drop down menu in the dialog supplies the list of all *SLO* program projects located in the specified directory. Select a project name from the list and click on the *Open* push button to open the project. If there is any project already opened, it will be closed, loaded dialogs are unloaded, and all variables are initialised before opening the specified project.

C.4.3  *Project* | *Close* option

This option allows the current project to be closed. When a project is closed, all the loaded *SLO* program dialogs are unloaded and all the connected *Fortran 90* input/output files are disconnected. Before closing the current project, however, the user
will be asked to save the current project\(^3\).

![Image: SLO Dialog for Opening an Existing Project]

**Figure C.4:** The *SLO* Dialog for Opening an Existing Project

### C.4.4 *Project | Save* option

This option allows the current project to be saved. When a project is saved, the current values of all the variables describing the model parameters, stope geometry constraints and economic factors are saved in corresponding files.

### C.4.5 *Project | Save as...* option

This option provides a dialog (Figure C.5) for saving the current project with a new name and/or in another directory. The user can enter the path of a new directory, to save the new project in the provided field. By default, the path of the current directory is displayed in this field. Click on the *Browse* button to locate/create the desired directory. A drop down menu is also provided to supply the list of all the *SLO* projects located in

---

\(^3\) The message, request for saving, is displayed regardless of whether or not there are any changes to the project.
that directory. Add the new project name to the list and click on the Save button to save the current project with the given name in the specified location.

![Image of SLO - Save As dialog box](image)

**Figure C.5:** The SLO Dialog for Saving the Current Project with a New Name/Directory

### C.4.6 Project | Exit option

This option allows the user to exit from the application. If any project is opened and exit from the SLO program is requested, the user will be asked to save the current project before exiting. The SLO program window is then closed.

### C.5 INPUT EDITION

This option allows the user to define and modify the three project files as well as the data file. The Edit option from the main menu leads to various sub-options (Figure C.6) which allows editing of all inputs of the optimisation.
C.5.1  Edit | Block Model option

This option allows the definition of the block model parameters, through a tabbed dialog, with four tab controls as described below.

The Select Mode sub-dialog: In the first sub-dialog (Figure C.7), the user can specify the block model format, that is, the XYZ mode or the IJK mode.
Block model type
Select the model type of the block model:
C X/Z mode
(* UK. mode
More info:
Select XYZ mode if you have co-ordinates of your model and block sizes. The SLO will return the No. of blocks in each dimension and the block volume.
Select UK mode if you have the number of blocks in each dimension of your model. The SLO will return the block volume and the co-ordinates of each sub-region. It is assumed that the origin of the model is set at zero.

Figure C.7: Mode Selection for Block Model Input

The XYZ Mode sub-dialog: The second sub-dialog (Figure C.8) provides the definition of the block model in the XYZ mode. Required data entry includes the minimum and maximum co-ordinates of the block model and the length, width and height of the fixed blocks in the specified fields. Click on the Calculate push button to calculate and display: the block volume; the number of blocks in the three orthogonal directions; and the total number of blocks in the block model. Click on the Reset push button to initialise the variables.

The IJK Mode sub-dialog: The third sub-dialog (Figure C.9) provides the definition of the block model in the IJK mode. Data entry includes the length, width and height of the fixed blocks as well as the number of blocks in the three orthogonal directions in the specified fields. Click on the Calculate push button to calculate and display the block volume, the total number of blocks in the block model and the maximum co-ordinates of the block model. The (relative) minimum co-ordinates of the block model are assumed zero. Click on the Reset push button to initialise the variables.
Figure C.8: The SLO Dialog which allows the Definition of the Block Model in the XYZ Mode

Figure C.9: The SLO Dialog which allows the Definition of the Block Model in the IJK Mode
The *Options* sub-dialog: The fourth sub-dialog (Figure C.10) provides the selection of the block model options, that is, the existence of the sub-regions within the block model and the type of the main data. In the *select regions mode* group, specify whether or not the block model contains any sub-regions. In the *data file type* group, specify whether the main data contain assay values or dollar values. Use the mouse or cursor keys to shift between the choices.

![SLO - Block Model Definition](image)

**Figure C.10:** The Dialog Box for Choosing Options in the Block Model

### C.5.2 *Edit | Sub-regions option*

This option provides for the editing and the addition and deletion of the sub-regions within the block model. This menu item is only available if sub-regions have already been defined in the block model (via the *select regions mode* group of the *Options* tabbed dialog in the block model dialogs). By selecting this option, a dialog is displayed as shown in Figure C.11. Enter the total number of sub-regions within the block model in the specified field and click on the *Define* button, as many times as the number is entered for the total number of sub-regions, to define any sub-region parameters. As
the sub-regions are defined, their names are displayed in the drop down menu in the dialog. Select a sub-region from the menu list and click on the *Edit* button to edit the parameters of the selected sub-region. A sub-region may be deleted by selecting its name from the menu list and clicking on the *Delete* button.

![Image of the SLO - Sub Regions Definition dialog]

**Figure C.11:** The *SLO* Dialog for Manipulating Sub-Regions

By clicking on the *Define* button, another dialog (Figure C.12) is displayed which allows the user to define the parameters of the sub-region. Depending on whether the XYZ or IJK mode is selected, two different ways are used to define the sub-region parameters. These parameters are defined in a similar manner as the whole block model, except that the name and ID number of the sub-region are entered in the specified fields. By clicking on the *Edit* button, the same dialog is displayed, which allows the user to modify the name and other parameters of the sub-region.
C.5.3 *Edit* | *Stope Constraints* option

This option provides the definition of the stope geometry constraints for the entire block model or the existing sub-regions (if any) through the dialog, shown in Figure C.13. If there is no sub-region in the block model, the *sub-regions constraints* group in the dialog, is disabled. In this case, enter the minimum allowable sizes of the stope (for the entire block model) in the specified fields and click on the *Calculate* button to calculate and display the orders of neighbourhood in the three principal directions and the total number of blocks in a neighbourhood. Click on the *Reset* button to set the variables to their initial values.

If the user has already defined any sub-regions within the block model, the *Entire Model Constraints* group is disabled and the dialog allows the definition of the constraints for the sub-regions. Select a sub-region from the list provided in the drop down menu and click on the *Edit* button to define or modify the stope geometry.
constraints of the selected sub-region. In the new dialog (Figure C.14), enter the minimum allowable sizes of the stope (for the selected sub-region) in the specified fields and use the *Calculate* and the *Reset* buttons as for the definition of the entire block model.

![Figure C.13](image)

**Figure C.13**: The *SLO* Dialog for Defining the Stope Geometry Constraints (entire model)

![Figure C.14](image)

**Figure C.14**: The *SLO* Dialog for Defining the Stope Geometry Constraints (sub-regions)
C.5.4  Edit | Economic Factors option

This option provides for the definition of all the economic parameters applied in mining the deposit. This menu item is available only if the user has specified that the main data contains assay values (through the select regions mode group of the Options tabbed dialog in the block model dialogs). By selecting this menu option, a tabbed-dialog is displayed with six tab controls, as described below.

The Products sub-dialog: In the first sub-dialog (Figure C.15), the user can define the products contained in the deposit. In the main product group, enter the name of the main product in the provided field. Using the radio buttons, select an appropriate option to specify whether or not there are any by-products in the deposit. If the Yes choice is selected, the by-product group is enabled. Enter the total number of by-products in the supplied field to enable as many fields as needed for specifying product names. Enter the names of the by-products in their fields and click on the Accept button to accept the by-product names.

The Prices sub-dialog: The second sub-dialog (Figure C.16) supplies the fields for the definition of prices and the price units for all products. Each of the provided drop down menus contain a list of three price units, namely, dollar per tonne, cents per kilo and dollar per ounce. Enter the price of each (by) product and select an appropriate price unit from its corresponding menu list. Then click on the Accept push button to accept these inputs.

---

\(^{4}\) SLO allows processing of up to four by-products.
Figure C.15: The Dialog for Defining the Products of the SLO Project

Figure C.16: The Dialog for Defining Prices and Price Units in the SLO Program

The Costs sub-dialog: The third sub-dialog (Figure C.17) supplies the fields for the
definition of various costs encountered to extract the metal product from the deposit in various stages. Enter the average cost of mining a tonne of rock in the specified field of the *ore-based costs* group. In the *metal-based costs* group, enter the cost of processing and refining to obtain a tonne of metal product, as well as the administration and other costs in the specified fields. As the user enters these data, the total value of the metal-based costs is updated. Finally, accept these inputs by clicking on the *Accept* button.

![Image of SLO dialog box](image)

**Figure C.17:** The Dialog for Defining Costs in the SLO Program

The *Recoveries* sub-dialog: This sub-dialog (Figure C.18) supplies the fields, for the definition of various rates of recovery at the different stages, for the main product. It also allows the selection of the by-products for the further definition of their rates of recovery. Enter the rates of recovery of the main product at the mining, processing and refining stages, in terms of percent in the provided fields. As each rate is entered, its proportion and the total rate of recovery are updated and displayed in the provided fields. Also, provided is a check box, named "*Use only total recovery*". Review this
check box, if the total rate of recovery is available.

**Figure C.18:** The SLO Dialog for Defining the Rates of Recovery (main product)

If there is any by-product in the deposit, the *By-products* group is activated and the list of by-products becomes available in the drop down menu. Select a by-product and click on the *Edit* button. The new dialog, shown in Figure C.19, allows for the definition of the rates of recovery for the selected by-product. Enter the stage-rates or the total rate in the specified fields and click on the *Accept* button to update the proportions and save the values. Click on the *Cancel* button to discard the inputs. Click on the *Close* button to return to the previous dialog.

The *Grades* sub-dialog: This sub-dialog (Figure C.20) supplies the fields for definition of the grade units for different products (the grades of products are provided in the data file). Two grade units are available in SLO, that is, percent (%) and gram per tonne (ppm). Select the appropriate grade unit from the drop down menu list for each (by) product. Click on the *Accept* button to save the inputs.
By-products recovery

Mining Recovery:
Processing Recovery:
Refining Recovery:
Total Recovery:

Table

<table>
<thead>
<tr>
<th>By-product</th>
<th>Silver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Percent (%)</td>
<td>Proportion</td>
</tr>
<tr>
<td>90.0</td>
<td>0.900</td>
</tr>
<tr>
<td>95.0</td>
<td>0.850</td>
</tr>
<tr>
<td>90.0</td>
<td>0.900</td>
</tr>
<tr>
<td>Total Recovery:</td>
<td>88.8</td>
</tr>
</tbody>
</table>

Figure C.19: The SLO Dialog for Defining the Rates of Recovery (by-products)

Figure C.20: The SLO Dialog for Defining Grade Units

The Ore properties sub-dialog: This sub-dialog (Figure C.21) allows for the definition of the cut-off grade (of the main product) and the specific gravity of the different rock types. Enter the cut-off grade and the densities of the ore and waste in the specified fields and click on the Accept button to save the inputs.
Figure C.21: The SLO Dialog for Defining Ore Properties

C.5.5 **Edit | Main Data option**

This option provides the definition of the main data file and the data format in the file. It allows the user to view and edit data if necessary. By selecting this menu item, a dialog is displayed, as shown in Figure C.22. Enter the main data file name in the specified field. By default, the project name with the (*.dat) extension is displayed in the filed. Enter the Fortran format of the data file to read the \((i, j, k)\) address of the block, the block assay or the economic value. It may be necessary to view the data file for the correct format. Click on the View Assay Data button to view/edit the data file via Notepad, if the data file contains block assay values. Alternatively, click on the View BEV Data button to view/edit the data file via Notepad, if the data file contains block economic values. Click on the OK button to save the inputs, or click on the Cancel button to discard the inputs.
C.6 PRE OPTIMISATION

Various options available in this item from the main menu help the user to prepare the data for optimisation. Prior to optimisation, the user needs to have: the block data in terms of the block economic values in the binary format; determined the domain of the optimisation; and regulated the optimisation variables for the specified domain. Figure C.23 shows the available options for the Preoptimisation menu item.

C.6.1 Preoptimisation | Data Preparation Option

This option provides for the required preparation of the input data to be usable in the optimisation core and leads to the sub-options for two different data types. These are the assay data and the economic data types.

Preoptimisation | Data Preparation | Block Grade Values: Select this item if the data file contains assay values. In this case, the assay data are, firstly, converted from ASCII format into binary format and, then, the assay values are converted into economic values.
Preoptimisation | Data Preparation | Block Economic Values: Select this item if the data file contains economic values (BEV). In this case, the BEV data are converted from ASCII format into binary format. No further process is required.

C.6.2 Preoptimisation | Select Region Option

The user must determine the lower and upper bounds of the space, on which the optimisation algorithm will be applied. Selecting this option provides a dialog box, as shown in Figure C.24. Use the cursor keys to shift between the two radio button type choices. Select the Entire Block Model Optimisation radio button option if the optimisation is performed on the entire block model. Then Click on the OK push button to set the optimisation variables with respect to the entire model. Alternatively, the Sub-
regions Optimisation radio button option is used if the optimisation is only performed on a sub-region. Select a sub-region from the list of sub-regions available from the drop down menu. Click on the OK push button to set the optimisation variables with respect to the selected sub-region.

![SLO - Sub-regions Optimisation dialog](image)

**Figure C.24:** The SLO Dialog for Defining the Optimisation Domain

### C.6.3 Preoptimisation | Import Block Data Option

This option is used to read the economic values of the specified blocks into a 3D array variable.

### C.7 OPTIMISATION

The Run option from the main menu leads to an item for the performance of the optimisation.
C.7.1 Run | Optimise Option

Select this option to apply the 3D MVN algorithm, for the optimisation of the stope geometry, to the selected region.

C.8 POST OPTIMISATION

The Results option from the main menu leads to various sub-options (Figure C.25), which control the post optimisation activities. Using these sub-options, the user can plot the optimised stope and access the results.

![Figure C.25: Options Available for the Results Menu Item](image)

C.8.1 Results | Export Flag Data Option

This option performs a search through the blocks of the selected (sub)region and exports their flag data into the output file (*.out). The output file is opened, using
Notepad, so as to view the flag data file.

### C.8.2 Results | Summary Report Option

This option searches through the selected (sub) region and produces a summary report of the optimisation results. The summary report is displayed on a dialog, as shown in Figure C.26.

![SLO Summary Report](image)

Figure C.26: Example of the SLO Summary Report

### C.8.3 Results | Intermediate Results Option

The optimisation intermediate results are stored in the (*.res) file as the optimisation progresses over the blocks. Select this option to open the file and view these results.

### C.8.4 Results | Neighbourhood Results Option
Appendix C: The SLO User's Guide

The optimisation results concerning the neighbourhoods of each block are stored in the (*.nbv) file as the optimisation progresses over the blocks. Select this option to open the file and view these neighbourhood results.

C.8.5 Results | Test Results Option

All the computations and logical decisions made while performing the optimisation are stored in the (*.fst) file as the optimisation progresses over the blocks. Select this option to open the test file and view the results.

C.8.6 Results | Plot Plans/Sections Option

Use this option if it is desired to plot the optimised stope in the form of plans or sections. As the user selects this menu item, a dialog is displayed to provide various opportunities for plotting, as shown in Figure C.27. There are three main options available for plotting the stope.
Figure C.27: The SLO Dialog for Plotting the Optimum Stope

Plotting plans: Review the X-Y plans check box to enable the Plot Plans group in the dialog. Then choose the All radio button for plotting all the plans of the stope. Alternatively, choose the Specify radio button for plotting some plans of the optimised stope. In this case, the user should specify the range of the requested plans by entering the minimum and maximum limits of the range in the specified fields.

Plotting longitudinal section: Check the X-Z Sections check box to enable the Plot longitudinal sections group in the dialog. Choose the All radio button for plotting all the longitudinal sections of the stope. Alternatively, choose the Specify radio button for plotting some longitudinal sections of the optimised stope. In this case, the user should specify the range of the requested longitudinal sections by entering the minimum and maximum limits of the range in the specified fields.

Plotting cross sections: Check the Y-Z Sections check box to enable the Plot cross sections group in the dialog. Then choose the All radio button for plotting all the cross
sections of the stope. Alternatively, choose the Specify radio button for plotting some cross sections of the optimised stope. In this case, the user should specify the range of the requested cross sections by entering the minimum and maximum limits of the range in the specified fields.

Click on the OK push button to plot the requested plans and/or sections. Click on the Cancel push button to discard the inputs. The plotted plans/sections will be saved in the (*.plt) files.

C.8.7 Results | View Plots Option

Select this menu item to access the plot file (*.plt), via the Notepad application, and view the plotted plans/sections.

C.9 HELP

This option, from the main menu, provides some information concerning the SLO program and leads to the following sub-option.

C.9.1 Help | About SLO Option

This option displays a dialog containing information about the development of the SLO program together with the SLO logo, as shown in Figure C.1.