T-4586

A MATHEMATICAL ANALYSIS OF THE LERCHS AND GROSSMANN ALGORITHM AND THE NESTED LERCHS AND GROSSMANN ALGORITHM

by

Gary D. Bond

T-4586

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Doctor of Philosophy (Mathematical and Computer Sciences).

Golden, Colorado

Date March 9th, 1995

Signed: Gary D. Bond
Gary D. Bond

Approved: Robert Underwood
Dr. Robert Underwood
Thesis Advisor

Golden, Colorado

Date March 13, 1995

Graeme Fairweather
Dr. Graeme Fairweather
Professor and Head,
Department of Mathematical and
Computer Sciences

T-4586

ABSTRACT

The Lerchs and Grossmann algorithms are used in the surface mining industry to solve the Pit Limit Problem and to generate an approximate solution to the Extraction Sequence Problem. We revisit the theoretical proof of convergence of the algorithm and provide considerable examples to explain the algorithm.

We demonstrate that the Lerchs and Grossmann (LG) Algorithm is a graph theoretic implementation of the dual simplex algorithm which solves the Pit Limit Problem. We also prove that the LG Algorithm converges, in a finite number of iterations, to the smallest maximum valued pit limit, when multiple maximum valued pit limits exist, and through a simple change in definition the algorithm, it can be forced to converge to the largest maximum valued pit limit.

We also show that the Nested Lerchs and Grossmann (NLG) Algorithm uses Lagrangian relaxation on the volume constrained Pit Limit Problem to generate a sequence of nested pit limits whose average values decrease monotonically. Using the fact that the nested pit limits average values decrease monotonically, we prove they maximize the integral of the global cash flow function. These nested pit limits generate an approximate solution to the Extraction Sequence Problem.

Finally, we formulate the Extraction Sequence Problem, which we argue must be a zero-one integer programming problem. Thus, it is classified as an NP-hard problem and ϵ-approximate algorithms to the problem are also NP-hard. Thus, it is impossible to prove that the NLG Algorithm's approximate solution is within some bound of the optimal solution.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I thank all of those individuals who supported me and provided moral support throughout this seemingly never-ending process. Without their kind, reliable, and interesting support this effort might never have been completed.

I am deeply indebted to Dr. Underwood and Dr. Pruess for their close readings and critical reviews, which definitely improved the quality of the result.

I also thank the remaining committee members Dr. Boes, Dr. Bell, and Dr. Dagdelen for their patience during the lengthy oral review and their willingness to review the tedious final paper. I especially appreciate the generous guidance and motivation provided by Fred Seymour.

DEDICATION

I dedicate this work to my father, Luther Ross Bond, and my brother, Richard David Bond. I dedicate it to my father, for stressing the importance of a quality education and the benefits of the quest for knowledge. I dedicate it to my brother, who died in the services of his country, for showing me that anything worth being is worth paying a high price.

Chapter 1
INTRODUCTION

In the surface mining industry, two difficult problems must be solved during the planning phase of a surface mine exploitation. These are the *Pit Limit Problem*, deciding what should be mined, and the *Extraction Sequence Problem*, deciding the order in which to extract what should be mined. Although this is a simplistic view of the mine planning process, it suffices for purposes of this research. For a thorough discussion of the mine planning process, see Dagdelen (1985).

The scale of most surface mine models, which typically consist of tens of thousands of elements, makes manual solution of pit limit problems a slow and tedious task. Thus, computerized models and algorithms have been developed to solve the problem. The most widely used computer model is the regular 3-D fixed *block model*, where the orebody is divided into fixed-size *blocks* (Kim78). The horizontal and vertical dimensions of each block are consistent and usually the vertical dimension corresponds to the height of the bench.

Kim (1978) describes four classes of optimizing algorithms used in solving pit limit problems: dynamic programming algorithms, linear programming algorithms, network flow algorithms, and graph theoretic algorithms. In addition to the optimizing algorithms, a variety of non-optimizing but good heuristic algorithms exist.

Dynamic programming algorithms for solving the pit limit problem have been developed by Lerchs and Grossmann 1965 and Koenigsberg 1982. Lerchs and Grossmann present a dynamic programming algorithm which solves the two dimensional pit limit problem, where the two-dimensional pit limit problem finds the pit limit for

a vertical slice of the block model. Koenigsberg then extended the two dimensional algorithm by adding a sophisticated search.

In Picard 1976, the pit limit problem was demonstrated to be equivalent to finding the maximum flow in a network. Thus, solution of the pit limit problem could use any of the wide range of network flow algorithms currently available. Yegulalp and Arias 1992 compare and contrast the various maximum flow algorithms as applied to the pit limit problem.

The first, and most famous, graph theoretic algorithm was published by Lerchs and Grossmann 1965. Other than the heuristic algorithms, this algorithm has been favored by the surface mining industry. Recently, a new graph theoretic algorithm, published by Zhao and Kim 1990, has received considerable attention. The attention it has received is probably a result of the authors' claim that it performs much better (runs faster) than the Lerchs and Grossman algorithm.

Several algorithms have been proposed for solving the extraction sequence problem. The algorithm developed by Dagdelen 1985, and discussed by Dagdelen and Johnson 1986 and Tachefine et. al. 1994, uses Lagrangian relaxation to reduce the extraction sequence problem to solving a collection of pit limit problems. Although the algorithm has a sound mathematical basis, it has been attacked by some authors, specifically Zhao and Kim 1994, who claim the algorithm cannot solve the problem. Tolwinski and Underwood 1992 and Wang and Sevim 1992 present heuristic search algorithms for solving the extraction sequence problem. Gershon 1982 presents a linear programming model, a mixed integer programming formulation (Gershon 1983), a simple heuristic algorithm (Gershon 1986c), and a blending based algorithm (Gershon 1986a), each of which can be applied to the extraction sequence problem. In addition, Gershon 1986b presents a survey of mine scheduling approaches. Zhang, et.

al. 1986 present an interactive optimization algorithm which combines inventory theory, dynamic programming, and computer simulation. Fytas and Calder 1986 present an interactive simulation model which combines simulation for long range production scheduling and linear programming for short range scheduling.

Another algorithm, described by Lerchs and Grossmann in their seminal paper "Optimum Design of an Open-Pit Mine" (Lerchs and Grossmann 1965), uses the concept of nested pits to solve the extraction sequence problem. This algorithm, called the Nested Lerchs and Grossmann algorithm, generates a sequence of nested pits by reducing the value of all blocks in the pit by a constant amount and solving the resulting pit limit problem.

Although numerous algorithms exist for solving the pit limit problem and the extraction sequence problem, we chose to analyze the algorithms developed by Lerchs and Grossmann. These algorithms are widely used, and we feel, not well understood.

Lerchs and Grossmann paper is terse and difficult to follow. This research is an extension of their algorithms. We revisit the development of the Lerchs and Grossmann algorithm, expanding upon the algorithm and its associated proofs, frequently using examples to clarify concepts. We prove previously unproven assertions made by the authors with respect to the Nested Lerchs and Grossmann algorithm. We also develop extensions to the algorithm to address some of its shortcomings.

1.1 The Block Model

The mining industry usually begins the mine planning process by defining a *block model* representation of the ore deposit in question. The block model partitions the deposit into uniform rectangular blocks, where the block's height and base dimensions are based upon operational considerations. A typical block model will contain on the

order of tens of thousands of individual blocks. Figure 1.1 shows a cross section of a three-dimensional block model, which contains five benches (horizontal collection of blocks) and nine pillars (vertical collection of blocks).

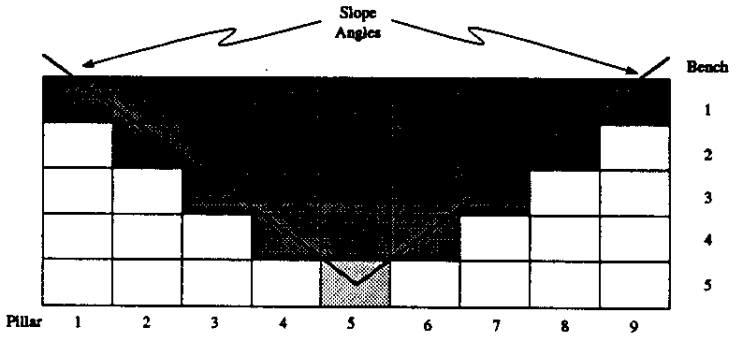


Figure 1.1: Cross Section of Block Model and Accessibility Cone

Both the pit limit problem and extraction sequence problem are optimized with respect to block values. The optimal pit limit solution is the pit limit with the maximum net value of blocks within the pit limit. The optimal extraction sequence solution is the extraction sequence which either maximizes the integral of the cash flow function or maximizes the net discounted cash flow.

Assigning values to blocks is a complicated endeavor which requires estimating the amount of recoverable metal within each given block, estimating the market price of the recoverable metal, and allocating the costs of mining the block. An estimated block value may be obtained by combining these components.

The recoverable metal of a block is based upon estimates of mineral grade and recovery percentages. The mineral grade is estimated using geostatistical techniques, which interpolate between measured observations from drill hole assays. Recovery percentages are based upon historical performance of the recovery process. The product of these two figures provides an estimate of recoverable metal for the block.

Estimating the market price of the recoverable metal can be performed in several manners. Metal prices can be based upon projections of historical data or by simply using judgement. Frequently, pit limit analyses consider a range of metal prices.

Mining costs consists of extraction costs, processing costs, and overhead. Extraction costs include the variable cost of hauling and an allocated percentage of the fixed costs of hauling capital. Processing costs include the variable cost of extracting the metal from the ore and an allocated percentage of the fixed costs of processing capital. Overhead costs are an allocated percentage of other operations.

It is not our intent to describe how block values should be developed or how mine planning should be performed. We have assumed an impartial view of these matters. Our intent is to discuss the mathematical issues relating to the Lerchs and Grossmann algorithms.

The Pit Parameterization algorithm developed, by G. Matheron in 1975 and discussed in detail in (François-Bongarçon and Maréchal 1976; François-Bongarçon and Guibal 1982; Dagdelen and François-Bongarçon 1982; and Coléou 1983), identifies a collection of optimal pit limits independent of the economic assumptions discussed above. The Pit Parameterization algorithm identifies an isovalue contour for each different setting of an economic parameter. By identifying the pits contained within the contours additional factors may be considered in selecting comparable pit limits.

1.2 An Overview of the Pit Limit Problem

Both the pit limit problem and the extraction sequence problem contain one critical constraint, the pit wall slope angle constraint. Pit wall slope angles define cones of accessibility for each block (i.e., accessibility cones). For a given block, the accessibilty cone defines the collection of blocks which must be mined before that

block becomes accessible. For example, Figure 1.1 shows the accessibility for the block in the fifth pillar and on the fifth bench (lightly shaded block). The heavily shaded blocks are blocks contained in the accessibilty cone. All heavily shaded blocks must be removed before the lightly shaded block becomes *accessible* (may be mined).

Lerchs and Grossmann recasts the block model into a directed graph where nodes in the graph correspond to blocks in the block model, and arcs in the graph correspond to direct dependencies between blocks. Figure 1.2 shows the directed graph representation of the block model depicted in Figure 1.1. Each node is denoted with a round symbol. Each arc is denoted with a directed line segment, where the direction of the arrow indicates the direction of dependency. Again, heavy shading is used to denote blocks in the accessibility cone of the lightly shaded block. This example is typical of a 45 degree slope angle, assuming cubic blocks. Node symbol shading, as for the node in the second bench of the first pillar, denotes blocks which are inaccessible. These nodes are considered inaccessible since continuation of the slope angle would require mining blocks which are not contained within the block model.

In order to solve the pit limit problem, an algorithm must identify the collection of blocks which meet pit wall slope angle constraints and have the maximum net value. Figure 1.3 shows a representative optimal pit limit. Each block within the pit limit has had every block within its accessibility cone identified as part of the pit limit.

1.3 An Overview Of The Extraction Sequence Problem

Once the pit limit has been identified, the problem of determining the extraction sequence must be solved. Although some algorithms combine these two steps, we believe separating the problems may make the solution of the problems compu-

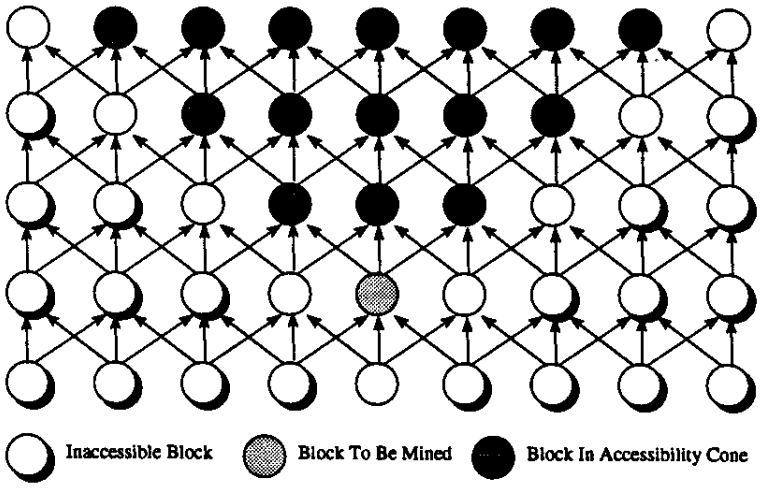


Figure 1.2: Directed Graph Representation of Block Model

tationally tractable and easier to implement. Two possible optimizing criteria may be considered in solving the extraction sequence problem. Maximizing the integral of the cash flow function and maximizing the discounted cash flow. We do not address which criteria is better.

Maximizing the integral of the cash flow function has relevance with respect to the payback period of the investment. Figure 1.4 shows two possible cash flow functions for the same pit limit and the original investment amount. The figure on the left has the larger integral value and the shorter payback period.

Maximizing the discounted cash flow is another relevant measure. The discounted cash flow, or net present value, of the extraction sequence is computed based upon a discount rate. Each block's value is multiplied by a discount multiplier raised to a power, where the power is determined by the blocks position in the extraction sequence.

The Nested Lerchs and Grossmann algorithm defines an extraction sequence based upon a sequence of nested pits. The algorithm generates a sequence of nested

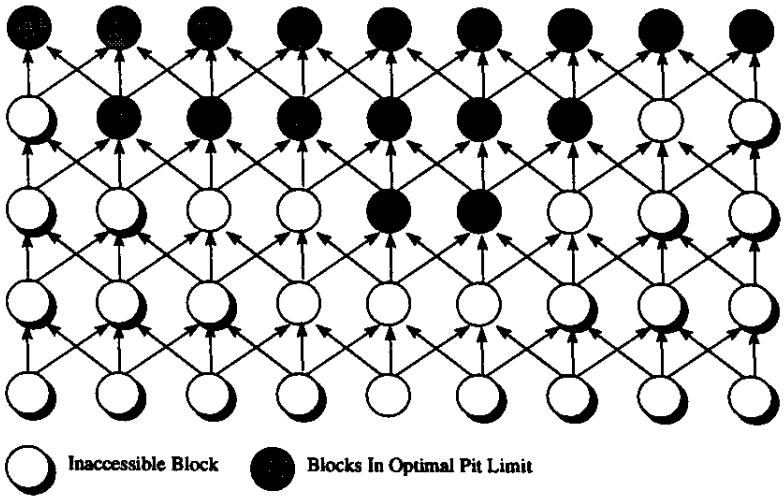


Figure 1.3: Representative Optimal Pit Limit

pits, where each consecutive nested pit increases the total volume (number of blocks) of the pit. The extraction sequence is obtained by mining the smallest nested pit, then the next smallest nested pit, etc., until the final nested pit has been mined. Figure 1.5 shows a representative collection of nested pits. The authors argue, but do not prove, that the given algorithm maximizes the integral of the cash flow function.

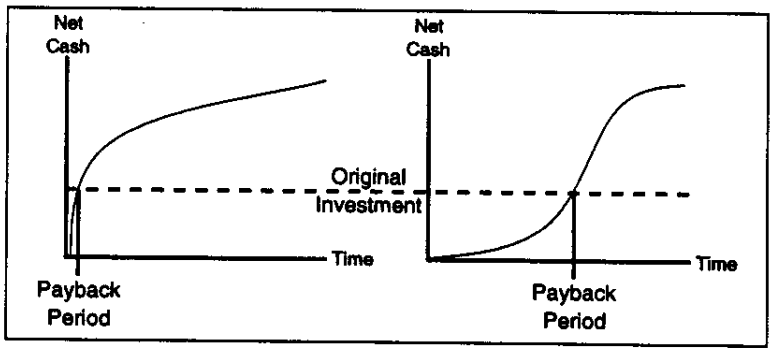


Figure 1.4: Payback Period Comparison

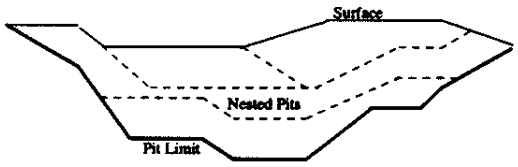


Figure 1.5: Cross Sectional View of Nested Pits

1.4 Research Summary

The goals of this research were: *i*) demystify and thoroughly explain the graph-theoretic pit limit algorithm developed by Lerchs and Grossmann, *ii*) define and prove, how, and if, the Nested Lerchs and Grossmann algorithm maximizes the cash flow function; and *iii*) develop an algorithm which maximizes the net present value of the extraction sequence.

Although we were successful in achieving our first two goals, we were unable to develop an algorithm which maximizes the net present value of the extraction sequence. We believe the problem is NP Hard; thus, finding an optimal solution is computationally intractable. We also believe the extraction sequence generated using our extended version of the Nested Lerchs and Grossmann algorithm gives a near optimal solution. Assuming the problem is NP Hard, it is not possible to prove the solution's error is within a given bound of the optimal.

In Chapter 2, we review Lerchs and Grossmann's pit limit algorithm (the LG Algorithm). We define common graph terminology and graph terminology specific to the LG Algorithm. We describe the steps of the LG Algorithm and provide a complete detailed example. The purpose of this chapter is to give the reader a thorough understanding of the LG Algorithm.

In Chapter 3, we develop the mathematical foundation of the LG Algorithm. We

discuss each step of the algorithm and clarify concepts through numerous examples. We also restate the authors proofs of the convergence of the LG Algorithm. We have added considerable commentary about the proofs and provided examples of their concepts in an effort to clarify the development.

In Chapter 4, we address the solution obtained using the LG Algorithm when the directed graph contains multiple maximum valued closures. A closure is a collection of blocks, such that for each block in the closure, every block upon which it is dependent is also in the closure. Thus, a feasible pit limit is equivalent to a closure. The solution obtained when a directed graph contains multiple maximum valued closures has not been addressed in the mining literature. We prove that the smallest maximum valued closure and the largest maximum valued closure are unique. We also prove that the maximum valued closure obtained when using the LG Algorithm is the smallest maximum valued closure. In addition, we prove that the LG Algorithm can be easily modified to obtain the largest maximum valued closure.

In Chapter 5, we view the LG Algorithm in a linear programming context. Through an example, we demonstrate that the LG Algorithm operates in the dual space of the pit limit problem. In other words, the LG Algorithm starts with an infeasible solution and through a series of changes of bases, with each change of basis moving the solution closer to feasibility, finds a feasible solution. The first feasible solution is also the optimal solution. The linear programming context enhances our understanding of steps performed by the LG Algorithm. Specifically, we show the LG Algorithm's normalization step is required when the current dual solution is infeasible. The normalization step brings the infeasible solution back into dual feasibility.

In Chapter 6, we develop the mathematical foundation for the Nested Lerchs and Grossmann algorithm (NLG Algorithm). We develop the mathematical programming

formulation of the problem, additional terminology relating to the algorithm, and provide a detailed example. We also prove the value of the offset parameter (the constant value subtracted from each node in the directed graph) equals the average value of the blocks contained in the increment between consecutive nested pits. We also prove that the nested pits generated by the NLG Algorithm have the following property: from smallest to largest nested pit, the nested pit's average value decreases monotonically. Thus, the NLG Algorithm's ordering principal is to maximize the average value of the nested pits.

In Chapter 7, we prove the nested pits generated by the NLG Algorithm maximizes the integral of the *global* cash flow function. We refer to the global cash flow function, since the NLG Algorithm provides no information on ordering nodes contained in the increment between consecutive nested pits. Ordering the nodes in the increment is called the *local ordering problem.* This problem was first identified by Dagdelen 1985, which he called the *gapping problem.* We present two approaches for finding the solution to the local ordering problem, which are based upon properties of the NLG Algorithm.

In Chapter 8, we demonstrate a shortcoming of the NLG Algorithm. When the optimizing criteria is discounted cash flow, the NLG Algorithm can be shown to define a suboptimal extraction sequence. We use a binary valued directed graph, where all non-positive valued blocks have the same value, and all positive valued blocks have the same value, to demonstrate this characteristic. We consider binary valued graphs to define the simplest type of problem.

We also develop the mathematical programming formulation of the discounted cash flow problem. We demonstrate the formulation must be a zero-one integer programming problem. Thus, the problem is NP-Hard. Thus, as implied by NP

theory, it is not possible to develop a polynomial order algorithm which can identify the optimal solution. In addition, it is not possible to prove that an algorithm provides a solution which is within some bound of the optimal solution.

Chapter 2
THE LERCHS AND GROSSMANN ALGORITHM

The graph theoretic algorithm developed by H. Lerchs and I. F. Grossmann 1965 (the *LG Algorithm*) identifies the maximum valued closure in a directed graph. We describe each step of the LG Algorithm and apply the steps to an example problem.

In Section 2.1, common graph terminology and notation is developed. In Section 2.2, terminology and notation specific to the LG Algorithm is developed. In Section 2.4, the steps of the LG Algorithm are described and applied to an example problem. A reader who is familiar with the LG Algorithm may choose to skip Section 2.4.

2.1 Common Graph Terminology

The graph theoretic development of the LG Algorithm requires the use of numerous commonly accepted terms and notation. These terms are stated, defined, and, where appropriate, examples are given. A few non-standard terms, an augmented graph, node dependence, and node independence, are also introduced.

A directed graph $G = (X, A)$ is defined by the sets X and A. The set of X contains elements called the *nodes* of G. A specific node in X is denoted with a lower case x and a subscript indicating its number; for example, x_i. The set A contains elements which are ordered pairs. Each ordered pair $a_{i,j} = (x_i, x_j)$ is called an *arc* of G. For any arc $a_{i,j}$, node x_i is the *source node*, and node x_j is the *terminal node*. An *edge* $e_{i,j} = [x_i, x_j]$ is an unordered pairing of nodes such that $a_{i,j} \in A$ or $a_{j,i} \in A$. Thus, an arc considers direction while an edge does not. Note that arcs are denoted

with parentheses and edges are denoted with square brackets.

A *path* consists of a sequence of *arcs*; such that for each arc in the sequence the arc's terminal node is the source node of the succeeding arc. Consider the directed graph in Figure 2.1. The sequence of arcs

$$\{a_{7,5} = (x_7, x_5), a_{5,3} = (x_5, x_3), a_{3,4} = (x_3, x_4), a_{4,1} = (x_4, x_1)\}$$

defines a path between nodes x_7 and x_1. Conversely, no path exists between node x_1 and any other node since both arcs containing x_1 terminate at node x_1.

A path which starts and ends with the same node forms a *circuit.* Again consider the directed graph in Figure 2.1. The path

$$\{a_{7,5} = (x_7, x_5), a_{5,3} = (x_5, x_3), a_{3,6} = (x_3, x_6), a_{6,7} = (x_6, x_7)\}$$

defines the only circuit in the directed graph. A directed graph containing no circuits will be referred to as a directed acycle graph.

A *chain* consists of a sequence of edges, such that each edge has one node in common with the succeeding edge. Again consider the directed graph in Figure 2.1. The sequence of edges

$$\{e_{1,3} = [x_1, x_3], e_{3,5} = [x_3, x_5], e_{5,2} = [x_5, x_2]\} = [x_1, x_3, x_5, x_2]$$

defines a chain between nodes x_1 and x_2. The shorthand notation on the right is frequently used. Since a chain consists of edges, rather than arcs, the direction of the arcs is irrelevant.

A chain which starts and ends with the same node forms a *cycle.* Using Figure

2.1 again, we see the sequence of edges

$$\{e_{3,4} = [x_3, x_4], e_{4,1} = [x_4, x_1], e_{1,3} = [x_1, x_3]\}$$

forms a chain. Again, the direction of arcs is irrelevant.

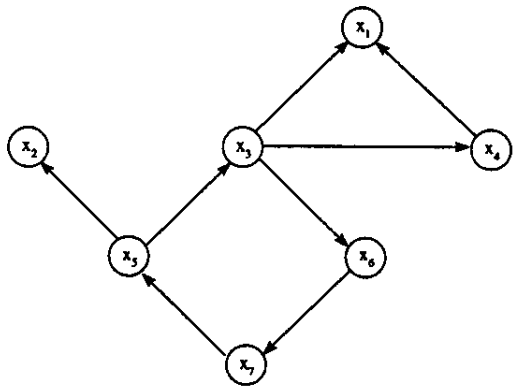


Figure 2.1: Paths, Circuits, Chains and Cycles

If there exists a path from node x_i to node x_j then node x_i is *dependent* upon node x_j. If there does not exist a path from node x_i to node x_j then node x_i is *independent* of node x_j. Dependence of node x_i upon node x_j will be denoted as $x_i \rightarrow x_j$ or $x_j \leftarrow x_i$. Independence of node x_i from node x_j will be denoted as $x_i \not\rightarrow x_j$ or $x_j \not\leftarrow x_i$. Any arc $a_{i,j} = (x_i, x_j)$ implies node x_i is *directly dependent* upon the node x_j. Thus, direct dependence is identified by a path containing a single arc.

For example, consider the directed graph shown in Figure 2.2. Node x_{11} is dependent upon node x_1, since there exists a path from x_{11} to x_1. Node x_{11} is directly dependent upon node x_7, since there exists a path containing a single arc from node x_{11} to node x_1. Node x_{11} is independent of node x_{10}, since there does not exist a path from node x_{11} to node x_{10}.

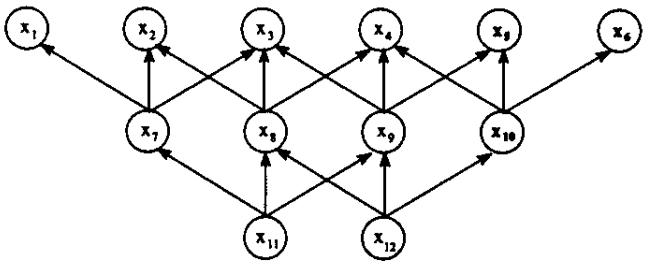


Figure 2.2: Node Dependence, Node Independence, and Closures

The closure of a node, defined in a directed circuitless graph, is a key concept which is critical to the research which follows. The *closure of a node* x_i consists of the set of all nodes which x_i is dependent upon. Consider Figure 2.2. The closure of node x_8 consists of node x_8 and all nodes which x_8 is dependent upon, nodes x_2, x_3 and x_4. We use the function $\Gamma(x_i)$, which maps from $X \rightarrow X$, to define the closure of node x_i. The function $\Gamma(x_i)$ may be defined recursively as:

$$
\begin{aligned}
\Gamma_0(x_i) &= \{x_i\}, \\
\Gamma_1(x_i) &= \Gamma_0(x_i) \cup \{x_j : a_{k,j} = (x_k, x_j) \in A \ , \ x_k \in \Gamma_0(x_i)\}, \\
\Gamma_2(x_i) &= \Gamma_1(x_i) \cup \{x_j : a_{k,j} = (x_k, x_j) \in A \ , \ x_k \in \Gamma_1(x_i)\}, \\
&\vdots \\
\Gamma_h(x_i) &= \Gamma_{h-1}(x_i) \cup \{x_j : a_{k,j} = (x_k, x_j) \in A \ , \ x_k \in \Gamma_{h-1}(x_i)\}.
\end{aligned}
$$

The subscripting denotes the recursion level. The zero level closure Γ_0 always consists of the single node x_i. The first level closure Γ_1 adds to the zero level closure those nodes which node x_i is directly dependent upon. Each level of the recursive definition adds an additional layer of nodes, where the new nodes have at least one node in the previous level which is directly dependent upon it. The final level closure,

level h, contains all nodes which node x_i is dependent upon.

The number of recursion levels depends upon the nodes location within the directed graph. For example, node x_1 will require only a single recursion level, node x_8 requires two recursion levels. Where it is not relevant, we will drop the recursion level subscript. Thus, $\Gamma(x_i)$ denotes the closure of node x_i, irrespective of the number of recursion levels it actually contains.

For example consider the directed graph in Figure 2.2. The closure of node x_{11} is computed as:

$$\begin{aligned}
\Gamma_0(x_{11}) &= \{x_{11}\}, \\
\Gamma_1(x_{11}) &= \Gamma_0(x_{11}) \cup \{x_j : a_{k,j} \in A, x_k \in \Gamma_0(x_{11})\}, \\
&= \{x_{11}\} \cup \{x_j : a_{11,j} \in A, x_k \in \{x_{11}\}\}, \\
&= \{x_{11}\} \cup \{x_7, x_8, x_9\}, \\
\Gamma_2(x_{11}) &= \Gamma_1(x_{11}) \cup \{x_j : a_{k,j} \in A, x_k \in \Gamma_1(x_{11})\}, \\
&= \{x_7, x_8, x_9, x_{11}\} \cup \{x_j : a_{k,j} \in A, x_k \in \{x_7, x_8, x_9, x_{11}\}\}, \\
&= \{x_7, x_8, x_9, x_{11}\} \cup \{x_1, x_2, x_3, x_4, x_5\},
\end{aligned}$$

Thus, the closure of node x_{11} is

$$\Gamma(x_{11}) = \{x_1, x_2, x_3, x_4, x_5, x_7, x_8, x_9, x_{11}\}.$$

In addition to the closure of a node, we will frequently refer to the *closure of a node set.* If $Y \subseteq X$ then the *closure of the node set* Y is the union of the closures of

the nodes in Y, or

$$\Gamma(Y) = \bigcup_{x_i \in Y} \Gamma(x_i). \tag{2.1}$$

In Figure 2.2, the closure of the node set $Y = \{x_7, x_{12}\}$ is

$$\Gamma(Y) = \Gamma(\{x_7\}) \cup \Gamma(\{x_{12}\}) = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{12}\}.$$

A set of nodes Y, where $\forall x_i \in Y, \Gamma(x_i) \subseteq Y$, is called a *closure* or a *closed set.* For example, the set $Y = \{x_2, x_4, x_5, x_6, x_{10}\}$ is a closure, since $Y = \bigcup \Gamma(x_i),\ \forall x_i \in Y$.

The concept of an *augmented graph* also plays an important role in the development of the LG Algorithm. For a directed graph $G = (X, A)$, consider adding a new node x_0 and a corresponding set of additional arcs $\{a_{0,i} : \forall x_i \in X\}$. Let $X' = X \cup \{x_0\}$ and $A' = A \cup \{a_{0,i} : \forall x_i \in X\}$. The graph $G' = (X', A')$ is called the *augmented graph* of the directed graph G. Figure 2.3 shows the augmented graph of the directed graph shown in Figure 2.2.

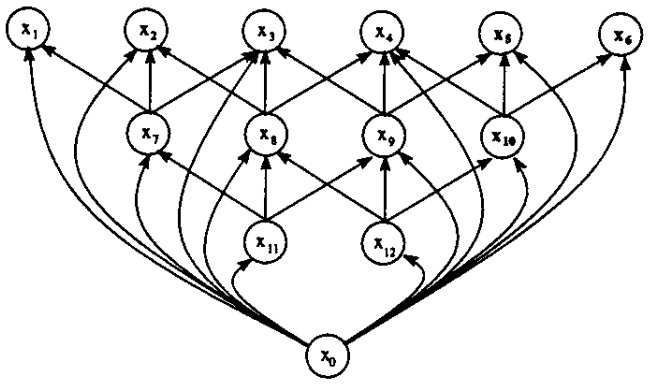


Figure 2.3: The Augmented Graph

The additional node x_0 is called the *artificial root* for reasons which are explained

below. The additional arcs $a_{0,i}$, are called the artificial arcs. In Figure 2.3, the artificial arcs are denoted with curved arcs. The non-artificial arcs are denoted with straight arcs and are called *real arcs.*

The LG Algorithm uses *connected trees* throughout its steps. A *connected tree* is a graph without any cycles, but with a chain of edges between every pair of nodes. A connected tree $T = (X', A'_T)$ defined in the augmented graph G', consists of a subset of arcs, $A'_T \subseteq A'$, and *all* nodes in X'. Since a connected tree contains all nodes in the augmented graph and only a small subset of its arcs there are many possible connected trees for a given augmented graph. The connected tree will always contain at least one artificial arc, and the artificial root is always the source of all artificial arcs.

Figures 2.4 and 2.5 show two different connected trees, defined in the augmented graph shown in Figure 2.3. The connected tree in Figure 2.4 is defined by the set of artificial arcs, the curved arcs. Note that for each arc $a_{i,j}$ in this tree, the artificial root x_0 is the source node. The connected tree shown in Figure 2.5 is defined by both artificial arcs and non-artificial (real) arcs.

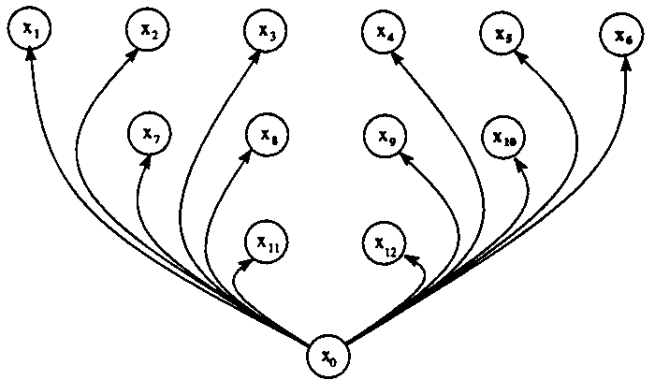


Figure 2.4: The Connected Tree Defined By The Artificial Arcs In G'

Another important term used in the LG Algorithm relates to the direction of arcs, with respect to the artificial arc, within the connected tree. An arc is said to *point away* from the artificial arc if the arc's terminal node precedes the arc's source node in the chain of edges connecting the source node to the artificial root.

For example, consider arc $a_{10,4} = (x_{10}, x_4)$ in the chain $[x_4, x_{10}, x_6, x_0]$. Since the arc's terminal node, x_4, precedes the arc's source node, x_{10}, the arc is said to point away from the artificial root. Note that *all* artificial arcs point away from the artificial root.

An arc is said to *point towards* the artificial root if the arc's source node precedes the arc's terminal node in the chain of edges connecting the source node to the artificial root.

For example, consider arc $a_{11,7} = (x_{11}, x_7)$ in the chain connecting node x_{11} and the artificial root x_0, $[x_{11}, x_7, x_1, x_0]$. Since the arc's source node, x_{11}, precedes the arc's terminal node, x_7, in the chain, the arc is said to point towards the artificial root.

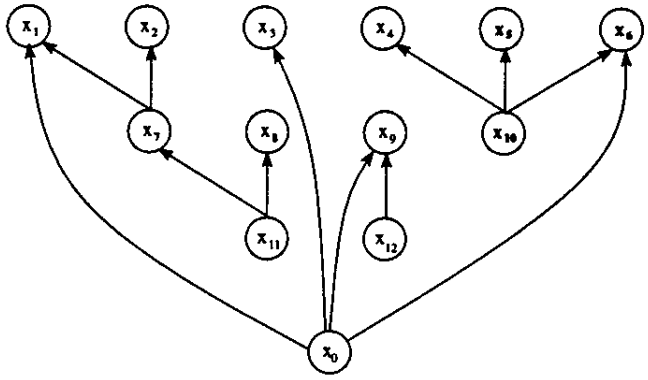


Figure 2.5: A Connected Tree Defined By Artificial And Real Arcs In G'

Severing any arc $a_{i,j} = (x_i, x_j) \in A'_T$ partitions the connected tree T into two

disjoint trees: the tree containing the artificial root x_0 is called the *rooted tree*; the tree not containing x_0 is called a *branch*. The severed arc $a_{i,j}$ is said to *support* the branch. The node x_i or x_j contained in the branch is called the *branch root*. Repeating the process by severing an arc in the branch, results in the formation of a *twig*. Specifically, severing any arc $a_{k,l} = (x_k, x_l)$ in a branch partitions the branch into two disjoint trees: the tree not containing the branch root is called a twig. The node x_k or x_l contained in the twig is called the *twig root*. Thus, only one rooted tree exists, branches define pieces of the rooted tree, and twigs define pieces of branches.

The *branch function* $B(x_i)$, is a mapping from $X \rightarrow X$, which defines the set of nodes contained in the branch with branch root x_i. The artificial root x_0 is never contained in a branch.

For example consider Figure 2.5. Severing the arc $a_{7,1}$ forms a branch containing the set of nodes $\{x_2, x_7, x_8, x_{11}\}$. The branch's root is node x_7. Thus, $B(x_7) = \{x_2, x_7, x_8, x_{11}\}$. Severing the arc $a_{11,7}$ forms a twig in the branch $B(x_7)$ containing the set of nodes $\{x_8, x_{11}\}$. The twig's root is node x_{11}. Thus, $B(x_{11}) = \{x_8, x_{11}\}$ denotes the branch with root x_{11}; or the twig with root x_{11}, when considered with respect to the branch $B(x_7)$. Note that the branch $B(x_{10}) = \{x_4, x_5, x_{10}\}$ does not include node x_6. This is because the branch $B(x_{10})$ is formed by severing the arc $a_{10,6}$, which places node x_6 in the rooted tree.

2.2 LG Algorithm Graph Terminology

Given the terms defined above, we can say the optimizing criteria of the pit limit problem is to define the maximum valued closure in the directed graph $G = (X, A)$. The LG Algorithm creates the augmented graph $G' = (X', A')$ from the directed graph $G = (X, A)$. Then the LG Algorithm creates a sequence of trees within the

augmented graph G'. The trees are connected trees, but we shall drop the adjective connected since all references to trees shall assume the tree is a connected tree. Within each connected tree the arcs are classified in terms of *direction* and *strength.* Note that arc classifications only apply to arcs in the tree T, not the full set of arcs contained in the augmented graph G'. In addition to arc classifications, nodes are classified based upon their *strength.*

The LG Algorithm (Lerchs and Grossmann 1965) introduces two types of arc classifications and a single node classification. The *directional classification of an arc* refers to the direction the arc points, either towards or away from the artificial root. The *strength classification of an arc* is determined by the net value of nodes in the branch supported by the arc. The *strength classificaton of a node* is determined by the strength classification of the set of arcs in the chain of edges between the node and the artificial root.

The *directional classification of arcs* classifies each arc in the connected tree T as either a p-arc or a m-arc. The directional classification of arcs depends upon the arc's direction in relation to the artificial root x_0. Put simply, a p-arc points away from x_0, a m-arc points toward x_0.

Formally, a *p-arc* is any arc, $a_{s,t} = (x_s, x_t)$, such that severing the arc results in the arc's terminal node x_t being located in the branch. Thus, the source node x_s resides in the rooted tree. The branch $B(x_t)$ is called a *p-branch.* To demonstrate, consider Figure 2.5. Severing the arc $a_{10,5}$ places the arc's terminal node x_5 in the branch $B(x_5)$; thus, the arc is a p-arc and the branch $B(x_5) = \{x_5\}$ is a p-branch.

A *m-arc* ia any arc, $a_{s,t} = (x_s, x_t)$, such that severing the arc results in the arc's terminal node x_t being located in the rooted tree. Thus, the source node x_s resides in the branch. The branch $B(x_s)$ is called a *m-branch.* Severing the arc $a_{7,1}$, in Figure

2.5, places the terminal node x_1 in the rooted tree; thus, the arc is a m-arc and the branch $B(x_7)$ is a m-branch.

In Figure 2.5, the set of all p-arcs is

$$\{a_{0,1}, a_{0,3}, a_{0,9}, a_{0,6}, a_{7,2}, a_{10,4}, a_{10,5}, a_{11,8}\},$$

and the set of all m-arcs is

$$\{a_{7,1}, a_{10,6}, a_{11,7}, a_{12,9}\}.$$

We shall use the function $\nu(x_i)$ to denote the value of node x_i. When applied to a set of nodes, the function $\nu(Y)$ gives the *value of the node set* Y. Thus, $\nu(Y)$, maps $X \rightarrow \Re$, and is defined as $\nu(Y) = \sum_{x_i \in Y} \nu(x_i)$. The composite function $\nu(B(x_i))$ gives the *value of the branch* with branch root x_i. Similarly, the composite function $\nu(\Gamma(Y))$ denotes the value of the closure of the set of nodes Y.

Recall that the optimizing criteria of the pit limit problem is to determine the *maximum valued closure* in a directed graph, the closure whose value is greater than or equal to the value of all other closures.

The *strength classification of arcs* classifies each arc in the connected tree T as either a strong arc or a weak arc. The strength of an arc is determined by the directional classification of the arc and the value of the branch supported by the arc. A *strong p-arc* is any p-arc which supports a positive valued branch ($\nu(B_{x_i}) > 0$). A *strong m-arc* is any m-arc which supports a non-positive valued branch ($\nu(B_{x_i}) \leq 0$). Arcs which are not strong are weak. Thus, a *weak p-arc* supports a non-positive valued branch, and a *weak m-arc* supports a positive valued branch.

In Chapter 4, we demonstrate that the location of the equality in the arc

strength classification definitions determines which closure, when the directed graph contains multiple maximum valued closures, the LG Algorithm converges to.

The *strength classification of nodes* classifies nodes as either a strong node or a weak node. It is determined by the strength classification of arcs, in the chain of edges between the node and the artificial root. All nodes which contain at least one strong arc in the chain between them and the artificial root are classified as *strong nodes.* Any node which does not have a strong arc in the chain between them and the artificial root are classified as *weak nodes.*

Figure 2.6 shows the connected tree depicted in Figure 2.5, where node values have been placed inside the node symbols, the node designations have been moved to the left of the node symbols, and the different arc styles and node shapes represent arc and node classifications. Line styles indicate directional classification of arcs: a dashed arc denotes a m-arc, a solid arc denotes a p-arc. Line thickness indicates strength classification of arcs: a thick arc denotes a strong arc, a thin arc denotes a weak arc. Node symbols indicate strength classification of nodes: a square denotes a strong node, a circle denotes a weak node. Table 2.1 identifies the branch root, branch value, and the directional and strength classification of all arcs in Figure 2.6. From the table we see arc $a_{0,1}$ is the only strong arc, thus nodes in the branch $B(x_1)$ are the only strong nodes.

The concept of a *normalized tree* also plays an important role in the LG Algorithm. A *normalized tree* is a connected tree where all strong arcs are *adjacent* to the artificial root x_0. An arc is adjacent to the artificial root x_0 if x_0 is one of the arc's nodes. Since all arcs containing the artificial root have the artificial root as the source node, we also know that an arc adjacent to the artificial root has the artificial root as its source node. The initialization step of the LG Algorithm forms a normalized tree.

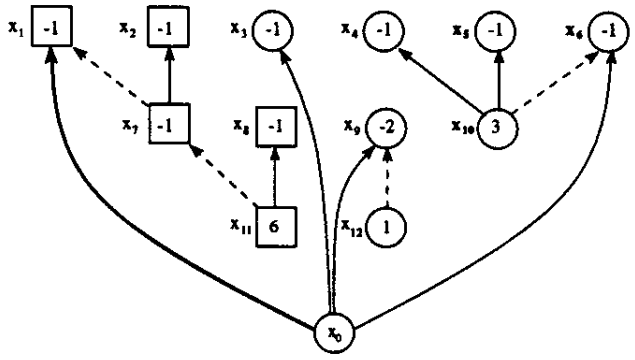


Figure 2.6: Arc and Node Classifications

Table 2.1: Arc Classifications

Arc	Branch Root x_{br}	Branch Value $\nu(B(x_{br}))$	Directional Classification	Strength Classification
$a_{0,1}$	x_1	2	p-arc	strong
$a_{0,3}$	x_3	-1	p-arc	weak
$a_{0,9}$	x_9	-1	p-arc	weak
$a_{0,6}$	x_6	0	p-arc	weak
$a_{7,2}$	x_2	-1	p-arc	weak
$a_{10,4}$	x_4	-1	p-arc	weak
$a_{10,5}$	x_5	-1	p-arc	weak
$a_{11,8}$	x_8	-1	p-arc	weak
$a_{7,1}$	x_7	3	m-arc	weak
$a_{10,6}$	x_{10}	1	m-arc	weak
$a_{11,7}$	x_{11}	5	m-arc	weak
$a_{12,9}$	x_{12}	1	m-arc	weak

Each iteration of the algorithm modifies the normalized tree to form a new connected tree; the connected tree, if required, is then forced to become a normalized tree.

The connected tree shown in Figure 2.6 is a normalized tree. The connected tree contains a single strong arc, arc $a_{0,1}$. Since all strong arcs (in this case a single arc) are adjacent to the artificial root the tree is also a normalized tree.

2.3 An Overview of the LG Algorithm

The LG Algorithm is an iterative algorithm which has been proven to converge to the maximum valued closure in a directed graph. In Chapter 5 we demonstrate that in a mathematical programming framework the LG Algorithm would be considered a dual algorithm. It starts with an infeasible solution, iterates through a sequence of primal infeasible solutions, with each iteration moving the solution closer to a primal feasible solution, until a primal feasible solution is identified. The first primal feasible solution is the optimal solution.

At each iteration, a *solution* is determined by the set of strong nodes. In the context of the pit limit problem, each strong node identifies a block which should be mined. Each weak node identifies a block which should not be mined. Thus, a primal feasible solution is a solution where the set of strong nodes defines a closure in the directed graph. In this section, and the next, when we refer to a solution we are referring to the current set of strong nodes, not necessarily the final solution to the pit limit problem. We attempt to make the distinction clear at all times.

The LG Algorithm operates on the augmented graph G' rather than the directed graph G. The algorithm generates a sequence of non-repeating normalized trees, until the set of strong nodes defines a closure in the graph G.

The initial step is to construct a normalized tree within G'. The initial normal-

ized tree may be constructed in several ways, but the simplest is to use the set of artificial arcs to define the initial normalized tree.

In each iteration of the algorithm, a new tree, denoted as $\hat{T}_{i+1}$, is formed by replacing an arc in the normalized tree T_i with an arc contained in the graph G (but not already in T_i). The arc introduced is selected from the set of arcs in G (but not in T_i) which have a strong node as its source node and a weak node as its terminal node. These arcs represent a dependency between a strong node and a weak node. By introducing one of these arcs into the normalized tree, the LG Algorithm is attempting to move the current solution closer to a feasible solution. Adding an arc to a connected tree causes a cycle to form within the tree. Thus, at least one arc must be removed from the tree. The LG Algorithm uses a simple rule (described below) to select the arc to remove. The rule's simplicity can cause the new tree $\hat{T}_{i+1}$ to be a non-normalized tree. We consider these operations to be a *transformation of the tree.* This transformation is performed by the *MoveTowardFeasibility* procedure (described below).

Since the new tree $\hat{T}_{i+1}$ is not guaranteed to be a normalized tree, a normalization procedure is required. The normalization procedure is another tree transformation. The normalization procedure selectively replaces an arc in the tree $\hat{T}_{i+1}$ with an artificial arc, forming a tree we denote as T_{i+1}. These operations are performed in the *NormalizedTree* procedure (described below).

When the tree T_{i+1} does not contain any strong nodes which have a direct dependency upon a weak node, the iteration process is terminated. When the set of strong nodes does not have any dependencies upon any weak nodes, the set of strong nodes must define a closure in the graph. Thus, the set of strong nodes defines a feasible solution. The first feasible solution defines the maximum valued closure of

the graph G.

Psuedocode for the **LG Algorithm** is:

$G' = (X', A') \leftarrow AugmentGraph(G = (X, A))$
$T_0 = (X', A'_{T_0} = \{a_{0,j} : x_j \in X\}) \leftarrow InitNormalizedTree(G')$
$T_i \leftarrow T_0$
while $SolutionNotFeasible(T_i)$
    $\hat{T}_{i+1} \leftarrow MoveTowardFeasibility(T_i)$
    $T_{i+1} \leftarrow NormalizeTree(\hat{T}_{i+1})$
end while
$S \leftarrow \{x_i : x_i \in X$ and x_i is a strong node$\}$.

2.4 The LG Algorithm

In this section we provide a detailed description of the steps of the LG Algorithm. An example is provided to explain the operations of each step. This section is somewhat tedious, thus the knowledgeable reader may choose to skip to the next section.

The *AugmentGraph* procedure creates the *augmented graph* $G' = (X', A')$. The augmented graph is formed by adding the artificial root x_0 to the node set X, and adding artificial arcs $a_{0,i} = (x_0, x_i)$ to the arc set A. Each artificial arc, $a_{0,i}$, connects the artificial root to node x_i. Thus, X' and A' denote the augmented node set and augmented arc set, respectively.

Figure 2.7 shows the augmented graph which will be used, throughout this section, to describe the steps of the LG Algorithm. Figure 2.8 shows the initial normalized tree T_1. Figures 2.9-2.11 show the three normalized trees, T_2, T_3, and T_4, obtained at the completion of the first three iterations. Figure 2.12 shows the non-normalized tree $\hat{T}_5$ formed after the *MoveTowardFeasibility* procedure in the fourth iteration. Figure 2.13 shows the normalized tree T_5 obtained after the *NormalizeTree*

procedure is completed in the fourth iteration. Figures 2.14-2.16 show the final three normalized trees, T_6, T_7 and T_8, obtained at the completion of the final three iterations. The operations performed during each iteration are described below.

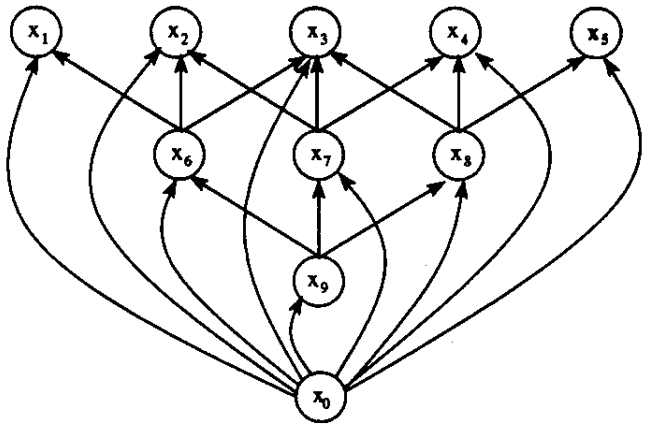


Figure 2.7: The Augmented Graph

The *InitNormalizedTree* procedure forms the initial normalized tree in the augmented graph G'. The initial normalized tree is specified by the arc set $A'_{T_0} \subseteq A$, which contains only artificial arcs $a_{0,i}$. The LG Algorithm does not require this to be the initial normalized tree, any normalized tree may be used as the initial normalized tree. Using the artificial arcs, we are guaranteed the tree is normalized, since *all* arcs in the tree are adjacent to the artificial root x_0. Recall that a *normalized* tree has all strong arcs adjacent to the artificial root.

Another benefit to using the artificial arcs to define the initial normalized tree, is the ease with which arcs and nodes may be classified. The initial set of strong arcs, $A'_S \subseteq A'_T$, is easily identified as those arcs supporting a node with positive value. The set of strong nodes, S, is the set of positive valued nodes. The set of weak arcs, A'_W, and weak nodes, W, are therefore $A'_W = A'_T \sim A'_S$ and $W = X \sim S$, respectively.

The initial set of strong nodes is not guaranteed to be a feasible solution (although in some rare cases it may be). Since it contains all positive valued nodes in the directed graph and excludes all non-positive valued nodes, the initial set of strong nodes will have the largest value of any infeasible or feasible solution.

The initial normalized tree T_1, for the augmented graph shown in Figure 2.7, is shown in Figure 2.8. The same representations, as used for Figure 2.9, are used here. The figure has shifted node designations to the left of the node symbols, and placed node values inside the symbols.

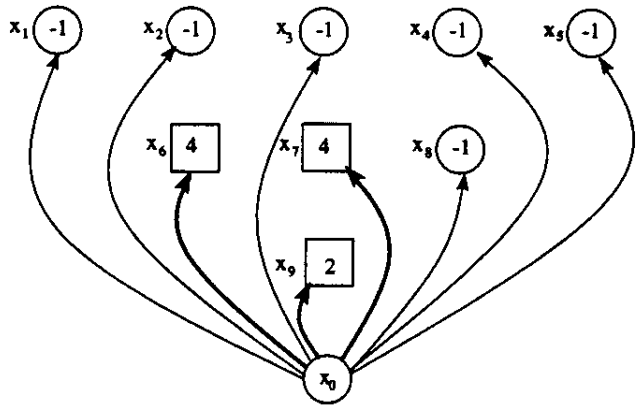


Figure 2.8: The Initial Normalized Tree T_1

Table 2.2 identifies the branch root, branch value, and the directional and strength classifications for each arc in the normalized tree T_1. The initial set of strong nodes S consists of all positive valued nodes, $S = \{x_6, x_7, x_9\}$, and has a value of $\nu(S) = 10$. The initial set of weak nodes W consists of all non-positive valued nodes. The initial solution, determined by the set of strong nodes, defines an upper bound on the value of all solutions.

After initialization, the current solution is tested for feasibility by the *Solution-NotFeasible* procedure. A solution is feasible when the set of strong nodes S defines

Table 2.2: Arc Classifications For The Initial Normalized Tree T_i

Arc	Branch Root x_{br}	Branch Value $\nu(B(x_{br}))$	Directional Classification	Strength Classification
$a_{0,1}$	x_1	-1	p-arc	weak
$a_{0,2}$	x_2	-1	p-arc	weak
$a_{0,3}$	x_3	-1	p-arc	weak
$a_{0,4}$	x_4	-1	p-arc	weak
$a_{0,5}$	x_5	-1	p-arc	weak
$a_{0,6}$	x_6	4	p-arc	strong
$a_{0,7}$	x_7	4	p-arc	strong
$a_{0,8}$	x_8	-1	p-arc	weak
$a_{0,9}$	x_9	2	p-arc	strong

a closure in the graph G. Thus, if there exists a strong node $x_s \in S$ which has a direct dependency upon any weak node $x_w \in W$, the set of strong nodes cannot define a closure in G and the solution is not feasible. In other words, if there exists an arc $a_{s,w} = (x_s, x_w) \in A$ then the current solution is infeasible. In order to move the solution closer to a feasible solution, the arc $a_{s,w}$ should be introduced into the normalized tree T_i

The *MoveTowardFeasibility* procedure introduces the arc $a_{s,w}$ into the tree T_i. Since adding an arc to a connected tree will form a cycle, and cause the connected tree to cease to be a tree, some existing arc in the tree must be removed. The algorithm always chooses to remove the artificial arc between the artificial root x_0 and the root of the branch containing the strong node x_s, which is denoted as node x_{r_s}. Since the tree is a connected tree there must exist such an arc.

The *MoveTowardFeasibility* procedure replaces arc a_{0,r_s} with arc $a_{s,w}$, forming the tree $\hat{T}_{i+1}$. The tree $\hat{T}_{i+1}$ may not be a normalized tree, but usually it is. To identify the root of the strong branch x_{r_s}, the *MoveTowardFeasibility* routine traverses

the chain between the strong node x_s and the artificial root x_0, searching for the arc adjacent to the artificial root. Denote this arc as a_{0,r_s}, since we know the artificial root must be the source node. The procedure then replaces the arc a_{0,r_s} with the arc $a_{s,w}$, forming the tree $\hat{T}_{i+1}$.

To describe the operations performed in the *MoveTowardFeasibility* procedure, we use the first three iterations of the example problem.

The *MoveTowardFeasibility* procedure introduces an arc found in the directed graph G, but not already in the normalized tree, into the initial normalized tree T_1. The arc introduced must represent a direct dependency between a strong node and a weak node. Each strong node $S = \{x_6, x_7, x_9\}$ in the initial normalized tree T_1 has at least one weak node which they are directly dependent upon. Node x_6 is directly dependent upon weak nodes $\{x_1, x_2, x_3\}$, node x_7 is directly dependent upon weak nodes $\{x_2, x_3, x_4\}$, and node x_9 is directly dependent upon weak nodes $\{x_8\}$. We choose to introduce arc $a_{9,8} = (x_9, x_8)$.

Once the arc to introduce is selected, the *MoveTowardFeasibility* procedure must identify the root of the strong branch containing the strong node x_9. Node x_9 is the strong node and also the root of the strong branch. Based upon the LG Algorithm's simple rule of removing the artificial arc supporting the branch which contains the strong node, arc $a_{0,9}$ is the arc to be removed.

The *MoveTowardFeasibility* procedure replaces the arc $a_{0,9} \in A'_T$ with the arc $a_{9,8} \in A$ forming the tree $\hat{T}_2$, shown in Figure 2.9. In the tree $\hat{T}_2$ the following arc and node classifications result. The arc $a_{9,8}$ is a m-arc and supports a branch with positive value, thus it is a weak arc. The arc $a_{0,8}$ is a p-arc and supports a branch with positive value, thus it is a strong arc. Node x_8 has a strong arc in the chain of edges between it and the artificial root x_0; thus, it is reclassified as a strong node.

The classifications of all other arcs and nodes remain unchanged.

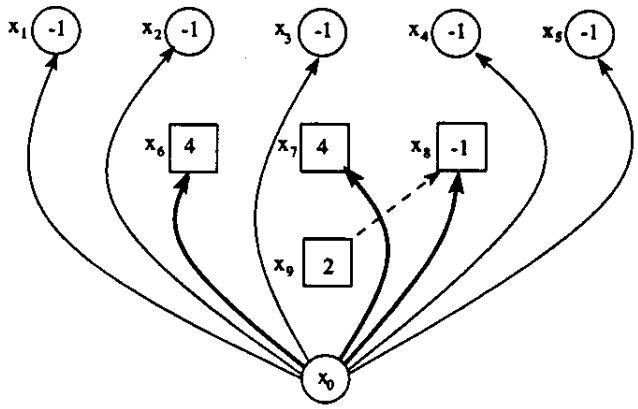


Figure 2.9: The Normalized Tree T_2 (End of Iteration 1)

Since a normalized tree is defined as a tree where all strong arcs are adjacent to the artificial root x_0, the tree $\hat{T}_2$ is normalized. Thus, $T_2 = \hat{T}_2$. In the tree T_2, the value of the strong node set has decreased to $\nu(S) = \nu(\{x_6, x_7, x_8, x_9\}) = 9$. The reduction in the value of the strong node set is sensible since the introduction of an arc, representing a direct dependency between a strong node and a weak node, must either decrease the value of the solution or leave the solution's value unchanged. The solution's value is unchanged when the weak node, x_8 in this iteration, has zero value.

In the second iteration, three of the four strong nodes in Figure 2.9 have direct dependencies on weak nodes. Node x_9 does not have a direct dependency upon a weak node. We chose to introduce the arc $a_{8,4} = (x_8, x_4)$. Again applying the simple rule for identifying the arc to remove, we see arc $a_{0,8}$ will be removed.

Figure 2.10 shows the tree $\hat{T}_3$, obtained after the transformation performed by the *MoveTowardFeasibility* procedure, which replaces the arc $a_{0,8} \in A'_T$, in T_2, with the arc $a_{8,4}$. The resulting arc classification changes are: the arc $a_{8,4}$ is a m-arc which supports a positive valued branch, thus it is a weak arc; and, the arc $a_{0,4}$ is a p-

arc which supports a non-positive valued branch, thus it remains a weak arc. The classification of arc $a_{0,4}$ as weak, causes nodes x_8 and x_9 to no longer have a strong arc in the chain between them and the artificial root. Thus, these nodes are reclassified as weak nodes. The classifications of all other arcs and nodes remain unchanged.

All strong arcs are adjacent with the artificial root x_0, thus the tree is normalized and $T_3 = \hat{T}_3$. The resulting value of the strong node set has again decreased to $\nu(S) = \nu(\{x_6, x_7\}) = 8$.

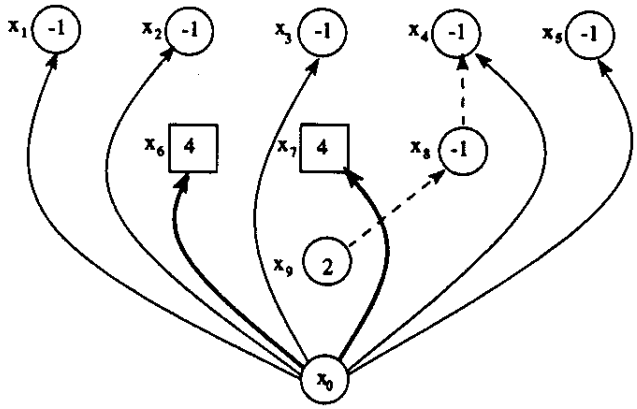


Figure 2.10: The Normalized Tree T_3 (End of Iteration 2)

In the third iteration, we replace arc $a_{0,4}$ with arc $a_{7,4}$, forming the tree $\hat{T}_4$ which is shown in Figure 2.11. The resulting arc classification changes are: the arc $a_{7,4}$ is a m-arc which supports a positive valued branch, thus it is a weak arc; and, the arc $a_{0,4}$ is a p-arc which supports a positive valued branch, thus it is a strong arc. Thus, all nodes in the branch $B(x_4)$ become strong. The classifications of all other arcs and nodes remain unchanged. Since all strong arcs are adjacent with the artificial root x_0 the tree is normalized; and $T_4 = \hat{T}_4$. In the tree T_4, the value of the strong node set remains constant, $\nu(S) = \nu(\{x_4, x_6, x_7, x_8, x_9\}) = 8$, but notice that the size of the strong node set in T_4 is larger than the size of the strong node set in T_3.

Review of the changes in the strong node set, after each of the first three iterations, indicates an important property of the LG Algorithm. This property will be proven to hold in Section 3.3. The property may be summarized as: either the value of the strong node set will decrease, or the value will remain constant but the *size* of the strong node set will increase. In the first two iterations, the value of the node set decreased. In the third iteration, the value of the node set remained constant but the size of the strong node set increased. This property is used to prove the LG Algorithm converges to the optimal solution in a finite number of iterations.

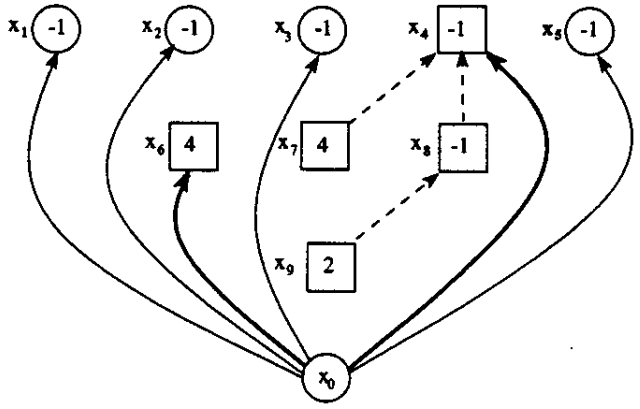


Figure 2.11: The Normalized Tree T_4 (End of Iteration 3)

In each of the first three iterations the tree formed by the *MoveTowardFeasibility* procedure was a normalized tree. In the fourth iteration, we see a non-normalized tree formed. We use this iteration to describe the operations of the *NormalizeTree* procedure.

Three of the strong nodes in Figure 2.11 have direct dependencies upon weak nodes. We chose to introduce the arc $a_{8,5}$. Thus, the LG Algorithm requires arc $a_{0,4}$ be removed. Introducing the arc $a_{8,5}$ forms the non-normalized tree $\hat{T}_5$, shown

in Figure 2.12. The tree is non-normalized since there exists a strong arc which is non-adjacent to the artificial root, arc $a_{8,4}$.

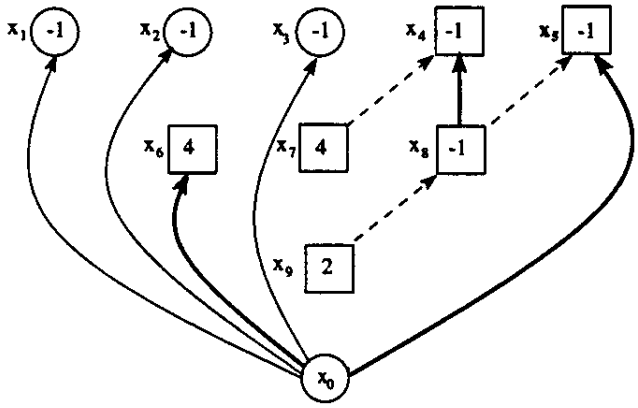


Figure 2.12: The Non-Normalized Tree $\hat{T}_5$

At the completion of the *MoveTowardFeasibility* procedure the following arc classification changes have occurred: the arc $a_{8,4}$ has changed from a m-arc to a p-arc, and since it supports a positive valued branch it is a strong arc; the arc $a_{8,5}$ is a m-arc which supports a positive valued branch, thus it is a weak arc; and, the arc $a_{0,5}$ is a p-arc which supports a positive valued branch, thus it is a strong arc. Since the arc $a_{8,4}$ is a strong arc which is not adjacent to the artificial root x_0, the tree $\hat{T}_5$ is a non-normalized tree. Since the tree is non-normalized, the *NormalizeTree* procedure transforms the tree to a normalized tree.

The *NormalizeTree* procedure performs a sequence of transformations which replaces each strong arc which is non-adjacent to the artificial root x_0 with an artificial arc. After completing these transformations, the final tree is guaranteed to be a normalized tree.

Depending upon the directional classification of each strong arc which is non-adjacent to the artificial root x_0, one of the following *NormalizeTree* transformations

is performed. Let $a_{s,t} = (x_s, x_t)$ denote a strong arc which is non-adjacent to the artificial root, where x_s denotes the source node and x_t denotes the terminal node. If the strong arc $a_{s,t}$ is a p-arc, then it is replaced by the artificial arc $a_{0,t}$. If the strong arc $a_{s,t}$ is a m-arc, then it is replaced by the artificial arc $a_{0,s}$.

The transformations performed by the *NormalizeTree* procedure prevent the *inappropriate* allocation of value between nodes. Inspection of Figure 2.12 demonstrates what is meant by *inappropriate* allocation of value between nodes. Partition nodes in the branch containing the strong arc non-adjacent to the artificial root into two sets, $Y_s = \{x_4, x_7\}$ and $Y_w = \{x_5, x_8, x_9\}$. Where Y_s contains nodes in the twig supported by the strong arc. Computing the value of the two sets, $\nu(Y_s) = 3$ and $\nu(Y_w) = 0$, we see that set Y_s has positive value and set Y_w has non-positive value. Thus, nodes in Y_w are classified as strong nodes because the net positive value of nodes in Y_s has been allocated to the branch containing them. Since nodes in Y_s are independent of nodes in Y_w, there is no justification for allocating the positive value of Y_s to the non-positive value of Y_w. Thus, the arc $a_{8,4}$ is inappropriately allocating the net positive value of nodes in Y_s to nodes in Y_w.

To prevent the inappropriate allocation, the strong p-arc $a_{8,4}$ is replaced with the arc $a_{0,4}$; forming the normalized tree T_5, shown in Figure 2.13. This transformation forces the nodes in Y_w to stand alone and be classified as strong based upon their own net value, not value obtained from nodes which are independent of them.

As a result of the normalization procedure the following arc and node classifications result. The introduced arc $a_{0,4}$ is a p-arc which supports a positive valued branch, thus it is a strong arc and all nodes in the branch are classified as strong. The arc $a_{0,5}$ remains a p-arc, but since it now supports a branch with non-positive value it is reclassified as weak, and all nodes in the branch are reclassified as weak.

Table 2.3: Arc Classifications For Normalized Tree T_8

Arc	Branch Root x_{br}	Branch Value $\nu(B(x_{br}))$	Directional Classification	Strength Classification
$a_{0,3}$	x_3	1	p-arc	strong
$a_{0,4}$	x_4	3	p-arc	strong
$a_{0,5}$	x_5	0	p-arc	weak
$a_{6,1}$	x_1	-1	p-arc	weak
$a_{6,2}$	x_2	-1	p-arc	weak
$a_{6,3}$	x_6	2	m-arc	weak
$a_{7,4}$	x_7	4	m-arc	weak
$a_{8,5}$	x_8	1	m-arc	weak
$a_{9,8}$	x_9	2	m-arc	weak

The new set of strong nodes defines the smallest maximum valued closure in the tree T_5 with value $\nu(S) = \nu(\{x_4, x_6, x_7\}) = 7$. Note that, the value of the strong node set in T_5 has decreased from the value of the strong node set in T_4.

The next three iterations of the LG Algorithm form the normalized trees T_6, T_7, and T_8, by introducing the arcs $a_{6,1}$, $a_{6,2}$, and $a_{6,3}$, respectively. These trees are shown in Figures 2.14, 2.15, and 2.16, and will not be described in detail. The transformations performed by the *NormalizeTree* procedure were not required in forming these trees.

Table 2.3 defines for each arc in the normalized tree T_8, its branch root, the value of the branch it supports, and its directional and strength classifications. The tree contains two strong arcs, both adjacent to the artificial root, thus the tree is a normalized tree. In addition, the set of strong nodes, $S = \{x_1, x_2, x_3, x_4, x_6, x_7\}$, defines a closure in the directed graph G.

The termination condition requires no strong node be directly dependent upon any weak node. Where a direct dependency is implied by an arc between two nodes.

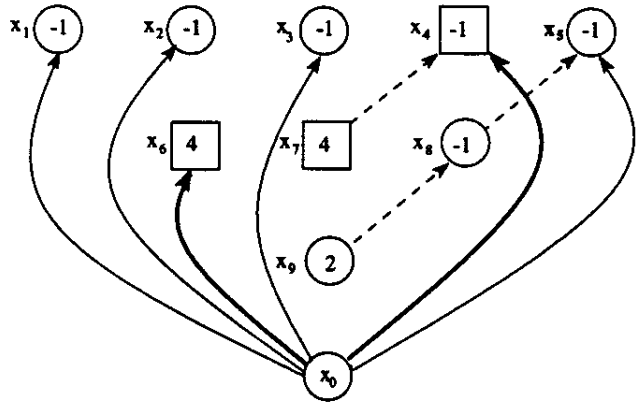


Figure 2.13: The Normalized Tree T_5 (End of Iteration 4)

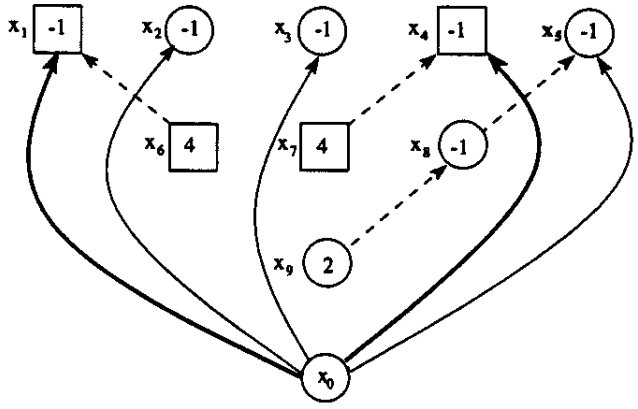


Figure 2.14: The Normalized Tree T_6 (End of Iteration 5)

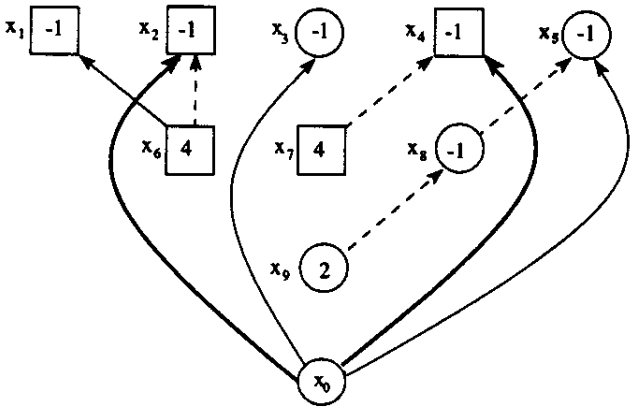


Figure 2.15: The Normalized Tree T_7 (End of Iteration 6)

The condition is checked in the *SolutionNotFeasible* procedure. When this condition is met, the set of strong nodes defines a closure in the directed graph G; and, thus is a feasible solution. We shall denote the *final normalized tree* as T_f.

From Figures 2.7 and 2.16 we see that none of the strong nodes in Figure 2.16 has a direct dependency upon any weak node. Therefore, the set of strong nodes, $S = \{x_1, x_2, x_3, x_4, x_6, x_7\}$, defines a closure in the original directed graph G; and, as is proven below, the closure also is the maximum valued closure.

This completes the example. In the next chapter we restate the properties used to prove the LG Algorithm is guaranteed to converge to the maximum valued closure in a directed graph in a finite number of iterations. In Chapter 4, we will prove that the LG Algorithm, as described here, converges to the smallest maximum valued closure in a directed graph with multiple maximum valued closures.

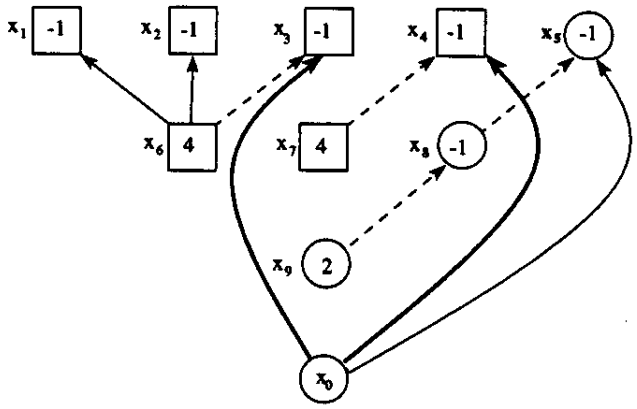


Figure 2.16: The Normalized Tree T_8 (End of Iteration 7)

Chapter 3
THEORETICAL DEVELOPMENT OF THE LG ALGORITHM

In this chapter, we restate the properties and theorems developed by H. Lerchs and I. F. Grossmann 1965. These properties and theorems prove the LG Algorithm converges, in a finite number of iterations, to the maximum valued closure in a directed graph. Although the arguments are based upon those given by Lerchs and Grossmann, we have added additional information to the proofs, made the notation precise, and we provide examples demonstrating the arguments.

LG Property 1 proves that if a node, x_s, belongs to the maximum valued closure of a normalized tree, then all the nodes in the branch $B(x_s)$ must also belong to the maximum valued closure of the *normalized tree.*

LG Property 2 proves the set of strong nodes, S, in a normalized tree, T, defines the maximum valued closure in the *normalized tree.*

LG Theorem I, uses LG Properties 1 and 2, to prove that if a normalized tree can be constructed in a directed graph G, such that the set of strong nodes, S, is a closure of the graph G, then S is a maximum valued closure in G.

LG Property 3 is a detailed property, which essentially proves a key relationship between the value of m-branches and p-branches in the chain of edges formed by the *MoveTowardFeasiblity* procedure.

LG Theorem II, uses LG Property 3, to prove the LG Algorithm constructs a normalized tree, in a finite number of iterations, such that the set of strong nodes, S, is a closure of the graph G. Thus, LG Theorem I in conjunction with LG Theorem II, proves the LG Algorithm converges, in a finite number of iterations, to the maximum

valued closure of a directed graph.

3.1 Definitions

In this section, terms used throughout this research effort are formally defined. In addition, several properties of normalized trees and closures in directed graphs are proven. Throughout this paper the symbol $\subseteq$ is intended to mean "is a subset of or equal to". The symbol $\subset$ is intended to mean "is a strict subset of".

Definition 1 *A* **directed graph** *consists of a set of nodes and a set of arcs connecting nodes. Denote a directed graph as $G = (X, A)$; where X is the set of nodes in G and A is the set of arcs in G.*

Definition 2 *In a directed graph, if there exists a path of arcs connecting node x_i to node x_j, then node x_i is said to* **depend** *upon node x_j.*

Definition 3 *In a directed graph, if there exists an arc with source node x_i and terminal node x_j, then node x_i is said to be* **directly dependent** *upon node x_j.*

Definition 4 *A* **closure** *in a directed graph is a set of nodes such that: for each node x_i in the set, every node which x_i depends upon is also in the set.*

Definition 5 *The* **augmented graph**, *of the directed graph $G = (X, A)$, is a directed graph which adds an additional node and additional arcs to the directed graph G. Denote the augmented graph as $G' = (X', A')$; where $X' = X \cup \{x_0\}$ and $A' = A \cup \{a_{0,i}\}$, $\forall x_i \in X$.*

Definition 6 *A* **partial graph**, *∂G, of a directed graph $G = (X, A)$, is a directed graph containing all nodes and a subset of the arcs in G. Thus, $\partial G = (X, \partial A)$, where $\partial A \subseteq A$.*

Definition 7 *A* **subgraph** *of a directed graph $G = (X, A)$, is a directed graph containing a subset of the nodes and a subset of the arcs connecting those nodes.*

Definition 8 *In the augmented graph $G' = (X', A')$, the node x_0 is called the* **artificial root**. *All other nodes are called* **real nodes**.

Definition 9 *In the augmented graph $G' = (X', A')$, each arc having the artificial root as it's source node is called an* **artificial arc**. *Thus, the set of arcs $\{a_{0,i}\}$, $\forall x_i \in X$ is the set of artificial arcs. All other arcs in A' are called* **real arcs**.

Definition 10 *A* **connected tree**, *is a directed graph, which contains a single chain of edges between every pair of nodes. Thus, a connected tree containing $n + 1$ nodes, contains n arcs.*

Definition 11 *A* **connected tree defined in a graph** *is a partial graph containing the minimal number of arcs required to maintain connectivity between nodes in the graph. Thus, a connected tree defined in the augmented graph, $G' = (X', A')$, may be denoted as $T = (X', A'_T)$, where $A'_T \subseteq A'$.*

Definition 12 *If an arc is severed (or removed) in a connected tree, a non-connected graph is formed. The non-connected graph consists of two subgraphs: a subgraph containing the root of the tree (the artificial root), and a subgraph not containing the root of the tree. The subgraph not containing the root of the tree is called a* **branch**.

Definition 13 *If an arc is severed (or removed) in a branch, two subgraphs of the branch are formed: the subgraph containing the root of the branch, and the subgraph not containing the root of the branch. The subgraph not containing the root of the branch is called a* **twig**.

Definition 14 *A* **p-arc** *is an arc such that, severing the arc places the arc's terminal node in the branch. A p-arc points away from the artificial root.*

Definition 15 *A* **m-arc** *is an arc such that, severing the arc places the arc's source node in the branch. A m-arc points towards the artificial root.*

Definition 16 *A* **strong p-arc** *is a p-arc supporting a positive valued branch.*

Definition 17 *A* **weak p-arc** *is a p-arc supporting a non-positive valued branch.*

Definition 18 *A* **strong m-arc** *is a m-arc supporting a non-positive valued branch.*

Definition 19 *A* **weak m-arc** *is a m-arc supporting a positive valued branch.*

Definition 20 *An arc is* **adjacent to a node** *if the node is either the arc's source node or terminal node.*

Definition 21 *A* **strong node** *is any node such that, in the chain of edges connecting the node to the artificial root, there exists at least one strong arc. The artificial arc is never classified as a strong node nor a weak node.*

Definition 22 *A* **weak node** *is any node such that, in the chain of edges connecting the node to the artificial root, there does not exist a strong arc.*

Definition 23 *A* **normalized tree** *is a connected tree defined in the augmented graph G' such that, every strong arc in the tree is adjacent to the artificial root. A normalized tree is also a partial graph.*

Definition 24 *A* **closure in a normalized tree** *is a closure defined in terms of the partial set of arcs which define the normalized tree; not the full set of arcs which define the augmented graph.*

To see the difference between the maximum valued closure in a directed graph and the maximum valued closure in a normalized tree, formed from the directed graph, consider the example problem discussed in Section 2.4. We found the maximum valued closure of the directed graph has a value of 4 and contains the following set of nodes:

$$\{x_1, x_2, x_3, x_4, x_6, x_7\}.$$

Now consider the normalized tree, T_5, shown in Figure 2.13. The smallest maximum valued closure of T_5, has a value of 7 and consists of the nodes x_4, x_6, and x_7.

To see that the set $\{x_4, x_6, x_7\}$ defines a maximum valued closure in the normalized tree, consider the remaining set of nodes. Including any node in $\{x_1, x_2, x_3, x_5\}$ reduces the value of the closure by 1. Including node x_8 reduces the value of the closure by 2. This follows since x_8 is dependent upon node x_5. Thus, to include x_8, node x_5 must also be included to define a closure. Including node x_9 will not change the value of the closure. Thus, the set of strong nodes, $\{x_4, x_6, x_7\}$, is a maximum valued closure of the normalized tree T_5.

Hence, the maximum valued closure of the normalized tree T_5 differs from the maximum valued closure of the directed graph. In general, the maximum valued closure of a tree is greater than or equal to the maximum valued closure of the graph from which the tree was obtained.

Lemma 1 *In a normalized tree, all arcs adjacent to the artificial root are p-arcs.*

Proof: All arcs adjacent to the artificial root are artificial arcs. Thus, the artificial root, x_0, is the arc's source node. Severing an artificial arc places the arc's terminal node in the branch. Hence, by Definition 14, the arc must be a p-arc □

Lemma 2 *In a normalized tree, all strong arcs are p-arcs.*

Proof: By Definition 23, all strong arcs are adjacent to the artificial root. By Lemma 1, all arcs adjacent to the artificial root are p-arcs. Hence, all strong arcs are p-arcs. □

Lemma 3 *In a normalized tree, all m-arcs are weak and support positive valued branches.*

Proof: By Lemma 2, all strong arcs are p-arcs. Thus, all m-arcs must be weak. By Definition 19, a weak m-arc supports a positive valued branch. □

Lemma 4 *Let $T = (X', A'_T)$ be a normalized tree defined in the augmented graph $G' = (X', A')$, and partition the nodes in T into two sets: the set of strong nodes, S_T; and, the set of weak nodes, W_T. Thus, $X = S_T \cup W_T$. If S_T is a closure in T, then it is not possible for T to contain an arc having a source node in S_T and a terminal node in W_T.*

Proof: Assume there exists an arc $a_{s,w} \in A'_T$, such that $x_s \in S_T$ and $x_w \in W_T$. By Definition 5, x_s is directly dependent upon x_s. Thus, S_T cannot define a closure in T. Therefore, T cannot contain any arcs connecting nodes in S_T to nodes in W_T. □

Lemma 5 *In a normalized tree $T = (X', A'_T)$, each p-branch defines a closure in T and no m-branch defines a closure in T.*

Proof: A p-branch is formed by severing a p-arc. Let $a_{b_p,b_r} = (x_{b_p}, x_{b_r})$ denote a p-arc in T. By Definition 5, node x_{b_p} is directly dependent upon x_{b_r}; but, node x_{b_r} is not dependent upon node x_{b_p}. After severing the arc, the branch $B(x_{b_r})$ contains a set of nodes such that: for each node in the branch, the branch contains every other node which it is dependent upon. Thus, by Definition 24, the branch is a closure in T. Hence, every p-branch defines a closure in T.

A m-branch is formed by severing a m-arc. Let $a_{b_r,b_p} = (x_{b_r}, x_{b_p})$ denote a m-arc in T. By Definition 3, node x_{b_r} is directly dependent upon node x_{b_p}. After severing the arc, the node x_{b_r} is in the branch and the node x_{b_p} is not in the branch. Thus, the branch $B(x_{b_r})$ does not contain a set of nodes such that: for each node in the branch, the branch contains every other node which it is dependent upon. Hence, m-branches do not define a closure in T. □

To demonstrate Lemma 5, consider the normalized tree shown in Figure 3.1. The lemma claims that each p-branch defines a closure in the tree. Consider severing the p-arc $a_{15,10}$, which forms the p-branch $B(x_{10}) = \{x_{10}, x_{11}, x_{16}, x_{19}\}$. For each node in the branch, all nodes which they depend upon are also included in the branch. Specifically: nodes x_{10} and x_{11} are not dependent upon any nodes; node x_{16} is dependent upon nodes x_{10} and x_{11}, both of which are contained in the branch; and node x_{19} is dependent upon node x_{16}, which is contained in the branch. As the lemma claims, the p-branch defines a closure in the normalized tree.

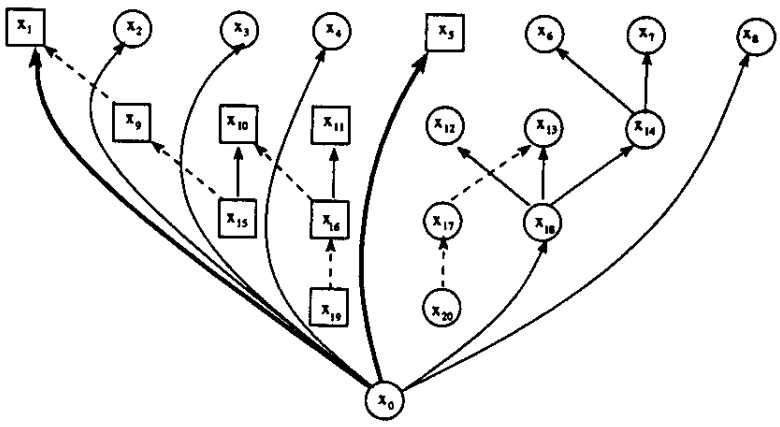


Figure 3.1: Closures in a Normalized Tree

The lemma also claims that it is not possible for a m-branch to define a closure in the tree. Consider the m-branch formed by severing the arc $a_{17,13}$. The m-branch contains nodes $B(x_{17}) = \{x_{17}, x_{20}\}$. Unlike a p-branch, a m-branch will always contain one node which is dependent upon a node which is not contained in it. Specifically, the branch root. For the m-branch $B(x_{17})$, node x_{17} is dependent upon node x_{13}. But, node x_{13} is not contained in the branch. Thus, the m-branch does not define a closure in the normalized tree.

Lemma 6 *If S is the maximum valued closure in the directed graph $G = (X, A)$, and S_T is the maximum valued closure in the normalized tree, $T = (X', A'_T)$, derived from G's augmented graph $G' = (X', A')$. Then $\nu(S_T) \geq \nu(S)$.*

Proof: From Definition 23, $A'_T \subseteq A'$. From Definition 5, $A \subset A'$. Partitioning the arcs in A'_T into the set of real arcs $A'_T \cap A$ and the set of artificial arcs $A'_T \cap (A' \sim A)$, we find $A'_T \cap A \subseteq A$. The set of artificial arcs represent the artificial root's direct dependencies upon real nodes. Since T cannot have more arcs than G, there cannot be more dependencies in T than there are in G. Hence, $\nu(S_T) \geq \nu(S)$. □

To demonstrate Lemma 6, consider the normalized tree in Figure 2.8, which was derived from the augmented graph in Figure 2.7. The normalized tree's maximum valued closure has a value of 10 and contains the following set of nodes $\{x_6, x_7, x_9\}$.

Now consider introducing an additional dependency (any arc in the augmented graph which is not in the normalized tree) into the normalized tree. As is shown below, introducing additional dependencies into a graph may reduce the value of the graph's maximum valued closure, or leave it unchanged, but will never increase it's value.

For example, consider adding arc $a_{8,5}$, which connects a weak node to a weak node. Adding this arc to the normalized tree makes node x_8 dependent upon node

x_5. The value of the maximum valued closure of the resulting graph (since it contains a cycle the graph is not a tree) equals the value of the maximum valued closure of the normalized tree. Adding the arc forms a graph with a maximum valued closure which has a value less than or equal to the value of the maximum valued closure in the normalized tree.

Consider adding arc $a_{9,7}$, which connects a strong node to a strong node. The graph's maximum valued closure does not change, and the value of the graph's maximum valued closure is less than or equal to the maximum valued closure in the normalized tree.

Although there does not exist an arc connecting a weak node to a strong node, assume an arc exists in the augmented graph, which connects node x_8 to node x_7. Adding this arc makes node x_8 dependent upon node x_7. Again, the graph's maximum valued closure is the same as the normalized tree's maximum valued closure and the value remains unchanged.

Now consider adding arc $a_{7,4}$, connecting a strong node to a weak node. This is the type of arc introduced in the LG Algorithm's *MoveTowardFeasibilty* procedure. Introducing this arc makes node x_7 dependent upon node x_4 in the new graph. Thus, the new graph's maximum valued closure would consists of the set of nodes $\{x_4, x_6, x_7, x_9\}$ and have a value of 9. Thus, the value of the graph's maximum valued closure is less than the normalized tree's maximum valued closure.

Therefore, we find that adding an arc connecting a weak node to a weak node, a strong node to a strong node, or a weak node to a weak node, forms a graph having a maximum valued closure with value equal to the value of the normalized tree's maximum valued closure. Introducing, an arc between a strong node and a weak node forms a graph having a maximum valued closure with value less than the value

of the normalized tree's maximum valued closure. Adding more arc's to the new graph will have the same effect.

Thus, as Lemma 6 states, the value of a normalized tree's maximum valued closure must be greater than or equal to the value of the maximum valued closure in the graph from which it was formed.

Lemma 7 *If $Y_1 \subseteq X$ and $Y_2 \subseteq X$ are closed sets in the graph $G = (X, A)$, then $Y_1 \cup Y_2$ is a closed set in G.*

Proof: For any node $x_i \in Y_1, \Gamma(x_i) \subseteq Y_1$, otherwise Y_1 cannot be a closed set. Thus, $\Gamma(x_i) \subseteq Y_1 \cup Y_2$. A similar argument holds for any node $x_i \in Y_2$. □

Lemma 8 *If $Y_1 \subseteq X$ and $Y_2 \subseteq X$ are closed sets in the graph $G = (X, A)$, then $Y_1 \cap Y_2$ is a closed set in G.*

Proof: For any node $x_i \in Y_1 \cap Y_2$, we know $\Gamma(x_i) \subseteq Y_1$ and $\Gamma(x_i) \subseteq Y_2$ otherwise Y_1 and Y_2 would not be closure sets. Thus, $\Gamma(x_i) \subseteq Y_1 \cap Y_2$. □

3.2 LG Theorem I

The Pit Limit Problem is to identify the maximum valued closure in a directed graph. Formally, it can be stated: for the directed graph $G = (X, A)$,

$$\begin{aligned} \text{max.} \quad & \nu(S) \\ \text{s.t.:} \quad & S \subseteq \text{X}, \\ & \Gamma(S) = \text{S}. \end{aligned}$$

To solve the Pit Limit Problem, the LG Algorithm forms an augmented graph, $G' = (X', A')$ from the directed graph $G = (X, A)$. The algorithm then generates a

sequence of normalized trees, $T_i = (X', A'_{T_i})$, until the set of strong nodes, S_{T_i}, in the normalized tree defines a closure in G.

LG Property 2 says: the set of strong nodes in a normalized tree, T_i, defines the maximum valued closure in T_i, not necessarily in the directed graph G. LG Theorem I says: when the set of strong nodes in T_i defines a closure in G then the set of strong nodes is the maximum valued closure in G.

When the set of strong nodes in T_i defines a closure in G, then there does not exist a strong node with a direct dependency upon a weak node. Thus, LG Theorem I is the basis for the LG Algorithm's stopping criteria: terminate the iteration process if there does not exist a direct dependency between a strong node and a weak node.

LG Property 1 says: if a node x_s belongs to the maximum valued closure in T_i, then all nodes in the branch $B(x_s)$ must also belong to the maximum valued closure in T_i. This property is used to prove LG Property 2.

3.2.1 LG Property 1

The proof of LG Property 1 is by contradiction. The proof assumes there exists a normalized tree with a branch containing: a set of nodes which are not in the maximum valued closure of the normalized tree, and the branch's root is in the maximum valued closure of the normalized tree. Based upon the assumed properties, it is demonstrated that by adding the nodes in the branch, a closure in the normalized tree can be formed where: the closure's value is greater than the value of the nodes in the original maximum valued closure, thus, contradicting the assertion that the original maximum valued closure is actually the maximum valued closure. Consequently, all nodes in a branch, with a branch root which is contained in the maximum valued closure of the normalized tree, must also be contained in the

maximum valued closure of the normalized tree. Following the proof, an example demonstrates its steps.

Lemma 9 (LG Property 1) *If a node x_s belongs to the maximum valued closure, S_T, of a normalized tree T, then all the nodes in the branch $B(x_s)$ must also belong to S_T.*

Proof: Assume $x_s \in S_T$, $x_w \in B(x_s)$, and $x_w \in W_T = X \sim S_T$. Thus, x_s is contained in the maximum valued closure of the normalized tree and x_w is not; and, x_w is contained in a branch having a strong node as it's root. By Lemma 4, all arcs between nodes in W_T and nodes in S_T, must have their terminal node in S_T. Denote this set of arcs with A_{W_T,S_T}.

There exists one arc in A_{W,S_T} which is a m-arc. To see this consider the chain of edges,

$$[x_w, \ldots, x_q, x_p, x_s, \ldots, x_0],$$

connecting node x_w to the artificial root, x_0, in T. The chain must go through node x_s, since node x_w is in the branch $B(x_s)$. If x_p is the first node in the chain which belongs to S_T, then $x_q \in W_T$. By Lemma 4, all arc's between nodes in W_T and S_T must have their terminal node in S_T, arc $a_{q,p} = (x_q, x_p)$ must be an m-arc, and $B(x_q)$ must be a m-branch.

All other arcs connecting nodes in $B(x_q)$ and nodes in S_T must be p-arcs. Assume there exists a second m-arc, a_{q_2,p_2}, connecting node $x_{q_2} \in B(x_q)$ with node $x_{p_2} \in S_T$. Then there exists two chains of edges connecting node x_{q_2} with the artificial root:

$$[x_{q_2}, \ldots, x_q, x_p, \ldots, x_s, \ldots, x_0],$$

and

$$[x_{q_2}, x_{p_2}, \ldots, x_0].$$

Thus, there exists a cycle in the tree. But, by Definiton 10, a tree can contain only a single chain of edges between any two nodes. Hence, there cannot exist two m-arcs between nodes in $B(x_q)$ and nodes in S_T.

Since T is normalized, each p-twig in $B(x_q)$ must have non-positive value. Let $B_p(x_q)$ be the *pruned branch*, which remains after all p-twigs have been removed. The p-twigs are removed to ensure $S_T \cup B_p(x_q)$ defines a closure in T. After removing the p-twigs, the only dependency between a node in $B_p(x_q)$ and a node in S_T, is the dependency between node x_q and x_p. But, since $x_p \in S_T$, $S_T \cup B_p(x_q)$ defines a closure in T. In addition, removing the non-positive valued p-twigs from $B(x_q)$ can only cause $\nu(B_p(x_q)) \geq \nu(B(x_q)) > 0$.

Thus, the value of the closure $S_T \cup B_p(x_q)$ is:

$$\nu(S_T \cup B_p(x_q)) = \nu(S_T) + \nu(B_p(x_q)), \tag{3.1}$$

$$> \nu(S_T). \tag{3.2}$$

But, this contradicts the assertion that S_T is the maximum valued closure in T.

Thus, it is not possible to have a node with the following properties: the node is *not* part of the normalized tree's maximum valued closure ($x_w \in W_T$); and, the node is contained in a branch which has a root node which is contained in the normalized tree's maximum valued closure ($x_w \in B(x_s)$). Thus, if a branch contains a node which is part of the tree's maximum valued closure than all nodes in the branch are part of the tree's maximum valued closure. □

To demonstrate the argument, consider the tree shown in Figure 3.2. By the statement of the lemma, the tree is normalized. The argument assumes S_T is the maximum valued closure in the tree, and there exists nodes $x_s \in S_T$, $x_w \in B(x_s)$, and $x_w \in W = X \sim S_T$. In the figure, we find

$$S_T = \{x_1, x_5, x_9, x_{10}, x_{11}, x_{15}, x_{16}, x_{19}\}.$$

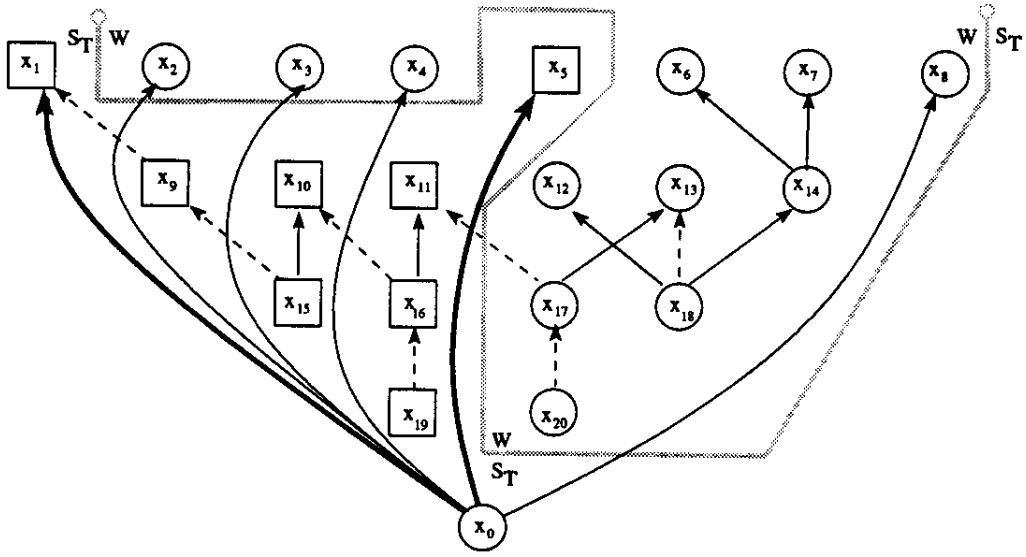


Figure 3.2: Demonstration of LG Property 1

Let x_{16} and x_{18} correspond to nodes x_s and x_w, respectively. Thus, the chain of edges from x_w to x_0 consists of the following sequence of nodes:

$$[x_{18}, x_{13}, x_{17}, x_{11}, x_{16}, x_{10}, x_{15}, x_9, x_1, x_0].$$

Nodes x_{17} and x_{11} correspond to nodes x_q and x_p, respectively.

The argument continues by identifying the following: all arcs between nodes in W and nodes in S_T have their terminal node in S_T; and, exactly one of these arcs is a m-arc. In Figure 3.2, there is only one arc between nodes in W and nodes in S_T, and this arc is a m-arc, arc $a_{17,11}$. Severing arc $a_{17,11}$ forms the branch

$$B(x_{17}) = \{x_6, x_7, x_{12}, x_{13}, x_{14}, x_{17}, x_{18}, x_{20}\}.$$

If the branch, $B(x_{17})$, has non-positive value then arc $a_{17,11}$ would be a strong m-arc, and the tree could not be normalized. It would contain a strong arc non-adjacent to the artificial root.

If the branch, $B(x_{17})$, has positive value then arc $a_{17,11}$ would be a weak m-arc, but the the set of strong nodes, S_T, cannot be the maximum valued closure in the tree. This follows since: $S_T \cup B(x_{17})$ is a closure in the tree; and, it's value is greater than the value of S_T.

Hence, it is not possible for the tree in Figure 3.2 to be a normalized tree. To convert it to a normalized tree, all nodes in the branch $B(x_{17})$ must be reclassified as strong nodes.

Review of the normalized trees shown in Figures 2.8-2.16, demonstrates that for any branch having a root node contained in the normalized tree's maximum valued closure, all nodes in the branch are in the normalized tree's maximum valued closure.

3.2.2 LG Property 2

LG Property 2 says: if S is the set of strong nodes in a normalized tree T, then the smallest maximum valued closure in the normalized tree is defined by S. The proof provided by H. Lerchs and I. F. Grossmann 1965 is:

> If we note that any p-branch of T is a closed set of T, but that an m-branch is not a closed set of T, this property follows directly from LG Property 1. If the tree has no strong nodes, that is, no strong arcs, then the maximum closure is the empty set.

Although we agree that the argument is correct, it is not clear that the conclusion follows. Therefore, we present a more descriptive proof of the property. The authors' term *closed set* is equivalent to our term closure.

Lemma 10 (LG Property 2) *If S_T is the set of strong nodes in a normalized tree T, then a maximum valued closure in the normalized tree is defined by S_T.*

Proof: Severing each artificial arc in T forms a collection of branches B. From Lemma 1, each branch in B must be a p-branch. From Lemma 5, each p-branch defines a closure in T.

Partition B into it's set of strong p-branches, B_S, and it's set of weak p-branches, B_W. By Definitions 16 and 17, we know for each $B_i \in B_S$, $\nu(B_i) > 0$ and for each $B_i \in B_w$, $\nu(B_i) \leq 0$. Since the nodes in each p-branch defines a closure in T, by Lemma 7, the union of the nodes in the branches in B_S define a closure in T. In addition, since each branch in B_W has non-positive value, B_S is a maximum valued closure in T.

Since B_S is a maximum valued closure in T, by Lemma 9, every node in B_S is in the maximum valued closure of T, and every node in B_W is *not* in the maximum valued closure of T. Thus, $S_T = B_S$. Hence, S_T is a maximum valued closure in T. □

To demonstrate consider the normalized tree, T_2, in Figure 2.9. The set of strong p-branches is:

$$B_S = \{B(x_6), B(x_7), B(x_8)\} = \{x_6, x_7, x_8, x_9\},$$

each having positive value. The set of weak p-branches is:

$$B_W = \{B(x_1), B(x_2), B(x_3), B(x_4), B(x_5)\},$$

each having non-positive value. In addition, every node in B_S is a strong node, and every node in B_W is a weak node.

The normalized tree's maximum valued closure is defined by the set of nodes $S = \{x_6, x_7, x_8, x_9\}$, which equals the nodes contained in the set of branches B_S. Thus, in accordance with LG Property 2, the set of strong nodes defines a maximum valued closure in the normalized tree.

3.2.3 LG Theorem I

Theorem 1 (LG Theorem I) *Let $G = (X, A)$ be a directed graph and S be it's maximum valued closure. Also, let T be a normalized tree defined in G's augmented graph, and S_T be it's maximum valued closure. If a normalized tree T can be constructed such that S_T defines a closure in G, then $S_T = S$.*

Proof: By Lemma 6, we know

$$\nu(S_T) \geq \nu(S). \tag{3.3}$$

If S_T defines a closure in G, we know from Equation 3.3, that the value of S_T is greater than any other closure in G. Thus, $S_T = S$. If the strong node set S_T is empty, then $S_T = S = \emptyset$. □

3.3 LG Theorem II

LG Theorem II says: in following the steps of the LG Algorithm, the set of strong nodes in the normalized tree converges to the maximum valued closure of the directed graph G in a finite number of iterations. Thus, the theorem proves the LG Algorithm requires a finite number of steps to construct a normalized tree, where the set of strong nodes defines a closure in G. Then by LG Theorem I, we know the set of strong nodes in the normalized tree defines a maximum valued closure in G.

The proof of LG Theorem II uses LG Property 3. Before LG Property 3 can be discussed, we must investigate the detailed operations of the LG Algorithm's *MoveTowardFeasibility* and *NormalizeTree* procedures. Recall that the *MoveToward-Feasibility* procedure replaces an arc in the current normalized tree, T_i, with an arc not contained in T_i. Forming a new tree $\hat{T}_{i+1}$, which may or may not be a normalized tree.

If $\hat{T}_{i+1}$ is a non-normalized tree, the *NormalizeTree* procedure performs additional transformations which form the normalized tree T_{i+1}. Each transformation performed by the *NormalizeTree* procedure replaces an arc in $\hat{T}_{i+1}$ with an artificial arc.

To be more specific, in the *MoveTowardFeasibility* procedure, the introduced arc is an arc with a strong source node, x_s, and a weak terminal node x_w. The arc is denoted as $a_{s,w}$. In the tree T_i, the root of the branch containing x_s is denoted as node x_{r_s}, and the root of the branch containing x_w is denoted as x_{r_w}. These nodes

are called the root of the strong branch and the root of the weak branch respectively. The arc which is removed from T_i is the artificial arc having x_0 as it's source node and x_{r_s} as it's terminal node. Thus, in the tree $\hat{T}_{i+1}$, the following chain of edges exists:

$$[x_{r_s}, \ldots, x_s, x_w, \ldots, x_{r_w}, x_0].$$

If the tree $\hat{T}_{i+1}$ is a non-normalized tree, the *NormalizeTree* procedure replaces strong arc's in the chain (Chain 3.3) with artificial arcs. The *NormalizeTree* procedure searches the chain, starting with node x_{r_s}, until it encounters a strong arc. Let a_{p_s,p_t} denote a strong arc in the chain. The transformation replaces arc a_{p_s,p_t} with the artificial arc a_{0,p_t}. The *NormalizeTree* procedure then continues searching, replacing each strong arc encountered.

In this section, we develop relationships between the value of the m-branches and p-branches, formed by severing the arc's in the chain of edges in Chain 3.3. These relationships are used to prove LG Property 3, which says: in Chain 3.3, each m-branch has positive value, and it's value is greater than any p-branch which precedes it in the chain. This property is used to prove the *MoveTowardFeasibility* procedure does not create any strong m-arcs. It is also used to prove the *NormalizeTree* procedure does not create any strong m-arcs. Thus, the only strong arcs formed by these procedures are p-arcs.

3.3.1 Arc Properties and the *MoveTowardFeasibility* Procedure

In this section, we investigate changes in the value of the branches, which have as their branch roots, the nodes in the Chain 3.3. We identify how the value of these branches relate to the branch $B(x_{r_s})$. Since branch $B(x_{r_s})$ is a strong branch, by Lemma 2, $\nu(B(x_{r_s})) > 0$.

The example presented in the previous chapter will be used to examine the results of this section. Figures 3.3 and 3.4 repeat trees T_4 and $\hat{T}_5$, which were previously shown in Figures 2.11 and 2.12. Tree T_4 corresponds to the normalized tree T_i. The non-normalized tree $\hat{T}_5$ corresponds to the non-normalized tree $\hat{T}_{i+1}$, which is generated by the *MoveTowardFeasibility* procedure. The generalized notation of some nodes has been added to the right of the respective node in Figures 3.3 and 3.4.

To reacquaint the reader, tree $\hat{T}_5$ is formed by replacing arc $a_{0,4}$, in tree T_4, with $a_{8,5}$. The correspondences between nodes in the example and the generalized notation are: the weak node x_w is node x_5, the strong node x_s is node x_8, the root of the strong node branch x_{r_s} is node x_4, and the root of the weak node branch x_{r_w} is node x_5. The chain of edges between node x_{r_s} and the artificial root x_0 in the tree $\hat{T}_{i+1}$ is defined by the sequence of nodes $[x_4, x_8, x_5, x_0]$.

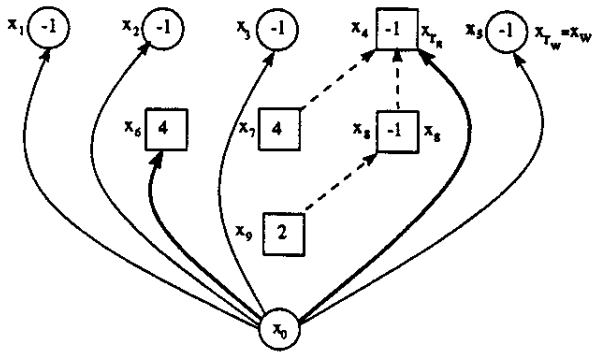


Figure 3.3: The Normalized Tree $T_4 = T_i$

To develop the required relationships we partition Chain 3.3 into the following three sub-chains:

$$SC_1 = [x_{r_s}, \ldots, x_s], \tag{3.4}$$

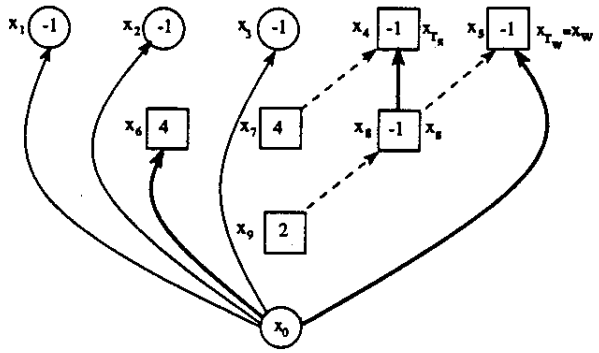


Figure 3.4: The Non-Normalized Tree $\hat{T}_5 = \hat{T}_{i+1}$

$$SC_2 = [x_s, x_w], \tag{3.5}$$

$$SC_3 = [x_w, \ldots, x_{r_w}, x_0]. \tag{3.6}$$

Sometimes branches in trees T_i and $\hat{T}_{i+1}$ are referred to within an individual equation. Thus, our notation must distinquish which tree the branch is in. Therefore, let $B_i(x)$ denote a branch in tree T_i and $\hat{B}_{i+1}(x)$ denote a branch in the tree $\hat{T}_{i+1}$.

Arcs In The Sub-Chain $[x_{r_s}, \ldots, x_s]$

Consider sub-chain SC_1, which contains the set of arcs between x_{r_s} and x_s. In the tree T_i, the nodes in SC_1 were contained in the branch $B_i(x_{r_s}$. The branch containing the strong node x_s. Thus, for each node in $B_i(x_{r_s})$, the chain to the artificial root, x_0, went through the branch's root node, x_{r_s}. In addition, since $B_i(x_{r_s})$ is a strong p-branch

$$\nu(B_i(x_{r_s})) > 0. \tag{3.7}$$

In forming $\hat{T}_{i+1}$, arc a_{0,r_s} was replaced with arc $a_{s,w}$. As a result, the directional

classification of each arc in SC_1 changes: a p-arc in T_i becomes a m-arc in $\hat{T}_{i+1}$; and, a m-arc in T_i becomes a p-arc in $\hat{T}_{i+1}$.

First consider some edge $e_{p,q}$ in SC_1. This edge was contained in the branch $B_i(x_{r_s})$ in T_i. Forming the chain of edges from x_s to x_0 we have:

$$[x_s, \ldots, x_p, x_q, \ldots, x_{r_s}, x_0]. \tag{3.8}$$

Now consider the same edge in the tree $\hat{T}_{i+1}$. Forming the chain of edges from x_{r_s} to x_0 we have:

$$[x_{r_s}, \ldots, x_q, x_p, \ldots, x_s, x_w, \ldots, x_{r_w}, x_0]. \tag{3.9}$$

Severing the edge in Chain 3.8, forms a branch having node x_p as it's root. Severing the edge in Chain 3.9, forms a branch having node x_q as it's root.

Now assume edge $e_{p,q}$ corresponds to a p-arc in T_i. Since it is a p-arc, from Chain 3.8, we find x_q must be the arc's source node and x_p must be it's terminal node. Now consider the same edge in the tree $\hat{T}_{i+1}$. From Chain 3.9 we see, severing the arc makes node x_q the root of the branch. Thus, the arc's source node is in the branch. Therefore, by Definition 15, the arc must be m-arc in $\hat{T}_{i+1}$. Assuming edge $e_{p,q}$ corresponds to a m-arc in T_i, and following the same steps, demonstrates it will become a p-arc in $\hat{T}_{i+1}$.

In addition to changing the directional classification of arcs in SC_1, *MoveTowardFeasibility* transformation may cause the value of the branches supported by these arcs to change. As demonstrated above, severing edge $e_{p,q}$ in $\hat{T}_{i+1}$ forms the branch $\hat{B}_{i+1}(x_q)$. Severing the same edge in T_i forms the branch $B_i(x_p)$. Using Chains 3.8 and 3.9, we find that, in $\hat{T}_{i+1}$, the branch $\hat{B}_{i+1}(x_q)$ consists of the chain of edges

$[x_{r_s}, \ldots, x_q]$. Thus, it's value is given by:

$$\nu(\hat{B}_{i+1}(x_q)) = \nu(B_i(x_{r_s})) - \nu(B_i(x_p)). \tag{3.10}$$

To demonstrate the above results, consider Figures 3.3 and 3.4. Chain 3.8 is

$$[x_8, x_4, x_0], \tag{3.11}$$

and Chain 3.9 is

$$[x_4, x_8, x_5, x_0]. \tag{3.12}$$

Subchain SC_1 contains a single edge, and is given by the following sequence of nodes $[x_4, x_8]$. This edge corresponds to the arc $a_{8,4}$. Thus, $x_p = x_8$ and $x_q = x_4$. In T_4 the arc is a m-arc, in $\hat{T}_5$ it is a p-arc.

Computing the terms in Equation 3.10, we have:

$$\begin{aligned} \nu(B_i(x_{r_s})) &= \nu(B_4(x_4)), \\ &= \nu(\{x_4, x_7, x_8, x_9\}), \\ &= 4, \end{aligned} \tag{3.13}$$

and

$$\begin{aligned} \nu(B_i(x_p)) &= \nu(B_4(x_8)), \\ &= \nu(\{x_8, x_9\}), \end{aligned} \tag{3.14}$$

$$= 1.$$

Combining Equations 3.14 and 3.15, we have

$$\begin{aligned} \nu(\hat{B}_{i+1}(x_q)) &= \nu(B_i(x_{r_s})) - \nu(B_i(x_p)), \\ \nu(\hat{B}_5(x_4)) &= \nu(B_4(x_4)) - \nu(B_4(x_8)), \\ &= 4 - 1, \\ &= 3. \end{aligned}$$

Inspection of Figure 3.4 indicates, $\hat{B}_5(x_4) = \{x_4, x_7\}$ has a value of 3. Thus, demonstrating the computation of Equation 3.10.

Now we wish to relate the value of branches, formed by severing an arc in SC_1, with respect to the branch $B_i(x_{r_s})$. To do this, we shall investigate p-branches and m-branches separately.

Consider a p-arc a_{p_s,p_t} in T_i. Severing the arc forms the branch $B_i(x_{p_t})$. Since T_i is a normalized tree, we know $\nu(B_i(x_{p_t})) \leq 0$. Combining this with Equations 3.7 and 3.10 we find:

$$\begin{aligned} \nu(\hat{B}_{i+1}(x_{p_s})) &= \nu(B_i(x_{r_s})) - \nu(B_i(x_{p_t})), \\ &\geq \nu(B_i(x_{r_s})). \end{aligned} \tag{3.15}$$

Since a_{p_s,p_t} is a p-arc in T_i it will become a m-arc in $\hat{T}_{i+1}$. Thus, we can say, in $\hat{T}_{i+1}$: each m-branch, formed by severing an arc in SC_1, has value greater than or equal to the value of the strong branch in T_i. Unfortunately, the example problem does not

contain such an arc.

In Equation 3.16, we denoted the p-arc in T_i as a_{p_s,p_t}. In $\hat{T}_{i+1}$, the p-arc is an m-arc. To make the relation reflect the arc's directional classification in $\hat{T}_{i+1}$, we shall denote the m-arc as a_{m_s,m_t}. In $\hat{T}_{i+1}$, the m-arc's source node, x_{m_s}, corresponds to the p-arc's source node x_{p_s}. Thus, we can express the relation in Equation 3.16 as:

$$\nu(\hat{B}_{i+1}(x_{m_s})) \geq \nu(B_i(x_{r_s})). \tag{3.16}$$

Now consider a m-arc a_{m_s,m_t} in T_i. Severing the arc forms the branch $B_i(x_{m_s})$. Since T_i is a normalized tree, we know $\nu(B_i(x_{m_s})) > 0$. Combining this with Equations 3.7 and 3.10 we find:

$$\begin{aligned} \nu(\hat{B}_{i+1}(x_{m_t})) &= \nu(B_i(x_{r_s})) - \nu(B_i(x_{m_s})), \\ &< \nu(B_i(x_{r_s})). \end{aligned} \tag{3.17}$$

Since a_{m_s,m_t} is a m-arc in T_i it will become a p-arc in $\hat{T}_{i+1}$. Thus, we can say, in $\hat{T}_{i+1}$: each p-branch, formed by severing an arc in SC_1, has value less than the value of the strong branch in T_i.

Again, the relation in Equation 3.18 is not in the desired form, since the m-arc in T_i is a p-arc in $\hat{T}_{i+1}$. To make the relation reflect the arc's directional classification in $\hat{T}_{i+1}$, we shall denote the p-arc as a_{p_s,p_t}. In $\hat{T}_{i+1}$, the p-arc's terminal node, x_{p_t}, corresponds to the m-arc's terminal node x_{m_t}. Thus, we can express the relation in Equation 3.18 as:

$$\nu(\hat{B}_{i+1}(x_{p_t})) < \nu(B_i(x_{r_s})). \tag{3.18}$$

Fortunately, the example problem does contain such an arc, $a_{8,4} = (x_8, x_4) = (x_{m_s}, x_{m_t})$. Since the arc is a m-arc in T_4, it will be a p-arc in $\hat{T}_5$. Thus, the p-branch $\hat{B}_{i+1}(x_{m_t}) = \hat{B}_5(x_4) = \{x_4, x_7\}$ has a value of 3, which is less than the value of the strong branch in T_4, $B_i(x_{r_s}) = B_4(x_4) = \{x_4, x_7, x_8, x_9\} = 4$.

The Arc $a_{s,w}$

Sub-chain SC_2 contains the sole edge $e_{s,w}$, which corresponds to arc $a_{s,w}$. Arc $a_{s,w}$ was not part of T_i. It was the arc introduced by the *MoveTowardFeasibility* procedure. In $\hat{T}_{i+1}$ the arc will be a m-arc, and it obviously supports branch $B_i(x_{r_s})$. Thus, we have the following equality in terms of value:

$$\nu(\hat{B}_{i+1}(x_s)) = \nu(B_i(x_{r_s})). \tag{3.19}$$

Thus, we can say, in $\hat{T}_{i+1}$: the sole m-branch, formed by severing an arc in SC_2, has value equal to the value of the strong branch in T_i. In the example, $a_{s,w} = a_{8,5}$ has the following expected properties: it did not exist in T_4, it is a m-arc, and it supports a branch with value, $\nu(\hat{B}_5(x_8)) = \nu(\{x_4, x_7, x_8, x_9\}) = 4$, equal to the strong branch in T_4.

Again, since we wish the relation to reflect the arc's directional classification in $\hat{T}_{i+1}$, we shall denote the m-arc as a_{m_s,m_t}. The m-arc's source node, x_{m_s} corresponds to the node x_s. Thus, relation 3.19 can be stated as:

$$\nu(\hat{B}_{i+1}(x_{m_s})) \;=\; \nu(B_i(x_{r_s})). \tag{3.20}$$

Arcs In The Sub-Chain $[x_w, \ldots, x_{r_w}, x_0]$

Consider subchain SC_3, which contains the set of arcs between x_w and x_0. In T_i, the nodes in SC_3 were contained in the branch $B_i(x_{r_w})$. The branch containing the weak node x_w. Thus, for each node in $B_i(x_{r_w})$, the chain to the artificial root, x_0, went through the branch's root node x_{r_w}. In addition, since $B_i(x_{r_w})$ is a weak p-branch

$$\nu(B_i(x_{r_w})) \leq 0. \tag{3.21}$$

The *MoveTowardFeasibility* procedure adds arc $a_{s,w}$, which connects node x_s to node x_w. Thus, forming the branch $\hat{B}_{i+1}(x_s)$ under the node x_w. Thus, it is obvious that, in $\hat{T}_{i+1}$, arcs in SC_3 will retain their directional classifications: p-arcs in T_i will remain p-arcs in $\hat{T}_{i+1}$; and, m-arcs in T_i will remain m-arcs in $\hat{T}_{i+1}$.

Since $B_i(x_{r_s})$ is moved under x_w, its value is added to the value of each branch supported by arcs in SC_3. Since $\nu(B_i(x_{r_s})) > 0$, the value of each branch supported by an arc in SC_3 will increase. Consider any edge, $e_{p,q}$, in SC_3. In T_i we can form the chain from x_w to x_0:

$$[x_w, \ldots, x_p, x_q, \ldots, x_{r_w}, x_0]. \tag{3.22}$$

In $\hat{T}_{i+1}$ we can form the chain from x_{r_s} to x_0:

$$[x_{r_s}, \ldots, x_s, x_w, \ldots, x_p, x_q, \ldots, x_{r_w}, x_0]. \tag{3.23}$$

Thus, for any edge $e_{p,q}$, we have the following value of the branch $\hat{B}_{i+1}(x_p)$:

$$\nu(\hat{B}_{i+1}(x_p)) = \nu(B_i(x_{r_s})) + \nu(B_i(x_p)). \tag{3.24}$$

In the example, the only edge in this sub-chain is edge $e_{5,0}$, which corresponds to arc $a_{0,5}$. Thus, $x_p = x_5$ and $x_q = x_0$. In T_4, we have

$$\begin{aligned} \nu(B_i(x_p)) &= \nu(B_4(x_5)), \\ &= \nu(\{x_5\}), \\ &= -1, \end{aligned} \tag{3.25}$$

and

$$\begin{aligned} \nu(B_i(x_{r_s})) &= \nu(B_4(x_4)), \\ &= \nu(\{x_4, x_7, x_8, x_9\}), \\ &= 4. \end{aligned} \tag{3.26}$$

In $\hat{T}_5$, we have

$$\begin{aligned} \nu(\hat{B}_{i+1}(x_p)) &= \nu(\hat{B}_5(x_5)), \\ &= \nu(\{x_4, x_5, x_7, x_8, x_9\}), \\ &= 3. \end{aligned} \tag{3.27}$$

Thus, demonstrating the computation of Equation 3.24.

Now we wish to relate the value of branches, formed by severing an arc in SC_3, with respect to $B_i(x_{r_s})$. To do this, we shall investigate p-branches and m-branches separately.

Consider a p-arc a_{p_s,p_t} in T_i. Severing the arc forms the branch $B_i(x_{p_t})$. Since T_i is a normalized tree, we know $\nu(B_i(x_{p_t})) \leq 0$. Combining this fact with Equations 3.7 and 3.24 we find:

$$\begin{aligned} \nu(\hat{B}_{i+1}(x_{p_t})) &= \nu(B_i(x_{r_s})) + \nu(B_i(x_{p_t})), \\ &\leq \nu(B_i(x_{r_s})). \end{aligned} \tag{3.28}$$

Thus, we can say, in $\hat{T}_{i+1}$: each p-branch, formed by severing an arc in SC_3, has value less than or equal to the value of the strong branch in T_i. In the example, the arc $a_{0,5}$ is a p-arc in tree T_4 and remains a p-arc in tree $\hat{T}_5$. The value of branch $\hat{B}_{i+1}(x_p)$ is $\nu(\hat{B}_{i+1}(x_5)) = \nu(\{x_4, x_5, x_7, x_8, x_9\}) = 3$, which is less than the value of the strong branch.

Consider a m-arc a_{m_s,m_t} in the normalized tree T_i. Severing the arc forms the branch $B_i(x_{m_s})$. Since T_i is a normalized tree, we know $\nu(B_i(x_{m_s})) > 0$. Combining this fact with Equations 3.7 and 3.24 we find:

$$\begin{aligned} \nu(\hat{B}_{i+1}(x_{m_s})) &= \nu(B_i(x_{r_s})) + \nu(B_i(x_{m_s})), \\ &> \nu(B_i(x_{r_s})). \end{aligned} \tag{3.29}$$

Thus, we can say, in $\hat{T}_{i+1}$: each m-branch, formed by severing an arc in SC_3, has value less than the strong branch in T_i. In the example, no m-arcs exist in SC_3.

3.3.2 LG Property 3

LG Property 3 asserts that all m-arcs in the chain of edges defined by the sequence of nodes, $[x_{r_s}, \ldots, x_s, x_w, \ldots, x_{r_w}, x_0]$, in the tree $\hat{T}_{i+1}$, are weak; and, each such m-arc supports a branch with value greater than the branch supported by any preceding p-arc in the chain.

Lemma 11 (LG Property 3) *Let* $[x_{r_s}, \ldots, x_s, x_w, \ldots, x_{r_w}, x_0]$ *denote the chain of edges in* $\hat{T}_{i+1}$*, the tree formed by the* MoveTowardFeasibility *procedure. Consider any m-branch formed by severing a m-arc* a_{m_s,m_t} *in the chain, and any p-branch formed by severing p-arc* a_{p_s,p_t}*, which precedes* a_{m_s,m_t} *in the chain. Then,*

$$\nu(\hat{B}_{i+1}(x_{m_s})) \; > 0, \tag{3.30}$$

and

$$\nu(\hat{B}_{i+1}(x_{m_s})) \; > \; \nu(\hat{B}_{i+1}(x_{p_t})), \tag{3.31}$$

where $\hat{B}_{i+1}(x)$ *denotes a branch in* $\hat{T}_{i+1}$.

Proof: Let a_{m_s,m_t} denote a m-arc in the chain, and $\hat{B}_{i+1}(x_{m_s})$ be the branch formed in $\hat{T}_{i+1}$, when the arc is severed. By Equations 3.16, 3.20, and 3.30, we find

$$\nu(\hat{B}_{i+1}(x_{m_s})) \; \geq \; \nu(B_i(x_{r_s})). \tag{3.32}$$

Since, $B_i(x_{r_s})$ is a strong p-branch, $\nu(B_i(x_{r_s})) > 0$. Thus,

$$\nu(\hat{B}_{i+1}(x_{m_s})) \; \geq \; 0. \tag{3.33}$$

To prove the relation in Equation 3.31, partition the chain into the subchains: $SC_1 = [x_{r_s}, \ldots, x_s]$, $SC_2 = [x_s, x_w]$, and $SC_3 = [x_w, \ldots, x_{r_w}, x_0]$.

Consider a m-arc and a preceding p-arc on the subchain SC_1. By Equations 3.16 and 3.18 we know,

$$\nu(\hat{B}_{i+1}(x_{m_s})) \geq \nu(B_i(x_{r_s})), \tag{3.34}$$

and

$$\nu(\hat{B}_{i+1}(x_{p_t})) < \nu(B_i(x_{r_s})), \tag{3.35}$$

respectively. Since the right hand sides of Equations 3.34 and 3.35 are equal, the left hand sides have the following relation:

$$\nu(\hat{B}_{i+1}(x_{m_s})) > \nu(\hat{B}_{i+1}(x_{p_t})). \tag{3.36}$$

Thus, the value of any m-branch is greater than the value of any preceding p-branch, where both branches are formed by severing an arc in SC_1.

Now, consider the m-arc in SC_2. By Equation 3.20 we know,

$$\nu(\hat{B}_{i+1}(x_{m_s})) = \nu(B_i(x_{r_s})). \tag{3.37}$$

The only p-arcs which precede this m-arc are those p-arcs in SC_1. The right hand sides of Equations 3.35 and 3.37 are equal. Thus, the left hand sides have the following relation.

$$\nu(\hat{B}_{i+1}(x_{m_s})) > \nu(\hat{B}_{i+1}(x_{p_t})). \tag{3.38}$$

Thus, the value of this m-branch is greater than any preceding p-branch.

Now, consider a m-arc and a p-arc on the subchain SC_3. By Equations 3.29 and 3.30 is,

$$\nu(\hat{B}_{i+1}(x_{p_t})) \leq \nu(B_i(x_{r_s})). \tag{3.39}$$

and

$$\nu(\hat{B}_{i+1}(x_{m_s})) > \nu(B_i(x_{r_s})). \tag{3.40}$$

Thus, the relation between the left hand sides of Equations 3.39 and 3.40, we have:

$$\nu(\hat{B}_{i+1}(x_{m_s})) > \nu(\hat{B}_{i+1}(x_{p_t})). \tag{3.41}$$

Hence, the value of any m-branch is greater than the value of any p-branch, where both branches are formed by severing an arc in SC_3. There are no p-branchs in SC_2. From Equation 3.34, the value of any p-branch formed by severing an arc in SC_1 is less than or equal to the value of the strong branch, $B_i(x_{r_s})$. From Equation 3.40, the value of any m-branch formed by severing an arc in SC_3 is greater than the value of the strong branch. Thus, the value of any m-branch formed by severing an arc in SC_3 is greater than the value of any p-branch preceding it in the branch.

Hence we have proven Equation 3.31 holds for any m-branch and any p-branch which precedes it in the chain. □

3.3.3 Arc Properties Resulting From the *NormalizeTree* Procedure

If the tree, $\hat{T}_{i+1}$, formed by the *MoveTowardFeasibility* procedure is a non-normalized tree, then the *NormalizeTree* procedure performs additional transformations to normalize the tree. These *normalizing transformations* replace a strong arc, which is non-adjacent to the artificial root, with an artificial arc.

Only branches, formed by severing an arc in the chain

$$[x_{r_s}, \ldots, x_s, x_w, \ldots, x_{r_w}, x_0],$$

can have a value change. Thus, all arcs not in the chain will retain the strength and directional classification they possessed in the normalized tree T_i. Thus, strong arcs which are non-adjacent to the artificial root must be an arc in the chain.

By Lemma 11, we know the *MoveTowardFeasibility* procedure cannot create a strong m-arc in the chain, since each m-branch, formed by severing an arc in the chain, will have positive value. Therefore, only strong p-arcs may be found in the chain.

The *NormalizeTree* procedure searches the chain, starting with node x_{r_s}, until it encounters a strong p-arc, a_{p_s,p_t}. The transformation replaces a_{p_s,p_t} with the artificial arc a_{0,p_t}. Let $\hat{B}_{i+1}(x_{p_t})$ denote the p-branch in the non-normalized tree $\hat{T}_{i+1}$, and let $B_{i+1}(x_{p_t})$ denote the p-branch in the normalized tree T_{i+1}. Since $\nu(\hat{B}_{i+1}(x_{p_t})) > 0$, we know $\nu(B_{i+1}(x_{p_t})) > 0$. Thus, in T_{i+1}, arc a_{0,p_t} is a strong p-arc. Hence all nodes in $B_{i+1}(x_{p_t})$ will be strong.

After each normalizing transformation, the *NormalizeTree* procedure continues searching, replacing each strong p-arc encountered. Although each normalizing transformation removes a positive valued p-twig from the branch $\hat{B}_{i+1}(x_{r_w})$, we know

the m-arcs which follow the p-arc will remain weak. This follows from Lemma 11, since the value of each m-branch is greater than the value of any preceding p-branch. But, it is possible for the normalizing transformations to cause $\hat{B}_{i+1}(x_{r_w})$ to have non-positive value, thus, making nodes which remain in the branch weak.

Lemma 12 *Of the strong p-arcs replaced in the* NormalizeTree *procedure, the last strong p-arc replaced supported the branch with greatest value in* $\hat{T}_{i+1}$.

Proof: Let $\{a_{s_1,t_1}, a_{s_2,t_2}, \ldots, a_{s_n,t_n}\}$ denote the set of strong p-arcs replaced in the normalizing transformations, where a_{s_1,t_1} is the first p-arc replaced, and a_{s_n,t_n} is the last. Let

$$T^1_{i+1}, T^2_{i+1}, \ldots, T^n_{i+1},$$

denote the sequence of trees formed *before* each normalizing transformation. Also, let

$$B^1_{i+1}(x), B^2_{i+1}(x), \ldots, B^n_{i+1}(x)$$

denote branches in the sequence of trees formed by the normalizing transformations.

In the tree $\hat{T}_{i+1}$, a_{s_n,t_n} supported a branch containing all nodes contained in the branches formed by the arcs in $\{a_{s_1,t_1}, \ldots, a_{s_{n-1},t_{n-1}}\}$. This can be expressed as:

$$\hat{B}_{i+1}(x_{t_n}) \;=\; \bigcup_{j=1}^{n} B^j_{i+1}(x_{t_j}). \tag{3.42}$$

In the tree T^{n-1}_{i+1},

$$\nu(B^n_{i+1}(x_{t_n})) \;>\; 0. \tag{3.43}$$

Otherwise, it would be a weak arc, and a normalizing transformation would not be

required.

Thus, taking the value of the non-intersecting node sets in 3.42, we have

$$\nu(\hat{B}_{i+1}(x_{t_n})) = \sum_{j=1}^{n} \nu(B^j_{i+1}(x_{t_j})), \tag{3.44}$$

$$= \nu(B^n_{i+1}(x_{t_n})) + \sum_{j=1}^{n} \nu(B^j_{i+1}(x_{t_j}). \tag{3.45}$$

Using Equation 3.43, we obtain the result

$$\nu(\hat{B}_{i+1}(x_{t_n})) > \sum_{j=1}^{n} \nu(B^j_{i+1}(x_{t_j})). \tag{3.46}$$

□

An example may help. Consider the non-normalized tree shown in Figure 3.5. The chain $[x_8, x_3, x_7, x_2, x_6, x_1, x_0]$ consists of the strong p-arcs $\{a_{7,3}, a_{6,2}, a_{0,1}\}$.

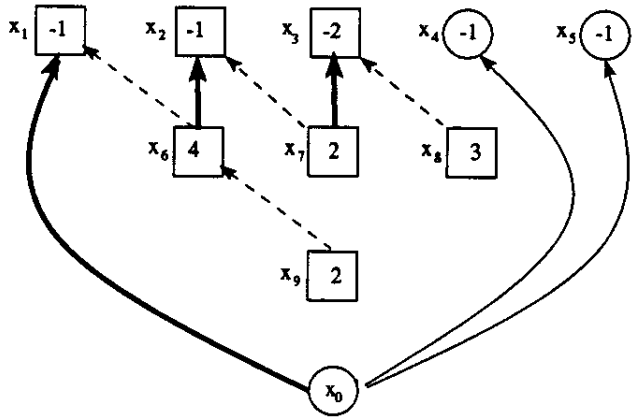


Figure 3.5: The Non-Normalized Tree T^1_{i+1} Before The First Normalization Transformation

The *NormalizeTree* procedure traverses the chain searching for the first strong arc. The first strong arc encountered is arc $a_{7,3}$. The normalizing transformation

replaces arc $a_{7,3}$ with arc $a_{0,3}$, which forms the tree, T^2_{i+1}, shown in Figure 3.6.

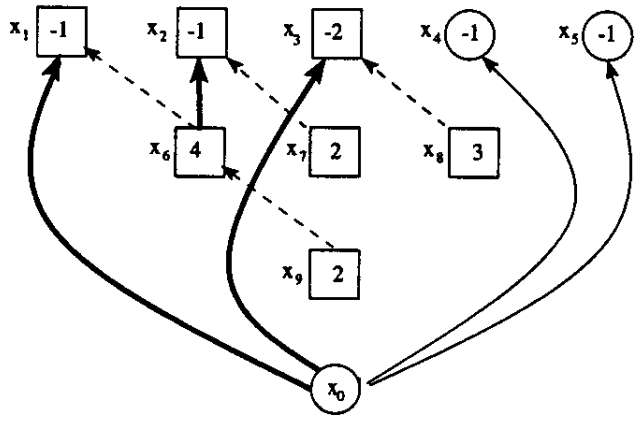


Figure 3.6: The Non-Normalized Tree T^2_{i+1} Before The Second Normalization Transformation

The twig supported by $a_{7,3}$, $B^1_{i+1}(x_3) = \{x_3, x_8\}$, has a value of 1. Thus, in T^1_{i+1}, nodes $\{x_3, x_8\}$ are classified as strong nodes. Removing the p-twig $\hat{B}_{i+1}(x_3)$ from the branch $\hat{B}_{i+1}(x_1)$ reduces the value of the branch by 1. Thus, decreasing the value of the branch supported by the strong p-arc $a_{6,2}$ to 1, and the decreasing the value of the branch supported by the p-arc $a_{0,1}$ to 5. Thus, these arcs remain strong.

The next strong arc encountered is the p-arc $a_{6,2}$. The normalization transformation replaces arc $a_{6,2}$ with arc $a_{0,2}$, forming the tree T^3_{i+1}, which is shown in Figure 3.7. As will be discussed, this tree also corresponds to the tree formed at the completion of the *NormalizeTree* procedure.

The twig supported by $a_{6,2}$, $B^2_{i+1}(x_2) = \{x_2, x_7\}$, has a value of one. Thus, in T^2_{i+1}, nodes $\{x_2, x_7\}$ are classified as strong nodes. Removing the p-twig reduces the value of the branch by 1. But, the value of the last remaining strong arc, $a_{0,1}$, remains positive. Thus, it is still classified as a strong arc.

The next strong arc encountered is the p-arc $a_{0,1}$. The normalization transfor-

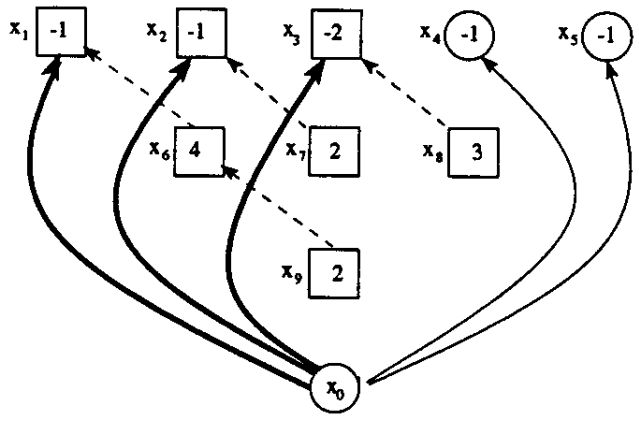


Figure 3.7: The Normalized Tree T_{i+1} After Completion Of The NormalizeTree Procedure

mation replaces arc $a_{0,1}$ with the same arc. Thus, this final transformation changes nothing. It is for this reason that, Figure 3.7, also displays the final normalized tree. Although, in the example, the final normalizing transformation was unnecessary, in the context of Lemma 12, the last strong p-arc would be considered arc $a_{0,1}$. As the Lemma claims, the last p-arc, $a_{0,1}$, supports a branch having greater value than any previously removed p-arc.

To demonstrate the three properties used in the Lemma's proof, consider Figures 3.5-3.7. Each strong p-arc replaced supported a positive valued branch when it was replaced. The strong p-arc $a_{7,3}$ supported the branch $B(x_3) = \{x_3, x_8\}$ which had a value of one. The strong p-arc $a_{6,2}$ supported the branch $B(x_2) = \{x_2, x_7\}$ which also had a value of one. The last strong p-arc $a_{0,1}$ supported the branch $B(x_1) = \{x_1, x_6, x_9\}$ which had a value of five.

The last strong p-arc originally supported a branch containing every previously replaced strong p-arc. The last strong p-arc replaced was arc $a_{0,1}$ which supported the branch $B(x_1) = \{x_1, x_2, x_3, x_6, x_7, x_9\}$. This branch contained each of the previously

replaced strong p-arcs $a_{7,3}$ and $a_{6,2}$.

The last strong p-arc supported a positive valued branch after the next to last strong p-arc was replaced. The value of the last strong p-arc $a_{0,1}$ was five after the next to last strong p-arc $a_{6,2}$ was replaced. It should be noted that if the last strong p-arc replaced is not an artificial arc then the nodes remaining in the branch after the last strong p-arc is replaced would become weak. The fact that the last strong p-arc supports the greatest valued branch in the chain is used in the proof of LG Theorem II.

Returning to the example presented in the previous chapter, we can follow the results of the normalization transformation. Figure 3.8 repeats the normalized tree T_5 shown in Figure 2.13. The normalized tree T_5 is obtained after the non-normalized tree $\hat{T}_5$, see Figure 3.4, has been normalized. The strong p-arc $a_{8,4}$ is the first strong arc encountered in the chain $[x_4, x_8, x_5, x_0]$. The arc is replaced by the arc $a_{0,4}$. As a result of the transformation, the arc $a_{8,4}$ is eliminated, thus preventing the allocation of the net positive value of nodes $Y_s = \{x_4, x_7\}$ to the set of nodes $Y_w = \{x_5, x_8, x_9\}$. The allocation should be prevented since the nodes in Y_s are independent of the nodes in Y_w.

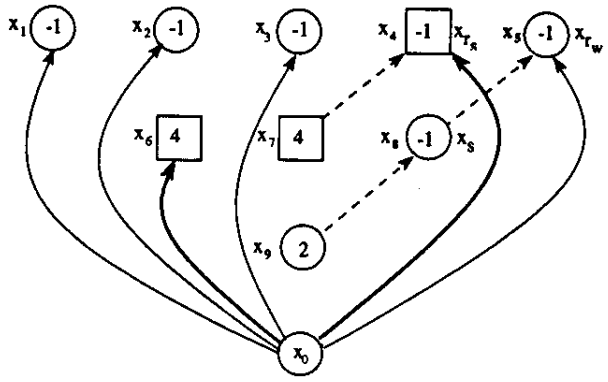


Figure 3.8: The Normalized Tree T_5

To summarize, the *NormalizeTree* procedure breaks away strong p-twigs replacing the p-arcs supporting the p-twigs with p-arcs which are adjacent with the artificial root x_0. Since the p-twigs have positive value the nodes in the p-twigs will remain strong. If the last strong p-arc replaced is not the artificial arc supporting the branch then the nodes which remain in the branch after the last strong p-arc is replaced will become weak, thus preventing the net positive value of the strong p-twigs removed from the branch from supporting the nodes remaining in the branch after the last strong p-arc is replaced.

3.3.4 LG Theorem II

LG Theorem II proves the LG Algorithm converges to its solution in a finite number of iterations. An outline of the theorem's proof is as follows: in a finite graph there exists a finite number of trees which can be formed from it; no two normalized trees, formed by the LG Algorithm, are equal; thus, the LG Algorithm will eventually form every necessary normalized tree.

Although the outline implies the LG Algorithm could enumerate every possible normalized tree, the proof demonstrates the following with respect to the strong node set in successive normalized trees: 1) either the value of the strong node set decreases; or, 2) the value of the strong node set remains constant but the size of the strong node set increases. Thus, the LG Algorithm will never form a normalized tree whose strong node set has a value greater than the value of the strong nodes in the current normalized tree.

We have not proven the LG Algorithm's rate of convergence. But, it is our hope that our development of the LG Algorithm provides a basis for such an analysis.

Before LG Theorem II is proven we supply the following lemma. The lemma

is based upon the operations of the *MoveTowardFeasibility* and *NormalizeTree* procedures. Therefore, it requires the following terminology.

Let G be a directed graph from which the following trees are generated: T_i, $\hat{T}_{i+1}$, and T_{i+1}. The tree T_i is a normalized tree. The tree $\hat{T}_{i+1}$ is the tree formed from the tree T_i, by the *MoveTowardFeasibility* procedure. The tree T_{i+1} is the normalized tree formed from the tree $\hat{T}_{i+1}$, by the *NormalizeTree* procedure. The *MoveToward-Feasibility* procedure replaces the arc a_{0,r_s}, in the tree T_i, with the arc $a_{s,w}$, forming the tree $\hat{T}_{i+1}$. Node x_{r_s} is the root of the branch containing the strong node x_s. The *NormalizeTree* procedure replaces all strong p-arcs, denoted as a_{p_s,p_t}, in the chain of edges $[x_{r_s}, \ldots, x_s, x_w, \ldots, x_{r_w}, x_0]$ in the tree $\hat{T}_{i+1}$, with arcs a_{0,p_t}, forming tree T_{i+1}.

Lemma 13 *If T_i, $\hat{T}_{i+1}$, and T_{i+1} are the trees described above, S_{T_i} is the set of strong nodes in T_i, and x_{p_t} is the terminal node of the last strong p-arc replaced by the* NormalizeTree *procedure, then the set of strong nodes in the normalized tree T_{i+1} is given by:*

$$S_{T_{i+1}} = (S_{T_i} \sim B_i(x_{r_s})) \cup \hat{B}_{i+1}(x_{p_t}). \tag{3.47}$$

Proof: Let $B(x)$, $\hat{B}_{i+1}(x)$ and $B_{i+1}(x)$ denote branches in the trees T_i, $\hat{T}_{i+1}$ and T_{i+1}, respectively.

In the transformations in forming normalized tree T_{i+1} from T_i, the only nodes which can have their strength classification changed are nodes in $\hat{B}_{i+1}(x_{r_w})$. Nodes in this branch can be partitioned into the two branches $B_i(x_{r_s})$ and $B_i(x_{r_w})$. In the tree T_i, all nodes in $B_i(x_{r_s})$ are strong, all nodes in $B_i(x_{r_w})$ are weak.

If x_{p_t} denotes the last strong p-arc replaced by the *NormalizeTree* procedure, then branch $\hat{B}_{i+1}(x_{p_t})$ contains the set of nodes in $\hat{B}_{i+1}(x_{r_w})$, which will be strong in

T_{i+1}.

The result follows from these properties. □

To demonstrate, start with S_{T_i}, the set of strong nodes in T_i. Remove $B_i(x_{r_s})$, the set of strong nodes which are contained in $B_i(x_{r_s})$; and, thus may be classified as weak in T_{i+1}. Then, add $\hat{B}_{i+1}(x_{p_t})$, the set of nodes in $B_i(x_{r_s})$ which are classified as strong nodes, giving the result.

Theorem 2 (LG Theorem II) *In following the steps of the LG Algorithm, the set of strong nodes in the normalized tree converges to the maximum valued closure of the directed graph G in a finite number of iterations.*

Proof: As the number of trees in a finite graph is finite, we only have to show that no normalized tree can repeat itself in the sequence $T_0, T_1, \ldots, T_n$. Each normalized tree is characterized by: its set of strong nodes S_{T_i}, and the value of the strong node set $\nu(S_{T_i})$. It will be shown that either: the value decreases during an iteration; or, the value remains constant, but the size of set S_T increases. Thus, any two normalized trees will differ either in their value or in their set of strong nodes.

In the normalized tree T_{i+1}, the strong node set is given by Equation 3.47. Taking the value of the sets in Equation 3.47, we obtain the value of the set of strong nodes in the normalized tree T_{i+1}:

$$\nu(S_{T_{i+1}}) = \nu(S_{T_i}) - \nu(B_i(x_{r_s})) + \nu(\hat{B}_{i+1}(x_{p_t})). \tag{3.48}$$

From Equations 3.18 and 3.29, we know

$$\nu(\hat{B}_{i+1}(x_{p_t})) \leq \nu(B_i(x_{r_s})). \tag{3.49}$$

Combining the relation in Equation 3.49 with 3.48 we find

$$\nu(S_{T_{i+1}}) < \nu(S_{T_i}), \tag{3.50}$$

or

$$\nu(S_{T_{i+1}}) = \nu(S_{T_i}); \tag{3.51}$$

Equation 3.50 holds when $\nu(\hat{B}_{i+1}(x_p)) < \nu(B_i(x_{r_s}))$, and Equation 3.51 holds when $\nu(\hat{B}_{i+1}(x_p)) = \nu(B_i(x_{r_s}))$.

Since Equation 3.18 is a strict inequality, $\nu(\hat{B}_{i+1}(x_p)) = \nu(B_i(x_{r_s}))$ can only occur when the strong p-arc a_{p_s,p_t} is in the sub-chain $[x_w, \ldots, x_{r_w}, x_0]$. The branch $B(x_{p_t})$, in this sub-chain must contain the branch $B(x_{r_s})$. Thus, $S_i \subset S_{i+1}$.

Thus, either: the value of the strong node set decreases; or, the value does not change, and the size of the strong node set increases. Hence, any two normalized trees must differ in value or in their strong node sets. □

Table 3.1 shows the strong node set S, its size $|S|$, and its value $\nu(S)$, for each normalized tree shown in Figures 2.8-2.16. As the table shows: the value of the strong node set either decreases; or, the value remains constant, but the size of the strong node set increases, which is in accordance with LG Theorem II.

Table 3.1: Value And Size Of Strong Node Set In Normalized Trees

Tree	S	$\nu(S)$	$\lvert S\rvert$
T_1	$\{x_6, x_7, x_9\}$	10	3
T_2	$\{x_6, x_7, x_8, x_9\}$	9	4
T_3	$\{x_6, x_7\}$	8	2
T_4	$\{x_4, x_6, x_7, x_8, x_9\}$	8	5
T_5	$\{x_4, x_6, x_7\}$	7	3
T_6	$\{x_1, x_4, x_6, x_7\}$	6	4
T_7	$\{x_1, x_2, x_4, x_6, x_7\}$	5	5
T_8	$\{x_1, x_2, x_3, x_4, x_6, x_7\}$	4	6

Chapter 4
THE LG ALGORITHM AND MULTIPLE MAXIMUM VALUED CLOSURES

Although the previous chapter proved the LG Algorithm converges to the maximum valued closure in a directed graph, it did not address the issue of multiple maximum valued closures. In this chapter, we prove the LG Algorithm converges to the smallest maximum valued closure in the graph. We also describe how the LG Algorithm can be modified to converge to the largest maximum valued closure in the graph.

In Section 4.1, an example of a directed graph with multiple maximum valued closures is presented. In Section 4.2, we prove the smallest and largest maximum valued closures are unique. In Section 4.3, we prove the LG Algorithm converges to the smallest maximum valued closure and that the LG Algorithm can easily be modified to converge to the largest maximum valued closure.

4.1 An Example

The directed graph in Figure 4.1 contains four maximum valued closures. The node values are shown inside the node symbols and the node designations are shown to the left of the node symbols. Figure 4.2 shows the smallest of the four maximum valued closure Z_1. Nodes in the maximum valued closure have their node symbol shadowed. Figures 4.3, 4.4, 4.5 show maximum valued closures Z_2, Z_3, and Z_4, respectively.

The four maximum valued closures have a net value of four and consist of the

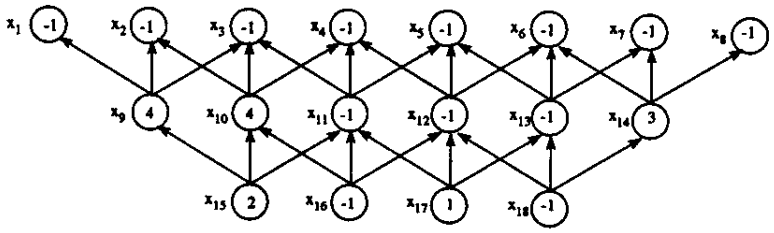


Figure 4.1: A Graph With Multiple Maximum Valued Closures

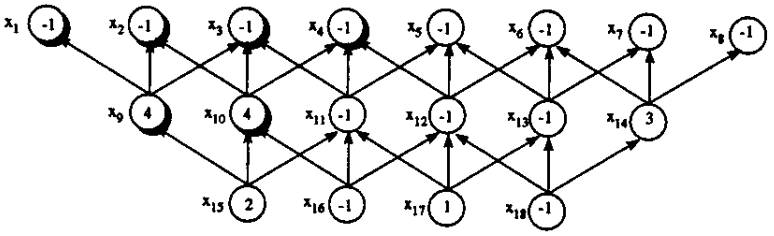


Figure 4.2: Maximum Valued Closure Z_1

following node sets:

$$Z_1 = \{x_1, x_2, x_3, x_4, x_9, x_{10}\},$$

$$Z_2 = \{x_1, x_2, x_3, x_4, x_5, x_9, x_{10}, x_{11}, x_{15}\},$$

$$Z_3 = \{x_1, x_2, x_3, x_4, x_6, x_7, x_8, x_9, x_{10}, x_{14}\},$$

$$Z_4 = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{14}, x_{15}\}.$$

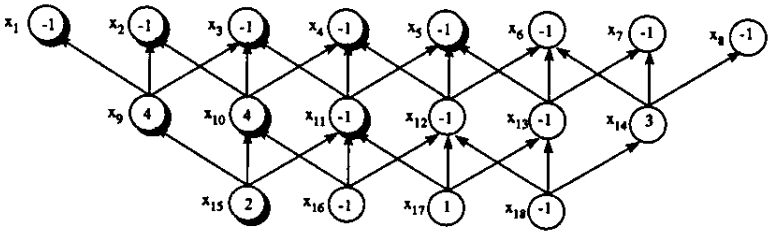


Figure 4.3: Maximum Valued Closure Z_2

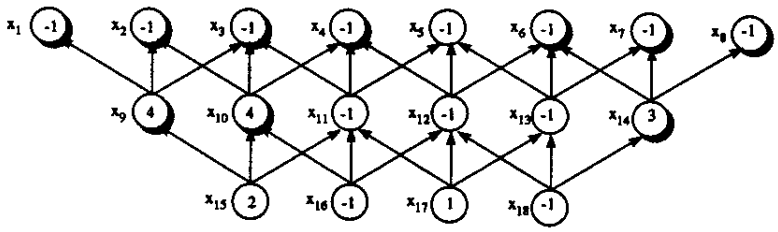


Figure 4.4: Maximum Valued Closure Z_3

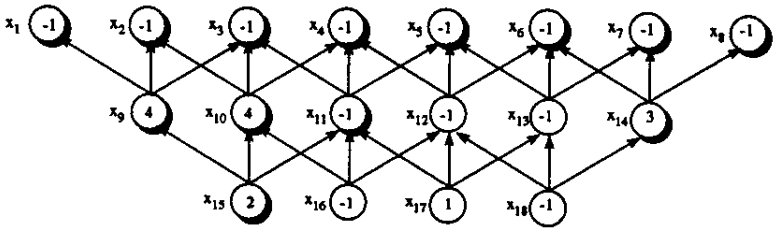


Figure 4.5: Maximum Valued Closure Z_4

Notice the following properties, which are proven in the next section: the smallest maximum valued closure Z_1 is a subset of each of the other maximum valued closures; the largest maximum valued closure Z_4 contains every other maximum valued closure; and, the value of the difference between closure sets $Z_i \sim Z_j,\ i \neq j$ equals zero. For example, the difference between closure sets Z_1 and Z_2 is $Z_2 \sim Z_1 = \{x_5, x_{11}, x_{15}\}$, which has value zero.

4.2 Properties of Multiple Maximum Valued Closures

Let $Z^* = \{Z_1, Z_2, \ldots, Z_n\}$ denote the collection of all *maximum* valued closures in the directed graph $G = (X, A)$. Thus, for any closure $Z_i \in Z^*$ we know

$$\nu(Z_i) \;=\; \nu^*, \tag{4.1}$$

and, for any closure $\Gamma(Y)$ in G,

$$\nu(\Gamma(Y)) \leq \nu^*. \tag{4.2}$$

Consider any two sets Z_i and Z_j in Z^* where $i \neq j$. Partition the set Z_i into the sets $Z_i \cap Z_j$ and $Z_i \sim Z_j$, where $(Z_i \cap Z_j) \cap (Z_i \sim Z_j) = \emptyset$ and $Z_i = (Z_i \cap Z_j) \cup (Z_i \sim Z_j)$. Thus, the set Z_i can be expressed as

$$Z_i = (Z_i \cap Z_j) \cup (Z_i \sim Z_j), \tag{4.3}$$

the size of Z_i can be expressed as

$$|Z_i| = |Z_i \cap Z_j| + |Z_i \sim Z_j|, \tag{4.4}$$

and, the value of Z_i can be expressed as

$$\nu(Z_i) = \nu(Z_i \cap Z_j) + \nu(Z_i \sim Z_j), \tag{4.5}$$

where, from Equation 4.1, we know $\nu(Z_i) = \nu^*$.

Lemma 14 *For any two sets Z_i and Z_j in Z^*, where $i \neq j$, $Z_i \cap Z_j \neq \emptyset$.*

Proof: Assume $Z_i \cap Z_j = \emptyset$. The value of their union is

$$\nu(Z_i \cup Z_j) = \nu(Z_i) + \nu(Z_j), \tag{4.6}$$

$$= \nu^* + \nu^*, \tag{4.7}$$

$$= 2\nu^*. \tag{4.8}$$

Thus, $\nu(Z_i \cup Z_j) > \nu^*$, which contradicts Equation 4.2. Thus, $Z_i \cap Z_j \neq \emptyset$. □

Lemma 15 *For any two sets Z_i and Z_j in Z^*, where $i \neq j$, $\nu(Z_i \sim Z_j) = \nu(Z_j \sim Z_i)$.*

Proof: Select any two sets Z_i and Z_j in Z^*. From Equation 4.5 we know

$$\nu(Z_i) = \nu(Z_i \cap Z_j) + \nu(Z_i \sim Z_j), \tag{4.9}$$

and

$$\nu(Z_j) = \nu(Z_i \cap Z_j) + \nu(Z_j \sim Z_i). \tag{4.10}$$

From Equation 4.1, we know $\nu(Z_i) = \nu(Z_j) = \nu^*$. Hence, the right hand sides of Equation 4.9 and Equation 4.10 are equal. Equating the right hand sides and subtracting $\nu(Z_i \cap Z_j)$ from both sides we obtain the result $\nu(Z_i \sim Z_j) = \nu(Z_j \sim Z_i)$. □

Lemma 16 *For any two sets Z_i and Z_j in Z^*, where $i \neq j$, $\nu(Z_i \sim Z_j) = \nu(Z_j \sim Z_i) = 0$.*

Proof: From Lemma 15 we know $\nu(Z_i \sim Z_j) = \nu(Z_j \sim Z_i)$. Thus, it will only be shown that $\nu(Z_i \sim Z_j) = 0$. Select any two sets Z_i and Z_j in Z^*.

Assume $\nu(Z_i \sim Z_j) < 0$. From Equations 4.5 and 4.1, we Know

$$\nu(Z_i) = \nu(Z_i \cap Z_j) + \nu(Z_i \sim Z_j), \tag{4.11}$$

and

$$\nu(Z_i) = \nu^*. \tag{4.12}$$

Equating right hand sides and solving for $\nu(Z_i \cap Z_j)$, we obtain

$$\nu(Z_i \cap Z_j) = \nu^* - \nu(Z_i \sim Z_j). \tag{4.13}$$

By the assumption $\nu(Z_i \sim Z_j) < 0$; hence, $\nu(Z_i \cap Z_j) > \nu^*$. By Lemma 14, the set $Z_i \cap Z_j$ is a closure in G. But, this is not possible since it contradicts Equation 4.2. Thus, we know $\nu(Z_i \sim Z_j) \not< 0$.

Now we show $Z_i \sim Z_j$ cannot have positive value either. Assume $\nu(Z_i \sim Z_j) > 0$. Partition the sets Z_i and Z_j into the following three sets: $Z_i \cap Z_j$, $Z_i \sim Z_j$ and $Z_j \sim Z_i$. Thus,

$$(Z_i \cup Z_j) = (Z_i \cap Z_j) \cup (Z_i \sim Z_j) \cup (Z_j \sim Z_i), \tag{4.14}$$

and the intersection of any pair of the three sets is empty. Thus, the value of the union of Z_i and Z_j is

$$\nu(Z_i \cup Z_j) = \nu(Z_i \sim Z_j) + \nu(Z_j \sim Z_i) + \nu(Z_i \cap Z_j), \tag{4.15}$$

By the assumption that $\nu(Z_i \sim Z_j) > 0$, we can drop $\nu(Z_i \sim Z_j)$ from the right hand side of the equation and obtain the relation

$$\nu(Z_i \cup Z_j) > \nu(Z_j \sim Z_i) + \nu(Z_i \cap Z_j). \tag{4.16}$$

From Equation 4.5 we know the right hand side of Equation 4.16 equals $\nu(Z_i)$. Thus, replacing the right hand side with $\nu(Z_i)$ we obtain

$$\nu(Z_i \cup Z_j) > \nu(Z_i). \tag{4.17}$$

By Equation 4.1, we know $\nu(Z_i) = \nu^*$. By Lemma 7, we know $Z_i \cup Z_j$ is a closure in G. Thus, contradicting Equation 4.2. Thus, $\nu(Z_i \sim Z_j) \not> 0$.

Since $\nu(Z_i \sim Z_j) \not< 0$ and $\nu(Z_i \sim Z_j) \not> 0$, we know $\nu(Z_i \sim Z_j) = 0$. □

Lemma 17 *If the set Z_i is the smallest set in Z^* then $Z_i \subset Z_j$ for any j where $i \neq j$. Let $Z_i \subset Z_j$ denote Z_i is a strict subset of Z_j, and let $|\cdot|$ denote the size of its set argument.*

Proof: It is obvious that the lemma can only hold if Z_i is the smallest set in Z^*. Thus, let Z_i be the smallest closure in Z^* and Z_j be any other set in Z^*. Hence, $|Z_i| \leq |Z_j|$.

Assume $Z_i \not\subset Z_j$. We know from Equations 4.3 and 4.4, that the set Z_i and its size can be expressed as

$$Z_i = (Z_i \cap Z_j) \cup (Z_i \sim Z_j), \tag{4.18}$$

and

$$|Z_i| = |Z_i \cap Z_j| + |Z_i \sim Z_j|, \tag{4.19}$$

respectively. We know from the assumption that $|Z_i \sim Z_j| > 0$ and from Lemma 14 that $|Z_i \cap Z_j| > 0$. Thus, we Equation 4.19 gives the relationship

$$|Z_i| > |Z_i \cap Z_j|. \tag{4.20}$$

Thus, the set Z_i is smaller than the set $Z_i \cap Z_j$. We also know from Lemma 8, that the set $Z_i \cap Z_j$ is a closure.

From Equation 4.5, that the value of Z_i is given as

$$\nu(Z_i) = \nu(Z_i \cap Z_j) + \nu(Z_i \sim Z_j). \tag{4.21}$$

From Lemma 16, we know $\nu(Z_i \sim Z_j) = 0$. Thus, Equation 4.21, reduces to

$$\nu(Z_i) = \nu(Z_i \cap Z_j). \tag{4.22}$$

Since $Z_i \in Z^*$, its value equals ν^*. Thus, $\nu(Z_i \cap Z_j) = \nu^*$.

Thus, we know Z_i is larger than $Z_i \cap Z_j$, $Z_i \cap Z_j$ is a closure, and $\nu(Z_i \cap Z_j) = \nu^*$. Hence the assumption, $Z_i \not\subset Z_j$, contradicts the assertion that Z_i is the smallest maximum valued closure in the directed graph. Therefore, $Z_i \subset Z_j$. Obviously, the relationship in Equation 4.20 would be an equality when the assumption is reversed. □

Theorem 3 *If Z_i is the smallest set in Z^* and Z_j is any other set in Z^*, then $|Z_i| < |Z_j|$. In other words, the smallest set in Z^* is unique.*

Proof: Assume for some $i \neq j$ that $|Z_i| = |Z_j|$ and $Z_i \neq Z_j$. Thus, $|Z_i \sim Z_j| > 0$, which implies $|Z_i \cap Z_j| < |Z_i|$. From Equation 4.5 we know

$$\nu(Z_i) = \nu(Z_i \cap Z_j) + \nu(Z_i \sim Z_j). \tag{4.23}$$

We know from Lemma 16 that $\nu(Z_i \sim Z_j) = 0$. Thus, $\nu(Z_i \cap Z_j) = \nu^*$.

Since $Z_i \cap Z_j$ is a smaller closure than both Z_i and Z_j, and $\nu(Z_i \cap Z_j) = \nu^*$, we have contradicted the assumption that Z_i and Z_j are the smallest closures in Z^*. Thus, there can only be a single smallest set in Z^*. □

Lemma 18 *If Z_i is the largest set in Z^* and Z_j is any other set in Z^*, then $Z_j \subset Z_i$.*

Proof: It is obvious that the lemma can only hold if Z_i is the largest set in Z^*. Thus, let Z_i be the largest set in Z^* and Z_j be any other set in Z^*. Thus $|Z_j| \leq |Z_i|$.

Assume $Z_j \not\subset Z_i$. Thus, $|Z_j \sim Z_i| > 0$. The value of the union of Z_i and Z_j is

$$\nu(Z_i \cup Z_j) = \nu(Z_i \cap Z_j) + \nu(Z_i \sim Z_j) + \nu(Z_j \sim Z_i), \tag{4.24}$$

$$= \nu(Z_i) + \nu(Z_j \sim Z_i). \tag{4.25}$$

Where Equation 4.25 follows from Equation 4.5. From Lemma 16 we know, $\nu(Z_i \sim Z_j) = 0$. Thus, the set $Z_i \cup Z_j$ is a closure in the graph G with value equal to ν^* and $|Z_i \cup Z_j| > |Z_i|$. But, this contradicts the assumption that Z_i is the largest set in Z^*. Thus, $Z_j \subset Z_i$. □

Theorem 4 *If Z_i is the largest set in Z^* and Z_j is any other set in Z^*, then $|Z_j| < |Z_i|$. In other words, the largest set in Z^* is unique.*

Proof: Assume for some $j \neq i$ and $j = 1, \ldots, n$ that $|Z_i| = |Z_j|$ and $Z_i \neq Z_j$. Thus, $|Z_i \sim Z_j| > 0$. The value of the union of Z_i and Z_j is

$$\nu(Z_i \cup Z_j) = \nu(Z_i \cap Z_j) + \nu(Z_i \sim Z_j) + \nu(Z_j \sim Z_i), \tag{4.26}$$

$$= \nu(Z_i) + \nu(Z_j \sim Z_i). \tag{4.27}$$

We know from Lemma 16 that $\nu(Z_j \sim Z_i) = 0$. Thus, the set $Z_i \cup Z_j$ has value equal to ν^*. We also know $Z_i \cup Z_j$ is a closure and is larger than Z_i. Thus, contradicting the assumption that Z_i is the largest set in Z^*. Therefore, the largest set in Z^* must be unique. □

4.3 The Maximum Valued Closure Obtained With the LG Algorithm

In Section 4.2, we proved the smallest set in Z^* is a unique set which is a subset of every other set in Z^*. We also proved the largest set in Z^* is a unique set which contains every other set in Z^*. Let Z_1 and Z_n denote the smallest and largest sets in Z^*, respectively. Thus, $Z_1 \subset Z_k$ for $k = 2, \ldots, n$ and $Z_k \subset Z_n$ for $k = 1, \ldots, n-1$.

In Theorem 4.3, we prove the set of strong nodes S_f, in the final normalized tree generated by the LG Algorithm, equals the smallest maximum valued closure, symbolically $S_f = Z_1$. In Corollary 1, we prove a modified version of the LG Algorithm can be forced to converge to the largest maximum valued closure, Z_n. The modified version of the LG Algorithm changes the location of the equality signs in the strength classification of arc relationships.

Lemma 19 *Let $Z^* = \{Z_1, Z_2, \ldots, Z_n\}$ be the collection of all maximum valued closures of the directed graph $G = (X, A)$, where $\nu^* = \nu(Z_k)$ for $k = 1, \ldots, n$. Also, let $|Z_1| < |Z_k|$, $k = 2, \ldots, n$. Then for any set of nodes $Y \subseteq X$,*

$$\nu(\Gamma(Y) \sim Z_1) = 0$$

if $\Gamma(Y) = Z_k$ for some $k \in \{1, \ldots, n\}$ and

$$\nu(\Gamma(Y) \sim Z_1) < 0$$

for any other set of nodes Y.

Proof: If $\Gamma(Y) = Z_k$ for some $k \in \{1, \ldots, n\}$, then $\nu(\Gamma(Y) \sim Z_1) = 0$.

Assume $\nu(\Gamma(Y) \sim Z_1) > 0$. Consider the set $(\Gamma(Y) \sim Z_1) \cup Z_1$. It is obvious

that,

$$\Gamma(Y) \cup Z_1 = (\Gamma(Y) \sim Z_1) \cup Z_1, \tag{4.28}$$

and

$$\begin{aligned} \nu(\Gamma(Y) \cup Z_1) &= \nu((\Gamma(Y) \sim Z_1) \cup Z_1), && (4.29)\\ &= \nu((\Gamma(Y) \sim Z_1)) + \nu(Z_1), && (4.30)\\ &= \nu((\Gamma(Y) \sim Z_1)) + \nu^*, && (4.31)\\ &\geq \nu^*. && (4.32) \end{aligned}$$

Thus, $\nu(\Gamma(Y)) \leq 0$. □

Lemma 20 *Let $Z^* = \{Z_1, Z_2, \ldots, Z_n\}$ be the collection of all maximum valued closures of the directed graph $G = (X, A)$, where $\nu^* = \nu(Z_k)$ for $k = 1, \ldots, n$. Also, let $|Z_1| < |Z_k|$, $k = 2, \ldots, n$. If S_f is the set of strong nodes in the final normalized tree, T_f, of the LG Algorithm, then it is possible to eliminate all arcs between nodes in $S_f \sim Z_1$ and Z_1.*

Proof: Since Z_1 is a closure, we know there does not exist a node in Z_1 which is dependent upon a node in $S \sim Z_1$. Assume there exists an arc, $a_{i,j} = (x_i, x_j)$, such that $x_i \in S \sim Z_1$ and $x_j \in Z_1$.

If the $a_{i,j}$ is a p-arc, then the value of the branch $B(x_j)$ must have non-positive value. □

Theorem 5 *Let $Z^* = \{Z_1, Z_2, \ldots, Z_n\}$ be the collection of all maximum valued closures of the directed graph $G = (X, A)$, where $\nu^* = \nu(Z_k)$ for $k = 1, \ldots, n$. Also, let*

$|Z_1| < |Z_k|$, $k = 2, \ldots, n$. *If S_f is the set of strong nodes in the final normalized tree, T_f, of the LG Algorithm, then $S_f = Z_1$.*

Proof: Consider any node $x_i \in S \sim Z_1$. Since all nodes in Z_1 are independent of all nodes in $\Gamma(x_i) \sim Z_1$, the nodes in $\Gamma(x_i) \sim Z_1$ must be contained in a collection of p-branches. Let $B_{\Gamma(x_i)\sim Z_1}$ denote this collection of branches.

Assume $B_{\Gamma(x_i)\sim Z_1}$ contains at least one positive valued branch, denoted as B_p. By Lemma 19, we find

$$\nu(\Gamma(x_i) \sim Z_1) \leq 0, \tag{4.33}$$

$$\nu(B_{\Gamma(x_i)\sim Z_1}) \leq 0, \tag{4.34}$$

$$\sum_{j=1}^{m} \nu(B_j) \leq 0, \tag{4.35}$$

$$\nu(B_p) + \sum_{j=1, j\neq p}^{m} \nu(B_j) \leq 0, \tag{4.36}$$

$$\sum_{j=1, j\neq p}^{m} \nu(B_j) \leq -\nu(B_p). \tag{4.37}$$

Thus, $B_{\Gamma(x_i)\sim Z_1}$ must contain at least one non-positive valued branch, denoted as B_{np}.

In addition, there must exist some node in B_p which is dependent upon a node in B_{np}. To see this, assume there does not exist a node in B_p which is dependent upon a node in B_{np}. Then, since the set of nodes in $B_{\Gamma(x_i)\sim Z_1}$ defines a closure, the set of node in branch B_p define a closure. By Lemma 7, we know $Z_1 \cup B_p$ is a closure. Computing it's value gives $\nu(Z_1)+\nu(B_p) > \nu^*$. Thus, contradicting the assertion that Z_1 is a maximum valued closure. Hence, it is not possible for $B_{\Gamma(x_i)\sim Z_1}$ to contain a positive valued branch.

Thus, each branch in $B_{\Gamma(x_i) \sim Z_1}$ has non-positive value, each branch is weak, and all nodes in $\Gamma(x_i) \sim Z_1$ are weak. Hence, node x_i must be weak.

Hence, all nodes in $S \sim Z_1$ must be weak. □

For example, consider the normalized tree in Figure 4.6. This is the final normalized tree of the example problem discussed in the previous sections of this chapter. The tree contains six strong nodes: $\{x_1, x_2, x_3, x_4, x_9, x_{10}\}$. These nodes correspond to the smallest maximum valued closure in the graph.

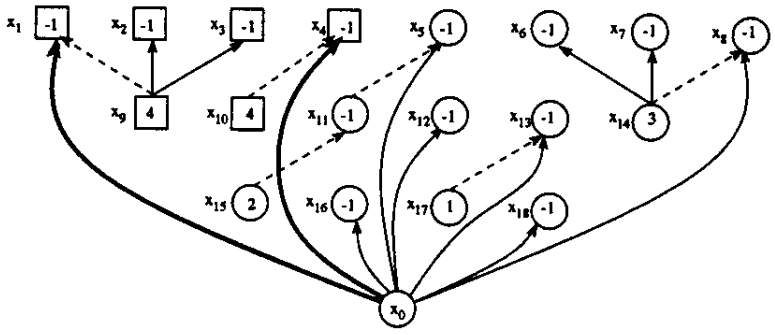


Figure 4.6: The Final Normalized Tree With Multiple Maximum Valued Closures

Table 4.1 provides, for each arc in the normalized tree, the root of the branch formed by severing the arc, the branch's value, and the arc's directional and strength classification. The normalized tree contains eight artificial arcs, which must be p-arcs. Of these p-arcs, only two support positive valued branches. These are the strong p-arcs. Each of the remaining p-arcs support non-positive valued branches, thus they are weak p-arcs.

For each maximum valued closure in the graph, those nodes which are not in the smallest maximum valued closure are weak. This follows since each maximum valued closure (other than the smallest) consists of a collection of branches: some of which are strong and some of which are weak. Each strong branch contains nodes

Table 4.1: Arc Classifications For Tree In Figure 24

Arc	Branch Root x_{br}	Branch Value $\nu(B(x_{br}))$	Directional Classification	Strength Classification
$a_{0,1}$	x_1	1	p-arc	strong
$a_{0,4}$	x_4	3	p-arc	strong
$a_{0,5}$	x_5	0	p-arc	weak
$a_{0,8}$	x_8	0	p-arc	weak
$a_{0,12}$	x_{12}	-1	p-arc	weak
$a_{0,13}$	x_{13}	0	p-arc	weak
$a_{0,16}$	x_{16}	-1	p-arc	weak
$a_{0,18}$	x_{18}	-1	p-arc	weak
$a_{9,1}$	x_9	2	m-arc	weak
$a_{9,2}$	x_2	-1	p-arc	weak
$a_{9,3}$	x_3	-1	p-arc	weak
$a_{10,4}$	x_{10}	4	m-arc	weak
$a_{11,5}$	x_{11}	1	m-arc	weak
$a_{14,6}$	x_6	-1	p-arc	weak
$a_{14,7}$	x_7	-1	p-arc	weak
$a_{14,8}$	x_3	1	m-arc	weak
$a_{15,11}$	x_{15}	2	m-arc	weak
$a_{17,13}$	x_{17}	1	m-arc	weak

in the smallest maximum valued closure. Each weak branch contains nodes not in the smallest maximum valued closure, and these branches have non-positive value. Consider the branch supported by arc $a_{0,5}$. This arc supports a zero valued branch. Taking the union of the nodes in this branch with the nodes in the smallest maximum valued closure forms another maximum valued closure. But, since the LG Algorithm defines a weak p-arc to be any arc supporting a non-positive valued branch, the branch is weak.

Corollary 1 *Let Z_n be the largest maximum valued closure in a directed graph $G = (X, A)$. Also let S' denote the set of strong nodes in the final normalized tree generated by the LG Algorithm, when the* alternative strength classification definitions *are used by the algorithm. Then $S' = Z_n$.*

The alternative strength classification definitions *are: define a strong p-arc as a p-arc which supports a non-negative branch; define a weak p-arc as a p-arc which supports a negative branch; define a strong m-arc as a m-arc which supports a negative branch; and, define a weak m-arc as a m-arc which supports a non-negative branch. The alternative strength classification definitions simply change the location of the equality sign, with respect to the original strength classification definitions.*

Proof: Changing the strength classification of a p-arc from p-arcs supporting branches with positive value to p-arcs supporting branches with non-negative value enables a p-branch to be strong if it supports a zero valued branch.

Consider the collection of nodes in the set $Z_n \sim Z_1$, where we know from Lemma 16 that $\nu(Z_n \sim Z_1) = 0$. From Theorem 5, we know the set of strong nodes S in the final normalized tree T_f generated by the LG Algorithm equals Z_1. Thus, the set of nodes $Z_n \sim Z_1$ must be classified as weak.

Let B_W denote the set of branches containing the nodes in the set $Z_n \sim Z_1$. These branches are formed by severing the p-arcs between the artificial root x_0 and the branch root. These p-branches must have zero value, otherwise the branch would be strong.

Using the *alternative strength classification definitions*, the p-branches in B_W would be classified as strong, and nodes in these branches would be classified as strong. Thus, all nodes in $Z_n \sim Z_1$ would be classified as strong and the set of strong nodes would include $Z_n \sim Z_1$ in addition to Z_1. Hence, the resulting set of strong nodes would equal the largest maximum valued closure Z_n. □

Using the *alternative strength classification definitions*, we obtain the final normalized tree shown in Figure 4.7. As Corollary 1 states, changing the definition of a strong p-arc, from a p-arc supporting a branch with positive value to a p-arc supporting a branch with non-negative value, causes the set of strong nodes in the final normalized tree to equal the largest maximum valued closure.

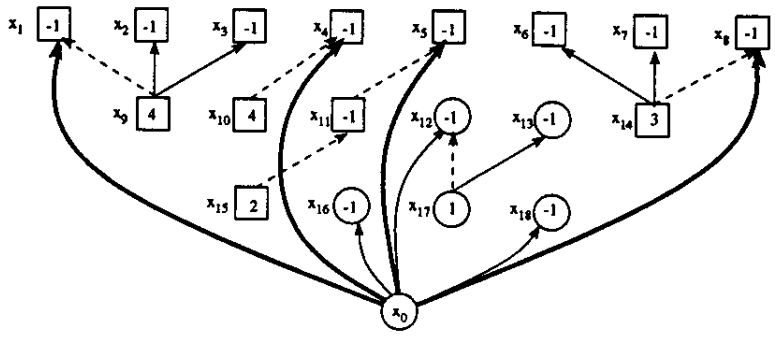


Figure 4.7: The Final Normalized Tree Using The Alternative Strength Classification Definitions

Each arc, in Figure 4.6, supporting a zero valued branch is classified as a strong arc under the alternative strength classification definitions. From Table 4.1, we see

there are three zero valued branches: $B(x_5)$, $B(x_8)$, and $B(x_{13})$. Taken in conjunction with the nodes in branches $B(x_1)$ and $B(x_4)$, the nodes in branches $B(x_5)$, $B(x_8)$ define a closure. Thus, all nodes in $B(x_5)$ and $B(x_8)$ remain classified as strong.

But, nodes in $B(x_{13})$, taken in conjunction with nodes in $B(x_1)$ and $B(x_4)$, do not define a closure. Node x_{17} is also dependent upon node x_{12}. Thus, another iteration of the LG Algorithm is required. The *MoveTowardFeasibility* procedure replaces the arc $a_{0,13}$ with the arc $a_{17,12}$. The p-arc $a_{0,12}$ supports the branch $B(x_{12}) = \{x_{12}, x_{13}, x_{17}\}$. The branch has a value of negative one. Thus, the p-arc is classified as a weak p-arc and all nodes in the branch are classified as weak.

Chapter 5
LINEAR PROGRAMMING FORMULATION OF THE PIT LIMIT PROBLEM

In this chapter, we develop the linear programming formulation for the *pit limit problem* and its dual formulation. We shall use, primal and dual, to refer to the two formulations. We assume the reader is familiar with linear programming concepts.

We relate the steps of the LG Algorithm to the steps performed by the Simplex Algorithm, in solving the dual formulation of the pit limit problem. Specifically, we show:

- The *InitNormalizeTree* procedure is equivalent to finding the initial basic feasible solution in the Simplex Algorithm.
- The *MoveTowardFeasibility* procedure is equivalent to one or more change of bases in the Simplex Algorithm.
- The *NormalizeTree* procedure is equivalent to a change of basis in the Simplex Algorithm, and the change of basis is only required when the basis generated by the *MoveTowardFeasibility* procedure is dual infeasible.
- The *SolutionNotFeasible* procedure identifies the solution to be feasible when the dual solution is optimal.

In addition to the relationships described above, we show relationships exist between nodes and arcs in the LG Algorithm's normalized trees, and the Simplex Algorithm's tableau. Specifically, we find:

- The value of the set of strong nodes in each normalized tree equals the value of the objective function.
- The strength classification of each node in the normalized tree is indicated by the objective function coefficients of the slack variables.
- The set of arcs having a strong source node and a weak terminal node is indicated by the objective function coefficients of the dual decision variables associated with the real arcs in the augmented graph.
- The value of each branch in the normalized tree equals the value of associated dual decision variables in the basis.

Although the LG Algorithm and the Simplex Algorithm do have a considerable number of similarities, some differences do exist. The two primary differences are:

- Selection of the arc to remove from the normalized tree (or, in the linear programming context, select the leaving basic variable).
- The final solution reached when the directed graph contains multiple maximum valued closures.

In Section 5.1, the primal pit limit linear program is formulated. In Section 5.2, the dual pit limit linear program is formulated. In Section 5.3, the dual linear program is converted to the *standard form* assumed by the Simplex Algorithm. In Section 5.4, we review the steps of the simplex algorithm. In Section 5.5, we compare the augmented graph of the LG Algorithm and the augmented graph of the dual linear program. In Section 5.6, we show the equivalences between the steps of the LG Algorithm and the Simplex Algorithm. In Section 5.7, we show the relationships

between nodes and arcs in the LG Algorithm's normalized trees, and the Simplex Algorithm's tableau. In Section 5.8, we show the dual linear program's tableau for each tree formed by the LG Algorithm, when applied to the example problem discussed in Chapter 2.

Essentially, we show the LG Algorithm is a graphical implementation of the Simplex Algorithm, as applied to the dual pit limit linear program. This development identifies several topics for future research. Since this chapter is rather lengthy, the reader knowledgeable about the Simplex Algorithm may wish to simply review Sections 5.1-5.3, and read Sections 5.6 and 5.7.

5.1 The Primal Pit Limit Linear Program

In Chapter 2, we developed the LG Algorithm using a directed graph $G = (X, A)$, where X and A denote the set of nodes and the set of arcs in the graph, respectively. In Chapter 3, we proved the LG Algorithm converges to a maximum valued closure in the graph. In Chapter 4, we proved the LG Algorithm converges to the smallest maximum valued closure, when the graph contains several maximum valued closures. Thus, the primal pit limit linear program must find a maximum valued closure in a directed graph. We will not restrict it to finding the smallest maximum valued closure.

Let X be a vector of non-negative real *decision variables* X; where the *ith* element of the vector corresponds to node x_i. Let C be a vector of real coefficients, where the *ith* element of the vector is the value of the *ith* node, or $C[i] = \nu(x_i)$. Thus, X is the decision variable vector and C is the cost, or value, vector.

We shall use a constraint to reflect the direct dependency implied by an arc in the graph. Recall that each arc, $a_{i,j} \in A$, implies a direct dependency between it's

Table 5.1: States Of The Direct Dependency Constraint

x_i	x_j	$-x_j + x_i \leq 0$
0	0	Feasible
0	1	Feasible
1	0	Infeasible
1	1	Feasible

source node, x_i, and it's terminal node, x_j. Represent each arc $a_{i,j}$ with the following constraint

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

Thus, the source node has a positive one coefficient and the terminal node has a negative one coefficient. Assuming the decision variable's, x_i and x_j, can only assume a value of zero or one (which will be demonstrated below), there exist four possible sets of values for the pair. Table 5.1 shows the four possible sets of values for the pair, and the feasibility of the constraint for each pair.

Let E be a matrix containing the coefficients of all arc constraints, and call this set of constraints the *accessibility constraints.* Thus, each row of E contains: a single positive one coefficient, a single negative one coefficient, with all remaining coefficients equal to zero.

In addition to the accessibility constraints, the pit limit problem also contains what we call the *single extraction constraints.* These constraints require the value of each decision variable to be less than or equal to one. Thus, ensuring the related node is extracted only once. The set of single extraction constraints can be expressed as $IX \leq 1$, where I is an appropriately dimensioned identity matrix.

Using the vectors, X and C, and the matrices, E and I, we can state the *primal*

pit limit linear program as:

$$\begin{aligned} \text{max.} \quad & C^T X, \\ \text{s.t.:} \quad & EX \leq 0, \\ & IX \leq 1, \\ & X \geq 0. \end{aligned}$$

The values 0 and 1 denote appropriately dimensioned vectors consisting of zeros or ones, respectively. The last set of constraints, $X \geq 0$, are called the *non-negativity constraints.* These constraints ensure decision variables have non-negative values, since a negative value is unreasonable in the context of the problem. The statement *max* $C^T X$ is called the *objective function.* The objective function states the optimizing criteria for the linear program. Combining the matrices E and I forms the *constraint matrix* for the pit limit linear program.

In general, a linear program consists of an objective function and a set of constraints. The objective of the linear program is to find the optimal feasible solution. A *solution* is a specification of decision variable values. A *feasible solution* is a solution which meets all constraint requirements. An *optimal feasible solution* is a feasible solution whose value is optimal with respect to the objective function. For an objective function which is to be maximized, an optimal feasible solution is a feasible solution whose value is greater than or equal to all other feasible solutions.

For the primal pit limit linear program, a feasible solution is a specification of decision variable values, such that each decision variable's value is in the real interval [0,1], and each of the accessibility constraints is met. An optimal feasible solution is a feasible solution such that the sum of the products (of the decision variable's value and the associated nodes value), is greater than or equal to every other feasible solution. In a graph theoretic framework, the primal pit limit problem is equivalent

to finding the maximum valued closure in a directed graph.

Although the primal pit limit linear program could be solved using the Simplex Algorithm (discussed in Section 5.4, its use would not take advantage of the special structure of the constraint matrix. Using the special structure of the constraint matrix reduces the computer resources required to solve the problem.

A typical block model consists of a large number of blocks, with each block having multiple direct dependencies. The number of direct dependencies for a given block depends upon the blocks location, the pit slope angles, and the common dimensions of the blocks. Assuming there are n blocks in the block model, and that each block is directly dependent upon m blocks, the primal pit limit linear program would have n variables and at least nm constraints. Thus, a small block model containing 40000 blocks, with each block having at least 9 direct dependencies (typically more), would result in a linear program having 40000 variables and at least 360000 constraints.

As was stated by Dagdelen 1985, the constraint matrix of the primal pit limit problem is a *unimodular* matrix. Linear programs which have a unimodular constraint matrix and integral bounds, have been proven, see (Ignizio 1984), to ensure decision variables assume integral values. Since, in the primal pit limit problem, the decision variables are bounded between zero and one, and the constraint matrix is unimodular, the decision variables are ensured to assume values of either zero or one. A value of one implies the node is contained in the pit limit, a value of zero implies the node is not contained in the pit limit.

Figure 5.1 shows the primal pit limit linear program for the problem discussed in Chapter 2. Each column corresponds to a decision variable which also corresponds to a node in the graph. Thus, above each column we identify the associated node. Each constraint row corresponds to an arc in an augmented graph very similar to the

augmented graph shown in Figure 2.7. The first twelve constraints are the accessibility constraints, which relate to the *real* arcs. The last nine constraints are the single extraction constraints, which relate to the *artificial* arcs. Thus, to the left of each row we identify the associated arc. Each coefficient in the objective function row relates to the value of the column's related node.

		NODES (x_i)										
	i	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9		
max.	z=	-1	-1	-1	-1	-1	4	4	-1	2		
s.t.:												
ARCS												
$a_{6,1}$		-1	0	0	0	0	1	0	0	0	<=	0
$a_{6,2}$		0	-1	0	0	0	1	0	0	0	<=	0
$a_{6,3}$		0	0	-1	0	0	1	0	0	0	<=	0
$a_{7,2}$		0	-1	0	0	0	0	1	0	0	<=	0
$a_{7,3}$		0	0	-1	0	0	0	1	0	0	<=	0
$a_{7,4}$		0	0	0	-1	0	0	1	0	0	<=	0
$a_{8,3}$		0	0	-1	0	0	0	0	1	0	<=	0
$a_{8,4}$		0	0	0	-1	0	0	0	1	0	<=	0
$a_{8,5}$		0	0	0	0	-1	0	0	1	0	<=	0
$a_{9,6}$		0	0	0	0	0	-1	0	0	1	<=	0
$a_{9,7}$		0	0	0	0	0	0	-1	0	1	<=	0
$a_{9,8}$		0	0	0	0	0	0	0	-1	1	<=	0
$a_{1,0}$		1	0	0	0	0	0	0	0	0	<=	1
$a_{2,0}$		0	1	0	0	0	0	0	0	0	<=	1
$a_{3,0}$		0	0	1	0	0	0	0	0	0	<=	1
$a_{4,0}$		0	0	0	1	0	0	0	0	0	<=	1
$a_{5,0}$		0	0	0	0	1	0	0	0	0	<=	1
$a_{6,0}$		0	0	0	0	0	1	0	0	0	<=	1
$a_{7,0}$		0	0	0	0	0	0	1	0	0	<=	1
$a_{8,0}$		0	0	0	0	0	0	0	1	0	<=	1
$a_{9,0}$		0	0	0	0	0	0	0	0	1	<=	1

Figure 5.1: The Primal Pit Limit Linear Program

The first twelve constraints are the accessibility constraints. Each accessibility constraint corresponds to a real arc in the augmented graph. For a given accessibility

constraint, the column with a positive one coefficient indicates the source node, the column with a negative one coefficient indicates the terminal node. For example, the first constraint relates to arc $a_{6,1}$; which has node x_6 as its source node and node x_1 as its terminal node.

The final nine constraints are the single extraction constraints. These constraints correspond to artificial arcs in the augmented graph (shown below in Figure 5.2); although each constraint contains only a single non-zero coefficient. Since a positive one in the accessibility constraints indicated the arc's source node, we also let the positive one coefficient, in the single extraction constraints, indicate the arc's source node. We assume each artifical arc terminates at an artificial node x_0. The graph formed by adding the artificial arcs and the artificial node is called the *augmented graph.*

The objective function is a linear function expressing the criteria for maximizing the linear program. In Figure 5.1, the row with the word "max." at the left denotes the objective function. Each coefficient in the objective function equals the value of the corresponding node. For example, node x_6 has a value of 4 and node x_9 has a value of 2. In the primal pit limit linear program, we wish to define a solution which: maximizes the value of the nodes, and meets all accessibility and single extraction constraints.

Figure 5.2 shows the augmented graph for the pit limit linear program. Node values are placed inside the node symbols, and node designations are placed to the left of each node symbol. We denote real arcs with straight lines and artificial arcs with curved lines. Notice that each artificial arc terminates at the artificial node x_0, whereas in the LG Algorithm the artificial arcs originate at x_0.

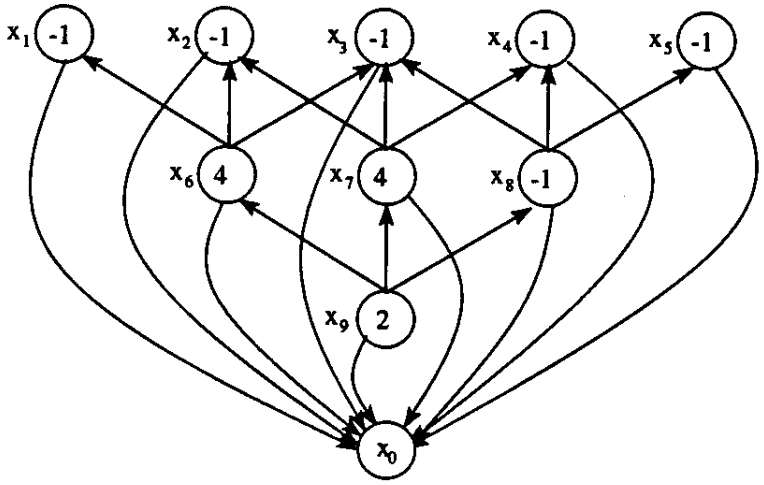


Figure 5.2: The Augmented Graph

5.2 The Dual Pit Limit Linear Program

The dual of the primal pit limit linear program is:

$$\begin{aligned} \text{min.} \quad & 0^T U + 1^T V, \\ \text{s.t.:} \quad & E^T U + IV \geq C, \\ & U \geq 0, \\ & V \geq 0. \end{aligned}$$

The vectors U and V are dual decision variable vectors. The matrix E is the same accessibility constraint matrix as was used in the primal. The vector C is the same node value vector. Again we let 0 denote an appropriately dimensioned vector of zeros, and 1 denote an appropriately dimensioned vector of ones.

Figure 5.3 shows the dual pit limit linear program for the problem discussed in Section 5.1. Each column corresponds to an arc in the augmented graph and a dual decision variable. For each column we denote the decision variable, the corresponding arc, and the source node and terminal node of each arc. For example, the column

with decision variable u_4 (shown at the top) corresponds to the real arc $a_{7,2}$; which has node x_7 as its source node and node x_2 as its terminal node. Similarly, the column with decision variable v_4 corresponds to the artificial arc $a_{4,0}$; which has node x_4 as its source node and the artificial root x_0 as its terminal node.

										DUAL VARIABLES (u_i,v_i)													
dual var.	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9		
										ARCS ($a_{i,j}$)													
arc $a_{i,j}$	a_{61}	a_{62}	a_{63}	a_{72}	a_{73}	a_{74}	a_{83}	a_{84}	a_{85}	a_{96}	a_{97}	a_{98}	a_{10}	a_{20}	a_{30}	a_{40}	a_{50}	a_{60}	a_{70}	a_{80}	a_{90}		
										ARC NODES ($a_{i,j}$=(x_i,x_j))													
source i	6	6	6	7	7	7	8	8	8	9	9	9	1	2	3	4	5	6	7	8	9		
term. j	1	2	3	2	3	4	3	4	5	6	7	8	0	0	0	0	0	0	0	0	0		
min.	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1		
NODES																							
x_1	-1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	>=	-1
x_2	0	-1	0	-1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	>=	-1
x_3	0	0	-1	0	-1	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	>=	-1
x_4	0	0	0	0	0	-1	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	>=	-1
x_5	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	>=	-1
x_6	1	1	1	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	>=	4
x_7	0	0	0	1	1	1	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	>=	4
x_8	0	0	0	0	0	0	1	1	1	0	0	-1	0	0	0	0	0	0	0	1	0	>=	-1
x_9	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	>=	2

Figure 5.3: The Dual Pit Limit Linear Program

The dual constraints, $E^TU + IV \geq C$, are *net flow* constraints. Each net flow constraint requires the net flow on arcs entering and leaving the related node to be greater than or equal to the value of the node. For example, consider the constraint with x_6 to its left. This constraint requires that the total outflow on arcs $a_{6,1}$, $a_{6,2}$, $a_{6,3}$, less the total inflow on arc $a_{9,6}$, be greater than or equal to 4.

The dual constraints, $U \geq 0$ and $V \geq 0$, are non-negativity constraints. They require that the flow on any given arc be non-negative. These constraints are not specified in the linear program shown in Figure 5.3. But, they are assumed to be there.

The dual's objective function, $min\ 0^T U + 1^T V$, implies the objective is to minimize the amount of flow on the arcs associated with the dual variables v_i. Each of these arcs is an artificial arc. Thus, the objective of the dual formulation is to minimize flow on the artificial arcs.

Since the constraint matrix in the dual formulation is unimodular, all dual variables, u_i and v_i, will have integral values. Assuming all node values in C are integral. A non-integral node value may cause some dual variables to have non-integral values.

5.3 Conversion Of The Dual To The Standard Form

The Simplex Algorithm is the general algorithm used for solving linear programming problems. The algorithm assumes the linear program is in a *standard form.* The assumed standard form maximizes an objective function subject to a set of equality constraints. The following paragraphs describe how we chose to convert the dual pit limit linear program into the standard form.

To convert the objective function, $minZ = O^T U + 1^T V$, from its minimization form to a maximization form, we multiplied by negative one. In the standard form, the dual pit limit linear program's objective function is

$$max.\ 0^T U - 1^T V.$$

Conversion of the dual constraints, from inequality constraints to equality constraints, requires several steps; where the steps followed depend upon the sign of the *bound* (the constraint's right hand side). Each dual constraint is a greater than or equal to inequality with either a negative or a non-negative bound.

A dual constraint with a negative bound can be expressed as

$$E_i^T U + v_i \geq c_i,$$

where E_i is the ith column of the accessibility matrix (which corresponds to the ith node), U is the dual decision variable vector, v_i is also a dual decision variable, and c_i is the negative bound. Dual constraints with a negative bound are converted as follows. The constraint is multiplied by negative one, which converts the inequality to a less than or equal to inequality and forces the constraint's bound to be positive. The resulting form of the constraint is

$$-E_i^T U - v_i \leq -c_i.$$

To convert the constraint, from an inequality to an equality, we introduce a unique *slack variable* s_i. The constraint then assumes the form

$$-E_i^T U - v_i + s_i = -c_i.$$

The slack variable has a value equal to the constraint's slack (the difference in the constraint's current value and its bound).

A dual constraint with a positive bound can be expressed as

$$E_i^T U + v_i \geq c_i.$$

Since the bound is positive, we do not need to multiply by a negative one. To convert the constraint, from an inequality to an equality, we introduce a unique

surplus variable s_i for each constraint. The surplus variable has a value equal to the constraint's surplus (the difference in the constraint's current value and its bound). The resulting form of the constraint becomes

$$E_i^T U + v_i - s_i = c_i.$$

Figure 5.4 shows, the dual pit limit linear program in standard form. In it's standard form: all constraints are equality constraints with non-negative bounds, and the objective function is to be maximized. For each column we identify its decision variable or slack variable, the related arc and it's source node and terminal node. We replaced each constraint's associated node with the constraint's corresponding basic variable (discussed in Section 5.4).

The introduction of the slack and surplus variables, s_i, adds additional artificial arcs to the augmented graph. For example, the slack variable, s_2, adds the artificial arc $a_{2,0}$. The new augmented graph is discussed in Section 5.5.

	DUAL VARIABLES (u_i,v_i)																				SLACK VARIABLES (s_i)									
dual var.	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	
arc $a_{i,j}$	a61	a62	a63	a72	a73	a74	a83	a84	a85	a96	a97	a98	a10	a20	a30	a40	a50	a60	a70	a80	a90	a01	a02	a03	a04	a05	a06	a07	a08	a09	
source i	6	6	6	7	7	7	8	8	8	9	9	9	1	2	3	4	5	6	7	8	9	0	0	0	0	0	0	0	0	0	
term. j	1	2	3	2	3	4	3	4	5	6	7	8	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	9	
max.	0	0	0	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0	0	0	0	0	0	0	0	
BASIC VAR.																															
s_1	1	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	=	1
s_2	0	1	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	=	1
s_3	0	0	1	0	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	=	1
s_4	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	=	1
s_5	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	=	1
x_6*	1	1	1	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	=	4
x_7*	0	0	0	1	1	1	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	=	4
s_8	0	0	0	0	0	0	-1	-1	-1	0	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	=	1
x_9*	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	=	2

Figure 5.4: The Dual Pit Limit Linear Program In Standard Form

Table 5.2: Relationships Between The Graph Formulation And The Linear Programming Formulations

Graph Formulation	Linear Programming Formulations
Node x_i	Primal Decision Variable x_i
Real Arc $a_{i,j}$	Dual Decision Variable u_k
Artificial Arc $a_{0,i}$	Dual Slack Variable s_i
Artificial Arc $a_{i,0}$	Dual Decision Variable v_i

If there are m arcs and n nodes in the directed graph, for which the pit limit linear program is being defined, then in the dual linear program there will be $m+2n$ decision variables and n constraints (excluding the nonnegativity constraints). There are m decision variables u_i, n decision variables v_i, and a total of n slack and surplus variables s_i.

In Table 5.2, we show relationships between the graph formulation and the linear programming formulations, of the pit limit problem. Note the following: the notation for the primal decision variables is identical to that used for nodes in the graph formulation; the subscript of the real node in each artificial arc corresponds to the subscript of its corresponding dual variable; and, the subscript of the dual decision variables corresponding to the real arcs does not necessarily agree with either subscript of its corresponding real arc.

5.4 Review Of The Simplex Algorithm

In this section, we review the steps of the Simplex Algorithm. Knowledgeable readers may choose to skip this section. Given a linear program, in the standard form, the Simplex Algorithm consists of the following steps.

Initialization Step. Define an initial basic feasible solution. A *solution* is any specification of decision variable values. A *feasible solution* is a solution where all con-

straints are satisfied. Since there are $m+2n$ decision variables and only n constraints we can set any $m+n$ decision variables to a value of zero and solve the constraint equations in terms of the remaining n decision variables. The $m+n$ decision variables set to zero are called the *non-basic variables* and the remaining n decision variables are called the *basic variables.* A *basic solution* is a solution defined in terms of the basic variables.

A basic variable is identified by a column with a single positive one coefficient in the column and zero coefficients everywhere else. In Figure 5.4, there are six basic variables $\{s_1, s_2, s_3, s_4, s_5, s_8\}$. Each basic variable has an association with the constraint in which the positive one coefficient is located. To denote the association, the decision variable is identified at the left of the constraint.

The columns associated with the basic variable's define a *basis.* Since Figure 5.4 contains only six basic variables, we must identify three additional basic variables in order to have a *full rank basis.* A full rank basis is a basis where the columns associated with the basic variables span the vector space defined by all columns in the linear program.

The Simplex Algorithm uses an approach called the *Big-M Method* to define a full rank basis when one does not exist. The Big-M Method adds *artificial variables* which are assigned large negative coefficient values in the objective function. The Simplex Algorithm is then used to drive these artificial variables from the basis.

Although the Big-M Method could be applied to the dual pit limit linear program, a simpler alternative exists. Inspection of the constraint matrix, in Figure 5.4, indicates we can form additional basic variables by adding some constraint rows to the objective function row, and thus force objective function coefficients to zero. For a decision variable to be made a basic variable we need only make the decision

variable's column contain a single positive one coefficient (all other coefficients must be zero). Notice that the columns associated with dual variables v_6, v_7 and v_9 have a single positive one coefficient and a single negative one coefficient in the objective function. To convert these variables into basic variables we simply add their associated constraints (the constraints with basic variables which have a star beside them in the basic variable column) to the objective function.

For example, adding the row associated with the node x_6 to the objective function converts the objective function coefficient, associated with dual variable v_6, to zero. Thus, making dual variable v_6 a basic variable. Introducing dual variables v_6, v_7 and v_9 into the basis gives the dual linear program shown in Figure 5.5. The dual pit limit linear program with a full rank basis is shown in Figure 5.5. Notice that introducing these dual variables into the basis forces the objective function to have a value of ten (shown at the far right of the objective function).

	DUAL VARIABLES (u_i, v_i)																				SLACK VARIABLES (s_i)									
dual var.	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	
max.	1	1	1	1	1	1	0	0	0	0	0	1	-1	-1	-1	-1	-1	0	0	-1	0	0	0	0	0	0	-1	-1	0	-1	= 10
BASIC VAR.																															
s_1	1	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	= 1
s_2	0	1	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	= 1
s_3	0	0	1	0	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	= 1
s_4	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	= 1
s_5	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	= 1
v_6	1	1	1	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	0	= 4
v_7	0	0	0	1	1	1	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	= 4
s_8	0	0	0	0	0	0	-1	-1	-1	0	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	= 1
v_9	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	= 2

Figure 5.5: The Dual Pit Limit Linear Program With A Full Rank Basis

Each dual constraint relates to a single basic variable. The basic variable with a positive one coefficient in the constraint. The value of the constraint's bound defines the value of the basic variable. In Figure 5.5, each basic variable $\{s_1, s_2, s_3, s_4, s_5, s_8\}$

has a value of one. The basic variables v_6, v_7, v_9 have values of 4, 4, and 2, respectively.

The objective function has a value of ten, shown to the right of the objective function row. Note that the value of the objective function equals the sum of the values of the dual variables $\{v_6, v_7, v_9\}$, which were the only decision variables with non-zero coefficients in the objective function.

Since an initial basic feasible solution has been identified, the dual linear program's *tableau* may be constructed. Figure 5.6 shows the dual pit limit linear program's *tableau.*

	DUAL VARIABLES (u_i, v_i)																					SLACK VARIABLES (s_i)								
dual var.	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9
	ARCS ($a_{i,j}$)																													
arc a_{ij}	a_{61}	a_{62}	a_{63}	a_{72}	a_{73}	a_{74}	a_{83}	a_{84}	a_{85}	a_{96}	a_{97}	a_{98}	a_{10}	a_{20}	a_{30}	a_{40}	a_{50}	a_{60}	a_{70}	a_{80}	a_{90}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	a_{06}	a_{07}	a_{08}	a_{09}
	ARC NODES ($a_{i,j}=(x_i, x_j)$)																													
source i	6	6	6	7	7	7	8	8	8	9	9	9	1	2	3	4	5	6	7	8	9	0	0	0	0	0	0	0	0	0
term. j	1	2	3	2	3	4	3	4	5	6	7	8	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	9
max.	-1	-1	-1	-1	-1	-1	0	0	0	0	0	-1	1	1	1	1	1	0	0	1	0	0	0	0	0	0	1	1	0	1 = -10
BASIC VAR.																														
$s_1=a_{01}$	1	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0 = 1
$s_2=a_{02}$	0	1	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0 = 1
$s_3=a_{03}$	0	0	1	0	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0 = 1
$s_4=a_{04}$	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0 = 1
$s_5=a_{05}$	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0 = 1
$v_6=a_{60}$	1	1	1	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	0 = 4
$v_7=a_{70}$	0	0	0	1	1	1	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0 = 4
$s_8=a_{08}$	0	0	0	0	0	0	-1	-1	-1	0	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0 = 1
$v_9=a_{90}$	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	= 2

Figure 5.6: The Initial Tableau of the Dual Pit Limit Linear Program

After identifying the initial basic feasible solution, the Simplex Algorithm checks to determine whether the *Stopping Criteria*, described below, has been met. If the Stopping Criteria is not met, the Simplex Algorithm performs an *Iteration Step.*

Iteration Step. The *Iteration Step* consists of three parts: Part (i) selecting the entering basic variable, Part (ii) identifying the leaving basic variable, and Part

(*iii*) determining the new basic feasible solution. The iteration step replaces the leaving basic variable with the entering basic variable, which forms a new basic feasible solution typically having a greater value. The change in basic variables is called a *change of basis.* At the completion of each iteration step, the *Stopping Criteria* is considered to determine whether the current solution is optimal.

In Part (*i*) an entering basic variable is selected. The entering basic variable must be a non-basic variable (identified by a negative objective function coefficient). Since the entering basic variable has a negative objective function coefficient, its introduction into the basis will improve the value of the objective function. Assuming the linear program is in the standard form, the entering basic variable is the non-basic variable with the largest negative objective function coefficient (in absolute terms). Thus selecting as the entering basic variable, the non-basic variable which provides the largest incremental change in the objective function (relative to the current solution).

In Part (*ii*) the leaving basic variable is selected. The leaving basic variable is determined by the constraint which imposes the greatest restriction on the increase in the entering basic variable. The entering basic variable is a non-basic variable with zero value in the current solution. Bringing the non-basic variable into the basis allows the variable's value to become positive. The amount by which the non-basic variable can increase must be restricted to ensure the new solution remains feasible (does not violate a constraint).

Each constraint, which has a positive coefficient in the entering basic variable's column, must be considered to determine which constraint imposes the greatest restriction on the value assumed by the entering basic variable. For each constraint a *test ratio* is computed. The test ratio equals the constraint's bound divided by the positive coefficient in the entering basic variable's column. The leaving basic variable

is the basic variable corresponding to the constraint with the smallest test ratio.

In Part (*iii*) the new basic feasible solution is determined. The new solution is defined in terms of the new basis: which consists of the entering basic variable and the previous basic variables (less the leaving basic variable). The new solution is obtained by using elementary row operations to convert the entering basic variable's column into a column with a single positive one coefficient and all other coefficients equal zero. The positive one coefficient will be located in the row corresponding to the most restrictive constraint, which corresponds to the leaving basic variable.

The highlighted column in Figure 5.6 identifies the selected entering basic variable to be dual variable u_{12}. Note that all non-basic variables have objective function coefficients equal to zero, one or negative one. Thus, there is no best entering basic variable. Each one results in the same incremental improvement of the objective function. Hence, any non-basic variable with a negative objective function coefficient may be selected.

The highlighted row in Figure 5.6 identifies the basic variable which must leave the basis, slack variable s_8. The test ratios computed for constraints with positive coefficients in the entering basic variable's column are: for the constraint relating to basic variable s_8, we have a test ratio of $1/1 = 1$; and, for the constraint relating to basic variable v_9, we have a test ratio of $2/1 = 2$. All other constraints have zero coefficients in the entering basic variables column. Thus, they impose no restriction on the value assumed by the entering basic variable. The smallest test ratio corresponds to the constraint associated with basic variable s_8. Hence, s_8 must be the leaving basic variable.

After completing each *Iteration Step*, the *Stopping Criteria* is evaluated to determine whether the current solution is optimal. The *Iteration Step* is repeated as

long as the current basic solution is not optimal.

Stopping Criteria. The Simplex Algorithm has reached the optimal solution if the objective function coefficient, of each non-basic variable, is non-negative. When each non-basic variable has a non-negative objective function coefficient, the solution must be optimal. Since it is not possible to improve the value of the objective function.

5.5 Comparison Of The Augmented Graphs

In this section, we compare the LG Algorithm's augmented graph, discussed in Chapter 2, with the augmented graph implied by the dual pit limit linear program. We then describe the relationship between dual decision variables and arc flows in the augmented graph; and the how the dual constraints force the net flow on arcs into a node to be equal to the absolute value of the node.

Figure 2.3 shows the LG Algorithm's augmented graph. This graph consists of the all nodes and all arcs found in the original directed graph, and the set of artificial arcs. Each artificial arc has the artificial root as it's source node and a real node as it's terminal node. The augmented graph, corresponding to the dual, contains the same components as the LG Algorithm's augmented graph; but, it also includes an additional set of artificial arcs. Each of these additional artificial arcs has the root node as it's terminal node and a real node as it's source node.

Figure 5.7 shows, for our example dual linear program, a partial graph defined with respect to the dual's augmented graph. Some of the artificial arcs in the augmented graph are not shown. As before, node designations are to the left of the node symbol and node values are placed inside the nodes. Arcs have been labeled with their associated dual variable. These dual variables define the amount of flow on the arc. Artificial arcs, arcs connected to the artificial root x_0, are curved. Real arcs,

arcs defined in the original directed graph, are straight.

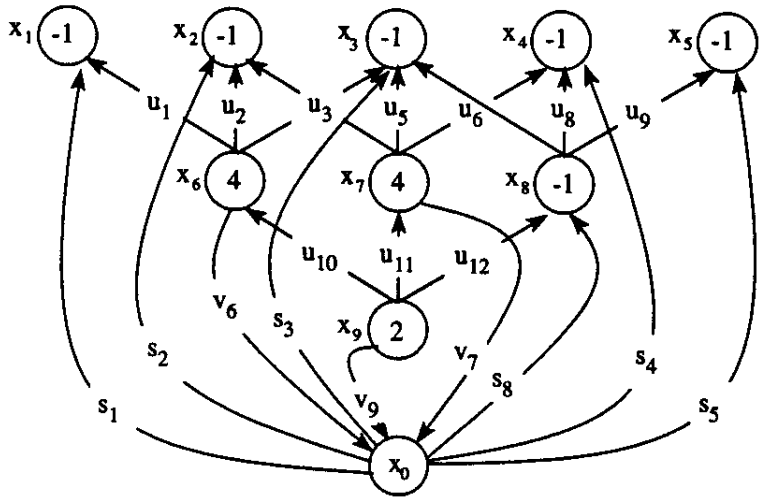


Figure 5.7: The Augmented Graph Corresponding to the Dual Pit Limit Linear Program

Each dual decision variable u_k corresponds to flow on a real arc in the original directed graph. If there are m arcs in the original directed graph, then there are m dual variables u_k. For example, dual decision variable u_1 corresponds to the flow on the real arc $a_{6,1}$. Similarly, dual decision variable u_{12} corresponds to the flow on the real arc $a_{9,8}$. Each dual decision variable, u_k, has a zero coefficient in the objective function. Thus, there is no "value added" associated with increasing flows on real arcs.

Each dual decision variable v_i corresponds to flow on an artificial arc $a_{i,0} = (x_i, x_0)$, which terminates at the artificial root x_0. Each dual decision variable s_i corresponds to flow on an artificial arc $a_{0,i} = (x_0, x_i)$, which originates at the artificial root x_0. The subscripts on these dual decision variables correspond to the real node in the artificial arc. Thus, dual decision variable v_5 corresponds to the artificial arc

$a_{5,0}$.

As stated above, Figure 5.7 does not contain all artificial arcs. The artificial arcs which are shown correspond to the basic variables contained in the initial basic feasible solution. Artificial arcs corresponding to non-basic variables are not shown.

In the Simplex Algorithm, the basis is defined in terms of the basic variables. Basic variables are the only variables whose value are not required to equal zero. All non-basic variables are required to have a value of zero. In Figure 5.8, we show the tree which is formed by the arcs corresponding to the basic variables. These are the only arcs which will have non-zero flows. The value of each decision variable indicates the flow on the corresponding arc. Thus, arc $a_{6,0}$, which corresponds to the basic variable v_6 has a flow of 4.

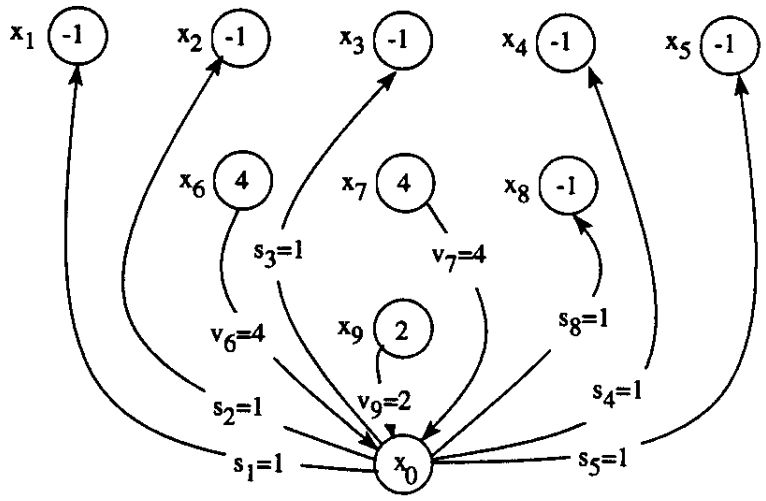


Figure 5.8: The Tree Corresponding To The Initial Basic Feasible Solution

The flow on each arc associated with a basic variable equals the absolute value of the *branch* supported by the arc. In Figure 5.8, each arc supports a branch containing a single node.

5.6 Equivalences Between The LG Algorithm And The Simplex Algorithm

In this section, we relate the steps of the LG Algorithm to steps in the Simplex Algorithm. Recall that the LG Algorithm consists of four steps: *InitNormalizeTree*, *MoveTowardFeasibility*, *NormalizeTree*, and *SolutionNotFeasible*. The *InitNormalizeTree* procedure defines the initial normalized tree. The *MoveTowardFeasibility* procedure replaces an arc in the normalized tree with an arc not contained in the normalized tree. When the transformation performed by the *MoveTowardFeasibility* procedure generates a non-normalized tree, the *NormalizeTree* procedure performs additional transformations, which replace arcs in the tree with artificial arcs not in the tree, until the tree is normalized. The *SolutionNotFeasible* procedure determines whether any additional transformations can be performed to improve the value of the current solution.

5.6.1 The InitNormalizedTree Procedure

In the *InitNormalizedTree* procedure, the initial normalized tree is generated. The standard approach defines the initial normalized tree using the artificial arcs. This approach guarantees the tree is normalized since all arcs are adjacent to the artificial root. Recall that a normalized tree is a tree where all strong arcs are adjacent to the artificial root.

In Section 5.4, the initial tableau of the dual pit limit linear program was developed. In Figure 5.8, the tree corresponding to the initial basic feasible solution is shown. As for the initial normalized tree, the tree associated with the initial basic feasible solution consists solely of artificial arcs; and, a single artificial arc connects each real node to the artificial root. Thus, the *InitNormalizedTree* procedure is equivalent

to finding the initial basic feasible solution in the Simplex Algorithm.

5.6.2 The MoveTowardFeasibility Procedure

In the *MoveTowardFeasibility* procedure an arc in the normalized tree is replaced with an arc not in the normalized tree. The introduced arc has a strong node as it's source node and a weak node as its terminal node.

Since each arc corresponds to a dual decision variable, the arc replacement above corresponds to a change of basis in the dual linear program. For example, consider the arc replacement performed in the initial iteration of the LG Algorithm, where arc $a_{9,8}$ replaces arc $a_{0,9}$. These arcs correspond to the highlighted column and row in Figure 5.9, respectively. In the initial tableau, the decision variable, u_{12}, defining the flow on arc $a_{9,8}$ is a non-basic variable, and the decision variable, s_9, defining the flow on arc $a_{0,9}$ is a basic variable. Thus, u_{12} is the entering basic variable and s_9 is the leaving basic variable.

	DUAL VARIABLES (u_i,v_i)																					SLACK VARIABLES (s_i)									
dual var.	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	
arc a_{ij}	a_{61}	a_{62}	a_{63}	a_{72}	a_{73}	a_{74}	a_{83}	a_{84}	a_{85}	a_{96}	a_{97}	a_{98}	a_{10}	a_{20}	a_{30}	a_{40}	a_{50}	a_{60}	a_{70}	a_{80}	a_{90}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	a_{06}	a_{07}	a_{08}	a_{09}	
source	1	6	6	6	7	7	7	8	8	8	9	9	9	1	2	3	4	5	6	7	8	9	0	0	0	0	0	0	0	0	0	
term.	j	1	2	3	2	3	4	3	4	5	6	7	8	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	9	
max.	Z	-1	-1	-1	-1	-1	-1	0	0	0	0	0	-1	1	1	1	1	1	0	0	1	0	0	0	0	0	0	1	1	0	1	=-10
BASIC VAR.																																
$s_1=a_{01}$	1	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	=	1
$s_2=a_{02}$	0	1	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	=	1
$s_3=a_{03}$	0	0	1	0	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	=	1
$s_4=a_{04}$	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	=	1
$s_5=a_{05}$	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	=	1
$v_6=a_{60}$	1	1	1	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	0	=	4
$v_7=a_{70}$	0	0	0	1	1	1	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	=	4
$s_8=a_{08}$	0	0	0	0	0	0	-1	-1	-1	0	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	=	1
$v_9=a_{90}$	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	-1	=	2

Figure 5.9: The Dual Tableau and The LG Algorithm's Initial Arc Transformation

The tableau will be updated by pivoting at the intersection of the highlighted row and highlighted column. The upper tableau in Figure 5.10, shows the resulting tableau. The change of basis introduces decision variable u_{12} into the basis; which is indicated by the u_{12} to the left of the last row, which previously related to the leaving basic variable s_9.

As a result of this change of basis we discover that the resulting dual basic solution is dual infeasible. The dual decision variable s_8 has a value of -1, which violates it's non-negativity constraint. In Section 5.7, we explain what caused the solution to become infeasible. To make the basic solution feasible a second change of basis must be performed.

We select decision variable v_8 to be the entering basic variable (denoted by the highlighted column in Figure 5.10), which requires s_8 to be the leaving basic variable (denoted by the highlighted row). The lower tableau shows the resulting tableau. The new basis contains v_8, which has a value of 1, and is feasible. This change of basis guarantees that the basic variables associated with artificial arcs agrees with the artificial arcs in the LG Algorithm's normalized tree.

The *MoveTowardFeasibility* procedure does not always require two changes of bases be performed. For example, consider the arc replacement performed in the second iteration of the LG Algorithm. In this iteration, the arc $a_{8,4}$ replaces arc $a_{0,8}$. The associated decision variables are indicated, by the highlighted row and column in the lower tableau of Figure 5.10. After the change of basis we obtain the tableau shown in Figure 5.11, which contains a feasible basic solution.

Thus, we see that the *MoveTowardFeasibility* procedure performs a change of basis. When the change of basis results in an infeasible basic solution, a second change of basis is performed. Frequently it requires two change of bases be performed.

DUAL VARIABLES (u_i,v_i)																					SLACK VARIABLES (s_i)										
dual var.	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	
ARCS ($a_{i,j}$)																															
arc a_{ij}	a_{61}	a_{62}	a_{63}	a_{72}	a_{73}	a_{74}	a_{83}	a_{84}	a_{85}	a_{96}	a_{97}	a_{98}	a_{10}	a_{20}	a_{30}	a_{40}	a_{50}	a_{60}	a_{70}	a_{80}	a_{90}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	a_{06}	a_{07}	a_{08}	a_{09}	
ARC NODES ($a_{i,j}=(x_i,x_j)$)																															
source i	6	6	6	7	7	7	8	8	8	9	9	9	1	2	3	4	5	6	7	8	9	0	0	0	0	0	0	0	0	0	
term. j	1	2	3	2	3	4	3	4	5	6	7	8	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	9	
max.	-1	-1	-1	-1	-1	-1	0	0	0	1	1	0	1	1	1	1	1	0	0	1	1	0	0	0	0	0	1	1	0	0	= -8
BASIC VAR.																															
$s_1=a_{01}$	1	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	= 1
$s_2=a_{02}$	0	1	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	= 1
$s_3=a_{03}$	0	0	1	0	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	= 1
$s_4=a_{04}$	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	= 1
$s_5=a_{05}$	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	= 1
$v_6=a_{60}$	1	1	1	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	0	= 4
$v_7=a_{70}$	0	0	0	1	1	1	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	= 4
$v_8=a_{80}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	0	0	0	0	0	0	0	0	-1	-1	0	0	0	0	0	0	0	1	1	= -1
$u_{12}=a_{98}$	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	= 2

DUAL VARIABLES (u_i,v_i)																					SLACK VARIABLES (s_i)										
dual var.	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	
ARCS ($a_{i,j}$)																															
arc a_{ij}	a_{61}	a_{62}	a_{63}	a_{72}	a_{73}	a_{74}	a_{83}	a_{84}	a_{85}	a_{96}	a_{97}	a_{98}	a_{10}	a_{20}	a_{30}	a_{40}	a_{50}	a_{60}	a_{70}	a_{80}	a_{90}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	a_{06}	a_{07}	a_{08}	a_{09}	
ARC NODES ($a_{i,j}=(x_i,x_j)$)																															
source i	6	6	6	7	7	7	8	8	8	9	9	9	1	2	3	4	5	6	7	8	9	0	0	0	0	0	0	0	0	0	
term. j	1	2	3	2	3	4	3	4	5	6	7	8	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	9	
max.	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	= -9
BASIC VAR.																															
$s_1=a_{01}$	1	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	= 1
$s_2=a_{02}$	0	1	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	= 1
$s_3=a_{03}$	0	0	1	0	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	= 1
$s_4=a_{04}$	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	= 1
$s_5=a_{05}$	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	= 1
$v_6=a_{60}$	1	1	1	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	0	= 4
$v_7=a_{70}$	0	0	0	1	1	1	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	= 4
$v_8=a_{80}$	0	0	0	0	0	0	-1	-1	-1	-1	-1	0	0	0	0	0	0	0	0	-1	-1	0	0	0	0	0	0	0	1	1	= -1
$u_{12}=a_{98}$	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	= 2

Figure 5.10: The Dual Tableau After The Initial LG Iteration

	DUAL VARIABLES (u_i, v_i)																				SLACK VARIABLES (s_i)									
dual var.	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	
	ARCS ($a_{i,j}$)																													
arc a_{ij}	a_{61}	a_{62}	a_{63}	a_{72}	a_{73}	a_{74}	a_{83}	a_{84}	a_{85}	a_{96}	a_{97}	a_{98}	a_{10}	a_{20}	a_{30}	a_{40}	a_{50}	a_{60}	a_{70}	a_{80}	a_{90}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	a_{06}	a_{07}	a_{08}	a_{09}	
	ARC NODES ($a_{i,j}=(x_i,x_j)$)																													
source i	6	6	6	7	7	7	8	8	8	9	9	9	1	2	3	4	5	6	7	8	9	0	0	0	0	0	0	0	0	
term. j	1	2	3	2	3	4	3	4	5	6	7	8	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	9	
max.	-1	-1	-1	-1	-1	-1	0	0	0	1	1	0	1	1	1	1	1	0	0	1	1	0	0	0	0	0	1	1	0	0	= -8
BASIC VAR.																															
$s_1=a_{01}$	1	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	= 1
$s_2=a_{02}$	0	1	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	= 1
$s_3=a_{03}$	0	0	1	0	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	= 1
$s_4=a_{04}$	0	0	0	0	0	1	-1	0	-1	-1	-1	0	0	0	0	-1	0	0	0	-1	-1	0	0	0	1	0	0	0	1	1	= 0
$s_5=a_{05}$	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	= 1
$v_6=a_{60}$	1	1	1	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	0	= 4
$v_7=a_{70}$	0	0	0	1	1	1	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	= 4
$u_8=a_{84}$	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	-1	-1	= 1
$u_{12}=a_{98}$	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	= 2

Figure 5.11: The Dual Tableau After The Second LG Iteration

5.6.3 The NormalizeTree Procedure

Similar to the *MoveTowardFeasibility* procedure, the *NormalizeTree* procedure replaces an arc in a non-normalized tree with an artificial arc not contained in the tree. Thus, the transformation again is equivalent to a change of basis. When the tree formed by the *MoveTowardFeasibility* procedure is normalized, no transformations are performed by the *NormalizeTree* procedure.

For example, consider the upper tableau in Figure 5.12. This tableau corresponds to the tree generated by the *MoveTowardFeasibility* procedure after the fourth iteration of the LG Algorithm. As in the previous section, the basic solution is infeasible. Thus, a second change of basis must be performed. The lower tableau shows the tableau formed after the second change of basis. But, the basic solution after this change of basis remains infeasible. The basic variable u_8 has a value of -3, which violates it's non-negativity constraint. Note that u_8 had the same value prior to the

second change of basis.

The basic solution in the lower tableau corresponds to the non-normalized tree $\hat{T}_5$ shown in Figure 2.12. The strong arc $a_{8,4}$ non-adjacent to the artificial root corresponds to the infeasible dual decision variable u_8. Thus, another change of basis is required. The change of basis is established by the arc replacement performed by the *NormalizeTree* procedure, arc $a_{8,4}$ is replaced by arc $a_{4,0}$. The associated leaving and entering basic variables are indicated by the highlighted row and column in the lower tableau of Figure 5.12.

Figure 5.13 shows the dual tableau after the change of basis. The resulting basic solution is feasible. Thus, the *NormalizeTree* procedure is required when the tree generated by the *MoveTowardFeasibility* procedure generates a dual basic solution which is infeasible. The *NormalizeTree* transformation makes the infeasible solution feasible.

5.6.4 The SolutionNotFeasible Procedure

The *SolutionNotFeasible* procedure determines whether there exists an arc in the augmented graph, which is not in the normalized tree, which has a strong source node and a weak terminal node. When such an arc exists, the set of strong nodes cannot define a closure in the directed graph, and the current solution cannot be feasible. When such an arc does not exist, the current solution is feasible, and the optimal solution has been reached.

In the Simplex Algorithm, the optimal solution has been obtained when all coefficients in the objective function row have non-positive value. Inspection of the lower tableau in Figure 5.14, which corresponds to the final normalized tree generated by the LG Algorithm, indicates an optimal solution has been identified. Thus, the

DUAL VARIABLES (u_i,v_i)																					SLACK VARIABLES (s_i)									
dual var.	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9

ARCS ($a_{i,j}$)																														
arc a_{ij}	a_{61}	a_{62}	a_{63}	a_{72}	a_{73}	a_{74}	a_{83}	a_{84}	a_{85}	a_{96}	a_{97}	a_{98}	a_{10}	a_{20}	a_{30}	a_{40}	a_{50}	a_{60}	a_{70}	a_{80}	a_{90}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	a_{06}	a_{07}	a_{08}	a_{09}

ARC NODES ($a_{i,j}=(x_i,x_j)$)																														
source	i	6	6	6	7	7	7	8	8	8	9	9	9	1	2	3	4	5	6	7	8	9	0	0	0	0	0	0	0	0	0	
term.	j	1	2	3	2	3	4	3	4	5	6	7	8	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	9	
max.		-1	-1	-1	0	0	0	0	0	0	1	0	0	1	1	1	1	1	0	1	1	1	0	0	0	0	0	1	0	0	0	= -4
BASIC VAR.																																
$s_1=a_{01}$		1	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	= 1
$s_2=a_{02}$		0	1	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	= 1
$s_3=a_{03}$		0	0	1	0	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	= 1
$u_9=a_{85}$		0	0	0	1	1	0	1	0	1	1	0	0	0	0	0	1	0	0	1	1	1	0	0	0	-1	0	0	-1	-1	-1	= 4
$s_5=a_{05}$		0	0	0	-1	-1	0	-1	0	0	-1	0	0	0	0	0	-1	-1	0	-1	-1	-1	0	0	0	1	1	0	1	1	1	= -3
$v_6=a_{60}$		1	1	1	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	0	= 4
$v_7=a_{70}$		0	0	0	1	1	1	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	= 4
$u_8=a_{84}$		0	0	0	-1	-1	0	0	1	0	0	1	0	0	0	0	-1	0	0	-1	0	0	0	0	0	1	0	0	1	0	0	= -3
$u_{12}=a_{98}$		0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	= 2

DUAL VARIABLES (u_i,v_i)																					SLACK VARIABLES (s_i)									
dual var.	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9

ARCS ($a_{i,j}$)																														
arc a_{ij}	a_{61}	a_{62}	a_{63}	a_{72}	a_{73}	a_{74}	a_{83}	a_{84}	a_{85}	a_{96}	a_{97}	a_{98}	a_{10}	a_{20}	a_{30}	a_{40}	a_{50}	a_{60}	a_{70}	a_{80}	a_{90}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	a_{06}	a_{07}	a_{08}	a_{09}

ARC NODES ($a_{i,j}=(x_i,x_j)$)																														
source	i	6	6	6	7	7	7	8	8	8	9	9	9	1	2	3	4	5	6	7	8	9	0	0	0	0	0	0	0	0	0	
term.	j	1	2	3	2	3	4	3	4	5	6	7	8	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	9	
max.		-1	-1	-1	-1	-1	0	-1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	= -7
BASIC VAR.																																
$s_1=a_{01}$		1	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	= 1
$s_2=a_{02}$		0	1	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	= 1
$s_3=a_{03}$		0	0	1	0	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	= 1
$u_9=a_{85}$		0	0	0	1	1	0	1	0	1	1	0	0	0	0	0	1	0	0	1	1	1	0	0	0	-1	0	0	-1	-1	-1	= 4
$v_5=a_{50}$		0	0	0	1	1	0	1	0	0	1	0	0	0	0	0	1	1	0	1	1	1	0	0	0	-1	-1	0	-1	-1	-1	= 3
$v_6=a_{60}$		1	1	1	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	0	= 4
$v_7=a_{70}$		0	0	0	1	1	1	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	= 4
$u_8=a_{84}$		0	0	0	-1	-1	0	0	1	0	0	1	0	0	0	0	-1	0	0	-1	0	0	0	0	0	1	0	0	1	0	0	= -3
$u_{12}=a_{98}$		0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	= 2

Figure 5.12: The Dual Tableau After The Fourth LG Iteration

	DUAL VARIABLES (u_i, v_i)																				SLACK VARIABLES (s_i)									
dual var.	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	
	ARCS ($a_{i,j}$)																													
arc a_{ij}	a_{61}	a_{62}	a_{63}	a_{72}	a_{73}	a_{74}	a_{83}	a_{84}	a_{85}	a_{96}	a_{97}	a_{98}	a_{10}	a_{20}	a_{30}	a_{40}	a_{50}	a_{60}	a_{70}	a_{80}	a_{90}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	a_{06}	a_{07}	a_{08}	a_{09}	
	ARC NODES ($a_{i,j}=(x_i, x_j)$)																													
source i	6	6	7	7	7	8	8	8	9	9	9	1	2	3	4	5	6	7	8	9	0	0	0	0	0	0	0	0	0	
term. j	1	2	3	2	3	4	3	4	5	6	7	8	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	9	
max.	-1	-1	-1	-1	-1	0	-1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1 = -7
BASIC VAR.																														
$s_1=a_{01}$	1	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0 = 1
$s_2=a_{02}$	0	1	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0 = 1
$s_3=a_{03}$	0	0	1	0	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0 = 1
$u_9=a_{85}$	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	-1	-1 = 1
$v_5=a_{50}$	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	-1	0	0	-1	-1 = 0
$v_6=a_{60}$	1	1	1	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	0 = 4
$v_7=a_{70}$	0	0	0	1	1	1	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0 = 4
$v_4=a_{40}$	0	0	0	1	1	0	0	-1	0	0	-1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	-1	0	0	-1	0 = 3
$u_{12}=a_{98}$	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-1	0 = 2

Figure 5.13: The Dual Tableau After The *NormalizeTree* Change of Basis

stopping criteria in the LG Algorithm is achieved when the Simplex Algorithm's stopping criteria has been met.

5.7 Relationships Between Normalized Trees And Dual Tableaus

In this section we look in greater detail at the relationships between normalized trees in the LG Algorithm and the decision variables and coefficient values of the associated dual tableau.

5.7.1 Value Relationship Between Solutions

As shown in Table 5.3, the value of the strong nodes in the initial normalized tree equals ten. Figure 5.6 shows the dual tableau corresponding to the initial basic feasible solution developed in Section 5.4. In the development of the dual linear program's formulation we discovered only the dual variables, v_i, had non-zero objective function

	DUAL VARIABLES (u_i, v_i)																				SLACK VARIABLES (s_i)									
dual var.	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	
	ARCS ($a_{i,j}$)																													
arc a_{ij}	a_{61}	a_{62}	a_{63}	a_{72}	a_{73}	a_{74}	a_{83}	a_{84}	a_{85}	a_{96}	a_{97}	a_{98}	a_{10}	a_{20}	a_{30}	a_{40}	a_{50}	a_{60}	a_{70}	a_{80}	a_{90}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	a_{06}	a_{07}	a_{08}	a_{09}	
	ARC NODES ($a_{i,j}=(x_i,x_j)$)																													
source i	6	6	6	7	7	7	8	8	8	9	9	9	1	2	3	4	5	6	7	8	9	0	0	0	0	0	0	0	0	
term. j	1	2	3	2	3	4	3	4	5	6	7	8	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	9	
max.	0	0	0	-1	-1	0	-1	0	0	-1	0	0	1	1	1	0	0	1	0	0	0	0	0	0	0	1	1	0	1	1	1 = -3
BASIC VAR.																															
$u_2=a_{62}$	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0 = 1
$v_2=a_{20}$	0	0	1	-1	0	0	0	0	0	-1	0	0	1	1	0	0	0	1	0	0	0	-1	-1	0	0	0	-1	0	0	0 = 2
$s_3=a_{03}$	0	0	0	1	1	0	1	0	0	1	0	0	-1	-1	-1	0	0	-1	0	0	0	1	1	1	0	0	1	0	0	0 = -1
$u_9=a_{85}$	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	-1	-1	= 1
$v_5=a_{50}$	0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0	1	0	0	1	1	0	0	0	0	-1	0	0	-1	-1 = 0
$u_1=a_{61}$	1	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0 = 1
$v_7=a_{70}$	0	0	0	1	1	1	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0 0 = 4
$v_4=a_{40}$	0	0	0	1	1	0	0	-1	0	0	-1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	-1	0	0	-1 0 = 3
$u_{12}=a_{98}$	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0 -1 = 2

	DUAL VARIABLES (u_i, v_i)																				SLACK VARIABLES (s_i)									
dual var.	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	
	ARCS ($a_{i,j}$)																													
arc a_{ij}	a_{61}	a_{62}	a_{63}	a_{72}	a_{73}	a_{74}	a_{83}	a_{84}	a_{85}	a_{96}	a_{97}	a_{98}	a_{10}	a_{20}	a_{30}	a_{40}	a_{50}	a_{60}	a_{70}	a_{80}	a_{90}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	a_{06}	a_{07}	a_{08}	a_{09}	
	ARC NODES ($a_{i,j}=(x_i,x_j)$)																													
source i	6	6	6	7	7	7	8	8	8	9	9	9	1	2	3	4	5	6	7	8	9	0	0	0	0	0	0	0	0	
term. j	1	2	3	2	3	4	3	4	5	6	7	8	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	9	
max.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1 = -4
BASIC VAR.																														
$u_2=a_{62}$	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0 = 1
$v_2=a_{20}$	0	0	1	-1	0	0	0	0	0	-1	0	0	1	1	0	0	0	1	0	0	0	-1	-1	0	0	0	-1	0	0	0 = 2
$s_3=a_{03}$	0	0	0	-1	-1	0	-1	0	0	-1	0	0	1	1	1	0	0	1	0	0	0	-1	-1	-1	0	0	-1	0	0	0 = 1
$u_9=a_{85}$	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	-1	-1	= 1
$v_5=a_{50}$	0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0	1	0	0	1	1	0	0	0	0	-1	0	0	-1	-1 = 0
$u_1=a_{61}$	1	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0 = 1
$v_7=a_{70}$	0	0	0	1	1	1	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0 0 = 4
$v_4=a_{40}$	0	0	0	1	1	0	0	-1	0	0	-1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	-1	0	0	-1 0 = 3
$u_{12}=a_{98}$	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0 -1 = 2

Figure 5.14: The Dual Tableau After The Eigth LG Iteration

coefficients. Thus, they were the only dual variables which could affect it's value. The initial basic feasible solution contains three such variables, $\{v_6, v_7, v_9\}$. The values of these dual variables are 4, 4 and 2, respectively, giving an objective function value of ten. Hence, we see the value of the strong nodes in the initial normalized tree equals the value of the objective function in the initial basic feasible solution.

In each tableau corresponding to a normalized tree, the value of the strong nodes equals the value of the objective function. Comparing objective function values, in Figures 5.10-5.14, with the value of the strong nodes in Table 5.3, we see this relationship always holds. For example, consider Figure 5.10. The tableau on the bottom corresponds to the normalized tree T_2, in Table 5.3. The objective function shows a value of -9. Because we transformed the objective function to place the coefficients on the same side as the objective function value, Z, we must negate the value shown on the right hand side. Thus, the objective function has a value of 9, as does the set of strong nodes in the normalized tree T_2.

5.7.2 Node Strength Classification Relationship

The strength classification of nodes in each normalized tree corresponds to objective function coefficient values for slack variables in the dual linear program. Linear programming theory, see (Luenberger 1982), has proven that the value of a slack variable's objective function coefficient equals the value of its associated primal decision variable. In the context of the dual pit limit linear program, each slack variable corresponds to some node in the directed graph. By the construction of the the dual linear program's decision variable names and subscripts, we know each slack variable s_i corresponds to node x_i. Thus, the value of each slack variable's objective function coefficient determines the value of the decision variable relating to that slack variable.

Hence, slack variable s_i's objective function coefficient determines the value of primal decision variable x_i.

In the lower tableau of Figure 5.10, four of the slack variables have an objective function coefficient value equal to positive one. The remaining slack variables have an objective function coefficient value equal to zero. The four with positive one coefficients are $\{s_6, s_7, s_8, s_9\}$. These slack variables correspond to decision variables, and nodes, $\{x_6, x_7, x_8, x_9\}$. Thus, in the context of the pit limit linear program, only these four nodes are contained in the current pit limit. From Table 5.3, these four nodes are the only strong nodes in normalized tree T_2.

Comparing the slack variable objective function coefficients, in Figures 5.10-5.14, with the set of strong nodes in Table 5.3, we see this relationship holds for each tableau associated with a normalized tree.

5.7.3 Relationship Between Arcs Available For Introduction And Objective Function Coefficients Of u_i

The LG Algorithm selects the arc to introduce into the normalized tree from those arcs, not currently in the tree, which have a strong source node and a weak terminal node. By introducing such an arc, the LG Algorithm attempts to make the normalized tree's solution (determined by its set of strong nodes) feasible. Of these arcs, the arc selected is arbitrary.

For example, consider the augmented graph and the normalized tree T_2, shown in Figures 2.7 and 2.9, respectively. We see there exists the following nine arcs, not contained in the normalized tree, where each arc has a strong source node and a weak terminal node

$$\{a_{6,1}, a_{6,2}, a_{6,3}, a_{7,2}, a_{7,3}, a_{7,4}, a_{8,3}, a_{8,4}, a_{8,5}\}.$$

Each of these arcs represent a direct dependency between a strong node and a weak node. The LG Algorithm selects one of these arcs to introduce into the normalized tree.

Comparing these arcs to the objective function coefficient values of their associated dual decision variables, we find we can identify the same set. Consider the lower tableau in Figure 5.10. This tableau corresponds to the normalized tree T_2. Each dual decision variable, u_i, with a negative one coefficient in the objective function, corresponds to one of the arcs

$$\{a_{6,1}, a_{6,2}, a_{6,3}, a_{7,2}, a_{7,3}, a_{7,4}, a_{8,3}, a_{8,4}, a_{8,5}\}.$$

All remaining dual decision variables, u_i, have a zero coefficient in the objective function.

Comparing the dual decision variables, u_i, objective function coefficients, in Figures 5.10-5.14, with the set of arcs, with a strong source node and a weak terminal node, in each normalized tree we see this relationship holds for each tableau associated with a normalized tree.

5.7.4 Entering Arc Selection Rule

As was identified above, dual decision variables, u_i, will have either a negative one or a zero coefficient value in the objective function. Inspection of the coefficient value's for the remaining decision variables, v_i and s_i, indicates they will have values of either zero or positive one. Thus, in the linear programming context, the choice for entering basic variable will be one of the decision variables, u_i. Specifically, one selected from among those decision variables, u_i, with a negative one coefficient. This

is in accordance with the Simplex Algorithm's rule for selecting the entering basic variable.

Since each available entering basic variable has the same objective function coefficient value, the choice of which one to select is arbitrary. This agrees with the LG Algorithm's arbitrary rule for selecting the arc to introduce into the normalized tree.

5.7.5 Leaving Arc Selection Rule Difference

Introducing an arc into the normalized tree, as discussed above, forms a graph which contains a cycle. Since the graph contains a cycle it cannot be a tree. To convert the graph to a tree, an arc must be removed. The LG Algorithm's rule, for selecting the leaving arc, is to remove the artificial arc between the artificial root and the root of the branch containing the strong node. Here, the strong node refers to the strong node which is the source node of the arc introduced into the tree. Removing this arc will convert the graph to a tree.

Adding an arc to the normalized tree, in the linear programming context, corresponds to adding a decision variable to the basis. Removing an arc corresponds to removing a decision variable from the basis. Since each arc in the LG Algorithm corresponds to a decision variable in the linear program, we call: the decision variable entering the basis, the entering basic variable, and the decision variable leaving the basis, the leaving basic variable. This terminology is standard in linear programming.

The Simplex Algorithm uses a more sophisticated rule for selecting the leaving basic variable. It chooses the decision variable based upon the most restrictive constraint (see section 5.4). An easy way to identify the most restrictive constraint is to investigate the cycle formed by the entering basic variable. First we demonstrate

the affects of the steps performed by the LG Algorithm. Then we show the Simplex Algorithm's rule for selecting the leaving basic variable would result in the same tree.

Consider the first iteration of the LG Algorithm, where arc $a_{9,8}$ is introduced into the normalized tree. Introducing this arc forms the cycle shown in Figure 5.15. All other arcs in the normalized tree have been removed from the figure. The following discussion uses the decision variable notation rather than the arc notation.

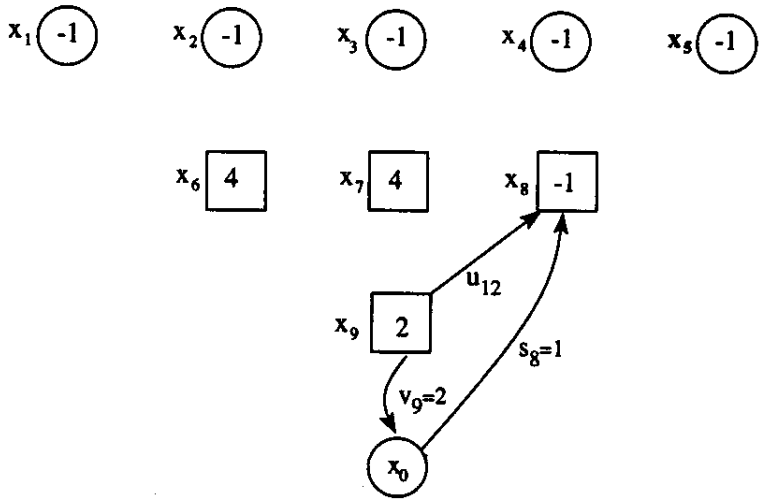


Figure 5.15: The Cycle Formed By Introducing Arc $a_{9,8}$

The LG Algorithm chooses to remove the arc corresponding to decision variable v_9. Removing this arc forces the flow on arcs u_{12} and s_8 to be 2 and -1, respectively. Recall that the dual constraints require net flows at a node to equal the absolute value of the node. Since node x_9 has an absolute value of 2, and only one arc connected to it, that arc must carry a flow of 2. Thus, decision variable u_{12} has a value of 2. Node x_8, which has an absolute value of 1, has arcs u_{12} and s_8 connected to it. Thus, the

flow on arc s_8 must be negative 1:

$$\begin{aligned} |\nu(x_8)| &= u_{12} + s_8, \\ 1 &= 2 + s_8, \\ s_8 &= -1. \end{aligned}$$

The value of these decision variables agree with those found in the upper tableau in Figure 5.10.

Arc s_8's negative flow violates the dual linear program's non-negativity constraints. Thus, the solution is dual infeasible. To make the solution dual feasible, another change of basis is performed. In the second change of basis, s_8 is the leaving basic variable and v_8 is the entering basic variable. This change of basis effectively reverses the direction of the arc connecting nodes x_0 and x_8. As shown in Figure 5.16, the flow on arc v_8 equals 1. Hence, the new basis is dual feasible.

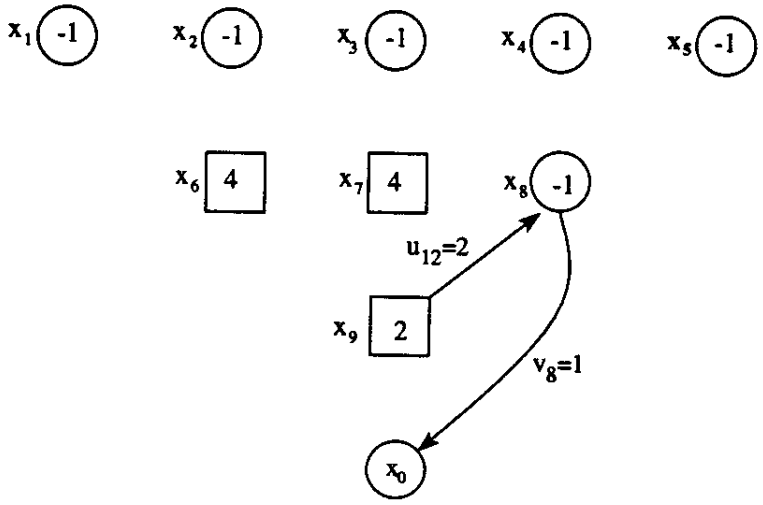


Figure 5.16: Arc Flows In The Cycle Formed By Introducing Arc $a_{9,8}$

Using the Simplex Algorithm's rule for identifying the leaving basic variable this basis would have been obtained in a single change of basis. Consider the cycle shown in Figure 5.15. The flow on arcs u_{12}, v_9 and s_8 can only increase, decrease, and decrease, respectively. The flows on the decreasing arcs, v_9 and s_8, are 2 and 1, respectively. Thus, arc s_8 has the more restrictive flow. Hence, the Simplex Algorithm's rule would select decision variable s_8 to be the leaving basic variable. The new basis would contain arcs u_{12} and v_9, each having a flow of 1. The net flows at node x_8 would be 2. The net flow at node x_9 would be 1.

The LG Algorithm's choice of arc to remove does not always result in an infeasible dual solution. Consider Figure 5.11, which shows the tableau formed after the LG Algorithm's second iteration. In this iteration, the arc $a_{8,4}$ replaces arc $a_{0,8}$. As the figure shows, all decision variables have non-negative values. Thus, meeting the dual linear program's non-negativity constraints.

Identifying the cycle formed when an arc is introduced into a normalized tree; and, eliminating the arc, among those arcs whose flow must be decreased, may provide a more efficient alternative to the LG Algorithm. Unfortunately, time did not permit implementing such an algorithm.

5.7.6 Branch Value Relationship

Each decision variable in the basis corresponds to an arc in the normalized tree. The value of each decision variable equals the absolute value of the nodes in the branch supported by the arc.

For example, consider the tableau shown in Figure 5.11 and its corresponding normalized tree, shown in Figure 2.10. Consider the arc $a_{9,8}$, which corresponds to decision variable u_{12}. This arc supports a branch whose value equals 2. The decision

variable has a value of 2. Similarly, the arc $a_{8,4}$, which corresponds to decision variable u_8, supports a branch whose value equals 1. The decision variable u_8 also has a value of 1. Comparing these figures we see this property holds for all arcs supporting positive valued branches. For arcs supporting negative valued branches, the decision variable equals the absolute value of the branch. For example, the arc $a_{0,1}$ supports a branch with a value of -1. Its associated decision variable, s_1, has a value of +1.

Comparison of each normalized tree and its associated tableau demonstrates this relationship always holds.

5.8 A Complete Example

In this section we show each tableau (not previously shown) generated by the *MoveTowardFeasibility* and *NormalizeTree* procedures.

Table 5.3 shows, for each tree formed by the LG Algorithm: the set of strong nodes, the value of the set of strong nodes, the arc selected to enter the normalized tree, and the arc selected to leave the normalized tree. Recall the tree $\hat{T}_5$ was a non-normalized tree. We shall use the selected arcs to determine the change of bases performed in the dual linear program.

In the first iteration arc $a_{9,8}$ replaces arc $a_{0,9}$. The corresponding entering and leaving basic variables are identified by the highlighted column and highlighted row in Figure 5.9. Note that the leaving basic variable does not agree with the leaving basic variable which would have been selected by the Simplex Algorithm (basic variable s_8 has a smaller test ratio than basic variable v_9). Performing the change of basis anyway, we obtain the upper tableau shown in Figure 5.10.

As a result of selecting the incorrect leaving basic variable the solution becomes infeasible. The dual variable s_8 has a negative value which violates it's non-negativity

Table 5.3: Example Problem Normalized Tree Transformations

Tree T_i	Strong Nodes S	Value $\nu(S)$	Entering Arc $a_{s,w}$	Leaving Arc a_{0,r_s}
T_1	$\{x_6, x_7, x_9\}$	10	$a_{9,8}$	$a_{0,9}$
T_2	$\{x_6, x_7, x_8, x_9\}$	9	$a_{8,4}$	$a_{0,8}$
T_3	$\{x_6, x_7\}$	8	$a_{7,4}$	$a_{0,7}$
T_4	$\{x_4, x_6, x_7, x_8, x_9\}$	7	$a_{8,5}$	$a_{0,4}$
$\hat{T}_5$	$\{x_4, x_5, x_6, x_7, x_8, x_9\}$	7	$a_{0,4}$	$a_{8,4}$
T_5	$\{x_4, x_6, x_7\}$	7	$a_{6,1}$	$a_{0,6}$
T_6	$\{x_1, x_4, x_6, x_7\}$	6	$a_{6,2}$	$a_{0,1}$
T_7	$\{x_1, x_2, x_6, x_7\}$	5	$a_{6,3}$	$a_{0,2}$
T_8	$\{x_1, x_2, x_3, x_6, x_7\}$	4	—	—

constraint. The solution is brought back into feasibility by replacing dual variable s_8 with dual variable v_8. The lower tableau in Figure 5.10 shows the resulting tableau which has a feasible basic solution. Thus, the first iteration of the LG Algorithm corresponds to two change of bases in the dual linear program. The first change of basis results in an infeasible solution, the second change of basis brings the dual basic solution back into feasibility.

The same relationships as identified for the initial tableau hold for the lower tableau shown in Figure 5.10. The objective function value agrees with the value of the set of strong nodes. Slack variables with positive coefficients correspond to the set of strong nodes. Dual variables u_i with negative objective function coefficients agree with the set of arcs with a strong source node and a weak terminal node.

An additional relationship (not discussed with respect to the initial tableau, although it is evident there as well) is the value of the arcs corresponding to basic variables. Each basic variable has a value equal to the value of the branch supported by the arc. For example, dual variable v_8 has a value of 1. Which corresponds to

the value of the branch supported by the arc $a_{8,0}$. Similarly, dual variable u_{12} has a value of 2 which corresponds to the value of node x_9, the only node in the branch supported by arc $a_{9,8}$. Thus, the value of basic variables corresponds to the value of the branch supported by the basic variable's associated arc.

The second iteration of the LG Algorithm replaces arc $a_{8,0}$ with arc $a_{8,4}$. Figure 5.11 shows the tableau obtained after performing the corresponding change of basis. From Figure 5.10, we see there was a choice in leaving basic variable (both basic variables s_4 or v_8 have an equal test ratio). Thus, the resulting solution is feasible and a second change of basis is not required.

The third iteration of the LG Algorithm replaces arc $a_{7,0}$ with arc $a_{7,4}$. Again the choice of leaving basic variable results in an infeasible solution (the upper tableau of Figure 5.17). A second change of basis is performed to bring the basic solution back into feasibility (the lower tableau).

The fourth iteration of the LG Algorithm replaces arc $a_{4,0}$ with arc $a_{8,5}$, which results in a non-normalized tree $\hat{T}_5$. The upper tableau in Figure 5.12 shows the resulting tableau. The new solution has two negative basic variables s_5 and u_8, which relate to arcs $a_{0,5}$ and $a_{8,4}$. Bringing the non-basic variable s_5 into the basis is performed to make the new basic solution feasible. But, since two basic variables have negative values a third change of basis must be performed.

The third change of basis corresponds to the normalization transformation in the LG Algorithm. Recall that arc $a_{8,4}$ was replaced by arc $a_{0,4}$ in the normalization transformation. The equivalent change of basis replaces the basic variable u_8 with the non-basic variable v_4. Figure 5.18 shows the resulting tableau, which corresponds to the normalized tree T_5 with one exception.

The set of strong nodes, determined by the slack variable objective function

	DUAL VARIABLES (u_i, v_j)																				SLACK VARIABLES (s_i)									
dual var.	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9
arc a_{ij}	a_{61}	a_{62}	a_{63}	a_{72}	a_{73}	a_{74}	a_{83}	a_{84}	a_{85}	a_{96}	a_{97}	a_{98}	a_{10}	a_{20}	a_{30}	a_{40}	a_{50}	a_{60}	a_{70}	a_{80}	a_{90}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	a_{06}	a_{07}	a_{08}	a_{09}
source i	6	6	6	7	7	7	8	8	8	9	9	9	1	2	3	4	5	6	7	8	9	0	0	0	0	0	0	0	0	0
term. j	1	2	3	2	3	4	3	4	5	6	7	8	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	9
max.	-1	-1	-1	0	0	0	0	0	0	1	0	0	1	1	1	1	1	0	1	1	1	0	0	0	0	0	1	0	0	0 = -4
BASIC VAR.																														
$s_1=a_{01}$	1	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0 = 1
$s_2=a_{02}$	0	1	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0 = 1
$s_3=a_{03}$	0	0	1	0	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0 = 1
$v_4=a_{74}$	0	0	0	-1	-1	0	-1	0	-1	-1	0	0	0	0	0	0	-1	0	0	-1	-1	-1	0	0	0	1	0	0	1	1 = -4
$s_5=a_{05}$	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0 = 1
$v_6=a_{60}$	1	1	1	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0 = 4
$u_6=a_{74}$	0	0	0	1	1	1	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0 = 4
$u_8=a_{84}$	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	-1	-1 = 1
$u_{12}=a_{98}$	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-1 = 2

	DUAL VARIABLES (u_i, v_j)																				SLACK VARIABLES (s_i)									
dual var.	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9
arc a_{ij}	a_{61}	a_{62}	a_{63}	a_{72}	a_{73}	a_{74}	a_{83}	a_{84}	a_{85}	a_{96}	a_{97}	a_{98}	a_{10}	a_{20}	a_{30}	a_{40}	a_{50}	a_{60}	a_{70}	a_{80}	a_{90}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	a_{06}	a_{07}	a_{08}	a_{09}
source i	6	6	6	7	7	7	8	8	8	9	9	9	1	2	3	4	5	6	7	8	9	0	0	0	0	0	0	0	0	0
term. j	1	2	3	2	3	4	3	4	5	6	7	8	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	9
max.	-1	-1	-1	-1	-1	0	-1	0	-1	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0	1	0	1	1	1	1 = -8
BASIC VAR.																														
$s_1=a_{01}$	1	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0 = 1
$s_2=a_{02}$	0	1	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0 = 1
$s_3=a_{03}$	0	0	1	0	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0 = 1
$v_4=a_{40}$	0	0	0	1	1	0	1	0	1	1	0	0	0	0	0	1	0	0	1	1	1	0	0	0	-1	0	0	-1	-1	-1 = 4
$s_5=a_{05}$	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0 = 1
$v_6=a_{60}$	1	1	1	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0 = 4
$v_7=a_{70}$	0	0	0	1	1	1	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0 = 4
$u_8=a_{84}$	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	-1	-1 = 1
$u_{12}=a_{98}$	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-1 = 2

Figure 5.17: The Dual Tableau After The Third LG Iteration

	DUAL VARIABLES (u_i, v_i)																				SLACK VARIABLES (s_i)									
dual var.	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	
	ARCS ($a_{i,j}$)																													
arc a_{ij}	a_{61}	a_{62}	a_{63}	a_{72}	a_{73}	a_{74}	a_{83}	a_{84}	a_{85}	a_{96}	a_{97}	a_{98}	a_{10}	a_{20}	a_{30}	a_{40}	a_{50}	a_{60}	a_{70}	a_{80}	a_{90}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	a_{06}	a_{07}	a_{08}	a_{09}	
	ARC NODES ($a_{i,j}=(x_i,x_j)$)																													
source i	6	6	6	7	7	7	8	8	8	9	9	9	1	2	3	4	5	6	7	8	9	0	0	0	0	0	0	0	0	0	
term. j	1	2	3	2	3	4	3	4	5	6	7	8	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	9	
max.	-1	-1	-1	-1	-1	0	-1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	= -7
BASIC VAR.																															
$s_1=a_{01}$	1	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	= 1
$s_2=a_{02}$	0	1	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	= 1
$s_3=a_{03}$	0	0	1	0	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	= 1
$u_9=a_{85}$	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	-1	-1	= 1
$v_5=a_{50}$	0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	-1	0	0	-1	= 0
$v_6=a_{60}$	1	1	1	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	0	= 4
$v_7=a_{70}$	0	0	0	1	1	1	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	= 4
$v_4=a_{40}$	0	0	0	1	1	0	0	-1	0	0	-1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	-1	0	0	-1	0	= 3
$u_{12}=a_{98}$	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	= 2

Figure 5.18: The Dual Tableau After The Fifth LG Iteration

coefficient's, does not agree with the set of strong nodes in the normalized tree T_5. In the normalized tree T_5, the p-arc $a_{0,5}$ supports the branch $B(x_5) = \{x_5, x_8, x_9\}$ which has zero value. Thus, the LG Algorithm classifies the arc and all nodes in the arc as weak. In the dual tableau, we see the objective function coefficients of the slack variables associated with nodes $\{x_5, x_8, x_9\}$ are one, which implies these nodes are strong. Recall that the LG Algorithm converges to the smallest maximum valued closure in the graph. Using the alternative arc strength classifications discussed in Section 4.3, the LG Algorithm can be forced to converge to the largest maximum valued closure in the graph. Using the alternative arc strength classifications, the nodes in branch $B(x_4)$ would be classified as strong, which would agree with their classification in the dual tableau. Thus, we conclude the dual linear program converges to some maximum valued closure in the graph (which one is uncertain), whereas the LG Algorithm converges to the smallest maximum valued closure.

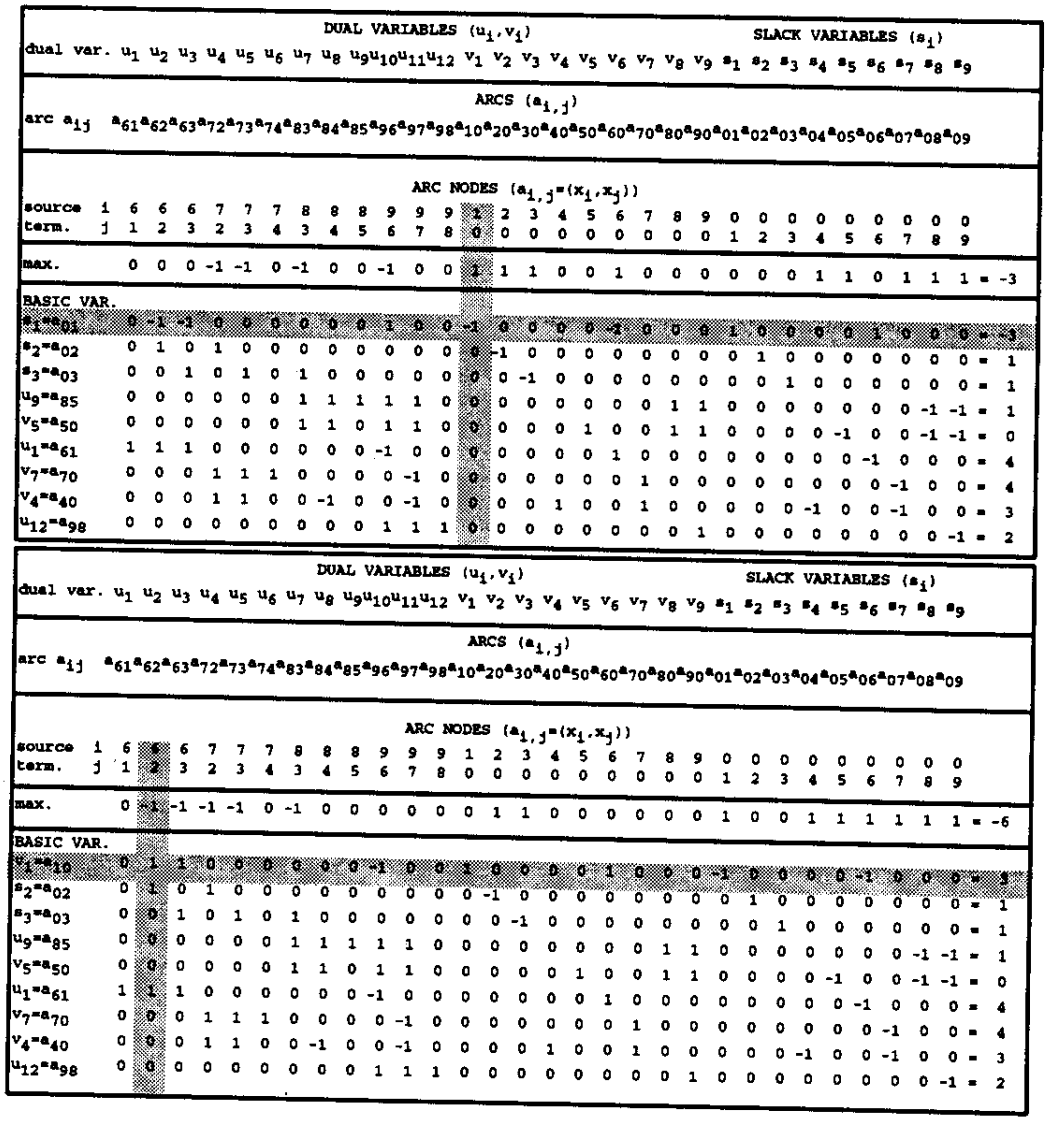
Figures 5.19,5.20, and 5.14 show the tableaus which correspond to the final three iterations of the LG Algorithm. Each iteration of the LG Algorithm corresponded to two change of bases in the dual. The bottom tableau of Figure 5.14 is the final tableau. In this tableau, all objective function coefficients are non-negative. Thus, indicating the optimal solution has been obtained.

Thus, we have demonstrated that the LG Algorithm is equivalent to solving the primal pit limit linear program's dual. The LG Algorithm defines a normalized tree which corresponds to the dual's initial basic feasible solution. The *MoveTowardFeasibility* procedure replaces an arc in the normalized tree with an arc between a strong node and a weak node. The transformation of the normalized tree corresponds to one or two change of bases in the dual linear program. Two change of bases are required when the dual basic solution is infeasible; with the second change of basis making the basic solution feasible. If the resulting tree is non-normalized, the *NormalizeTree* procedure performs an additional normalizing transformation. The normalizing transformation corresponds to an additional change of basis, which is required when the solution remains infeasible after the second transformation. Thus, each tree defined by the LG Algorithm corresponds to a basis in the dual linear program; where the arcs defining the normalized tree correspond to the dual's basic variables.

5.9 Conclusions

Since the Pit Limit Problem can be stated as a linear program it can be solved using the Simplex Algorithm. Thus, the reader may ask, why use the LG Algorithm? The primary justification for using the LG Algorithm is that it is more efficient, in terms of memory and processing requirements.

To demonstrate, consider a directed graph consisting of n nodes and $m = kn$



	DUAL VARIABLES (u_i, v_i)																			SLACK VARIABLES (s_i)								
dual var.	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9

	ARCS ($a_{i,j}$)																													
arc a_{ij}	a_{61}	a_{62}	a_{63}	a_{72}	a_{73}	a_{74}	a_{83}	a_{84}	a_{85}	a_{96}	a_{97}	a_{98}	a_{10}	a_{20}	a_{30}	a_{40}	a_{50}	a_{60}	a_{70}	a_{80}	a_{90}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	a_{06}	a_{07}	a_{08}	a_{09}

	ARC NODES ($a_{i,j}=(x_i,x_j)$)																													
source i	6	6	6	7	7	7	8	8	8	9	9	9	1	2	3	4	5	6	7	8	9	0	0	0	0	0	0	0	0	0
term. j	1	2	3	2	3	4	3	4	5	6	7	8	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	9

max.	0	0	0	-1	-1	0	-1	0	0	-1	0	0	1	1	1	0	0	1	0	0	0	0	0	0	1	1	0	1	1	1	=	-3
BASIC VAR.																																
$s_1=a_{01}$	0	-1	-1	0	0	0	0	0	0	1	0	0	-1	0	0	0	0	-1	0	0	0	1	0	0	0	0	1	0	0	0	=	-3
$s_2=a_{02}$	0	1	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	=	1
$s_3=a_{03}$	0	0	1	0	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	=	1
$u_9=a_{85}$	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	-1	-1	=	1
$v_5=a_{50}$	0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0	1	0	0	1	1	0	0	0	0	-1	0	0	-1	-1	=	0
$u_1=a_{61}$	1	1	1	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	0	=	4
$v_7=a_{70}$	0	0	0	1	1	1	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	=	4
$v_4=a_{40}$	0	0	0	1	1	0	0	-1	0	0	-1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	-1	0	0	-1	0	=	3
$u_{12}=a_{98}$	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-1	=	2

	DUAL VARIABLES (u_i, v_i)																			SLACK VARIABLES (s_i)								
dual var.	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9

	ARCS ($a_{i,j}$)																													
arc a_{ij}	a_{61}	a_{62}	a_{63}	a_{72}	a_{73}	a_{74}	a_{83}	a_{84}	a_{85}	a_{96}	a_{97}	a_{98}	a_{10}	a_{20}	a_{30}	a_{40}	a_{50}	a_{60}	a_{70}	a_{80}	a_{90}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	a_{06}	a_{07}	a_{08}	a_{09}

	ARC NODES ($a_{i,j}=(x_i,x_j)$)																													
source i	6	6	6	7	7	7	8	8	8	9	9	9	1	2	3	4	5	6	7	8	9	0	0	0	0	0	0	0	0	0
term. j	1	2	3	2	3	4	3	4	5	6	7	8	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	9

max.	0	-1	-1	-1	-1	0	-1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1	=	-6
BASIC VAR.																																
$v_1=a_{10}$	0	1	1	0	0	0	0	0	0	-1	0	0	1	0	0	0	0	1	0	0	0	-1	0	0	0	0	-1	0	0	0	=	1
$s_2=a_{02}$	0	1	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	=	1
$s_3=a_{03}$	0	0	1	0	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	=	1
$u_9=a_{85}$	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	-1	-1	=	1
$v_5=a_{50}$	0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0	1	0	0	1	1	0	0	0	0	-1	0	0	-1	-1	=	0
$u_1=a_{61}$	1	1	1	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	0	=	4
$v_7=a_{70}$	0	0	0	1	1	1	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	=	4
$v_4=a_{40}$	0	0	0	1	1	0	0	-1	0	0	-1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	-1	0	0	-1	0	=	3
$u_{12}=a_{98}$	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-1	=	2

Figure 5.19: The Dual Tableau After The Sixth LG Iteration

	DUAL VARIABLES (u_i,v_i)																				SLACK VARIABLES (s_i)								
dual var.	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9

	ARCS ($a_{i,j}$)																													
arc a_{ij}	a_{61}	a_{62}	a_{63}	a_{72}	a_{73}	a_{74}	a_{83}	a_{84}	a_{85}	a_{96}	a_{97}	a_{98}	a_{10}	a_{20}	a_{30}	a_{40}	a_{50}	a_{60}	a_{70}	a_{80}	a_{90}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	a_{06}	a_{07}	a_{08}	a_{09}

	ARC NODES ($a_{i,j}=(x_i,x_j)$)																													
source i	6	6	6	7	7	7	8	8	8	9	9	9	1	2	3	4	5	6	7	8	9	0	0	0	0	0	0	0	0	
term. j	1	2	3	2	3	4	3	4	5	6	7	8	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	9	
max.	0	0	0	-1	-1	0	-1	0	0	-1	0	0	1	1	1	0	0	1	0	0	0	0	0	0	0	1	1	0	1	1	1 = -3
BASIC VAR.																															
$u_2=a_{62}$	0	1	1	0	0	0	0	0	0	-1	0	0	1	0	0	0	0	1	0	0	0	-1	0	0	0	0	-1	0	0	0 = 3
$s_2=a_{03}$	0	0	-1	1	0	0	0	0	0	1	0	0	-1	-1	0	0	0	-1	0	0	0	1	1	0	0	0	1	0	0	0 = -2
$s_3=a_{03}$	0	0	1	0	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0 = 1
$u_9=a_{85}$	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	-1	-1	= 1
$v_5=a_{50}$	0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	1	0	0	1	1	0	0	0	0	0	-1	0	0	-1	-1 = 0
$u_1=a_{61}$	1	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0 = 1
$v_7=a_{70}$	0	0	0	1	1	1	0	0	0	0	-1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	-1	0	0 = 4
$v_4=a_{40}$	0	0	0	1	1	0	0	-1	0	0	-1	0	0	0	0	0	1	0	0	0	0	0	-1	0	0	-1	0	0	= 3	
$u_{12}=a_{98}$	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-1 = 2

	DUAL VARIABLES (u_i,v_i)																				SLACK VARIABLES (s_i)								
dual var.	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9

	ARCS ($a_{i,j}$)																													
arc a_{ij}	a_{61}	a_{62}	a_{63}	a_{72}	a_{73}	a_{74}	a_{83}	a_{84}	a_{85}	a_{96}	a_{97}	a_{98}	a_{10}	a_{20}	a_{30}	a_{40}	a_{50}	a_{60}	a_{70}	a_{80}	a_{90}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	a_{06}	a_{07}	a_{08}	a_{09}

	ARC NODES ($a_{i,j}=(x_i,x_j)$)																													
source i	6	6	6	7	7	7	8	8	8	9	9	9	1	2	3	4	5	6	7	8	9	0	0	0	0	0	0	0	0	
term. j	1	2	3	2	3	4	3	4	5	6	7	8	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	9	
max.	0	0	-1	0	-1	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	1	1	1	1	1	1 = -5
BASIC VAR.																															
$u_2=a_{62}$	0	1	1	0	0	0	0	0	0	-1	0	0	1	0	0	0	0	1	0	0	0	-1	0	0	0	0	-1	0	0	0 = 3
$v_2=a_{20}$	0	0	1	-1	0	0	0	0	0	-1	0	0	1	1	0	0	0	1	0	0	0	-1	-1	0	0	0	-1	0	0	0 = 2
$s_3=a_{03}$	0	0	1	0	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0 = 1
$u_9=a_{85}$	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	-1	-1	= 1
$v_5=a_{50}$	0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0	1	0	0	1	1	0	0	0	0	-1	0	0	-1	-1 = 0
$u_1=a_{61}$	1	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0 = 1
$v_7=a_{70}$	0	0	0	1	1	1	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-1	0	0 = 4
$v_4=a_{40}$	0	0	0	1	1	0	0	-1	0	0	-1	0	0	0	0	0	1	0	0	0	0	0	-1	0	0	-1	0	0	= 3	
$u_{12}=a_{98}$	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-1 = 2

Figure 5.20: The Dual Tableau After The Seventh LG Iteration

arcs, where k is the number of direct dependencies per node. The resulting constraint matrix, in the dual formulation, would have n rows and $m + n = (k + 1)n$ columns. Thus, a small pit containing 100000 blocks would have a constraint matrix with 100000 rows and 1000000 columns (assuming 9 arcs per node).

Assuming each column coefficient requires a byte of memory (the minimum number required to represent the range of values), each node would require approximately one megabyte of memory. For a problem containing 100000 nodes, the total memory required for the Simplex Algorithm would be 100 gigabytes. Even an inefficient graph representation of the problem should require no more than 1000 bytes of memory per node. Thus, the LG Algorithm would require about one-thousandth of that required by the Simplex Algorithm.

To compare processing requirements, consider the processing required to identify the arc to introduce into the normalized tree (entering basic variable and leaving basic variable) and the processing required to perform the *MoveTowardFeasibility* and *NormalizeTree* procedures (change of basis). We assume that all operations require roughly the same number of computer cycles, although we recognize the assumption is not typically valid, especially when disk memory fetches are required.

In the dual linear program, identifying the entering basic variable simply requires searching the objective function coefficients until a negative valued coefficient is located. At most, this would require $(k + 1)n$ compare operations. Once the entering basic variable is identified the column must be traversed and, for each row with a positive valued coeffient, the test ratio must be computed, requiring n additional operations.

Searching for the arc to introduce into the normalized tree requires checking each arc. At most, this would require kn compare operations and the additional

operations required to traverse the graph.

Now consider the operations required by the *MoveTowardFeasibility* and *NormalizeTree* procedures. Once an appropriate arc is selected the root of the branches containing the weak node and the strong node must be identified. This function would require at most $2b$ memory fetchs, where b is the number of benches in the pit (for real problems on the order of tens). In addition to each memory fetch a compare operation must be performed. Thus, identifying the branch roots requires a negligible amount of processing. Once the branch roots are identified, the operations performed require updating the value of the arcs between the two branch roots (addition operation), updating the status of the arcs (assignment operation), and updating the status of the nodes in one of the branches (assignment operation). In the worst case, this would require $2b$ addition operations and $n+2b$ assignment operations. Ignoring the $2b$ operations, this step would require approximately n assignment operations. Assuming an equivalent number of operations for the *NormalizeTree* procedure, we have a total of $2n$ operations.

To perform a change of basis, each row having a non-zero coefficient in the entering basic variable's column will be added to a multiple of the leaving basic variables row. Assuming a column will have at most $k-1$ non-zero coefficients, each row would require $(k+1)n$ multiplication and addition operations. Thus, requiring a total of $2(k-1)(k+1)n$ operations, or roughly $2k^2n$ operations. In our opinion, the algorithms require about the same amount of processing, with a slight edge held by the LG Algorithm. But, if memory swaps are considered, we imagine the large memory requirements of the Simplex Algorithm would increase its processing requirements significantly.

Although the LG Algorithm is more efficient in terms of memory and processing,

we do discover several interesting results from the analysis of the dual formulation and the example problem. First, we identified that the *NormalizeTree* procedure is required to ensure the solution stays dual feasible. Second, we identified that the *NormalizeTree* procedure could be eliminated by searching the chain of arcs running between the two branch roots and identifying the arc which should be removed, rather than automatically removing the artificial arc connecting the artificial root to the root of the strong branch. This rule would guarantee the transformation would result in a normalized tree.

Chapter 6
NESTED LERCHS AND GROSSMANN ALGORITHM

In Lerchs and Grossmann 1965, the authors point out that given a pit limit (or maximum valued closure in a graph) there are virtually an unlimited number of *ways* of reaching the pit limit. Each *way*, which corresponds to a different extaction sequence, may result in a different cash flow pattern. The authors continue by stating:

> An optimum digging pattern might be one in which the integral of the cash flow curve is maximum. The problem of designing intermediate pit contours can become extremely complex. The following analysis will highlight some properties of the pit model, and the results may provide a basis for the selection of intermediate contours.

The quotation indicates the lack of mathematical rigor supplied by the authors.

Although Lerchs and Grossmann 1965 does not supply the mathematical rigor justifying the use of what shall be called the Nested Lerchs and Grossmann (NLG) Algorithm, the mining industry adopted it.

In this chapter, we develop the mathematical basis of the NLG Algorithm. In Section 6.1, we develop the mathematical programming formulation of the NLG Algorithm. From this formulation, we find the NLG Algorithm uses the LG Algorithm to find the maximum valued closure in a set of new directed graphs which are derived from the original directed graph. These new graphs are called offset graphs. The maximum valued closures of the offset graphs are called nested closures.

In Section 6.2, we define terminology and introduce notation used in proving properties relating to the nested closures generated by the NLG Algorithm. In Section 6.3, we apply the NLG Algorithm to an example problem and generate a global cash flow function based upon the nested pits.

In Section 6.4, we prove: the nested closures generated by the NLG Algorithm do nest; that every base node in the smallest maximum valued closure of a directed graph has positive value; the average offset value of the increment between nested closures has zero value; that the average non-offset value of the increment between nested closures equals the offset parameter; and, that the average non-offset value of the nested closures, from smallest to largest, is monotonically decreasing.

6.1 Mathematical Programming Formulation of the NLG Algorithm

The NLG Algorithm solves a constrained version of the pit limit linear program in order to define a sequence of nested pits. The pit limit linear program, developed in Chapter 5, is restated here for convenience:

$$\begin{aligned} \max \quad & C^T X, \\ \text{s.t.:} \quad & EX \leq 0, \\ & IX \leq 1, \\ & X \geq 0. \end{aligned}$$

The vectors 1 and 0 denote vectors of positive ones and zeros, respectively; $EX \leq 0$ are accessibility constraints; and, $IX \leq 1$ are single extraction constraints.

Adding a volume constraint (an upper bound on the number of blocks in the pit limit) we have the *volume constrained pit limit linear program*:

$$\begin{aligned} \max \quad & C^T X, \\ \text{s.t.:} \quad & EX \leq 0, \\ & IX \leq 1, \\ & 1^T X \leq v, \\ & X \geq 0. \end{aligned}$$

The constraint $1^T X \leq v$ is the volume constraint, where v is an integer defining the maximum number of blocks which may be contained within the pit limit.

Relaxing the volume constraint, we obtain the *relaxed volume constrained pit limit linear program*:

$$\begin{aligned} \max \quad & C^T X - \lambda(1^T X - v), \\ \text{s.t.:} \quad & EX \leq 0, \\ & IX \leq 1, \\ & X \geq 0. \end{aligned}$$

Dropping the constant λv, which is irrelevant to the optimization (it is simply a constant offset), we have:

$$\begin{aligned} \max \quad & (C - \lambda 1)^T X, \\ \text{s.t.:} \quad & EX \leq 0, \\ & IX \leq 1, \\ & X \geq 0. \end{aligned}$$

In this form, the volume constrained pit limit linear program is equivalent to an unconstrained pit limit linear program where every block's value has been reduced by the constant λ. Thus, given a value of λ, the LG Algorithm can be used to solve the relaxed volume constrained pit limit linear program.

If $\lambda = 0$ then solving the relaxed volume constrained pit limit linear program is equivalent to solving the pit limit problem for the directed graph $G = (X, A)$. If

$\lambda > 0$, but it is not sufficiently greater than zero, then the solutions, for $\lambda = 0$ and $\lambda > 0$, will consist of the same set of nodes. This situation reflects a solution which does not meet the volume constraint. If λ is too large, then the resulting solution may meet the volume constraint, but there may exist a smaller value of λ which meets the volume constraint but has a solution containing more blocks. Thus, setting the value of λ is critical.

To generate a collection of nested pits, the relaxed volume constrained pit limit linear program must be solved for different values of λ. Each solution to a relaxed volume constrained pit limit linear program, which is unique from the other solutions, will be called a nested pit.

The simplistic approach, of gradually increasing the value of λ by a constant amount, will generate a collection of nested pits. But, there is no guarantee that all nested pits have been identified. Later in this chapter we prove, for each nested closure, the smallest value of λ equals the average value of the blocks contained in the increment between pit limits. This property allows the NLG algorithm to be modified to ensure that all nested pits are identified. Psuedocode for the **NLG Algorithm** is:

$\lambda_0 \leftarrow 0$; $S_{\lambda_0} \leftarrow LG(G)$
$k \leftarrow 1$; $\lambda_k \leftarrow \lambda_0$; $i \leftarrow 1$
while ($|S_{\lambda_k}| > 0$) **do**
    $\lambda_k \leftarrow \lambda_{k-1} + i\delta$
    $G_{\lambda_k} \leftarrow BuildOffsetGraph(G, \lambda_k)$
    $S_{\lambda_k} \leftarrow LG(G_{\lambda_k})$
    if ($|S_{\lambda_k}| < |S_{\lambda_{k-1}}|$)
        while ($\bar{\nu}(|S_{\lambda_{k-1}} \sim S_{\lambda_k}|) > \epsilon$) **do**
            $\lambda_k \leftarrow \bar{\nu}(|S_{\lambda_{k-1}} \sim S_{\lambda_k}|)$
            $G_{\lambda_k} \leftarrow BuildOffsetGraph(G, \lambda_k)$
            $S_{\lambda_k} \leftarrow LG(G_{\lambda_k})$
        end while
        $k \leftarrow k + 1$

```
            i ← 1
        else
            i ← i + 1
        end if
    end while
```

Using the NLG Algorithm, a collection of nested pits can be generated. The nested pits define an extraction sequence, and a cash flow function. Typically, the increment between nested pits contains more than a single node. Thus, the extraction sequence generated from the NLG Algorithm's nested pits is called a *global extraction sequence.* Similarly, the cash flow function is called the *global cash flow function.* In the next chapter, we show the extraction sequence generated by the NLG Algorithm, maximizes the integral of the global cash flow function.

Since the NLG Algorithm defines only a global extraction sequence, the problem of ordering blocks contained in the increment between nested closures remains. Since an increment may contain several thousand nodes, this problem should not be ignored. Ordering the blocks in the increment is called the *local ordering problem.* Sequences generated in solving these problems, shall be called *local extraction sequences.* In the next chapter, we present approaches for solving the local ordering problem, which are based upon properties of the NLG Algorithm.

6.2 NLG Algorithm Terminology

In this section, we define terminology used in developing the properties of the NLG Algorithm. As discussed in the previous section, the NLG Algorithm generates a sequence of graphs based upon the *original directed graph.* Each generated graph consists of the same nodes as the original, but the value of each node has been offset (decreased) by a constant value.

Let the directed graph shown in Figure 6.1, represent the original directed graph, $G = (X, A)$. Since node values in the original directed are not offset, we call the original directed graph the *non-offset graph.* We denote the non-offset graph with the symbol G_{λ_0}, where $\lambda_0 = 0$. Thus, $G_{\lambda_0} = G$. Let S_{λ_0} denote the maximum valued closure of G_{λ_0}. In Figure 6.1, S_{λ_0} contains all nodes.

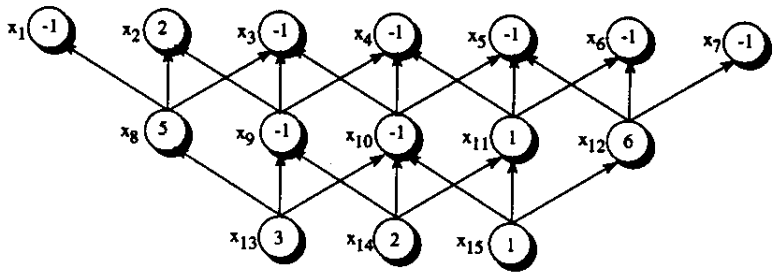


Figure 6.1: The Non-Offset Graph $G_{\lambda_0=0}$ and its Maximum Valued Closure S_{λ_0}

Let $G_\lambda(X, A)$ denote an *offset graph* of $G = (X, A)$, and let S_λ denote its smallest maximum valued closure. The offset graph contains the same set of nodes and arcs as G, but node values have been reduced by the real valued *offset parameter*, λ.

Previously, we used $\nu(x_i)$ to denote the value of a node in a directed graph. Since nodes have different values in different graphs, denote a node's value in a particular graph by adding a subscript to the value function. Thus, $\nu_{\lambda_0}(x_i) = \nu(x_i)$, denotes the *non-offset value* of x_i. Similarly, $\nu_\lambda(x_i)$, denotes the *offset value* of x_i. The offset value of x_i is computed with the following *offset transformation*,

$$\nu_\lambda(x_i) \;=\; \nu(x_i) - \lambda. \tag{6.1}$$

If $Y \subseteq X$, then $\nu_\lambda(Y)$ denotes its *offset value*, the set's value in the offset

graph G_λ; $\nu_{\lambda_0}(Y) = \nu(Y)$ denotes its *non-offset value.* Let $\bar{\nu}(Y) = \bar{\nu}_{\lambda_0}(Y)$ denote the average non-offset value of Y, and let $\bar{\nu}_\lambda(Y)$ denote the average offset value of Y. The offset value of Y can be computed as:

$$\begin{aligned} \nu_\lambda(Y) &= \sum_{x_i \in Y} \nu_\lambda(x_i), \\ &= \sum_{x_i \in Y} (\nu(x_i) - \lambda), \\ &= \sum_{x_i \in Y} \nu(x_i) - \sum_{x_i \in Y} \lambda, \\ &= \nu(Y) - |Y|\lambda. \end{aligned} \tag{6.2}$$

Let G_{λ_k} be an offset graph whose maximum valued closure S_{λ_k} has the following properties:

$$|S_{\lambda_k}| < |S_{\lambda_{k-1}}|; \tag{6.3}$$

and, for any $\delta > 0$,

$$|S_{\lambda_k - \delta}| = |S_{\lambda_{k-1}}|, \tag{6.4}$$

If Equations 6.3 and 6.4 hold, then G_{λ_k} is called the *consecutive offset graph* to $G_{\lambda_{k-1}}$; and, S_{λ_k} is called the *consecutive nested closure* of $S_{\lambda_{k-1}}$.

Since S_{λ_0} contains a finite number of nodes, it can contain only a finite number of consecutive nested closures. Denote the collection of consecutive offset graphs as: $\{G_{\lambda_0}, G_{\lambda_1}, \ldots, G_{\lambda_n}\}$. Denote the associated collection of consecutive nested closures as $S = \{S_{\lambda_0}, S_{\lambda_1}, \ldots, S_{\lambda_n}\}$.

In the simple examples provided in this chapter, each offset parameter may be represented as $\lambda = \lambda_{num}/\lambda_{den}$. Thus, the value of node x_i in the offset graph can be stated as:

$$\begin{aligned} \nu_\lambda(x_i) &= \nu(x_i) - \frac{\lambda_{num}}{\lambda_{den}}, \\ &= \frac{\lambda_{den}\nu(x_i) - \lambda_{num}}{\lambda_{den}}. \end{aligned} \tag{6.5}$$

Since the value of each node in an offset graph is divided by the same value, λ_{den}, figures showing offset graphs will display only the numerator, $\lambda_{den}\nu(x_i) - \lambda_{num}$, of each node's offset value. This format simplifies the presentation by not having to display real values within the node symbols.

For example, consider the offset graph, $G_{\lambda_1=4/7}$, shown in Figure 6.2. The values shown inside the node symbols are the numerator of the node's offset value, $\lambda_{den}\nu(x_i) - \lambda_{num}$. Thus, in the offset graph $G_{\lambda_1=4/7}$ the value of node x_9 is

$$\begin{aligned} \nu_{\lambda_1=4/7}(x_9) &= \nu(x_9) - \frac{4}{7}, \\ &= -1 - \frac{4}{7}, \\ &= \frac{-11}{7}. \end{aligned}$$

The value shown inside x_9's node symbol is -11, the numerator of its offset value.

The offset graph $G_{\lambda_1=4/7}$ has a different maximum valued closure than the non-offset graph G_{λ_0}. The shadowed node symbols in Figure 6.2 indicate the offset graph's

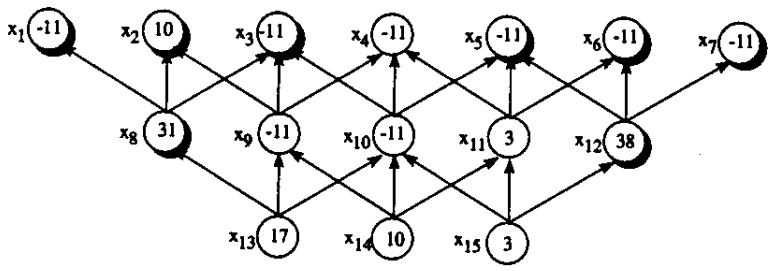


Figure 6.2: The Offset Graph $G_{\lambda_1=4/7}$ and its Maximum Valued Closure S_{λ_1}

maximum valued closure,

$$S_{\lambda_1} = \{x_1, x_2, x_3, x_5, x_6, x_7, x_8, x_{12}\},$$

which contains fewer nodes than the maximum valued closure of G_{λ_0}.

Let $I_{\lambda_k} = S_{\lambda_{k-1}} \sim S_{\lambda_k}$ denote the *kth increment*, or the set of nodes in nested closure $S_{\lambda_{k-1}}$ but not in nested closure S_{λ_k}. Thus, the first increment in the non-offset graph G_{λ_0} is:

$$I_{\lambda_1} = S_{\lambda_0} \sim S_{\lambda_1} = \{x_4, x_9, x_{10}, x_{11}, x_{13}, x_{14}, x_{15}\}.$$

The non-offset value of the first increment is:

$$\begin{aligned}
\nu_{\lambda_0}(I_{\lambda_1}) &= \nu(I_{\lambda_1}), \\
&= \nu(\{x_4, x_9, x_{10}, x_{11}, x_{13}, x_{14}, x_{15}\}), \\
&= \nu(x_4) + \nu(x_9) + \nu(x_{10}) + \nu(x_{11}) + \nu(x_{13}) + \nu(x_{14}) + \nu(x_{15}), \\
&= 4.
\end{aligned}$$

The average non-offset value of the first increment is:

$$\begin{aligned} \bar{\nu}_{\lambda_0}(I_{\lambda_1}) &= \frac{\nu(I_{\lambda_1})}{|I_{\lambda_1}|}, \\ &= 4/7. \end{aligned}$$

The offset value of the first increment, in the offset graph G_{λ_1}, is:

$$\begin{aligned} \nu_{\lambda_1}(I_{\lambda_1}) &= \nu_{\lambda_1}(\{x_4, x_9, x_{10}, x_{11}, x_{13}, x_{14}, x_{15}\}), \\ &= \nu_{\lambda_1}(x_4) + \nu_{\lambda_1}(x_9) + \nu_{\lambda_1}(x_{10}) \\ &\quad + \nu_{\lambda_1}(x_{11}) + \nu_{\lambda_1}(x_{13}) + \nu_{\lambda_1}(x_{14}) + \nu_{\lambda_1}(x_{15}), \\ &= \frac{-11 - 11 - 11 + 3 + 17 + 10 + 3}{7}, \\ &= 0. \end{aligned}$$

The *base node set* of a closure S, is the set of nodes which have no other nodes in the closure dependent upon them. Recall that a node x_i is dependent upon a node x_j if there exists a path from node x_i to node x_j. Thus, the base node set of any closure S can be defined as:

$$B_S = \{x_i : x_j \not\rightarrow x_i, \forall x_j \in S\}.$$

The base node set of the maximum valued closure S_{λ_0} in the non-offset graph G_{λ_0} is:

$$B_{S_{\lambda_0}} = \{x_{13}, x_{14}, x_{15}\};$$

and, for the maximum valued closure S_{λ_1}:

$$B_{S_{\lambda_1}} = \{x_8, x_{12}\}.$$

Inspection of Figures 6.1 and 6.2, indicates all base nodes have positive value. We prove below that this property holds for any smallest maximum valued closure in a directed graph.

6.3 An Example of the NLG Algorithm

In this section, we apply the NLG Algorithm to the non-offset graph shown in Figure 6.1. We then use the non-offset graph's nested closures to develop three functions: pit limit size as a function of offset parameter value, pit limit value as a function of offset parameter value, and pit limit value as a function of pit limit size. Representations of these figures where presented in Lerchs and Grossmann 1965, but a numerical example was never provided.

Applying the NLG Algorithm to the non-offset graph G_{λ_0}, we obtain five nested closures. This includes the non-offset graph's maximum valued closure and the final empty graph's, empty maximum valued closure, as nested closures. The nested closure S_{λ_0} is shown in Figure 6.1. The nested closure S_{λ_1} is shown in Figure 6.2. Figures 6.3, 6.4, and 6.5, show the final three nested closures, respectively.

Table 6.1 shows, for each nested closure: its node set, its non-offset value, its size, and its non-offset average value.

Table 6.2 shows, for each increment between consecutive closures: its non-offset value, its size, and its average non-offset value.

To ensure S_{λ_1} is the consecutive nested closure to S_{λ_0}, we must demonstrate

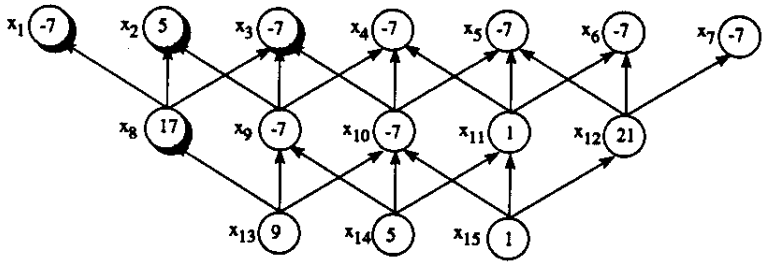


Figure 6.3: The Offset Graph $G_{\lambda_2=3/4}$ and its Maximum Valued Closure S_{λ_2}

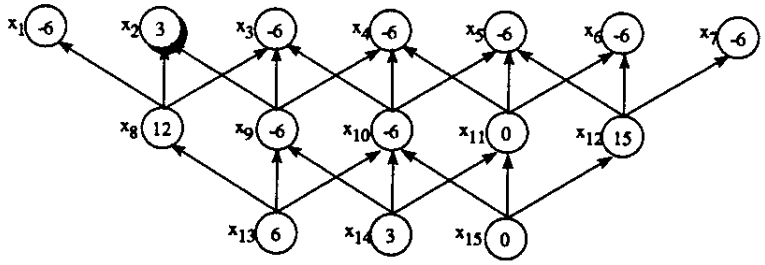


Figure 6.4: The Offset Graph $G_{\lambda_3=3/3}$ and its Maximum Valued Closure S_{λ_3}

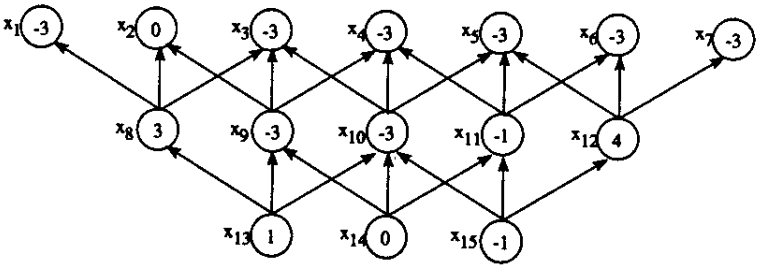


Figure 6.5: The Offset Graph $G_{\lambda_4=2/1}$ and its Maximum Valued Closure S_{λ_4}

Table 6.1: Nested Closure Values In G

Closure	Node Set S_{λ_k}	Value $\nu_{\lambda_0}(S_{\lambda_k})$	Size $\lvert S_{\lambda_k}\rvert$	Aver. Value $\bar{\nu}_{\lambda_0}(S_{\lambda_k})$
S_{λ_0}	$\{x_1, \ldots, x_{15}\}$	12	15	12/15
S_{λ_1}	$\{x_1, x_2, x_3, x_5, x_6, x_7, x_8, x_{12}\}$	8	8	8/8
S_{λ_2}	$\{x_1, x_2, x_3, x_8\}$	5	4	5/4
S_{λ_3}	$\{x_2\}$	2	1	2/1
S_{λ_4}	$\{\}$	0	0	—

Table 6.2: Closure Increment Average Values

Increment I_{λ_k}	Node Set $S_{\lambda_{k-1}} \sim S_{\lambda_k}$	Value $\nu_{\lambda_0}(I_{\lambda_k})$	Size $\lvert I_{\lambda_k}\rvert$	Aver. Value $\bar{\nu}_{\lambda_0}(I_{\lambda_k})$
I_{λ_1}	$\{x_4, x_9, x_{10}, x_{11}, x_{13}, x_{14}, x_{15}\}$	4	7	4/7
I_{λ_2}	$\{x_5, x_6, x_7, x_{12}\}$	3	4	3/4
I_{λ_3}	$\{x_1, x_3, x_8\}$	3	3	3/3
I_{λ_4}	$\{x_2\}$	2	1	2/1

that Equations 6.3 and 6.4 hold. From Table 6.1, we see that $|S_{\lambda_1}| < |S_{\lambda_0}|$. Thus, Equation 6.3 holds.

Now, consider the offset graph, $G_{\lambda_1-\delta}$, formed by decreasing λ_1 by $\delta > 0$. Thus, from Equation 6.2, we have

$$\begin{aligned} \nu_{\lambda_1-\delta}(I_{\lambda_1}) &= \nu(I_{\lambda_1}) - |I_{\lambda_1}|(\lambda_1 - \delta), \\ &= 4 - 7(4/7 - \delta), \\ &= 7\delta. \end{aligned}$$

Since $\nu_{\lambda_1-\delta}(I_{\lambda_1}) > 0$,

$$\begin{aligned} \nu_{\lambda_1-\delta}(S_{\lambda_1} \cup I_{\lambda_1}) &= \nu_{\lambda_1-\delta}(S_{\lambda_1}) + \nu_{\lambda_1-\delta}(I_{\lambda_1}), \\ &> \nu_{\lambda_1-\delta}(S_{\lambda_1}). \end{aligned}$$

Thus,

$$S_{\lambda_1-\delta} \subseteq S_{\lambda_1} \cup I_{\lambda_1}.$$

Table 6.3: Closure Increment Average Values

Set Y	Non-offset Value $\nu(Y)$	Offset Value $\nu_{\lambda_1-\delta}$
$\{x_4, x_{11}\}$	0	$-2(4/7-\delta) = -8/7+2\delta$
$\{x_4, x_9, x_{10}, x_{13}\}$	0	$-4(4/7-\delta) = -16/7+4\delta$
$\{x_4, x_9, x_{10}, x_{11}, x_{14}\}$	0	$-5(4/7-\delta) = -20/7+5\delta$
$\{x_4, x_{10}, x_{11}, x_{15}\}$	0	$-4(4/7-\delta) = -16/7+4\delta$
$\{x_4, x_9, x_{10}, x_{11}, x_{13}, x_{14}\}$	3	$3-6(4/7-\delta) = -3/7+6\delta$
$\{x_4, x_9, x_{10}, x_{11}, x_{14}, x_{15}\}$	1	$1-6(4/7-\delta) = -17/7+6\delta$
$\{x_4, x_9, x_{10}, x_{11}, x_{13}, x_{14}, x_{15}\}$	4	$4-7(4/7-\delta) = 7\delta$

To see that $S_{\lambda_1-\delta} = S_{\lambda_1} \cup I_{\lambda_1}$ consider Table 6.3. The table shows the offset value, in $G_{\lambda_1-\delta}$, for each candidate closure within I_{λ_1}. All base nodes in each candidate closure must have positive value. From Equation 6.2, we know the offset value of each candidate closure can be computed as:

$$\nu_{\lambda_1-\delta}(Y) = \nu(Y) - |Y|(\lambda_1 - \delta).$$

Since δ can be any positive value, assume it is a very small positive value. Thus, the only candidate closure having positive value is the one containing all nodes in I_{λ_1} (bottom row in table). Thus, $S_{\lambda_1-\delta}$ assumes its maximum value when it contains all nodes in I_{λ_1}. Note that $\delta = 0$ before this candidate closure has zero value.

Since $S_{\lambda_1+\delta}$ assumes its maximum value when it contains all nodes in I_{λ_1}, we know

$$\begin{aligned} S_{\lambda_1+\delta} &= S_{\lambda_1} \cup I_{\lambda_1}, \\ &= S_{\lambda_0}. \end{aligned}$$

Hence, Equation 6.4 holds.Therefore, $S_{\lambda_1=4/7}$ is the consecutive nested closure to S_{λ_0}. Although we do not demonstrate, S_{λ_2} can be shown to be the consecutive nested closure to S_{λ_1}, S_{λ_3} can be shown to be the consecutive nested closure to S_{λ_2}, etc.

Comparing the offset parameter values in Table 6.1, and the average value of the increments in Table 6.2, we see the offset parameter equals the average value of the increment. For example, $\lambda_3 = 3/3$, and

$$\begin{aligned}\bar{\nu}(I_{\lambda_3}) &= \frac{\nu(x_1)+\nu(x_3)+\nu(x_8)}{|I_{\lambda_3}|},\\ &= \frac{-1-1+5}{3},\\ &= \frac{3}{3}.\end{aligned}$$

The property that the offset parameter value equals the average non-offset value of the increment is proven to always hold below.

Another property of the nested closures formed by the NLG Algorithm, which we prove below, is that the offset value of the increment between two consecutive closures $S_{\lambda_{k-1}}$ and S_{λ_k} has an offset value equal to zero (with respect to the offset graph G_{λ_k}). For example,

$$\begin{aligned}\nu_{\lambda_3}(I_{\lambda_3}) &= \nu_{\lambda_3}(x_1)+\nu_{\lambda_3}(x_3)+\nu_{\lambda_3}(x_8),\\ &= \frac{-6}{3}+\frac{-6}{3}+\frac{12}{3},\\ &= 0.\end{aligned}$$

Thus, nested closures are formed by increasing the offset parameter's value until an

increment, having an offset value of zero, is obtained.

We can form the function shown in Figure 6.6 using the nested closures generated in our example problem. Recall that each value of the offset parameter, λ, generates a different offset graph. But, different offset graphs can have the same set of nodes in their maximum valued closure. For example, all offset graphs generated by offset parameter values in the half-open interval $[\lambda_0, \lambda_1)$, have their maximum valued closure defined by the set of nodes in S_{λ_0}. This follows from Equation 6.4. Thus, on each half-open interval $[\lambda_{k-1}, \lambda_k)$, the offset graph's maximum valued closure will consist of the set of nodes in $S_{\lambda_{k-1}}$. Thus, the function is a monotonically decreasing step function.

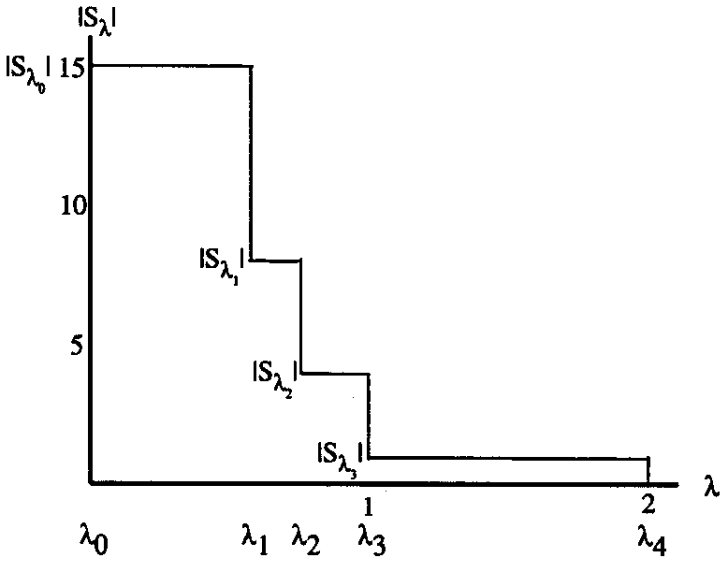


Figure 6.6: Pit Limit Size as a Function of the Offset Parameter

Figure 6.7 shows closure value as a function of the offset parameter. The function is a piecewise linear function (solid line) since the set of nodes in the maximum valued closure remains constant over the half-open interval $[\lambda_{k-1}, \lambda_k)$, but the value

of each node in the set is decreasing linearly. For this reason, each linear segment is denoted with the appropriate maximum valued closure. For example, on the interval $[\lambda_0, \lambda_1]$, the line segment is denoted as $\nu_\lambda(S_{\lambda_0})$.

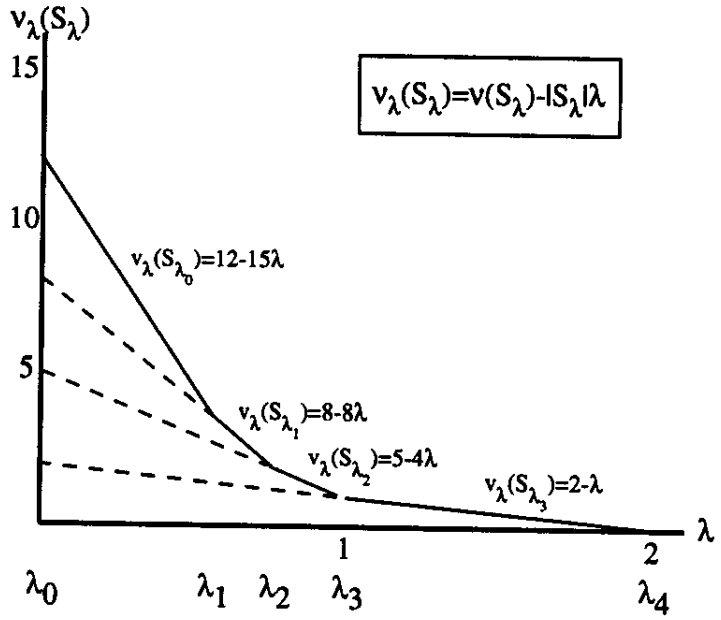


Figure 6.7: Pit Limit Value as a Function of the Offset Parameter

The dashed lines, shown in Figure 6.7, propagate each linear segment to indicate the value of the nested closure in the non-offset graph. For example, if $\lambda = 0$ then the linear function $\nu_\lambda(S_{\lambda_1}) = 8 - 8\lambda$ has a value of 8, which corresponds to the non-offset value of S_{λ_1}.

These functions demonstrate the behavior of the maximum valued closure in the offset graph, in terms of its size, offset value, and non-offset value. The size and non-offset value decreases at discrete values of λ. The offset value decreases linearly with λ. Knowing these relationships is important, but they are also fairly obvious.

The most important function obtained with the nested closures is the global

cash flow function. This function's independent variable is the size of the nested closure. its dependent variable is the value of the nested closure. The nested closures of the example generate the global cash flow function shown in Figure 6.8. The size and value of each nested closure defines a single point on the function. We call these points the *characteristic points* of the function.

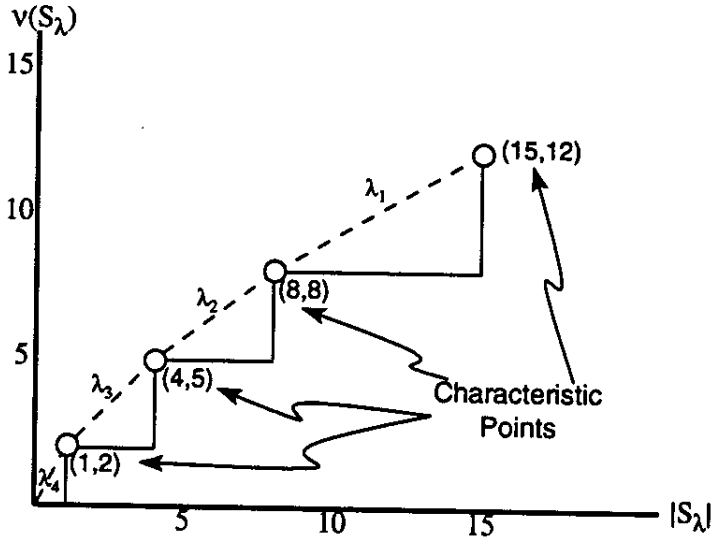


Figure 6.8: Global Cash Flow Function

Since the nested closures define only a few characteristic points, we call the function the global cash flow function. The global cash flow function assumes no value is received for a nested closure until the entire closure has been extracted. Thus, value remains constant until the entire closure is extracted, at which point an entire value of closure is assumed received.

The slope of the chord (dashed lines) connecting each pair of consecutive characteristic points, has a value equal to the offset parameter defining the initial characteristic point. In the next section, we prove that the offset parameter equals the

average value of the nodes in the increment between consecutive closures.

6.4 Properties of the NLG Algorithm

In this section we prove the following properties about the nested closures generated by the NLG Algorithm: the closures are nested, every base node has a positive value, the number of positive nodes in the non-offset graph's maximum valued closure is an upper bound on the number of nested closures obtainable from the graph; in the offset graph, each increment's value is zero; the offset parameter equals the average value of its associated increment; the offset parameter's value equals the *minimum* average value among all feasible increments. These proofs make considerable use of the following definitions.

The set $S = \{S_{\lambda_0}, S_{\lambda_1}, \ldots, S_{\lambda_n}\}$ contains *all* nested closures of the directed graph $G = (X, A)$, obtainable using the NLG Algorithm. Each *nested closure*, S_{λ_k}, is defined to be the *smallest* maximum valued closure in the offset graph G_{λ_k}. The indexing $\{0, 1, \ldots, n\}$ of the sets implies an ordering from the largest nested closure to the smallest nested closure. Thus, S_{λ_0} is the largest closure and S_{λ_n} is the smallest. In addition, S_{λ_0} is the smallest maximum valued closure of the non-offset graph G; obtained by applying the LG Algorithm to the non-offset graph G; and, S_{λ_n} is defined to be the empty set.

For each nested closure, S_{λ_k}, the offset parameter, λ_k, is defined to be the *minimum* offset parameter value; with respect to the set of all offset parameters which generate offset graphs having S_{λ_k} as their maximum valued closure. For any nested closure, $S_{\lambda_k} \in S$, there exists an infinity of offset parameter values which generate offset graphs having S_{λ_k} as their maximum valued closure. The definition of S assumes *each* offset parameter, λ_k, is the minimum possible value. The sequence

of offset parameters $\{\lambda_0, \lambda_1, \ldots, \lambda_n\}$ is monotonically increasing.

Any two nested closures having consecutive subscripts, $S_{\lambda_{k-1}}$ and S_{λ_k}, are called *consecutive closures.* Since S contains *all* nested closures obtainable using the NLG Algorithm, it is impossible for there to exist a nested closure between any two consecutive closures. Thus, for any two *consecutive* closures, $S_{\lambda_{k-1}}$ and S_{λ_k}, any offset parameter value in the half open interval $[\lambda_{k-1}, \lambda_k)$ will generate an offset graph having $S_{\lambda_{k-1}}$ as its smallest maximum valued closure.

Lemma 21 *Let $G = (X, A)$ be a directed graph and G_λ be a related offset graph with offset parameter λ. Also, let λ be sufficiently large enough such that the respective maximum valued closures, S and S_λ, of the graphs are not equal. Then $S_\lambda \subset S$. The symbol $\subset$ denotes a strict subset.*

Proof: Let T_f be the final normalized tree, defined by the LG Algorithm, in the non-offset graph $G = (X, A)$. Each arc in T_f will have one of the following classifications: (i) strong m-arcs (which are not possible in a normalized tree), (ii) strong p-arcs, (iii) weak p-arcs, and (iv) weak m-arcs.

In the offset graph G_λ, the value of each node, as defined by the offset transformation in Equation 6.1, is

$$\nu_\lambda(x_i) \;=\; \nu(x_i) - \lambda. \tag{6.6}$$

Thus, the value of each node in G_λ is less than its value in G.

In T_f, substitute each node's value, $\nu(x_i)$, with its offset value, $\nu_\lambda(x_i)$. The substitution does not change the structure of T_f, but it does decrease each branch's value. Let T_f' denote the tree after the value substitution. The decrease in each branch's value has the following consequences.

Arcs in classification (*ii*), strong p-arcs, support positive valued branches in T_f. If, after the substitution, the branch's value becomes negative the strong p-arc will become weak; if, after the substitution, the branch's value remains positive the p-arc will remain strong.

Arcs in classification (*iii*), weak p-arcs, support non-positive valued branches in T_f. After the substitution, the branch's value must remain negative; thus, the weak p-arc remains weak.

Arcs in classification (*iv*), weak m-arcs, support positive valued branches in T_f. If, after the substitution, the branch's value remains positive, the m-arc will remain weak. If, after the substitution, the branch's value becomes negative, the m-arc will become strong. But, since the arc is an m-arc, we know the arc cannot be adjacent to the artificial root. Thus, the tree fails to be normalized. Hence, the transformation required by the *NormalizeTree* procedure must be performed. The *NormalizeTree* procedure replaces the strong m-arc with an artificial arc adjacent to the artificial root. The artificial arc must be a p-arc, and we know it supports the same branch as the strong m-arc supported. Thus, the p-arc supports a negative valued branch and is classified as a weak p-arc. Thus, all nodes in the branch will remain classified as weak.

From the possible arc classification changes discussed above, we know the following about nodes in T_f': each weak node in T_f will remain weak; and, since $S \neq S_\lambda$, at least one strong node contained in T_f and S, will become weak. Thus, $S_\lambda \subset S$. □

Theorem 6 *If* $S = \{S_{\lambda_0}, S_{\lambda_1}, \cdots, S_{\lambda_n}\}$, *where* $\lambda_0 < \lambda_1 < \cdots < \lambda_n$, *is the set of all nested closures generated from the graph* $G = (X, A)$, *then* $S_{\lambda_n} \subset S_{\lambda_{n-1}} \subset \cdots \subset S_{\lambda_1} \subset S_{\lambda_0}$.

Proof: Start with the maximum valued closure, S_{λ_0}, of the graph $G_{\lambda_0} = (X, A)$. From

Lemma 21, its successive closure, S_{λ_1}, must be a subset of S_{λ_0}. Applying the same argument to the closure S_{λ_1}, we know $S_{\lambda_2} \subset S_{\lambda_1}$. Repeating this process for each successive closure, proves the property holds for all nested closures. □

Theorem 7 *Let S_{λ_i} and S_{λ_j} be any two nested closures in $S = \{S_{\lambda_0}, S_{\lambda_1}, \ldots, S_{\lambda_n}\}$, such that $i < j$. Then all nodes in S_{λ_j} are independent of all nodes in $S_{\lambda_i} \sim S_{\lambda_j}$.*

Proof: Since S_{λ_j} is a closure it is not possible for any node in S_{λ_j} to be dependent upon a node in $S_{\lambda_i} \sim S_{\lambda_j}$. □

Lemma 22 *The value of every base node in the smallest maximum valued closure of* any *directed graph is positive.*

Proof: A *base node* of a closure is a node which has no other nodes in the closure which are dependent upon it.

Assume a base node has zero value. Since the base node has no other nodes in the closure dependent upon it, it is possible to decrease the size of the maximum valued closure (by removing the zero valued base node from the closure) without decreasing the closure's value. This contradicts the property that the closure is the *smallest* maximum valued closure. Hence, a base node cannot have zero value.

Assume a base node has negative value. Since the base node has no other nodes in the closure dependent upon it, it is possible to increase the value of the closure. Since the node is a base node we may remove it from the closure and still have a closure. The value of the closure without the negative valued base node is greater than the value of the closure with the node. Thus, the closure with the node cannot be a maximum valued closure. Hence, a base node cannot have negative value.

Since base nodes, in the smallest maximum valued closure, cannot have negative or zero value, every base node must have positive value. □

Theorem 8 *If S_{λ_0} is the smallest maximum valued closure of the graph $G = (X, A)$, then the number of positive valued nodes in S, $|S^+_{\lambda_0}|$, is an upper bound on the number of nested closures obtainable with the NLG Algorithm.*

Proof: We know from Theorem 6 consecutive closures are nested; and, from Lemma 22 all base nodes are positive valued. Thus, the only possible way to form the nested closure S_{λ_k} from the closure $S_{\lambda_{k-1}}$ is to sufficiently decrease the value of each node enough to eliminate at least one positive valued node from S_{λ_k}. Thus, $|S^+|$ is an upper bound on the number of nested closures for a graph $G = (X, A)$. □

Theorem 9 is proven by showing non-zero offset values of the increment I_{λ_k} (where offset values refer to value in the offset graph G_{λ_k}), contradict properties of the set S. Specifically, if $\nu_{\lambda_k}(I_{\lambda_k}) > 0$ then there exists a closure in G_{λ_k} with value greater than S_{λ_k}. Thus, contradicting the property of S which says S_{λ_k} is the maximum valued closure of the offset graph G_{λ_k}. If $\nu_{\lambda_k}(I_{\lambda_k}) < 0$, then there exists an offset parameter value, less than λ_k, which defines an offset graph having S_{λ_k} as its maximum valued closure. Thus, contradicting the property of S which says λ_k is the minimum offset parameter value generating an offset graph having S_{λ_k} as its smallest maximum valued closure.

Theorem 9 *Let $S_{\lambda_{k-1}}$ and S_{λ_k} be any two consecutive maximum valued closures in $S = \{S_{\lambda_0}, S_{\lambda_1}, \ldots, S_{\lambda_n}\}$. Then the closure increment $I_{\lambda_k} = S_{\lambda_{k-1}} \sim S_{\lambda_k}$, has zero value in the offset graph G_{λ_k}, or $\nu_{\lambda_k}(I_{\lambda_k}) = 0$.*

Proof: Assume $\nu_{\lambda_k}(I_{\lambda_k}) > 0$. We know the sets I_{λ_k} and the S_{λ_k} are mutually exclusive and $S_{\lambda_{k-1}} = S_{\lambda_k} \cup I_{\lambda_k}$. Thus, in the offset graph G_{λ_k}, we have

$$\nu_{\lambda_k}(S_{\lambda_{k-1}}) \;=\; \nu_{\lambda_k}(S_{\lambda_k} \cup I_{\lambda_k}),$$

$$= \nu_{\lambda_k}(S_{\lambda_k}) + \nu_{\lambda_k}(I_{\lambda_k}).$$

By the assumption, we obtain the relationship

$$\nu_{\lambda_k}(S_{\lambda_{k-1}}) > \nu_{\lambda_k}(S_{\lambda_k}).$$

Since $S_{\lambda_{k-1}}$ is a closure, we have contradicted the assertion that S_{λ_k} is the smallest maximum valued closure of the offset graph G_{λ_k}. Hence, $\nu_{\lambda_k}(I_{\lambda_k}) \not> 0$.

Assume $\nu_{\lambda_k}(I_{\lambda_k}) < 0$. From the definition of S, we know S_{λ_k} is the maximum valued closure of the offset graph G_{λ_k}. Let $\lambda^* = \lambda_k - \frac{\nu_{\lambda_k}(I_{\lambda_k})}{|I_{\lambda_k}|} < \lambda_k$. In the offset graph G_{λ^*}, the value of the increment is

$$\begin{aligned}
\nu_{\lambda^*}(I_{\lambda_k}) &= \nu(I_{\lambda_k}) - \lambda^*|I_{\lambda_k}|, \\
&= \nu(I_{\lambda_k}) - (\lambda_k - \frac{\nu_{\lambda_k}(I_{\lambda_k})}{|I_{\lambda_k}|})|I_{\lambda_k}|, \\
&= (\nu(I_{\lambda_k}) - \lambda_k|I_{\lambda_k}|) - \nu_{\lambda_k}(I_{\lambda_k}), \\
&= \nu_{\lambda_k}(I_{\lambda_k}) - \nu_{\lambda_k}(I_{\lambda_k}), \\
&= 0.
\end{aligned}$$

Since the increment I_{λ_k} has zero value in the offset graph G_{λ^*}, I_{λ_k} remains excluded from the maximum valued closure S_{λ_k}. Thus, the offset parameter $\lambda^* < \lambda_k$ generates an offset graph having S_{λ_k} as its maximum valued closure. This contradicts the property of the set S which says λ_k is the smallest offset parameter value defining an offset graph with S_{λ_k} as its maximum valued closure. Therefore, $\nu_{\lambda_k}(I_{\lambda_k}) \not< 0$.

Since $\nu_{\lambda_k}(I_{\lambda_k}) \not> 0$ and $\nu_{\lambda_k}(I_{\lambda_k}) \not< 0$ the increment must have zero value in the

offset graph G_{λ_k}. □

Corollary 2 *Let $S_{\lambda_{k-1}}$ and S_{λ_k} be any two consecutive maximum valued closures in $S = \{S_{\lambda_0}, S_{\lambda_1}, \ldots, S_{\lambda_n}\}$. Then the offset parameter, λ_k, equals the average value of the closure increment, $I_{\lambda_k} = S_{\lambda_{k-1}} \sim S_{\lambda_k}$, in the non-offset graph G_{λ_0}.*

Proof: In the offset graph G_{λ_k}, the increment's, I_{λ_k}, value is

$$\nu_{\lambda_k}(I_{\lambda_k}) = \nu(I_{\lambda_k}) - |I_{\lambda_k}|\lambda_k. \tag{6.7}$$

From Theorem 9 we know the closure increment has zero value in the offset graph G_{λ_k}. Thus, setting the left hand side of Equation 6.7 to zero and solving for λ_k we have

$$\lambda_k = \frac{\nu(I_{\lambda_k})}{|I_{\lambda_k}|},$$

$$\lambda_k = \bar{\nu}(I_{\lambda_k}).$$

This proves that the offset parameter equals the average value of the closure increment in the non-offset graph. □

Theorem 10 *Let $S = \{S_{\lambda_0}, S_{\lambda_1}, \ldots, S_{\lambda_{n-1}}, S_{\lambda_n}\}$ be the set of nested closures, which can be generated using the NLG Algorithm, for the directed graph G. Then*

$$\bar{\nu}(S_{\lambda_0}) < \bar{\nu}(S_{\lambda_1}) < \cdots < \bar{\nu}(S_{\lambda_{n-1}}). \tag{6.8}$$

Note that since S_{λ_n} is the empty set, its average value is undefined. Thus, it has been dropped from the relation.

Proof: Assume for any two consecutive closures in S, say $S_{\lambda_{i-1}}$ and S_{λ_i}, that

$$\bar{\nu}(S_{\lambda_{i-1}}) \geq \bar{\nu}(S_{\lambda_i}). \tag{6.9}$$

Then

$$\begin{aligned}
\frac{\nu(S_{\lambda_{i-1}})}{|S_{\lambda_{i-1}}|} &\geq \frac{\nu(S_{\lambda_i})}{|S_{\lambda_i}|}, \\
\frac{\nu(S_{\lambda_i} \cup I_{\lambda_i})}{|S_{\lambda_i} \cup I_{\lambda_i}|} &\geq \frac{\nu(S_{\lambda_i})}{|S_{\lambda_i}|}, \\
\frac{\nu(S_{\lambda_i}) + \nu(I_{\lambda_i})}{|S_{\lambda_i}| + |I_{\lambda_i}|} &\geq \frac{\nu(S_{\lambda_i})}{|S_{\lambda_i}|} \\
|S_{\lambda_i}|(\nu(S_{\lambda_i}) + \nu(I_{\lambda_i})) &\geq (|S_{\lambda_i}| + |I_{\lambda_i}|)\nu(S_{\lambda_i}), \\
\nu(S_{\lambda_i}) + \nu(I_{\lambda_i}) &\geq (1 + \frac{|I_{\lambda_i}|}{|S_{\lambda_i}|})\nu(S_{\lambda_i}), \\
\nu(S_{\lambda_i}) + \nu(I_{\lambda_i}) &\geq \nu(S_{\lambda_i}) + \frac{|I_{\lambda_i}|}{|S_{\lambda_i}|}\nu(S_{\lambda_i}), \\
\nu(I_{\lambda_i}) &\geq \frac{|I_{\lambda_i}|}{|S_{\lambda_i}|}\nu(S_{\lambda_i}), \\
\frac{\nu(I_{\lambda_i})}{|I_{\lambda_i}|} &\geq \frac{\nu(S_{\lambda_i})}{|S_{\lambda_i}|}, \\
\bar{\nu}(I_{\lambda_i}) &\geq \bar{\nu}(S_{\lambda_i}).
\end{aligned} \tag{6.10}$$

But, if Relation 6.10 holds, then in the offset graph G_{λ_i}, S_{λ_i} would have non-positive value. To see this, consider the following. We know, by Corollary 2, $\lambda_i = \bar{\nu}(I_{\lambda_i})$. Thus, substituting λ_i for the left hand side of Relation 6.10, we have:

$$\lambda_i \geq \bar{\nu}(S_{\lambda_i}),$$

$$\begin{aligned} \lambda_i &\geq \frac{\nu(S_{\lambda_i})}{|S_{\lambda_i}|}, \\ \lambda_i|S_{\lambda_i}| &\geq \nu(S_{\lambda_i}), \\ 0 &\geq \nu(S_{\lambda_i}) - \lambda_i|S_{\lambda_i}|, \\ 0 &\geq \nu_{\lambda_i}(S_{\lambda_i}). \end{aligned} \tag{6.11}$$

The final step follows from Equation 6.2. Thus, S_{λ_i} has non-positive value in the offset graph G_{λ_i}. Hence, it is not possible for S_{λ_i} to be the maximum valued closure in G_{λ_i}. Therefore, it is not possible for Relation 6.9 to hold. □

Theorem 10 shows that the average value of the nested closures generated by the NLG Algorithm decreases monotonically. Thus, the smallest nested closure will have the largest average value of all nested closures. The final nested closure will have the smallest. We call this the NLG Algorithm's *ordering principal.*

Theorem 10 shows that the average value of the increments also decreases monotonically, starting with the increment I_{λ_n} and ending with the increment I_{λ_1}.

These properties will be used in the next chapter to develop approaches for solving the local ordering problem. This provides approaches for generating good complete extraction sequences. We use the term good since we were unable to prove these approaches define optimal local extraction sequences.

Chapter 7

MAXIMIZING THE GLOBAL CASH FLOW FUNCTION

In this chapter, we prove the global ordering defined by the NLG Algorithm, maximizes the *integral* of the cash flow function. A logical question to ask is, why maximize the integral of the *global* cash flow function. We know the value of a maximum valued closure remains constant, irrespective of the extraction sequence. The primary reason for maximizing the integral of the cash flow function is its relationship with the payback period of the investment.

Consider the cash flow functions depicted in Figure 7.1. Both functions reflect different extraction sequences for the same maximum valued closure. As indicated by the original investment line (dashed line) the function on the left has the shorter pay back period (when the cash flow equals the original investment). The function on the left also has a larger integral than the function on the right. Thus, the larger the integral value of the cash flow function the shorter the payback period of the investment.

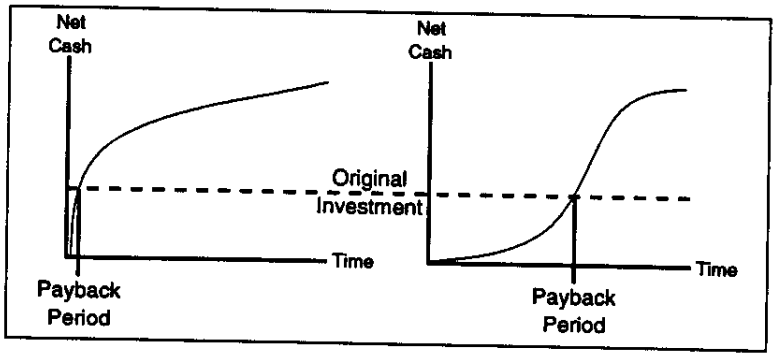


Figure 7.1: Payback Period Comparison

After proving the ordering defined by the NLG Algorithm maximizes the integral of the global cash flow function, we use the properties of the NLG Algorithm to develop two approaches for solving the local ordering problem. Recall that the NLG Algorithm provides no direct information for solving the local ordering problem. We demonstrate that the NLG Algorithm's ordering principal can be used to define local orderings.

In Section 7.1, an example problem is used to compare two global orderings: the global ordering generated by maximizing the *average* value of the closure (the NLG ordering); and, the global ordering generated by simply maximizing the value of the closure. The comparison demonstrates that the global cash flow function generated by maximizing the average value of the closure has a greater integral value.

In Section 7.2, we prove the integral of the cash flow function is maximized across all possible orderings of the characteristic points by ordering the characteristic points such that: the slopes of the chords between characteristic points is monotonically decreasing. Since the NLG Algorithm orders the characteristic points in such a manner, we prove the NLG Algorithm maximizes the global cash flow function.

In Section 7.3, we use properties of the NLG algorithm to develop two approaches for obtaining solutions to the *local ordering problem.* The first approach adds nodes to the smaller nested closure until the larger nested closure is reached. The second approach removes nodes from the larger nested closure until the smaller nested closure is reached. Both approaches generated the same local ordering for the example problem. We were unable to identify a better local ordering.

7.1 Comparison of Global Ordering Approaches

Consider the directed graph shown in Figure 7.2. The maximum valued closure of this graph, S, contains those nodes which have shadowed node symbols. The maximum valued closure contains three independent positive valued closures, as identified in Table 7.1. The graph was constructed to have independent closures to simplify the analysis. If the closures had been dependent, then the issue of dependencies between closures would have unnecessarily complicated the analysis.

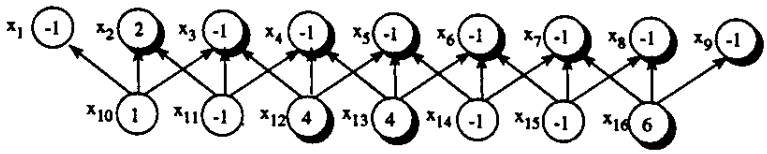


Figure 7.2: Example Directed Graph

Table 7.1 shows, for each of the three independent positive valued closures: its set of nodes, the number of nodes, its value, and its average value. Let I_k, $i = 1, 2, 3$ denote the three independent closures. From the table we find, I_1 has the largest value, and I_3 has the largest *average* value. We use I_k to denote the closures since each closure is also an increment, as defined in Chapter 6. From this point on, we shall refer to the closures as increments. Since the increments are independent, they can be ordered in $3! = 6$ ways.

It is important to note that the increments used are the increments generated by the NLG Algorithm. These increments were chosen since we wish to demonstrate in what sense the NLG Algorithm's ordering is optimal. In the next section, we prove the NLG Algorithm's ordering maximizes the integral of the *global* cash flow function, with respect to the increments defined by the NLG Algorithm. We also argue that

Table 7.1: Independent Closures

Closure Number k	Closure Nodes I_k	Number Nodes $\|I_k\|$	Closure Value $\nu(I_k)$	Average Value $\bar{\nu}(I_k)$
1	$\{x_3, x_4, x_5, x_6, x_{12}, x_{13}\}$	6	4	4/6=0.6667
2	$\{x_7, x_8, x_9, x_{16}\}$	4	3	3/4=0.7500
3	$\{x_2\}$	1	2	2/1=2.0000

the global cash flow function generated by the NLG Algorithm has an integral value greater than the integral value of any other set of increments, assuming the number of increments are equal. An increment different from the increments defined by the NLG Algorithm is easy to define. For example, nodes x_3, x_4, x_5, and x_{12} define an increment which is different from any of the NLG Algorithm's increments.

We wish to determine which of the six global orderings maximizes the integral value of the *global* cash flow function. Let π_j, $j = 1, \ldots, 6$, denote the six global orderings. Thus, if $\pi_6 = [3, 2, 1]$, then increment I_3 is the first increment in the extraction sequence, I_2 is the second, and I_1 is the third and final. We use $\pi_j(i)$ to denote the *ith* term in the permutation. Thus, $\pi_6(1) = 3$, $\pi_6(2) = 2$, and $\pi_6(3) = 1$.

Let $N_{\pi,k}$ denote the *kth* nested closure generated by ordering π. For the ordering π_6, we have $N_{\pi_6,1} = I_3$, $N_{\pi_6,2} = I_3 \cup I_2$, and $N_{\pi_6,3} = I_3 \cup I_2 \cup I_1$. In general, for an ordering π_j,

$$N_{\pi_j,k} = \bigcup_{i=1}^{k} I_{\pi_j(i)}. \tag{7.1}$$

Each nested closures define a characteristic point of the global cash flow function. Let $F_{\pi_j}(x)$ denote the global cash flow function generated by ordering π_j. It is formally defined in the next section.

Table 7.2: Nested Closures Defined By Ordering π_1

Nested Closure Number k	Nested Closure Nodes $N_{\pi_1,k}$	Number Nodes $\lvert N_{\pi_1,k}\rvert$	Nested Closure Value $\nu(N_{\pi_1,k})$
1	$\{x_3, x_4, x_5, x_6, x_{12}, x_{13}\}$	6	4
2	$\{x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{12}, x_{13}, x_{16}\}$	10	7
3	$\{x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{12}, x_{13}, x_{16}\}$	11	9

Table 7.2 shows, for each nested closure of the ordering $\pi_1 = [1, 2, 3]$: it's nodes, the number of nodes, and the its value. Using the number of nodes and the value of each nested closure to define the characteristic points of the global cash flow function, we obtain the function shown in Figure 7.3.

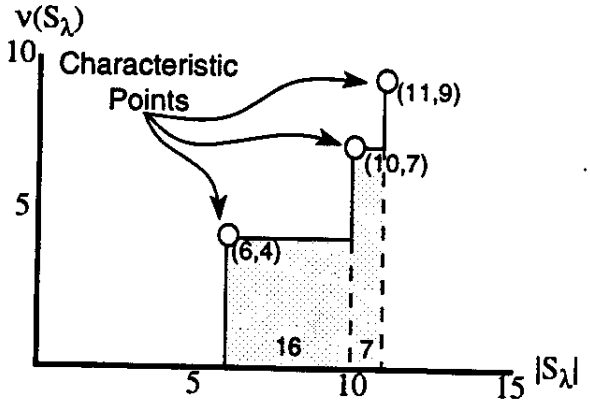


Figure 7.3: Cash Flow Function For Ordering π_1

Integrating $F_{\pi_1}(x)$ we have:

$$\int_0^{|S|} F_{\pi_1}(x)dx = \int_0^{|N_{\pi_1,1}|} 0dx + \int_{|N_{\pi_1,1}|}^{|N_{\pi_1,2}|} \nu(N_{\pi_1,1})dx + \int_{|N_{\pi_1,2}|}^{|N_{\pi_1,3}|} \nu(N_{\pi_1,2})dx,$$

Table 7.3: Nested Closures Defined By Ordering π_6

Nested Closure Number k	Nested Closure Nodes $N_{\pi_6,k}$	Number Nodes $\lvert N_{\pi_6,k}\rvert$	Nested Closure Value $\nu(N_{\pi_6,k})$
1	$\{x_2\}$	1	2
2	$\{x_2, x_7, x_8, x_9, x_{16}\}$	5	5
3	$\{x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{12}, x_{13}, x_{16}\}$	11	9

$$
\begin{aligned}
&= \int_0^6 0dx + \int_6^{10} 4dx + \int_{10}^{11} 7dx, \\
&= 0 + 16 + 7, \\
&= 23,
\end{aligned}
$$

where S is the maximum valued closure of the directed graph. The shaded regions in Figure 7.3 indicate the integral of the cash flow function. Each value above the horizontal axis in a shaded region is the value of the integral for that region.

Table 7.3 shows the same information for ordering $\pi_6 = [3, 2, 1]$. Figure 7.4 shows the resulting cash flow function.

Integrating $F_{\pi_6}(x)$ we have:

$$
\begin{aligned}
\int_0^{|S|} F_{\pi_6}(x)dx &= \int_0^{|N_{\pi_6,1}|} 0dx + \int_{|N_{\pi_6,1}|}^{|N_{\pi_6,2}|} \nu(N_{\pi_6,1})dx + \int_{|N_{\pi_6,2}|}^{|N_{\pi_6,3}|} \nu(N_{\pi_6,2})dx, \\
&= \int_0^1 0dx + \int_1^5 2dx + \int_5^{11} 5dx, \\
&= 0 + 8 + 30, \\
&= 38.
\end{aligned}
$$

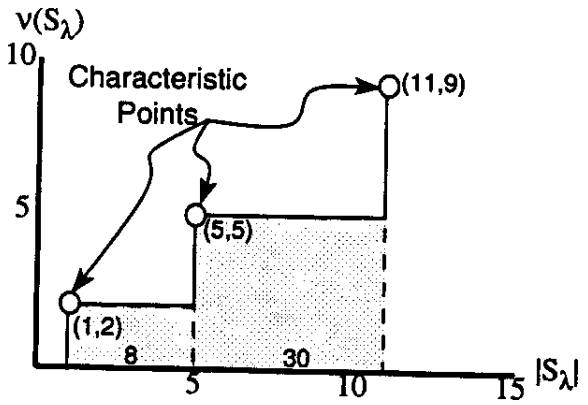


Figure 7.4: Cash Flow Function For Ordering π_6

Table 7.4: Integral Value Of Global Cash Flow Function For All Orderings

Ordering Number j	Ordering π_j	Integrated Cash Flow $\int_0^{\lvert S\rvert} F_{\pi_j}(x)dx$
1	[1, 2, 3]	23
2	[1, 3, 2]	28
3	[2, 1, 3]	25
4	[2, 3, 1]	33
5	[3, 1, 2]	36
6	[3, 2, 1]	38

Thus, we see different global orderings form cash flow functions with different integral values. Table 7.4 shows, for each of the six orderings: the ordering, and the value of the integral of the cash flow function. Thus, for the example problem, π_6 formed the global cash flow function with the largest integral value. This ordering is based upon maximizing the *average* value of the closure: I_3 had the largest average value, I_2 had the next to largest average value, and I_1 had the smallest average value. This is also the ordering generated by the NLG Algorithm.

In the next section, we prove that the integral of the global cash flow function is maximized (over a set of characteristic points), by the permutation which orders the characteristic points such that the slopes of the chords between characteristic points is monotonically decreasing. Since the NLG Algorithm orders the characteristic points in such a manner, we will have proven the NLG Algorithm maximizes the integral of the *global* cash flow function.

7.2 Maximizing The Integral Of The Global Cash Flow Function

Let $I_k = N_k \sim N_{k-1},\ k = 1, \ldots, n$, denote the n increments generated by the NLG Algorithm. For each increment, $I_k,\ k = 1, \ldots n$, we can define the following ordered pair $(|I_k|, \nu(I_k))$. Ignoring the fact that some increments will be dependent upon other increments, we can arrange the n ordered pairs in $n!$ ways. Let π denote a permutation of the n ordered pairs, such that $\pi(i)$ is the index of the ith ordered pair. Thus, under the permutation π the kth nested closure is:

$$N_{\pi,k} = \bigcup_{i=1}^{k} I_{\pi(i)}. \tag{7.2}$$

Thus, we have

$$\begin{aligned} N_{\pi,k} \sim N_{\pi,k-1} &= \bigcup_{i=1}^{k} I_{\pi(i)} \sim \bigcup_{i=1}^{k-1} I_{\pi(i)}, \\ &= \left(I_{\pi(k)} \cup \bigcup_{i=1}^{k-1} I_{\pi(i)} \right) \sim \bigcup_{i=1}^{k-1} I_{\pi(i)}, \\ N_{\pi,k} \sim N_{\pi,k-1} &= I_{\pi(k)}. \end{aligned} \tag{7.3}$$

For example, consider the example discussed in the previous section. The set of

increments I_k, $k = 1, 2, 3$, were identified in Table 7.1. The nested closure $N_{\pi_6,1} = I_3$, $N_{\pi_6,2} = I_3 \cup I_2$, and $N_{\pi_6,3} = I_3 \cup I_2 \cup I_1$.

Note that the notation π, k has a different meaning from the notation $\pi(k)$. The notation π, k denotes a value constructed from the $1, \ldots, k$ elements of the permutation π. The notation $\pi(k)$ denotes the *kth* element in the permutation π. For example, $N_{\pi,2}$ is a value constructed from the first two elements of the permutation, $I_{\pi(1)}$ and $I_{\pi(2)}$.

Thus, the permutation π, of n increments, will form $n + 1$ nested closures,

$$\{N_{\pi,0}, N_{\pi,1}, \ldots, N_{\pi,n}\}.$$

The nested closure $N_{\pi,0}$ is empty. The nested closure $N_{\pi,n}$ equals the maximum valued closure S. Each nested closure defines a characteristic point of the cash flow function. Denote the characteristic points as $(|N_{\pi,i}|, \nu(N_{\pi_i}))$, where $|N_{\pi,i}|$ is the size of the nested closure, and $\nu(N_{\pi_i})$ is the value of the nested closure. Thus, the global cash flow function defined by permutation π is:

$$F_\pi(x) = \begin{cases} \nu(N_{\pi,0}), & |N_{\pi,0}| \le x < |N_{\pi,1}|, \\ \nu(N_{\pi,1}), & |N_{\pi,1}| \le x < |N_{\pi,2}|, \\ \nu(N_{\pi,2}), & |N_{\pi,2}| \le x < |N_{\pi,3}|, \\ \vdots & \\ \nu(N_{\pi,i}), & |N_{\pi,i}| \le x < |N_{\pi,i+1}|, \\ \vdots & \\ \nu(N_{\pi,n-1}), & |N_{\pi,n-1}| \le x < |N_{\pi,n}|, \\ \nu(N_{\pi,n}), & |N_{\pi,n}| = x, \\ 0, & |N_{\pi,n}| < x. \end{cases}$$

The function is non-negative valued, and consists of n horizontal pieces.

For the ordering π_6 in the example, we have:

$$F_{\pi_6}(x) = \begin{cases} 0 & 0 \le x < 1, \\ 2, & 1 \le x < 5, \\ 5, & 5 \le x < 11, \\ 9, & 11 = x, \\ 0, & 11 < x. \end{cases}$$

The function $F_\pi(x)$ consists of $n-1$ horizontal pieces (or *steps*). The slope of the chord between the $(k-1)th$ and kth charactistic points is:

$$\begin{aligned} m_{\pi,k} &= \frac{\nu(N_{\pi,k}) - \nu(N_{\pi,k-1})}{|N_{\pi,k}| - |N_{\pi,k-1}|}, \\ &= \frac{\nu(N_{\pi,k} \sim N_{\pi,k-1})}{|N_{\pi,k} \sim N_{\pi,k-1}|}, \\ &= \frac{\nu(I_{\pi(k)})}{|I_{\pi(k)}|}. \end{aligned} \tag{7.4}$$

The last step follows from Equation 7.3.

Integrating $F_\pi(x)$ we obtain,

$$\begin{aligned} \int_{-\infty}^{\infty} F_\pi(x)dx &= \int_{|N_{\pi,1}|}^{|N_{\pi,n}|} F_\pi(x)dx, \\ &= \nu(N_{\pi,1})(|N_{\pi,2}| - |N_{\pi,1}|) + \cdots \\ &\quad +\nu(N_{\pi,k-1})(|N_{\pi,k}| - |N_{\pi,k-1}|) + \cdots \\ &\quad +\nu(N_{\pi,n-1})(|N_{\pi,n}| - |N_{\pi,n-1}|), \end{aligned}$$

$$
\begin{aligned}
&= \nu(N_{\pi,1})(|N_{\pi,2} \sim N_{\pi,1}|) + \cdots \\
&\quad + \nu(N_{\pi,k-1})(|N_{\pi,k} \sim N_{\pi,k-1}|) + \cdots \\
&\quad + \nu(N_{\pi,n-1})(|N_{\pi,n} \sim N_{\pi,n-1}|), \\
&= \nu(N_{\pi,1})|I_{\pi(2)}| + \cdots \\
&\quad + \nu(N_{\pi,k-1})|I_{\pi(k)}| + \cdots \\
&\quad + \nu(N_{\pi,n-1})|I_{\pi(n)}|, \\
&= \sum_{i=1}^{n-1} \nu(N_{\pi,i})|I_{\pi(i+1)}|. \qquad (7.5)
\end{aligned}
$$

The next to last step follows from Equation 7.3.

Theorem 11 *Let N_k, $k = 0, \ldots, n$, denote the $n+1$ nested closures, obtained using the NLG Algorithm, in the directed graph G. Let $I_k = N_k \sim N_{k-1}$, $k = 1, \ldots, n$, be the n increments between nested closures. Let π_d denote the permutation of the n ordered pairs, $\{(|I_1|, \nu(I_1)), \ldots, (|I_n|, \nu(I_n))\}$, such that*

$$
m_{\pi_d,1} > \cdots > m_{\pi_d,k} > m_{\pi_d,k+1} > \cdots > m_{\pi_d,n}, \qquad (7.6)
$$

where m is the slope of the chord between characteristic points, as in Equation 7.4. Denote the characteristic points, under the permutation π_d, as $(|N_{\pi_d,k}|, \nu(N_{\pi_d,k}))$, $k = 0, \ldots, n$. Then

$$
\int_{-\infty}^{\infty} F_{\pi_d}(x)dx > \int_{-\infty}^{\infty} F_{\pi}(x)dx, \qquad (7.7)
$$

for any other permutation, π, of the increments.

Proof: From Equation 7.5, we know

$$\begin{aligned}
\int_{-\infty}^{\infty} F_{\pi_d}(x)dx &= \sum_{i=1}^{n-1} \nu(N_{\pi_d,i})|I_{\pi_d(i+1)}|, \\
&= \sum_{i=1}^{k-2} \nu(N_{\pi_d,i})|I_{\pi_d(i+1)}| \\
&\quad +\nu(N_{\pi_d,k-1})|I_{\pi_d(k)}| \\
&\quad +\nu(N_{\pi_d,k})|I_{\pi_d(k+1)}| \\
&\quad +\nu(N_{\pi_d,k+1})|I_{\pi_d(k+2)}| \\
&\quad + \sum_{i=k+2}^{n-1} \nu(N_{\pi_d,i})|I_{\pi_d(i+1)}|, \\
&= \sum_{i=1}^{k-2} \nu(N_{\pi_d,i})|I_{\pi_d(i+1)}| \\
&\quad +\nu(N_{\pi_d,k-1})|I_{\pi_d(k)}| \\
&\quad +\nu(N_{\pi_d,k-1} \cup I_{\pi_d(k)})|I_{\pi_d(k+1)}| \\
&\quad +\nu(N_{\pi_d,k+1})|I_{\pi_d(k+2)}| \\
&\quad + \sum_{i=k+2}^{n-1} \nu(N_{\pi_d,i})|I_{\pi_d(i+1)}|, \\
&= \sum_{i=1}^{k-2} \nu(N_{\pi_d,i})|I_{\pi_d(i+1)}| \\
&\quad +\nu(N_{\pi_d,k-1})|I_{\pi_d(k)}| \\
&\quad +\nu(N_{\pi_d,k-1})|I_{\pi_d(k+1)}| \\
&\quad +\nu(I_{\pi_d(k)})|I_{\pi_d(k+1)}|
\end{aligned}$$

$$+\nu(N_{\pi_d,k+1})|I_{\pi_d(k+2)}|$$
$$+\sum_{i=k+2}^{n-1}\nu(N_{\pi_d,i})|I_{\pi_d(i+1)}|. \quad (7.8)$$

The final step follows since $\nu(N_{\pi_d,k}) = \nu(N_{\pi_d,k-1} \cup I_{\pi_d(k)})$. We return to Equation 7.8 below.

Now consider a different permutation, π, that interchanges the kth and $(k+1)st$ terms of permutation π_d. Thus,

$$I_{\pi(k)} = I_{\pi_d(k+1)}, \quad (7.9)$$

and

$$I_{\pi(k+1)} = I_{\pi_d(k)}, \quad (7.10)$$

and, for $i = 1, \ldots, k-1, k+2, \ldots n$,

$$I_{\pi(i)} = I_{\pi_d(i)}. \quad (7.11)$$

From Equation 7.2, the kth nested closure in permutation π is:

$$N_{\pi,k} = \bigcup_{i=1}^{k} I_{\pi(i)},$$
$$= \bigcup_{i=1}^{k-1} I_{\pi(i)} \cup I_{\pi(k)}. \quad (7.12)$$

Using Equations 7.9 and 7.11, we can express Equation 7.12 in terms of π_d:

$$\begin{aligned} N_{\pi,k} &= \bigcup_{i=1}^{k-1} I_{\pi_d(i)} \cup I_{\pi(k)}, \\ &= \bigcup_{i=1}^{k-1} I_{\pi_d(i)} \cup I_{\pi_d(k+1)}, \\ &= N_{\pi_d,k-1} \cup I_{\pi_d(k+1)}. \end{aligned}$$

Using Equation 7.2 again, we have

$$\begin{aligned} N_{\pi,k+1} &= \bigcup_{i=1}^{k+1} I_{\pi(i)}, \\ &= \bigcup_{i=1}^{k-1} I_{\pi(i)} \cup I_{\pi(k)} \cup I_{\pi(k+1)}. \end{aligned} \tag{7.13}$$

Using Equations 7.10 and 7.11, we can express Equation 7.13 in terms of π_d, we have:

$$\begin{aligned} N_{\pi,k+1} &= \bigcup_{i=1}^{k-1} I_{\pi_d(i)} \cup I_{\pi(k)} \cup I_{\pi(k+1)}, \\ &= \bigcup_{i=1}^{k-1} I_{\pi_d(i)} \cup I_{\pi_d(k+1)} \cup I_{\pi_d(k)}, \\ &= \bigcup_{i=1}^{k+1} I_{\pi_d(i)}, \\ &= N_{\pi_d,k+1}. \end{aligned}$$

Integrating F_π we have

$$\int_{-\infty}^{\infty} F_\pi(x)dx = \sum_{i=1}^{n-1} \nu(N_{\pi,i})|I_{\pi(i+1)}|,$$

$$
\begin{aligned}
&= \sum_{i=1}^{k-2} \nu(N_{\pi,i})|I_{\pi(i+1)}| \\
&\quad +\nu(N_{\pi,k-1})|I_{\pi(k)}| \\
&\quad +\nu(N_{\pi,k})|I_{\pi(k+1)}| \\
&\quad +\nu(N_{\pi,k+1})|I_{\pi(k+2)}| \\
&\quad + \sum_{i=k+2}^{n-1} \nu(N_{\pi,i})|I_{\pi(i+1)}|. \qquad (7.14)
\end{aligned}
$$

Expressing Equation 7.14 in terms of π_d, we have:

$$
\begin{aligned}
&= \sum_{i=1}^{k-2} \nu(N_{\pi_d,i})|I_{\pi_d(i+1)}| \\
&\quad +\nu(N_{\pi_d,k-1})|I_{\pi_d(k+1)}| \\
&\quad +(\nu(N_{\pi_d,k-1}) \cup I_{\pi_d(k+1)})|I_{\pi_d(k)}| \\
&\quad +\nu(N_{\pi_d,k+1})|I_{\pi_d(k+2)}| \\
&\quad + \sum_{i=k+2}^{n-1} \nu(N_{\pi_d,i})|I_{\pi_d(i+1)}|. \qquad (7.15)
\end{aligned}
$$

Expanding $\nu(N_{\pi_d,k-1}) \cup I_{\pi_d(k+1)})|I_{\pi_d(k)}|$ in Equation 7.15, we have:

$$
\begin{aligned}
&= \sum_{i=1}^{k-2} \nu(N_{\pi_d,i})|I_{\pi_d(i+1)}| \\
&\quad +\nu(N_{\pi_d,k-1})|I_{\pi_d(k+1)}| \\
&\quad +\nu(N_{\pi_d,k-1})|I_{\pi_d(k)}| \\
&\quad +\nu(I_{\pi_d(k+1)})|I_{\pi_d(k)}|
\end{aligned}
$$

$$+\nu(N_{\pi_d,k+1})|I_{\pi_d(k+2)}|$$
$$+\sum_{i=k+2}^{n-1}\nu(N_{\pi_d,i})|I_{\pi_d(i+1)}|. \tag{7.16}$$

Subtracting the integral of F_π from the integral of F_{π_d}, Equation 7.16 from 7.8, we have

$$\begin{aligned}\int_{-\infty}^{\infty} F_{\pi_d}(x) - F_\pi(x)dx \;=\; & \sum_{i=1}^{k-2}\nu(N_{\pi_d,i})|I_{\pi_d(i+1)}| \\ & +\nu(N_{\pi_d,k-1})|I_{\pi_d(k)}| \\ & +\nu(N_{\pi_d,k-1})|I_{\pi_d(k+1)}| \\ & +\nu(I_{\pi_d(k)})|I_{\pi_d(k+1)}| \\ & +\nu(N_{\pi_d,k+1})|I_{\pi_d(k+2)}| \\ & +\sum_{i=k+2}^{n-1}\nu(N_{\pi_d,i})|I_{\pi_d(i+1)}| \\ & -\sum_{i=1}^{k-2}\nu(N_{\pi_d,i})|I_{\pi_d(i+1)}| \\ & -\nu(N_{\pi_d,k-1})|I_{\pi_d(k+1)}| \\ & -\nu(N_{\pi_d,k-1})|I_{\pi_d(k)}| \\ & -\nu(I_{\pi_d(k+1)})|I_{\pi_d(k)}| \\ & -\nu(N_{\pi_d,k+1})|I_{\pi_d(k+2)}| \\ & -\sum_{i=k+2}^{n-1}\nu(N_{\pi_d,i})|I_{\pi_d(i+1)}|. \end{aligned} \tag{7.17}$$

Cancelling terms in Equation 7.17, we have

$$\begin{aligned}\int_{-\infty}^{\infty} F_{\pi_d}(x) - F_{\pi}(x)dx &= \nu(I_{\pi_d(k)})|I_{\pi_d(k+1)}| \\ &\quad -\nu(I_{\pi_d(k+1)})|I_{\pi_d(k)}|. \end{aligned} \tag{7.18}$$

Dividing Equation 7.18 by $|I_{\pi_d(k+2)}||I_{\pi_d(k+1)}|$ we obtain

$$\frac{\int_{-\infty}^{\infty} F_{\pi_d}(x) - F_{\pi}(x)dx}{|I_{\pi_d(k)}||I_{\pi_d(k+1)}|} = \frac{\nu(I_{\pi_d(k)})}{|I_{\pi_d(k)}|} - \frac{\nu(I_{\pi_d(k+1)})}{|I_{\pi_d(k+1)}|}. \tag{7.19}$$

Using Equation 7.4, for the two terms on the right hand side of Equation 7.19, we have

$$\frac{\int_{-\infty}^{\infty} F_{\pi_d}(x) - F_{\pi}(x)dx}{|I_{\pi_d(k)}||I_{\pi_d(k+1)}|} = m_{\pi_d,k} - m_{\pi_d,k+1}. \tag{7.20}$$

But, from Equation 7.7, we know $m_{\pi_d,k} > m_{\pi_d,k+1}$. Thus,

$$\begin{aligned}\frac{\int_{-\infty}^{\infty} F_{\pi_d}(x) - F_{\pi}(x)dx}{|I_{\pi_d(k)}||I_{\pi_d(k+1)}|} &> 0, \\ \int_{-\infty}^{\infty} F_{\pi_d}(x) - F_{\pi}(x)dx &> 0, \\ \int_{-\infty}^{\infty} F_{\pi_d}(x)dx - \int_{-\infty}^{\infty} F_{\pi}(x)dx &> 0, \\ \int_{-\infty}^{\infty} F_{\pi_d}(x)dx &> \int_{-\infty}^{\infty} F_{\pi}(x)dx. \end{aligned} \tag{7.21}$$

Thus, interchanging any two increments decreases the value of the integral of F_{π_d}. □

Theorem 11 tells us that the integral of the global cash flow function is max-

imized when the slope of the chords between characteristic points is monotonically decreasing.

Corollary 3 *The NLG Algorithm generates a collection of increments which maximizes the integral of the global cash flow function, with respect to those increments.*

Proof: From Theorem 2, we know that the offset parameter equals the average value of the increment. We also know that the average value of the increments decreases monotonically, when ordered as $I_{\lambda_n}, I_{\lambda_{n-1}}, \ldots, I_{\lambda_1}$. Thus,

$$\bar{\nu}(I_{\lambda_n}) > \bar{\nu}(I_{\lambda_{n-1}}) > \cdots > \bar{\nu}(I_{\lambda_1}).$$

From Equation 7.4, we find $m_i = \bar{\nu}(I_{\lambda_i})$. Thus, Equation 7.6 holds for the NLG Algorithm's increments. Thus, the slopes of the chords between characteristic points of the NLG Algorithm's global cash flow function are monotonically decreasing. Hence, from Theorem 11, we know the NLG Algorithm maximizes the integral of the global cash flow function. □

Theorem 11 and Corollary 3 assume the use of the NLG Algorithm's increments. Although it may appear as though there could exist a different set of increments, which form a global cash flow function whose integral value is greater than the integral value of the global cash flow function generated by the NLG Aglorithm, we do not believe such is possible.

To make comparisons between increment sets fair, the alternative set of increments is restricted to contain only as many increments as were generated by the NLG Algorithm. This requirement is justified since it is easy to identify an alternative set of increments, containing more increments, which generate a global cash flow function with a greater integral value. This is caused by how the measure, the integral of the

global cash flow function, is computed.

For example, consider the trivial closure consisting of two equal valued nodes (let ν denote their value) where both are immediately accessible. Applying the NLG Algorithm to this closure will result in a single increment containing both nodes. The cash flow function has two characteristic points $(0,0)$ and $(2,2\nu)$, and an integral value of zero. Now consider the cash flow function generated when each node is considered to define a different increment. The cash flow function would have the following characteristic points $(0,0)$, $(1,\nu)$, and $(2,\nu)$, and an integral value of ν.

We believe that the chords connecting the characteristic points, generated by the NLG Algorithm, of the cash flow function defines the convex hull of possible nested closures. Thus, every nested closure must fall on or below these chords. Therefore, it is not possible to identify an alternative set of increments, containing an equal number of increments, which has a larger integral value. But, this belief remains to be proven.

7.3 Local Node Ordering

The *local ordering problem* is the problem of ordering nodes contained in the increment between nested closures, $I_{\lambda_k} = S_{\lambda_{k-1}} \sim S_{\lambda_k}$. Since the increment between nested closures can, and typically does in actual problems, contain more than one positive valued node, we find a shortcoming in the NLG Algorithm. This problem is called the *gapping problem.*

To demonstrate, consider the directed graph in Figure 7.5. This graph's maximum valued closure consists of all 24 nodes. Applying the NLG Algorithm results in two nested closures: $S_{\lambda_1} = \{x_2\}$; and, $S_{\lambda_0} = \{x_1, x_2, \ldots, x_{24}\}$. Since the increment contains 23 nodes, in a graph having only 24 nodes, we find the nested closures, de-

fined by the NLG Algorithm, may not provide sufficient information for defining the entire cash flow function.

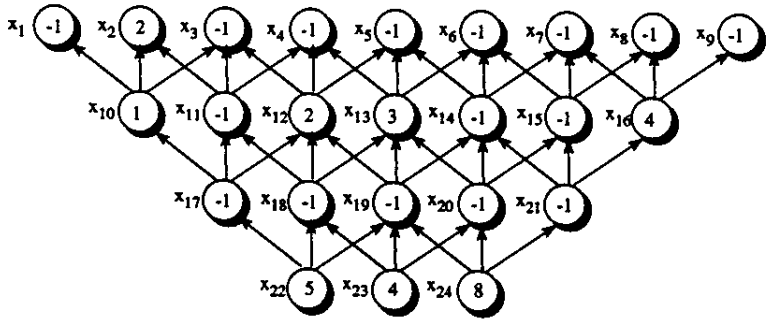


Figure 7.5: The Gapping Problem With The NLG Algorithm

In this section, we introduce two approaches which can be used to find a solution to the local ordering problem. Both approaches are based upon the NLG Algorithm's ordering principal. The first approach starts with the smaller nested closure and adds nodes until the larger nested closure is obtained. The second approach starts with the larger nested closure and removes nodes until the smaller nested closure is obtained. Thus, the first approach is called the *increasing closure approach*, the second approach is called the *decreasing closure approach*. The following sections describe these approaches.

Before the approaches are presented we must develop the restricted superset $2^{S^+_{\lambda_k}}|R$, which is the set of feasible base node sets in S_{λ_k}. Recall from Chapter 6, that G_{λ_0} denotes the non-offset graph and S_{λ_0} is its maximum valued closure. And, that each G_{λ_k} denotes an offset graph and S_{λ_k} is the offset graph's maximum valued closure. Also recall that the increment between two consecutive nested closures was

denoted as:

$$I_{\lambda_k} = S_{\lambda_k} \sim S_{\lambda_{k-1}}.$$

From Lemma 22, we know all base nodes in the smallest maximum valued closure of a directed graph must have non-positive value. Let $S^+_{\lambda_k}$ denote the set of positive valued nodes in S_{λ_k} and, let $2^{S^+_{\lambda_k}}$ denote the collection of all subsets defined by the nodes in $S^+_{\lambda_k}$. Thus, each set in $2^{S^+_{\lambda_k}}$ represents a base node set, which defines a closure in S_{λ_k}.

Some of these closures will have non-positive value and some will define the same closure. Thus, let $2^{S^+_{\lambda_k}}|R$ denote a restriction of $2^{S^+_{\lambda_k}}$ such that: it contains only those base node sets which define closures having positive value; and, for those base node sets which define the same closure, it eliminates all except for the smallest base node set. The restriction also removes any base node sets which can be partitioned into two subsets, where the closure of the first subset has positive value and the closure of the second subset has non-positive value. Let B_i denote a set in $2^{S^+_{\lambda_k}}|R$. Thus, $\Gamma(B_i)$ is a closure in $S^+_{\lambda_k}$ and $\nu(\Gamma(B_i)) > 0$.

For example, consider the maximum valued closure in Figure 7.5. It contains the following positive valued nodes:

$$S^+_{\lambda_0} = \{x_2, x_{10}, x_{12}, x_{13}, x_{16}, x_{22}, x_{23}, x_{24}\}.$$

Thus, the unrestricted set of its subsets, $2^{S^+_{\lambda_0}}$, contains $2^8 = 256$ sets. The closures of some of these subsets have non-positive value. For example, the base node set

$B = \{x_{12}\}$ has the following nodes in its closure:

$$\Gamma(\{x_{12}\}) = \{x_3, x_4, x_5, x_{12}\}.$$

Since the value of the closure ($\nu(\Gamma(\{x_{12}\}) = -1$) is non-positive, the base node set is excluded from the restricted set $2^{S^+_{\lambda_k}}|R$.

In some cases, different base node sets will define the same closure. For example, consider the base node sets $\{x_{16}, x_{24}\}$ and $\{x_{24}\}$. Since x_{16} is contained in the closure of x_{24}, we know:

$$\Gamma(\{x_{16}, x_{24}\}) = \Gamma(\{x_{24}\}). \tag{7.22}$$

Thus, $\{x_{16}, x_{24}\}$, the larger base node set, would be excluded from the restricted set $2^{S^+_{\lambda_k}}|R$.

Now consider the base node set $\{x_2, x_{12}\}$ which has positive value. This base node set can be partitioned into two subsets $\{x_2\}$ and $\{x_{12}\}$, where $\Gamma(x_2)$ has positive value (2) and $\Gamma(x_{12})$ has non-positive value (-1). Thus, the closure $\Gamma(x_2)$ provides support to the closure $\Gamma(x_{12})$. These base node sets are eliminated since they are never chosen in either algorithm.

After removing all base node sets, which define non-positive valued closures and/or generate duplicate closures, we obtain a total of 11 *feasible* base node sets. Table 7.5 shows each feasible base node set for the nested closure S_{λ_0}, and the value of its closure.

Each closure defines an increment with respect to the larger nested closure, S_{λ_0}. For example the base node set B_1 defines the closure $\Gamma(x_2) = \{x_2\}$. The base node

Table 7.5: Sets In The Restricted Set $2^{S^+_{\lambda_k}}|R$

Set Number i	Base Node Set $B_i \in 2^{S^+_{\lambda_0}}|R$	Value $\nu(\Gamma(B_i))$
1	x_2	2
2	x_2, x_{16}	3
3	x_2, x_{12}, x_{13}	3
4	x_{16}	1
5	x_{12}, x_{13}	1
6	x_{12}, x_{13}, x_{16}	2
7	$x_2, x_{12}, x_{13}, x_{16}$	4
8	x_2, x_{24}	7
9	x_{24}	5
10	x_{23}, x_{24}	9
11	x_{22}, x_{24}	13

set $B_2 = \{x_2, x_{16}\}$ defines the closure:

$$\begin{aligned} \Gamma(B_2) &= \Gamma(\{x_2, x_{16}\}), \\ &= \{x_2, x_7, x_8, x_9, x_{16}\}. \end{aligned}$$

With respect to the closure S_{λ_0}, both of these closures define increments. For example,

$$\begin{aligned} S_{\lambda_0} \sim \Gamma(B_1) &= S_{\lambda_0} \sim \{x_2\}, \\ &= \{x_1, x_2, \ldots, x_{24}\} \sim \{x_2\}, \\ &= \{x_1, x_3, x_4, \ldots, x_{24}\}, \end{aligned}$$

and

$$S_{\lambda_0} \sim \Gamma(B_2) = S_{\lambda_0} \sim \{x_2, x_7, x_8, x_9, x_{16}\},$$

$$= \{x_1, x_2, \ldots, x_{24}\} \sim \{x_2, x_7, x_8, x_9, x_{16}\},$$

$$= \{x_1, x_3, \ldots, x_6, x_{10}, \ldots, x_{15}, x_{17}, \ldots, x_{24}\}.$$

We compute the average value of these increments as before. For example,

$$\begin{aligned} \bar{\nu}(S_{\lambda_0} \sim \Gamma(B_1)) &= \bar{\nu}(\{x_1, x_3, x_4, \ldots, x_{24}\}), \\ &= \frac{\nu(\{x_1, x_3, x_4, \ldots, x_{24}\})}{|\{x_1, x_3, x_4, \ldots, x_{24}\}|}, \\ &= \frac{11}{23}. \end{aligned}$$

7.3.1 Increasing Closure Approach

The increasing closure approach uses the following NLG Algorithm property: to find a nested closure's consecutive nested closure remove the increment with the minimum average value. This property follows from the manner in which the NLG Algorithm gradually increases the offset parameter until an offset graph is formed which has a maximum valued closure that is smaller than the original nested closure. When considering the offset parameters, in the order $\lambda_1, \lambda_2, \ldots, \lambda_n$, their values are monotonically decreasing. Since the offset parameter equals the average value of the increment which it forms, the increments average values are monotonically decreasing. Reversing the order of the increments, their average values will be monotonically increasing.

The increasing closure approach starts with the smaller nested closure and adds groups of nodes, which create what we call an *increased closure*. The order in which the increased closures are formed is based upon the average value of the increment

they generate. The increased closure generating the increment with the smallest average value is selected first. Then the process is repeated.

We shall use the nested closures in Figures 7.5 and 7.6 to demonstrate. Figure 7.5 shows the larger nested closure, $S_{\lambda_0} = \{x_1, \ldots, x_{24}\}$. Figure 7.6 shows the smaller nested closure, $S_{\lambda_1} = \{x_2\}$. Thus, we wish to define a local ordering of the nodes in the increment:

$$\begin{aligned} I_{\lambda_1} &= S_{\lambda_0} \sim S_{\lambda_1}, \\ &= \{x_1, x_3, \ldots, x_{24}\}. \end{aligned}$$

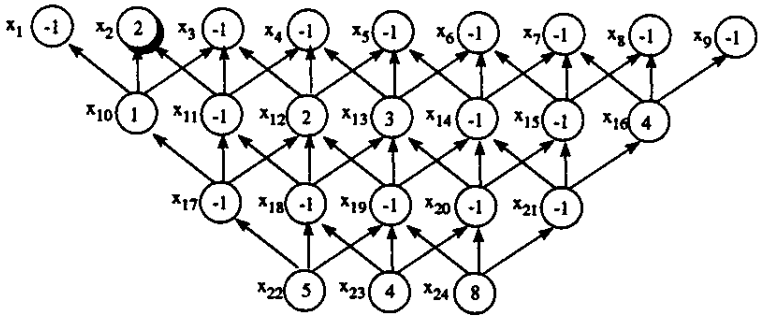


Figure 7.6: The Nested Closure $S_{\lambda_1} = \Gamma(B_1) = \{x_2\}$

The increasing closure approach starts with S_{λ_1} and adds groups of nodes until S_{λ_0} is obtained. To do this, we require each increased closure to contain all nodes in the previous closure. Some of the base node sets in $2^{S^+_{\lambda_0}}|R$ meet this requirement. Call these base node sets the *candidate* base node sets.

Table 7.6 shows, for each base node set in $2^{S^+_{\lambda_0}}|R$: the base node set, the positive valued nodes in the closure of the base node set, and the average value of the increment formed by the base node set.

Table 7.6: Increment Average Values

Set Number i	Base Node Set $B_i \in 2^{S^+_{\lambda_0}} \mid R$	Positive Valued Nodes in $\Gamma(B_i)$	Average Value $\bar{\nu}(S_{\lambda_0} \sim \Gamma(B_i))$
1	x_2	$\{x_2\}$	$11/23 = 0.4783$
2	x_2, x_{16}	$\{x_2, x_{16}\}$	$10/19 = 0.5263$
3	x_2, x_{12}, x_{13}	$\{x_2, x_{12}, x_{13}\}$	$10/17 = 0.5882$
4	x_{16}	$\{x_{16}\}$	$12/20 = 0.6000$
5	x_{12}, x_{13}	$\{x_{12}, x_{13}\}$	$12/18 = 0.6667$
6	x_{12}, x_{13}, x_{16}	$\{x_{12}, x_{13}, x_{16}\}$	$11/14 = 0.7857$
7	$x_2, x_{12}, x_{13}, x_{16}$	$\{x_2, x_{12}, x_{13}, x_{16}\}$	$9/11 = 0.8182$
8	x_2, x_{24}	$\{x_2, x_{12}, x_{13}, x_{16}, x_{24}\}$	$6/7 = 0.8571$
9	x_{24}	$\{x_{12}, x_{13}, x_{16}, x_{24}\}$	$8/8 = 1.0000$
10	x_{23}, x_{24}	$\{x_2, x_{12}, x_{13}, x_{16}, x_{23}, x_{24}\}$	$4/4 = 1.0000$
11	x_{22}, x_{24}	$\{x_2, x_{10}, x_{12}, x_{13}, x_{16}, x_{22}, x_{23}, x_{24}\}$	$4/1 = 4.0000$

It is easy to identify the candidate base node sets in Table 7.6 by using the positive valued nodes column. The candidate base node sets are those base node sets which define closures containing all nodes in S_{λ_1}. Since S_{λ_1} has only one base node, x_2, the candidate base node sets are those containing this node in their closures. Thus, the only candidate base node sets are:

$$\{B_2, B_3, B_7, B_8, B_{10}, B_{11}\}.$$

Base node set B_1 is excluded since it defines the smaller nested closure S_{λ_1}.

Among these base node sets, the base node set with the smallest average valued increment is base node set B_2. Thus, we select B_2 to define the first group of nodes added to S_{λ_1}. We denote the increased closure as $S_{\lambda_1,1}$. Thus,

$$S_{\lambda_1,1} = S_{\lambda_1} \cup \Gamma(B_2).$$

Figure 7.7 shows the closure $S_{\lambda_1,1}$.

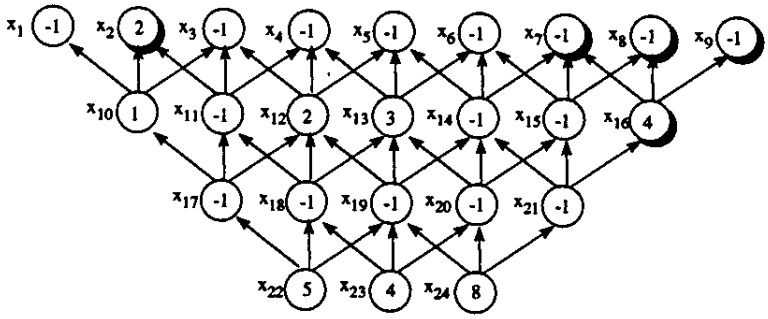


Figure 7.7: The Increased Closure $S_{\lambda_1,1}$

The closure $S_{\lambda_1,1}$ is increased using the same approach. Identify the candidate base node sets, those base node sets defining closures containing all base nodes in $S_{\lambda_1,1}$, which are $\{x_2, x_{16}\}$. The candidate base node sets are:

$$\{B_7, B_8, B_{10}, B_{11}\}.$$

Among these base node sets, base node set B_7 defines the increment with the smallest average value. Thus, the next increased closure is:

$$S_{\lambda_1,2} = S_{\lambda_1,1} \cup \Gamma(B_7).$$

Figure 7.8 shows the closure $S_{\lambda_1,2}$.

The candidate base node sets for $S_{\lambda_1,2}$ are:

$$\{B_8, B_{10}, B_{11}\}.$$

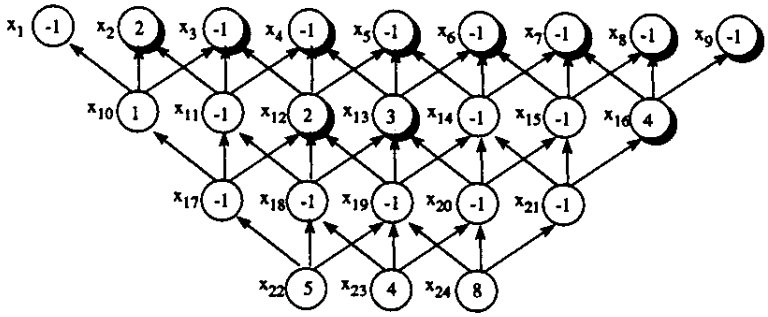


Figure 7.8: The Increased Closure $S_{\lambda_1,2}$

Base node set B_8 defines the increment with the smallest average value. Thus,

$$S_{\lambda_1,3} = S_{\lambda_1,2} \cup \Gamma(B_8).$$

Figure 7.9 shows the closure $S_{\lambda_1,3}$.

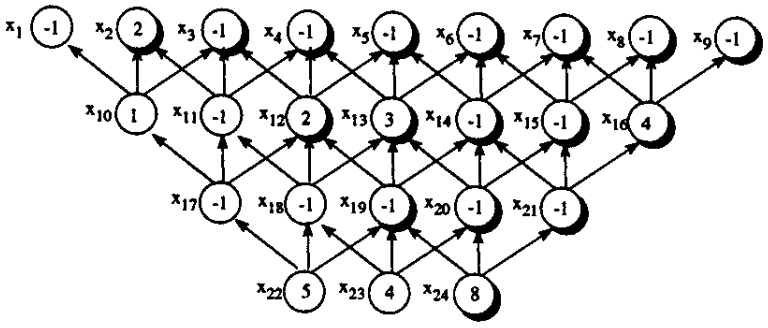


Figure 7.9: The Increased Closure $S_{\lambda_1,3}$

The increased closure $S_{\lambda_1,4}$ is formed from base node set B_{10}, which has an increment with an average value of 1.0. Figure 7.10 shows the resulting increased closure.

The increased closure $S_{\lambda_1,5}$ is formed by the base node set $\{x_{22}, x_{23}, x_{24}\}$ (not shown in the table). Figure 7.11 shows the resulting increased closure. This closure

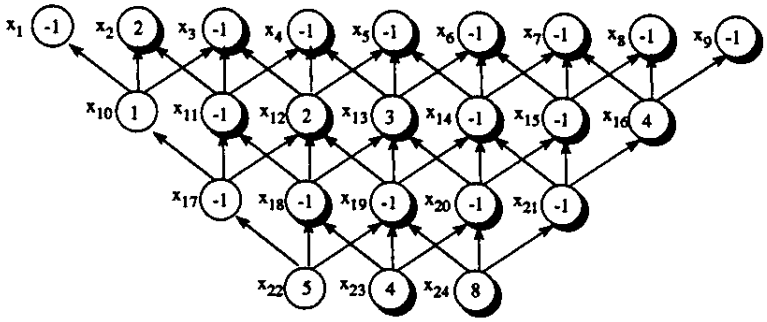


Figure 7.10: The Increased Closure $S_{\lambda_1,4}$

equals the larger nested closure S_{λ_0}.

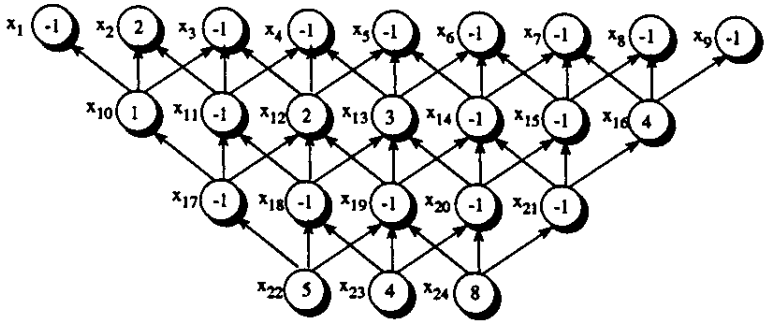


Figure 7.11: The Increased Closure $S_{\lambda_1,5} = S_{\lambda_0}$

Figure 7.12 shows the cash flow function defined by the increased closures

$$\{S_{\lambda_1,1}, \ldots, S_{\lambda_1,5}\}.$$

The light gray lines connect the characteristic points defined by the NLG Algorithm. The slopes of the chords between characteristic points, which equal the value of the offset parameters, are given by the values λ_1 and λ_2. The characteristic point defined by each increased closure is indicated in the figure. The values in the box below the graph are the cumulative values of the integral of the cash flow function.

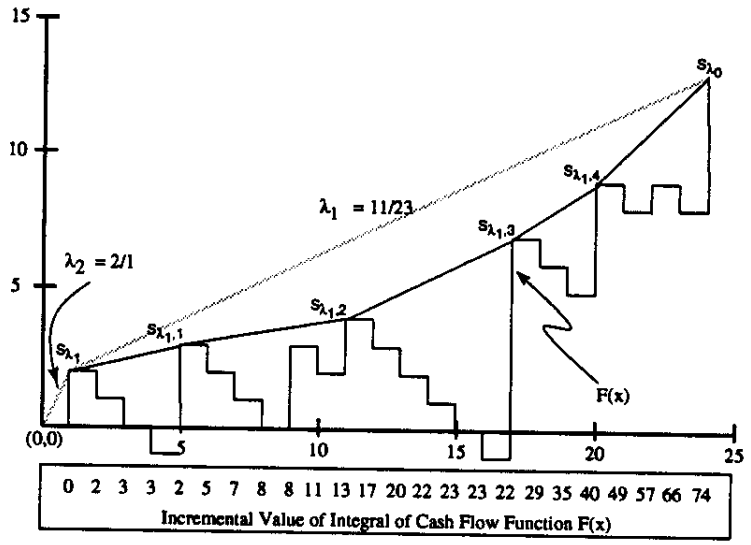


Figure 7.12: The Cash Flow Function When Nesting Based Upon Minimum Average Valued Increment

7.3.2 Decreasing Closure Approach

As proven in Theorem 10, the nested closures defined by the NLG Algorithm have the property that their average values are monotonically decreasing. When considered from the smallest to the largest. In the decreasing closure approach, we use this principal to generate a local ordering of nodes in the increment between nested closures. Let $S_{\lambda_0,k}$ denote the *kth* reduced closure, with respect to the nested closure S_{λ_0}.

The decreasing closure approach starts with the larger nested closure, S_{λ_0}, and removes nodes until the smaller nested closure is reached. The nodes which are removed are based upon the average value of the closures formed when a single base node is removed. The approach is to set, one at a time, the value of each base node in the current closure to zero. Then find the resulting maximum valued closure and its average value. Select the reduced closure, $S_{\lambda_0,1}$, from among those generated, to

be the closure with the maximum average value.

For example, consider the nested closure S_{λ_0}, shown in Figure 7.5, which has base nodes x_{22}, x_{23}, and x_{24}. Select a base node, say node x_{22}, set its value to zero, and solve for the resulting maximum valued closure. Figure 7.13 shows the resulting maximum valued closure, which has an average value of 9/20.

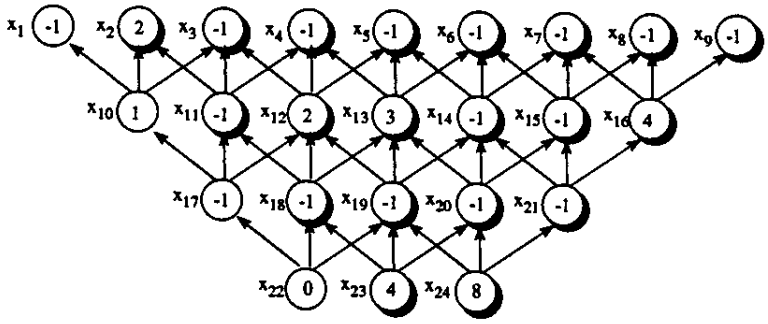


Figure 7.13: The Maximum Valued Closure When $\nu(x_{22}) = 0$

Resetting the value of base node x_{22} to its original value, select another base node, x_{23}, set its value to zero and solve for the resulting maximum valued closure. Figure 7.14 shows the resulting maximum valued closure, which has an average value of 9/23.

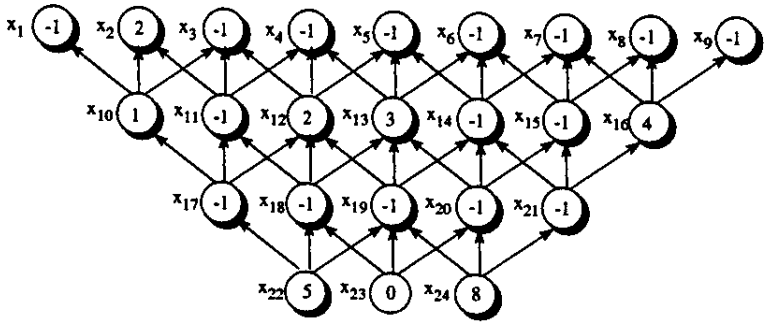


Figure 7.14: The Maximum Valued Closure When $\nu(x_{23}) = 0$

Resetting the value of base node x_{23} to its original value, select another base node, x_{24}, set its value to zero and solve for the resulting maximum valued closure. Figure 7.15 shows the resulting maximum valued closure, which has an average value of 6/22.

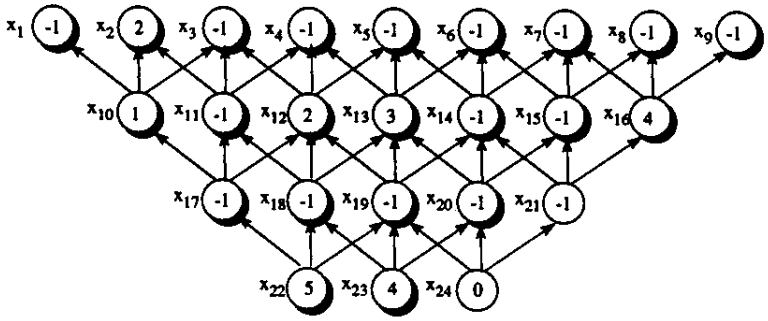


Figure 7.15: The Maximum Valued Closure When $\nu(x_{24}) = 0$

Based upon the average values of the resulting closures, select as the reduced closure, the closure which has the maximum average valued closure. In the example, this would be the closure formed when base node x_{22} was eliminated. Thus,

$$S_{\lambda_0,1} = \Gamma(\{x_{23}, x_{24}).$$

Once $S_{\lambda_0,1}$ has been identified, the process is repeated. The process is repeated until the consecutive nested closure, S_{λ_1} is obtained.

Table 7.7 shows, for each base node: the resulting maximum valued closures value, size and average value. The lines in the table seperate the base nodes for each reduced closure. For example, S_{λ_0} has three base nodes, x_{22}, x_{23}, and x_{24}; and, $S_{\lambda_0,1}$ has two base nodes, x_{23} and x_{24}. Thus, they are the only two base nodes considered in the second iteration.

Table 7.7: Closure Average Values

Eliminated Base Node	Closure Value	Closure Size	Closure Average Value
x_{22}	9	20	0.45*
x_{23}	9	23	0.39
x_{24}	6	22	0.27
x_{23}	7	17	0.41*
x_{24}	4	11	0.36
x_{24}	4	11	0.36*
x_2	2	10	0.20
x_{12}	3	5	0.60*
x_{13}	3	5	0.60*
x_{16}	3	7	0.43
x_2	1	4	0.25
x_{16}	2	1	2.00*

Table 7.8 shows for each reduced closure: its base nodes, its closure value, and the average value of its closure. The first and last reduced closures are the closures S_{λ_0} and S_{λ_1}, respectively. The first decreased closure in shown in Figure 7.11. The second, third, fourth, and fifth decreased closures are shown in Figures 7.10, 7.9, 7.8, and 7.7, respectively.

Generating the cash flow function based upon the reduced closures, we obtain the same function as for the increasing closure approach. Thus, the two approaches obtained the same local ordering.

Although we have not proven that these approaches define the optimal local ordering, our experience indicates they generate good local orderings. Further analysis is required to further develop these approaches.

Table 7.8: Reduced Closure Average Values

Reduced Closure	Base Nodes	Closure Value	Closure Size	Closure Average Value
$S_{\lambda_0,0}$	x_{22}, x_{23}, x_{24}	13	24	0.54
$S_{\lambda_0,1}$	x_{23}, x_{24}	9	20	0.45
$S_{\lambda_0,2}$	x_{24}	7	17	0.41
$S_{\lambda_0,3}$	$x_2, x_{12}, x_{13}, x_{16}$	4	11	0.36
$S_{\lambda_0,4}$	x_2, x_{16}	3	5	0.60
$S_{\lambda_0,5}$	x_2	2	1	2.00

Chapter 8
DISCOUNTING AND THE NLG ALGORITHM

In Chapter 7, we proved the NLG Algorithm maximizes the integral of the global cash flow function, and we used its ordering principal to develop two approaches for defining the complete cash flow function. Although we did not prove the complete cash flow function was optimal, our experience has indicated that it is at least a good solution.

In this chapter, we continue to consider the extraction sequence defined by the NLG Algorithm in conjunction with either of the two approaches developed in the previous chapter. Since a surface mining endeavor typically requires several years to finish, we wish to investigate the sensitivity of the extraction sequence to the time value of money. In other words, we wish to consider the discounted cash flows of the extraction sequence.

In this chapter, we demonstrate that the extraction sequence defined by the NLG Algorithm is not always optimal. We show that for some intervals of the discount multiplier, there exists an extraction sequence having a discounted cash flow greater than the discounted cash flow of the NLG Algorithm's extraction sequence. We use a binary valued directed graph (all positive valued nodes have equal value and all negative valued nodes have equal value) to demonstrate.

We also develop the mathematical programming formulation of the discounted extraction sequence problem. We then argue that the problem is NP-hard, since the formulation must be a zero-one integer program. Thus, as implied by NP-completeness theory, it is not possible to develop a polynomial order algorithm which

can identify the optimal solution. In addition, ϵ-approximation algorithms to zero-one integer programming problems are also NP-hard. Thus, it is not possible to prove that the NLG Algorithm provides an approximate solution that is within some ϵ-bound of the optimal solution.

8.1 The Time Value of Money

The time value of money simply stated says: a dollar today is worth more than a dollar tomorrow. Given the option of receiving a dollar today versus receiving a dollar a year from today, most rational individuals would prefer to receive the dollar today. They would choose to receive the dollar today because they know, if they wanted to, they could invest the dollar. And, in a year have more than a just the dollar. The additional amount received is the *return on the investment.* Call the return on the investment the *discount rate*, and denote it as α.

Using the example, we can express the amount received after a year as $1 + \alpha$. Similarly, an investment of ν_{pv} dollars would have a value of $\nu_{pv}(1 + \alpha)$ at the end of a year. Although the example used a period of one year, the discount rate can be relative to any *period* of time.

When the return of an investment is reinvested in the same investment then *compounding* occurs. Returning to the example, assume, rather than removing the money after the one year period, the investor leaves the entire amount for another year and the same rate of return holds for the following year. Then at the end of the second year the value of the investment is $\nu_{pv}(1 + \alpha)(1 + \alpha)$.

In general, if an investment of ν_{pv} dollars is invested for i periods, the same discount rate applies in each period, and compounding occurs in each period, then

the investment would have a value of

$$\nu_{fv} = \nu_{pv}(1+\alpha)^i. \tag{8.1}$$

Here ν_{fv} denotes the future value of the investment, and ν_{pv} denotes the present value of the investment.

The opposite of compounding is called *discounting*. Discounting determines the present value of a dollar amount received at a future time. For example, if an amount of $\nu_{pv}(1+\alpha)$ was received in a year then the present value of the amount would be ν_{pv}.

In general, we can solve Equation 8.1 for ν_{pv} to determine the *discounted* value of ν_{fv}, if ν_{fv} is received in i discounting periods into the future:

$$\nu_{pv} = \frac{\nu_{fv}}{(1+\alpha)^i}. \tag{8.2}$$

Letting

$$\delta = (1+\alpha)^{-1}, \tag{8.3}$$

be called the *discount multiplier*, we can substitute Equations δ^i for $(1+\alpha)^{-i}$ in Equation 8.2 and we have

$$\nu_{pv} = \nu_{fv}\delta^i. \tag{8.4}$$

Now let us return to the mine extraction problem. Recall that π denotes an ordering (or extraction sequence) of the blocks to be mined, where $\pi(i)$ is the index

of the *ith* block mined. Thus, if it is assumed that the value of the *ith* block mined is received at the end of the *ith* discounting period, then the present value of the cash flow generated by the *ith* mined block can be expressed as $\nu(x_{\pi(i)})\delta^i$. Hence, if there are n blocks to be mined, then the discounted cash flow (present value) of the extraction sequence π can be expressed as:

$$\nu(\pi)_{pv} = \sum_{i=1}^{n} \nu(x_{\pi(i)})\delta^i. \tag{8.5}$$

8.2 A Discounted Cash Flow Example

In this section we provide an example which shows there can exist an extraction sequence having a discounted cash flow which exceeds the discounted cash flow of the NLG extraction sequence. Consider the binary value directed graph shown in Figure 8.1. Nodes with shadowed node symbols (all nodes in the figure) define the directed graph's maximum valued closure. The directed graph is called a binary graph since all positive valued nodes have a value of 3 and all negative valued nodes have a value of -1.

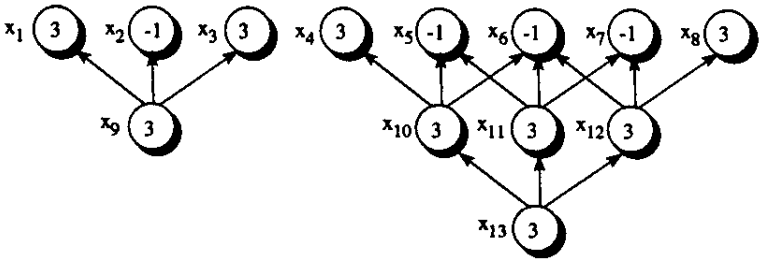


Figure 8.1: Binary Valued Directed Graph G_{λ_0}

We consider binary graphs to be the simplest interesting type of directed graph with respect to maximizing the discounted cash flow. The nodes in a unary directed

graph would all have the same value. Thus, the extraction sequence is irrelevant, all extraction sequences in a unary directed graph have the same value. For graphs having different valued nodes, it is easy to generate examples where extraction sequences have discounted cash flows which exceed the discounted cash flow of the NLG extraction sequence.

Applying the NLG Algorithm to the directed graph, we find the first consecutive closure S_{λ_1} when the offset parameter $\lambda_1 = 1$. Figure 8.2 shows the resulting offset graph $G_{\lambda_1=1}$. As in previous chapters, the values displayed within each node symbol is $\lambda_{den}\nu(x_i) - \lambda_{num}$. The numerator of the node's offset value. The increment between closures S_{λ_0} and S_{λ_1} contains nodes x_2 and x_9.

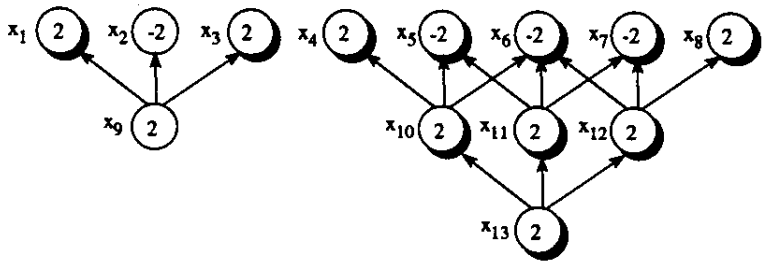


Figure 8.2: The Offset Graph $G_{\lambda_1=1}$

Continuing to reduce the offset parameter, we find the next consecutive closure S_{λ_2} when $\lambda_2 = 9/7$. Figure 8.3 shows the resulting offset graph $G_{\lambda_2=9/7}$. The increment between closures S_{λ_1} and S_{λ_2} is

$$I_{\lambda_2} = \{x_5, x_6, x_7, x_{10}, x_{11}, x_{12}, x_{13}\}.$$

The next consecutive closure S_{λ_3} occurs when the offset parameter $\lambda_3 = 3$, which eliminates all remaining nodes (figure not shown).

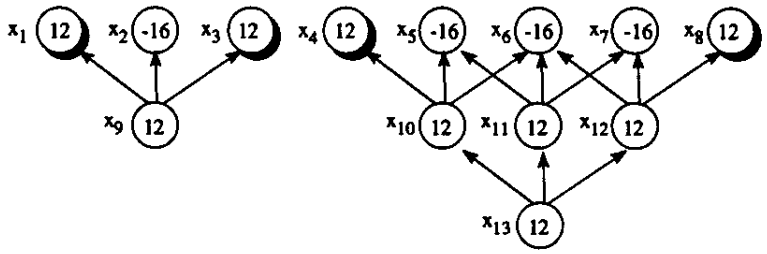


Figure 8.3: The Offset Graph $G_{\lambda_2=9/7}$

Based upon the NLG nested closures we can obtain the NLG extraction sequence shown in Table 8.1. The table specifies: the extraction sequence, the node's value, the incremental integrated value of the *non-discounted* cash flow function, and the discounted cash (assuming $\delta = 0.8$) for each node. The incremental integrated value of the non-discounted cash flow function is computed in the same manner as was presented in Chapter 7. Let π_{NLG} denote the NLG extraction sequence. The total discounted cash flow is obtained by summing the final column in the table. For the NLG extraction sequence, assuming $\delta = 0.8$, the total discounted cash flow is 8.0330.

The NLG extraction sequence starts with the positive valued nodes on the top bench. The next group of nodes are those nodes in the increment I_{λ_2}, which are contained in the larger closure on the right of Figure 8.3. The final group of nodes are those nodes in the increment I_{λ_1}, which are contained in the smaller closure on the left.

An alternative extraction sequence can be generated by simply switching the increments I_{λ_1} and I_{λ_2}. Table 8.2 shows, for the alternative extraction sequence, the same information as Table 8.1. Let π_{Alt} denote the alternative extraction sequence. The total discounted cash flow for the alternative extraction sequence, assuming $\delta = 0.8$, is 8.0878; which is slightly larger than the total discounted cash flow for the NLG

Table 8.1: NLG Defined Extraction Sequence π_{NLG}

Extraction Order i	Node x_i	Node Value $\nu(x_{\pi_{NLG}(i)})$	Integrated Value	Discounted Cash Flow $\nu(x_{\pi_{NLG}(i)})\delta^i$
1	x_1	3	0	2.4000
2	x_3	3	3	1.9200
3	x_4	3	9	1.5360
4	x_8	3	18	1.2288
5	x_5	-1	30	-0.3276
6	x_6	-1	41	-0.2620
7	x_{10}	3	51	0.6288
8	x_7	-1	64	-0.1676
9	x_{11}	3	76	0.4020
10	x_{12}	3	91	0.3216
11	x_{13}	3	109	0.2571
12	x_2	-1	130	-0.0685
13	x_9	3	150	0.1644

extraction sequence.

Comparing the integrated value of the non-discounted cash flow functions for both sequences, we see the NLG extraction sequence had a value of 150 and the alternative extraction sequence had a value of 146. This result follows from Theorem 11, which says the integral value of the NLG extraction sequence's non-discounted cash flow function is greater than that of any other extraction sequence.

As we claimed, we have proven there can exist an extraction sequence, different from the NLG extraction sequence, having a discounted cash flow greater than the NLG extraction sequence. Figure 8.4 shows the difference, $\pi_{NLG} - \pi_{Alt}$, in the discounted cash flows for $\delta \in (0, 1)$. For discount multiplier values greater than 0.85, the difference is positive. Indicating the discounted cash flow of π_{NLG} is greater than the discounted cash flow of π_{Alt}. But, for discount multiplier values less than 0.85,

Table 8.2: Alternative Extraction Sequence π_{Alt}

Extraction Order i	Node x_i	Node Value $\nu(x_{\pi_{Alt}(i)})$	Integrated Value	Discounted Cash Flow $\nu(x_{\pi_{Alt}(i)})\delta^i$
1	x_1	3	0	2.4000
2	x_3	3	3	1.9200
3	x_4	3	9	1.5360
4	x_8	3	18	1.2288
5	x_2	-1	30	-0.3276
6	x_9	3	41	0.7860
7	x_5	-1	55	-0.2096
8	x_6	-1	68	-0.1676
9	x_{10}	3	80	0.4020
10	x_7	-1	95	-0.1072
11	x_{11}	3	109	0.2571
12	x_{12}	3	126	0.2055
13	x_{13}	3	146	0.1644

the difference is negative. Indicating the discounted cash flow of π_{Alt} is greater than the discounted cash flow of π_{NLG}.

Although the example shows there can exist an extraction sequence having a discounted cash flow greater than the discounted cash flow of the NLG extraction sequence, there are mitigating factors which reduce the likelihood of this event occurring. The first mitigating factor is the large value of the annual rate of return (or discount rate). In the problem, the discounting period is determined by the time required to mine a single block. If it is possible to mine a block a day, then a discount multiplier of 0.85 would represent a discount rate of 0.176% per day. Annualizing the discount rate, would give an annual discount rate of 54.24%. A fairly large expected return on an investment. If more than one block can be mined in a day then the annualized discount rate increases. Thus, the point where the discounted cash flow of the NLG extraction sequence equalled the discounted cash flow of the alternative

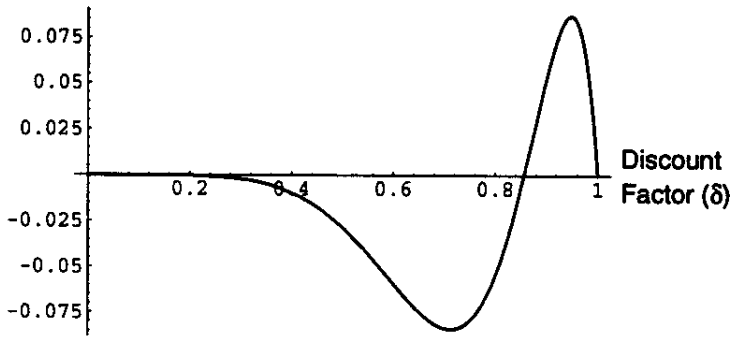


Figure 8.4: Difference In Discounted Value Between Extraction Sequences

extraction sequence occurred for a large expected rate of return.

Another mitigating factor, is the fairly small deviation in the discounted cash flows for the two sequences. The total non-discounted cash flow for the example was 23. From Figure 8.4, we see the discounted cash flow of π_{Alt} exceeded the discounted cash flow of π_{NLG} by only 0.075. A fairly negligible amount of 3 tenths of a percent of the total value. But, with respect to mathematical optimality, it does demonstrate that the NLG extraction sequence is not always optimal with respect to discounted cash flows.

8.3 Linear Programming Formulation Of The Discounted Extraction Sequence Problem

In this section, we develop the linear programming formulation of the extraction sequence problem. The extraction sequence problem may or may not consider discounting of the cash flows. In either case the formulation is identical, except for the presence of the discount multiplier in the objective function for the discounted cash flow formulation.

For a block model containing n blocks, the extraction sequence problem will consist of n periods. In each period at most one block will be selected as the block to extract. This block must be an accessible block. The formulation uses the same accessibility constraint matrix as was used in the linear programming formulation of the Pit Limit Problem. In addition, a block may be extracted only once.

Let C denote the vector of node values, E denote the accessibility constraint matrix, I the identity matrix, and X_i the vector of decision variables in the ith period. Let 1 and 0 denote appropriately dimensioned vectors of ones and zeros, respectively. Let δ denote the discount multiplier, n the number of blocks in the block model, and m the number of direct dependencies between blocks (arcs in the directed graph representation).

$$\max \quad \delta^1 CX_1 + \delta^2 CX_2 + \cdots + \delta^n CX_n$$

$$\begin{array}{lllll} 1^T X_1 & & & & \leq 1 \\ & 1^T X_2 & & & \leq 1 \\ & & \ddots & & \vdots \\ & & & 1^T X_n & \leq 1 \end{array}$$

$$\begin{array}{lllll} EX_1 & & & & \leq 0 \\ EX_1 + & EX_2 & & & \leq 0 \\ \vdots & \vdots & \ddots & & \vdots \\ EX_1 + & EX_2 + & \cdots + & EX_n & \leq 0 \end{array}$$

$$IX_1 + IX_2 + \cdots + IX_n \leq 1$$

The n constraints, $1^T X_i \leq 1$, prevent the extraction of more than a single block in a given period. It is possible for all decision variables to have a value of zero in a given period. The m constraints, $EX_1 + EX_2 + \cdots + EX_i \leq 0$, ensure the extracted block is accessible in the *ith* period. The constraints, $IX_1 + IX_2 + \cdots + IX_n \leq 1$, ensure no block is extracted in more than one period.

Since the problem does not allow for a portion of a block to be removed, the decision variables must be constrained to integral values of either zero or one. In the linear programming formulation of the Pit Limit Problem (Chapter 5), integer zero-one constraints were not required, since the constraint matrix was unimodular. The unimodularity of the constraint matrix quaranteed that the decision variables

would assume integral values.

Unfortunately, the constraint matrix for the extraction sequence problem is *not* unimodular. A necessary condition for a matrix to be unimodular is that every submatrix must have a determinant of 0, 1 or -1. To demonstrate that the extraction sequence problem's constraint matrix is not unimodular, consider the following submatrix. Select any row in the accessibility constraints, $EX_1 + EX_2 + \cdots + EX_i \leq 0$. For example, the first row in the set of constraints $EX_1 \leq 0$. At least two columns in the row will have non-zero coefficients, one coefficient will be positive and one coefficient will be negative. Select these columns. From the constraint $1^T X_1 \leq 1$ select the corresponding columns, which must have positive one coefficients. Thus, we obtain the following submatrix:

$$\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}.$$

Since the determinant of this submatrix has a value of 2, the constraint matrix cannot be unimodular. Hence, we must add the following zero-one constraints $X_i, j = 0, 1\ i = 1, \ldots, n,\ j = 1, \ldots, n$ to the problem's formulation. Thus making the extraction sequence problem a zero-one integer programming problem.

The theory of NP-completeness, discussed in detail in Horowitz and Sahni 1978, establishes two classes of problems, NP-hard and NP-complete. Problems in these classes are believed to be computationally intractable, meaning they probably cannot be solved by a polynomial order algorithm. Although the theory does not prove these problems cannot be solved by any polynomial order algorithm, considerable research, over many years, has yet to result in the development of a polynomial order algorithm for any one of them.

Essentially, the theory of NP-completeness says that if a problem is NP-hard

than it is probably not possible to obtain a polynomial order algorithm which can solve the problem. Hence, the only algorithms which are guaranteed to find the optimal solution are those algorithms which enumerate extremely large numbers of solutions.

Through the process of *algorithmic reducibility*, Horowitz and Sahni 1978 prove that the zero-one integer programming problem is NP-hard. Since the discounted cash flow problem is a zero-one integer program, it also is NP-hard. A more interesting result, provided by the same authors, relates to approximating algorithms.

The authors introduce three categories of approximation algorithms, algorithms which obtain a feasible solution which is close to the optimal solution. The categories differ in how *closeness* is measured. The following definitions were obtained from Horowitz and Sahni 1978.

Let A be an algorithm which generates a feasible solution to every instance I of a problem P. Let $F^*(I)$ be the value of the optimal solution to I and $\hat{F}(I)$ be the value of the feasible solution generated by A.

A is an *absolute approximation algorithm* for problem P if and only if for every instance I of P, $|F^*(i) - \hat{F}(I)| \leq k$ for some constant k.

A is an $f(n)$-*approximation algorithm* if and only if for every instance I of size n, $|F^*(I) - \hat{F}(I)|/F^*(I) \leq f(n)$. It is assumed that $F^*(I) > 0$.

An ϵ-*approximate* algorithm is an $f(n)$-approximate algorithm for which $f(n) \leq \epsilon$ for some constant ϵ.

The authors prove that the ϵ-approximation zero-one integer programming problem is NP-hard, which indicates it is not possible to obtain an ϵ-approximate algorithm to the zero-one integer programming problem which has polynomial order. Thus, it is not possible to obtain an ϵ-approximate algorithm to the extraction sequence prob-

lem. Since it is not possible to obtain an ϵ-approximate algorithm to the extraction sequence problem, it is not possible to prove that the NLG Algorithm provides an ϵ-bounded solution.

Although theory indicates it is not possible to prove the quality of the approximation obtained using the NLG Algorithm, it is our opinion that the NLG Algorithm, in conjunction with one of the local ordering approaches, does provide a good approximation to the optimal solution of the extraction sequence problem.

Chapter 9
CONCLUSIONS

In this chapter we summarize the key results of the effort and their practical implications. As discussed in the introduction, our goals were: *i*) demystify and thoroughly explain the graph-theoretic pit limit algorithm developed by Lerchs and Grossmann, *ii*) define and prove, how, and if, the Nested Lerchs and Grossmann algorithm maximizes the cash flow function; and *iii*) develop an algorithm which maximizes the net present value of the extraction sequence.

In Chapter 2 we described, using a complete 2-D example, the details of each step of the LG Algorithm. We demonstrated that the LG Algorithm starts with an infeasible solution consisting of all positive valued blocks in the pit. The algorithm then adds arcs, representing dependencies between blocks in the current pit and dependent blocks which are not part of the current pit, until the current pit is a feasible pit. We also demonstrated that the first current pit which is feasible is the optimal pit.

The LG Algorithm's process of starting with an infeasible pit and working to feasibility implied some relationship with respect to the dual simplex algorithm, which also starts from an infeasible solution and iterates until a feasible solution is found. Thus, in Chapter 5, we investigated the dual formulation of the pit limit problem. Our analysis demonstrated that the LG Algorithm is a graph theoretic implementation of the dual simplex algorithm with a minor deviation in the rule for selecting the leaving basic variable. Changing the rule for selecting the leaving basic variable (the arc removed from the normalized tree) would eliminate the need for the LG Algorihm's

NormalizeTree procedure.

In Chapter 3 we reviewed the proofs of convergence for the LG Algorithm presented by Lerchs and Grossmann. In addition to presenting Lerchs and Grossmann's proofs, we provided considerable annotation and examples to clarify concepts which were inadequately described. Although we found the proofs presented by Lerchs and Grossmann to be correct, we believe the additional information demystifies the proofs and the steps of the algorithm.

In Chapter 4 we discussed the solution obtained by the LG Algorithm in the context of a pit containing multiple maximum valued closures. We proved that the LG Algorithm will always identify the smallest (in number of blocks) maximum valued closure as the optimal closure. We also proved that changing the strength classification of arcs will force the LG Algorithm to identify the largest maximum valued closure.

In Chapter 6 we provided a complete example of the NLG Algorithm and proved several important results about the nested closures generated. We proved: *i*) that the closure increment between nested closures has zero value in the offset graph; *ii*) the offset parameter equals the average value of the closure increment in the non-offset graph; and, *iii*) the average value of the nested closures generated by the NLG Algorithm decrease monotonically (from smallest closure to largest).

In Chapter 7 we investigated Lerchs and Grossmann's claim that the NLG Algorithm maximizes the cash flow function. We prove that, in a restricted sense, the nested closures generated by the NLG Algorithm maximizes the global cash flow function. The conclusion is restricted by the fact that the claim is correct with respect to the collection of nested closures generated by the NLG Algorithm. We were unable to prove their claim with respect to all possible sets of nested closures, although we

do believe that the nested closures generated by the NLG Algorithm do define the convex hull of all closures.

In Chapter 7 we also considered the gapping problem, the collection of nodes contained in the increment between consecutive closures. We sketched two approaches, based upon the ordering principal of the NLG Algorithm, for generating a local ordering of nodes contained within the increment.

In Chapter 8 we considered the pit sequencing problem in conjunction with discounted cash flows. We demonstrated, through as simple an example as possible (binary valued pit), that the sequencing defined by the NLG Algorithm can be outperformed when discounting is considered. Although the result is mathematically correct, there exists mitigating factors which seem to make the result unlikely to be realized in a realistic problem. The mitigating factors are the large rate of return which would need to be assumed and the fairly negligible amount of deviation in the results.

In Chapter 8 we also formulate the discounted extraction sequence problem. The formulation demonstrates that the problem is an integer programming problem. Since the problem is an integer programming problem it is classified as an NP-hard problem, which implies that it is not possible to develop an algorithm which can guarantee the optimal solution, for any problem instance, in polynomial order time. In addition, theoretical work performed by others imply that it is not possible to generate a polynomial order algorithm which can provide an ϵ-approximate solution to any problem instance.

Thus, with respect to our goals, we claim to have demystified the steps of the LG Algorithm and to have proven in what sense the nested closures generated by the NLG Algorithm maximize the cash flow function. Unfortunately, we cannot claim to

have developed an algorithm which maximizes the net present value of the extraction sequence. What we can claim is that it is very unlikely, if not impossible, to define a polynomial order algorithm which can be guaranteed to maximize the net present value of the extraction sequence for any problem instance.

9.1 Practical Implications and Future Directions

In practical terms, we claim that by thoroughly understanding the LG Algorithm and the NLG Algorith we can improve the performance of these algorithms. Our dual simplex analysis identified an alternative rule for identifying the leaving basic variable. By changing the rule we can eliminate the *NormalizeTree* procedure. Rather than always eliminating the artificial arc between the artificial root and the root of the branch containing the weak node, the most constraining arc should be removed. This alternative rule would agree with the steps of the dual simplex algorithm and guarantee that the resulting solution remains dual feasible. A processing performance comparison of the two rules should be performed to identify the better rule.

In addition, understanding that the LG Algorithm always identifies the smallest maximum valued closure, when multiple maximum valued closures exist, warrants changing the strength classification definitions to identify the largest maximum valued closure. This analysis may identify important alternatives in selecting the desired optimal pit limit.

Our analysis of the NLG Algorithm provides insight into understanding the offset parameter and how it can be used to guarantee all nested closures are identified. Comparison of the average non-offset value of the increment with the offset parameter indicates whether a nested closure has been overlooked. If the offset parameter's value

is less than the average value of the increment than a nested closure has been skipped.

Although our proof that the NLG Algorithm maximizes the global cash flow function required an important restriction, we believe that the nested closures identified by the algorithm define the convex hull of nested closures. Providing this proof would be an important conclusion in the mathematical analysis of these algorithms.

Although our formulation of the discounted extraction sequence problem implies it is not possible to develop a polynomial order algorithm which is guaranteed to identify the optimal solution is impractical, developing a suite of test cases which can be tested against by the various heuristic algorithms is warranted. Thus, at least empirical evidence can be gathered to demonstrate the relative merits of the various existing algorithms and future algorithms.

REFERENCES CITED

Coléou, Thierry. 1983. Technical parameterization of reserves for open pit design and mine planning. Proceedings of the 21st APCOM Symposium.

Dagdelen, Kadri, and Dominique François-Bongarçon. 1982. Towards the complete double parameterization of recovered reserves in open pit mining. Proceedings of the 17th APCOM Symposium.

Dagdelen, Kadri. 1985. Optimum multi-period open pit mine production scheduling. Ph.D. Thesis, Colorado School of Mines. Thesis number T-3073.

Dagdelen, Kadri, and Thys Johnson. 1986. Optimum open pit mine production scheduling by Lagrangian parameterization. Proceedings of the 19th APCOM Symposium.

François-Bongarçon, D., and A. Maréchal. 1976. A new method for open-pit design: parameterization for the final pit contour. Proceedings of the 14th APCOM symposium.

François-Bongarçon, Dominique, and D. Guibal. 1982. Algorithms for parameterizing reserves under different geometrical constraints. Proceedings of the 17th APCOM Symposium.

Fytas, Kostas, and Peter N. Calder. 1986. A computerized model of open pit short and long range production scheduling. Proceedings of the 19th APCOM Symposium.

Gershon, M. E. 1982. A linear programming approach to mine scheduling optimization. Proceedings of the 17th APCOM Symposium.

Gershon, M. E. 1983. Mine scheduling optimization with mixed integer programming. Mining Engineering.

Gershon, M. E. 1986a. A blending-based approach to mine planning and production scheduling. Proceedings of the 19th APCOM Symposium.

Gershon, M. E. 1986b. Developments in computerized mine production scheduling. Society of Mining Engineers Transactions 280.

Gershon, M. E. 1986c. An open-pit production scheduler: algorithm and implementation. Society of Mining Engineers Transactions 280.

Horowitz, Ellis, and Sartaj Sahni. 1978. Fundamentals of computer algorithms. Computer Science Press.

Ignizio, J. 1984. Linear programming in single- and multiple-objective systems. Addison-Wesley Publishing Company.

Kim, Y. C. 1978. Ultimate pit limit design methodologies using computer models-the state of the art. Mining Engineering.

Koenigsberg, Ernest. 1982. The optimum contours of an open pit mine: an application of dynamic programming. Proceedings of the 17th APCOM Symposium.

Lerchs, Helmut, and Ingo F. Grossmann. 1965. Optimum design of open-pit mines. The Canadian Mining and Metallurgical Bulletin.

Luenberger, David. 1982. Linear and non-linear programming. Prentice Hall, Inc.

Picard, Jean-Claude. 1976. Maximal closure of a graph and applications to combinatorial problems. Management Science 22.

Tachefine, Beyime, François Soumis, and Godelieve Vanderstraten. 1994. A decomposition flow algorithm for the operations planning problem in open pit mines. Proceedings of the 24th APCOM Symposium.

Tolwinski, Boleslaw, and Robert Underwood. 1992. An algorithm to estimate the optimal evolution of an open pit mine. Proceedings of the 23rd APCOM Symposium.

Wang, Qing, and Hasan Sevim. 1992. Enhanced production planning in open pit mining through intelligent dynamic search. Proceedings of the 23rd APCOM Symposium.

Yegulalp, Tuncel, and J. A. Arias. 1992. A fast algorithm to solve the ultimate pit limit problem. Proceedings of the 23rd APCOM Symposium.

Zhang, Y. G., and Q. X. Yun. 1986. A new approach for production scheduling in

open pit mines. Proceedings of the 19th APCOM Symposium.

Zhao, Y., and Y. C. Kim. 1990. A new graph theory algorithm for optimal ultimate pit design. Preprint of article for the 23rd APCOM Symposium.

Zhao, Y., and Y. C. Kim. 1994. Optimum mine production scheduling using Lagrangian parameterization approach. Proceedings of the 24th APCOM Symposium.

Appendix A

Obtaining LG Algorithm Source Code

As part of our research the LG Algorithm was implemented in C++ (approximately 5000 lines of code). For a copy of the source code please contact:

Gary Bond

(303)393-7546

1175 Fillmore Street

Denver, Colorado 80206.