OCTANARY BRANCHING ALGORITHM

by

JAMES PATRICK BAILEY

B.S., Kansas State University, 2012

A THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Industrial and Manufacturing Systems Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas
2012

Approved by:

Major Professor
Dr. Todd Easton

Abstract

Integer Programs (IP) are a class of discrete optimization that have been used commercially to improve various systems. IPs are often used to reach an optimal financial objective with constraints based upon resources, operations and other restrictions. While incredibly beneficial, IPs have been shown to be NP-complete with many IPs remaining unsolvable.

Traditionally, Branch and Bound (BB) has been used to solve IPs. BB is an iterative algorithm that enumerates all potential integer solutions for a given IP. BB can guarantee an optimal solution, if it exists, in finite time. However, BB can require an exponential number of nodes to be evaluated before terminating. As a result, the memory of a computer using BB can be exceeded or it can take an excessively long time to find the solution.

This thesis introduces a modified BB scheme called the Octanary Branching Algorithm (OBA). OBA introduces eight children in each iteration to more effectively partition the feasible region of the linear relaxation of the IP. OBA also introduces equality constraints in four of the children in order to reduce the dimension of the remaining nodes. OBA can guarantee an optimal solution, if it exists, in finite time. In addition, OBA has been shown to have some theoretical improvements over traditional BB. During computational tests, OBA was able to find the first, second and third integer solution with 64.8%, 27.9% and 29.3% fewer nodes evaluated, respectively, than CPLEX. These integers were 44.9%, 54.7% and 58.2% closer to the optimal solution, respectively, when compared to CPLEX. It is recommended that commercial solvers incorporate OBA in the initialization and random diving phases of BB.

Dedication

I dedicate my work to my parents, Patrick and Carol Bailey. Without their support throughout the years, I would be pursuing my second passion of being a professional hobo.

Acknowledgments

Most importantly, I would like to thank Dr. Todd Easton. Not only has he made this research possible, but he has a great deal of influence on me in the last few years and in setting me on the right path towards accomplishing my goals. I would also like to thank Dr. John Wu, Dr. David Yetter, and Dr. Craig Spencer. These individuals have not only agreed to participate on my review committee, they also have given me a great deal of opportunities in the last few years that have shaped me into the individual I am today.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Integer programming is a class of discrete optimization in which the decision variables are required to have integer values. Formally, an Integer Program (IP) is defined as max $z = c^T x$ subject to $Ax \leq b$, $x \in \mathbb{Z}^n_+$ where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{mxn}$, and $b \in \mathbb{R}^m$. The feasible region is defined as $P = \{x \in \mathbb{Z}^n_+ | Ax \leq b\}$, which is the set of all integer points that satisfy the constraints of the IP. An IP is essentially a linear program with the additional requirement that all decision variables must have integer values.

The primary method of solving IPs is branch and bound (BB). This thesis presents a theoretically superior version of BB. This new algorithm is called the Octanary Branching Algorithm (OBA) and introduces a new method of partioning the solution space. Half of the children generated by OBA reduces the dimension of the problem by at least one. For the children where the dimension is not reduced, it is shown that the upper bound for the objective value is less than or equal to the upper bound for the objective value for children that reduce the dimension of the IP.

Integer programming has been used to model and manage a wide array of systems. Delta Airlines was the first to use an IP model to schedule flights. At the time, Delta had over 2,500 domestic flights daily that used about 450 aircrafts[28]. The use of this model was expected to save Delta $300 million dollars over the next three years.

IP models have also been used in financial plannning[22 26 7], including risk management for hedge fund portolios[12]. Numerical experiments have shown that IPs can be used to improve the performance of a portolio rebalancing strategies. It is also beneficial to combine various risk constraints to control different types of risk.

Some methods for generating Compressed Sensing matrices require solving an IP[3]. Compressed sensing is a technique for finding sparce solutions to undetermined linear systems. Compressed Sensing has applications in the identification of rare alleles and their carriers[25], surface characterization and metrology[19], high resolution single pixel cameras[18], Magnetic Resonance Imaging (MRI)[16 17] and many other areas.

There have been many other applications of integer programmin in medicine. Recently, mixed integer programs have been used for radiation therapy[14]. Optimal solutions were found quickly and these solutions were superior in tumor coverge while minimizing radiation damage to nearby tissues. IPs have also been used to match organ donors to potential recipients[27]. As a result, the efficiency of the transplant sytem has increased significantly in the United States.

Integer Programs have been shown to be beneficial to large variety of problems; however the limiting factor in the application of IPs is the computational complexity of the problem. Integer Programming is NP-complete in the strong sense[10]. Many IP's require an exponential amount of time to find an optimal solution. As a result, some integer programs are unsolvable. The Mixed Integer Programming Library (MIPLIB)[1] provides a 67,732 variable, 656 constraint mixed integer program that is unsolved even after years of researchers' attempts to solve it.

When an IP is unsolvable, users are forced to relax limiting factors such as constraints or variable restrictions on integrality. Relaxing the problem to allow for solvability might not even produce a feasible solution. Other methods, such as partioning the IP into smaller subproblems, often lead to an inferior IP. Solutions to a weaker IP are significantly less valuable than a solution to the desired model.

The primary method to solve IPs is the Branch and Bound algorithm (BB)[2 13]. BB is an an enumerative approach that uses a tree structure of bounds and constraints to generate potential solutions. BB garauntees an optimal solution to a bounded IP, if it exists, in finite time. If no integer solution exists, then BB reports no solution. Theoretically, BB requires an exponential number of iterations to solve to completion.

BB relies on the linear relaxation (LR) of the IP. The LR is the continuous formulation of the IP where all constraints are identical except the variables are not required to be integer. The LR solution includes x values, x^{*LR}, and the objective value, z^{*LR}.

BB begins by solving the LR of the IP with the solution x^{*LR} and z^{*LR}. If $x^{*LR} \notin \mathbb{Z}^n_+$ then select i such that $x_i^{*LR} \notin \mathbb{Z}$. Two new subproblems or nodes are generated and are referred to children of the LR. Both children are given the same constraints as the LR. The first "less than" child is given the additional constraint of $x_i \leq \lfloor x_i^{*LR} \rfloor$. The second "greater than" child is given the additional constraint of $x_i \geq \lfloor x_i^{*LR} \rfloor + 1$. All subproblems created are solved in the same fashion until all pendant nodes are fathomed. A node is fathomed if the LR is infeasible, integer, or z^{*LR} is inferior to an existing integer solution. The branching process can be seen in Figure 1.1 below.

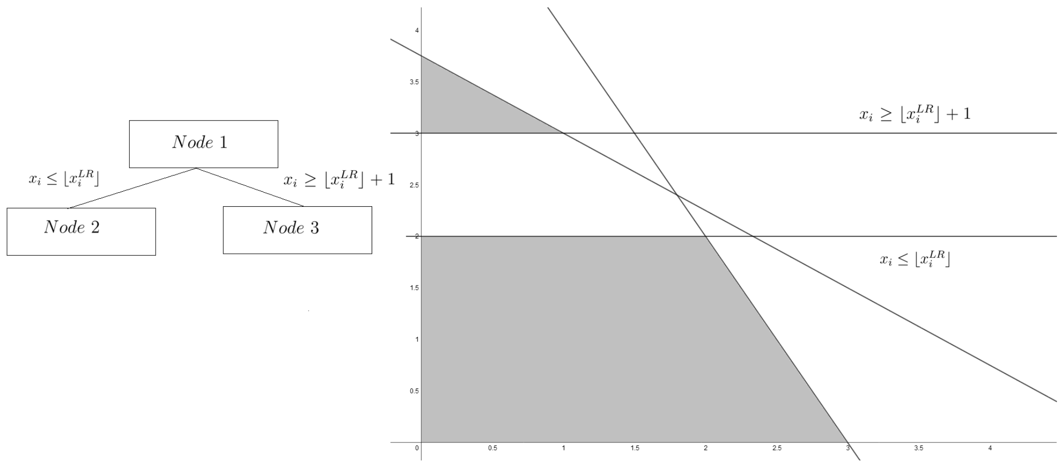

Node 1
$x_i \leq \lfloor x_i^{LR} \rfloor$
$x_i \geq \lfloor x_i^{LR} \rfloor + 1$
Node 2
Node 3
$x_i \geq \lfloor x_i^{LR} \rfloor + 1$
$x_i \leq \lfloor x_i^{LR} \rfloor$
4
3.5
3
2.5
2
1.5
1
0.5
0
0.5
1
1.5
2
2.5
3
3.5
4

Figure 1.1: *Partition for Branch and Bound*

Notice that a single branch eliminates all of the space in the LR where $\lfloor x_i^{LR} \rfloor < x_i < \lfloor x_i^{LR} \rfloor + 1$. BB may require an exponential number of nodes in order to find an optimal soluton. This is the primary disadvantage to using this method to solve IPs. Not only do problems take an exponential amount of time to solve, a tree can quickly grow large enough to exceed the memory capacity of a computer.

1.1 Motivation

The branch and bound algorithm has changed little since it was first used in 1960. Some improvements include various branching strategies that indicate which node is to be evaluated next and strategies that use variables with certain properties to branch on. However, of the existing strategies, there is no indication on which is best as the performance of a strategy varies greatly between problem classes and even instances in the same problem class.

The motivation for this research is derived from attempting find good integer solutions quickly. The goal for this research is to create a new branching algorithm that forces variables to take on integer values that are relatively close to the solution of the parent node's LR.

1.2 Research Contribution

This research's contribution is the development of a new branching algorithm called the Octanary Branching Algorithm (OBA). Unlike traditional BB, which uses only one variable to branch on, OBA uses two variables at each branch. In addition, OBA forces two variables to be constant in four of the eight children.

OBA grants a host of benefits over traditional BB. The four left most children of OBA have a dimension of at least one less than the parent node. OBA actually reduces the dimension by at least two whenever both variables used for branching are noninteger in the parent's solution. For the remaining four children, the upper bound for the objective value is greater than the upper bound of the objective value for the four left most children. These two things combined suggest that children with high objective values are easier to solve. By using a search strategy such as best bound, it is likely that many of the children that are more difficult to solve will be fathomed due to the existance of a superior integer solution.

In addition, OBA garauntees that at most n of the four left most children can be evaluated consecutively before fathoming a node. In the event that OBA never adds a redundant equality constraint, this bound can be reduced to $\lceil \frac{n}{2} \rceil$. During computational tests, OBA was able to find the first, second and third integer solution with 64.8%, 27.9% and 29.3% fewer nodes evaluated, respectively, than CPLEX. These integers were 44.9%, 54.7% and 58.2% closer to the optimal solution, respectively, when compared to CPLEX.

1.3 Thesis Outline

Chapter 2 provides the necessary background to understand the entirety of this thesis. In this section, BB is presented. Various search strategies, as well as their benefits. associated with BB are introduced. The methodology behind selecting particular variables to partition space is discussed.

Chapter 3 represents the majority of work for this research. The Octanary Branching Algorithm is explained in detail along with an example and the benefits associated with this algorithm. Proofs are given to show finite convergence, and that OBA correctly reduces the solution to find an optimal integer solution, if it exists. In addition, theoretical benefits are provided in this chapter.

Chapter 4 contains a computational study of OBA. OBA is compared to a commercially available IP software, CPLEX. This chapter presents the solution time for all methods. The interpretation of the data illustrates the benefits of OBA.

Chapter 5 contains a conclusion and summarizes the work. During work on this research, it was noted that the benefits of OBA could be used alongside other branching techniques, and some other future research topics are discussed.

Chapter 2

Background Information

This section is dedicated to providing the fundamental knowledge required to understand the entirety of this thesis. Only a small sample of the massive amounts of research that has been done in the field of Integer Programming is contained in this section. For further reading, see[21].

This section begins by giving a formal definition of an Integer Program. The Branch and Bound Algorithm is then presented as the primary tool for solving Integer Programs. Various search strategies and methods of selecting variables for branching are presented. An example of the Branch and Bound Algorithm is presented. Introduction to hyperplane branching is followed by an example of the Quaternary Hyperplane Branching Algorithm.

2.1 IP Definitions

An integer program (IP) is a discrete class of optimization problems consisting of an objective function subject to a set of constraints dependent on a set of integer decision variables. Formally, a bounded IP is defined as max $z = c^T x$ subject to $Ax \leq b$, $x \leq u$, $x \in \mathbb{Z}^n_+$ where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{mxn}$, $u \in \mathbb{Z}^n_+$, and $b \in \mathbb{R}^m$. The feasible region of an IP is defined as $P = \{x \in \mathbb{Z}^n_+ | Ax \leq b\}$. Thus, P describes the set of integer solutions that satisfy the constraints of the IP.

Solution techniques to solve IPs typically use the linear relaxation (LR) of the IP iteratively. The linear relaxation is defined as max $z = c^T x$ subject to $Ax \leq b$, $x \leq u$, $x \in \mathbb{R}^n_+$

where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{mxn}$, $u \in \mathbb{Z}_+^n$, and $b \in \mathbb{R}^m$. Let the feasible region of the LP be defined as $P^{LP} = \{x \in \mathbb{R}_+^n | Ax \leq b\}$.

Integer programs are NP-complete[23], which results in a great deal of effort to solve for the optimal solution even for some small instances. Linear programs, however, can be solved in polynomial time[11]. Commercial solvers are capable of solving most linear programs rapidly[8].

2.2 Branch and Bound

The Branch and Bound Algorithm (BB) and variants of it are most commonly used to solve IP's. BB garauntees to find an optimal solution, if it exists, in finite time. BB generates a tree structure of bounds and constraints to find an optimal solution.

BB begins by solving the linear relaxation of the IP and stores the solution as x^{*LR} with an objective value of z^{*LR}. These two values represent the root node, T_1 which is the linear relaxation of the IP, of the tree generated by BB. Denote the node being evaluated as the parent node T_p. If x^{*T_p} is noninteger, then find an x_i such that $x_i^{*T_p} \notin \mathbb{Z}_+$. Two new nodes for BB are generated and these nodes are referred to as children of the node T_p. Both children will begin the same feasible region as the parent node T_p. The first "less than" child, T_p^L, adds the additional constraint of $x_i \leq \lfloor x_i^{*T_p} \rfloor$. The second "greater than" child, T_p^G, includes the additional constraint of $x_i \geq \lfloor x_i^{*T_p} \rfloor + 1$.

BB continues until all pendant nodes are fathomed. A node is fathomed if its LR is infeasible, if the LR is integer, or if z^{*LR} is inferior to the best current integer solution. BB stores the best solution, x^* and z^*. If x^{*LR} is integer and $z^{*LR} > z^*$, then x^* and z^* are replaced with x^{*LR} and z^{*LR} respectively. If there are no more unfathomed pendant nodes, then BB terminates and reports an optimal solution if it exists. If there is no solution, then BB reports that the IP is infeasible. Formally,

The Branch and Bound Algorithm

Initialization

```
    Let $T = \{T_1\}$ be the enumeration tree and $T_1$ has the IP's linear relaxation.
    Set $z^* \leftarrow -\infty$ where $z^*$ is the current best integer solution.
    Set $totalnodes \leftarrow 1$.
Main Step
    While there exists an unfathomed pendant node.
        Let $T_p$ be any unfathomed pendant node of $T$.
        Solve the linear relaxation for $T_p$ and denote it as $z^{*T_p}$ and $x^{*T_p}$.
        If $T_p$ is infeasible, then mark $T_p$ as fathomed.
        If $x^{*T_p} \in \mathbb{Z}^n$, then mark $T_p$ as fathomed and if $z^{*T_p} > z^*$, then set $z^* \leftarrow z^{*T_p}$
                and set $x^* \leftarrow x^{*T_p}$.
        If $z^{*T_p} \leq z^*$, then mark $T_p$ as fathomed.
        If $T_p$ is unfathomed, then Begin.
            Select an $x_i$ such that $x_i^{*T_p} \notin \mathbb{Z}$.
            Set $\beta^{T_p} \leftarrow \lfloor x_i^{*T_p} \rfloor$.
            Create the following two new nodes with $T_p$ as the parent by adding the
                    following constraints to $T_p$.
            Set $T_p^L \leftarrow T_{totalnodes+1}$ with the constraint $x_i \leq \beta^{T_p}$ appended to $T_p$.
            Set $T_p^G \leftarrow T_{totalnodes+2}$ with the constraint $x_i \geq \beta^{T_p} + 1$ appended to $T_p$.
            Set $totalnodes \leftarrow totalnodes + 2$
        End if
    End while
Output
    If $z^* = -\infty$, then report the problem as infeasible; else report $z^*$ and $x^*$ as an
            optimal solution.
```

The branching step of BB is indeterminate. During any iteration, there are many nodes left that need to be evaluated. Selecting which node to evaluate next can have significant implications on the efficiency of BB. Memory capacities become a large issue when selecting a search strategy as the size of the tree can grow exponentially. However, it is possible that the solution time will greatly increase when attempting to use a strategy that decreases memory usage. Various search strategies used alongside BB are the subject of the next section.

2.2.1 Search Strategies

Early on, various search strategies were developed to decrease the amount of memory used when exploring the tree generated by BB. While memory issues have become less of a concern as computers have advanced, large branching structures still occur that require more resources than computers allow. As a result, strategies have been developed to avoid such difficulties. Generally, search strategies fall into the categories of depth first, breadth first, or best bound.

Depth first search is a method where BB evaluates a child node and keeps moving downward through the tree until all ancestors are fathomed. Once a fathomed node is found, then BB backtracks and repeats the process. Depth first strategies are typically given a direction to explore first, such as left or right.

Depth first search is a memory efficient method of exploring the evaluation tree. Let the depth of any node be denoted as d, where the root node has a depth of $d = 1$. The children of the root node have depth $d = 2$, and the children of a node with depth d' have a depth of $d = d' + 1$.

When evaluating a node at depth d, there are at most $d+1$ unevaluated nodes remaining in the tree. Because the number of unevaluated nodes is linear with the current depth of the node being evaluated, the amount of memory needed to store the tree is relatively small. In addition, depth first strategies tend to locate integer solutions quickly resulting in more nodes being fathomed from the tree.

There is no gaurantee, however, on the effectiveness of using a depth first strategy. It is possible to spend a great deal of time diving to find an integer solution that fathoms very few nodes in the tree. In addition, if an optimal solution does not appear on the left side of the tree but a depth first left strategy is being used, then the majority of the tree will have to be enumerated before finding an optimal solution. These issues can be magnified when dealing with large problems. Now that memory issues have become less severe, other search strategies have been devloped.

Breadth first search is a strategy that evaluates all nodes at depth d prior to evaluating any nodes at depth $d + 1$. Like depth first search, a direction is typically given to determine which side of the tree to start with. This allivietes the concern of whether or not the correct node was selected to explore that is associated with depth first search. Since time is not spent exploring the ancestors of a "useless" node, breadth first search can perform well.

The primary downfall of breadth first search is that it has no goal when exploring the tree. When evaluating a node at depth d, there are at most 2^d unevaluated nodes remaining in the tree. With an exponential relationship between the depth and number of nodes, memory issues rapidly become a limiting factor when attempting to solve IPs with breadth first search.

Many commercial solvers currently use some form of a best bound strategy[8]. Best bound uses the objective function as a means of determining which branch of the tree to explore first. The two nodes that have the parent with the highest z^{*LR} are evaluated first. The thought behind this strategy is that a better objective value is the best candidate for enumeration. The logic follows that an optimal integer solution would be the child of a node with a good objective value. Finding a good integer solution quickly can decrease the size of the current tree, prevent the need to evaluate many new nodess and leads to an improved solution time.

In an attempt to find a good integer solution quickly, a common practice is to use a hybrid of depth first and best bound search known as random diving[8 29]. In this strategy, BB uses the best child search for a set number of interations. At preset intervals, the algorithm switches to depth first search until the path is fathomed. This method makes an attempt to quickly discover a superior integer solution.

There is no indication as to which is the best search strategy as the performance varies greatly between problem classes and even instances in the same problem class. While one stategy may be effective for a certain class of problems, there is no guarantee that it will work well for all problems.

2.2.2 Branching Variable Selection

Prior to branching, BB must first select a variable to branch upon. BB only requires that if x_i is to be branched on, then $x_i^{*T_p} \notin \mathbb{Z}$. However, empirical evidence shows that the choice of x_i can be very important to the running time of the algorithm[21]. Often there exists a set of variables that when fixed at integer values, force all other variable to be integers. Because there is no robust method of determining such variables, often the priority of branching is user defined. Two common ways of selecting priorities are through degredation and penalties.

Degredation attempts to estimate the change in z^* caused by forcing x_i to be integral. Suppose $x_i^{*T_p}$ is noninteger. Define $f_i^{*T_p}$ such that $x_i^{*T_p} = \lfloor x_i^{*T_p} \rfloor + f_i^{*T_p}$. By branching on x_i, it could be expected that the objective function would decrease by $D_i^{-*T_p} = p_i^{-*T_p} f_i^{*T_p}$ for the left child and by $D_i^{+*T_p} = p_i^{+*T_p}(1 - f_i^{*T_p})$ for the right child. The coefficients $p_i^{-*T_p}$ and $p_i^{+*T_p}$ can be specified or estimated in several different ways.

Penalties involve more taxing calculations to determine the coefficients $p_i^{-*T_p}$ and $p_i^{+*T_p}$ in order to generate a lower bound on the change in z^{*T_p}. In early commercial solves penalties were used. However, emperical methods have shown that the cost of finding the penalties exceed the benefit of the information given[21].

Given $D_i^{-*T_p}$ and $D_i^{+*T_p}$, it is common to select x_i such that the minimum of $D_i^{-*T_p}$ and $D_i^{+*T_p}$ is maximized. The rationale is that maximizing the minimum decrease in the objective value will result in obtaining the optimal z^* value more quickly. Another common approach is to select x_i such that the maximum of $D_i^{-*T_p}$ and $D_i^{+*T_p}$ is maximized. The idea in doing this is that the branch with the lower objective value will quickly be fathomed by dominance.

2.2.3 A Branch and Bound Example

This section gives an example of how to solve an IP. For this example, no priority for branching on variables is given and depth first left search is used to evaluate the enumeration tree.

Example 2.2.1.

Consider the following problem:

$$\begin{aligned} \text{Maximize} \quad & 5x_1 + 4x_2 \\ \text{subject to} \quad & 2x_1 + 6x_2 \leq 15 \\ & 4x_1 + 3x_2 \leq 15 \\ & x_1, x_2 \geq 0,\ x_1, x_2 \in \mathbb{Z}. \end{aligned}$$

Figure 2.1 provides a graphical representation of the integer program with the constraints $2x_1 + 6x_2 \leq 15$, and $4x_1 + 3x_2 \leq 15$. The shaded region represents the feasible region of the linear relaxation of the integer program. Note that the the solution to the linear relaxation is noninteger and the only integer extreme point is (0,0).

Initially the root node, denoted as T_1, must be evaluated. The LR solution is $z^{*T_1} = \frac{115}{6}$ and $x^{*T_1} = (\frac{5}{2}, \frac{5}{3})$. For this example, depth first left search is used. In addition, no priority is given for which variable to branch on first.

In the root node, both decision variables are noninteger in the LR solution. Since no priority is given for variable selection, x_1 is selected. Constraints are created using the procedure described by BB. Since $x_1 = \frac{5}{2}$, the branches are $x_1 \leq \lfloor \frac{5}{2} \rfloor = 2$ and $x_1 \geq \lfloor \frac{5}{2} \rfloor + 1 = 3$. The branches lead to the less than child, $T_1^L = T_2$, and the greater than child, $T_1^G = T_3$.

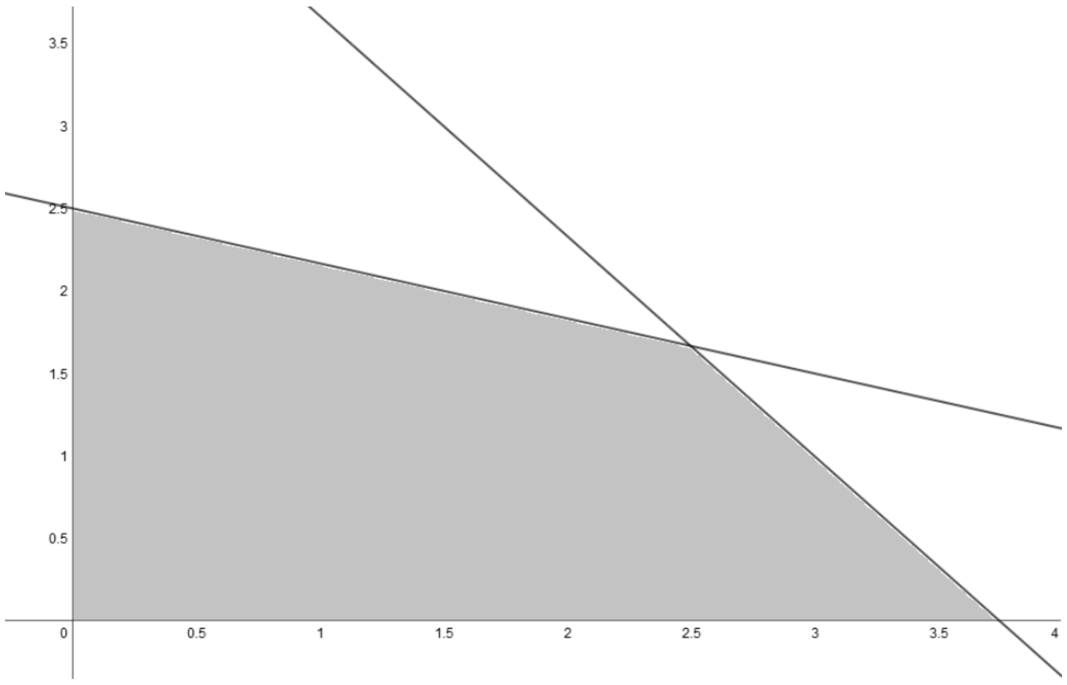


Figure 2.1: *Feasible Region for Example 2.2.1*

Figure 2.2 depicts the branching of the root node. For each node there are two branches, and each branch represents one constraint. These constraints are referred to as branching constraints. Finding the solution to the linear relaxation of a node and creating branching constraints is one iteration of BB. Figure 2.3 shows that the branching constraints divide the solution space into two subproblems. The shaded region represents the feasible region of the two subproblems. Note that the previous LR solution is eliminated in the both subproblems and that no feasible points are eliminated in the union of the two subproblems.

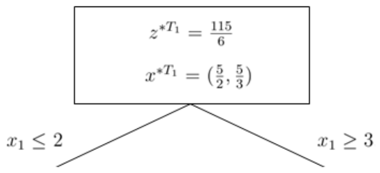

$z^{*T_1} = \frac{115}{6}$
$x^{*T_1} = (\frac{5}{2}, \frac{5}{3})$
$x_1 \leq 2$
$x_1 \geq 3$

Figure 2.2: *Branching on Node 1*

For the next iteration, the $x_1 \leq 2$ branch is followed due to depth first left search. The

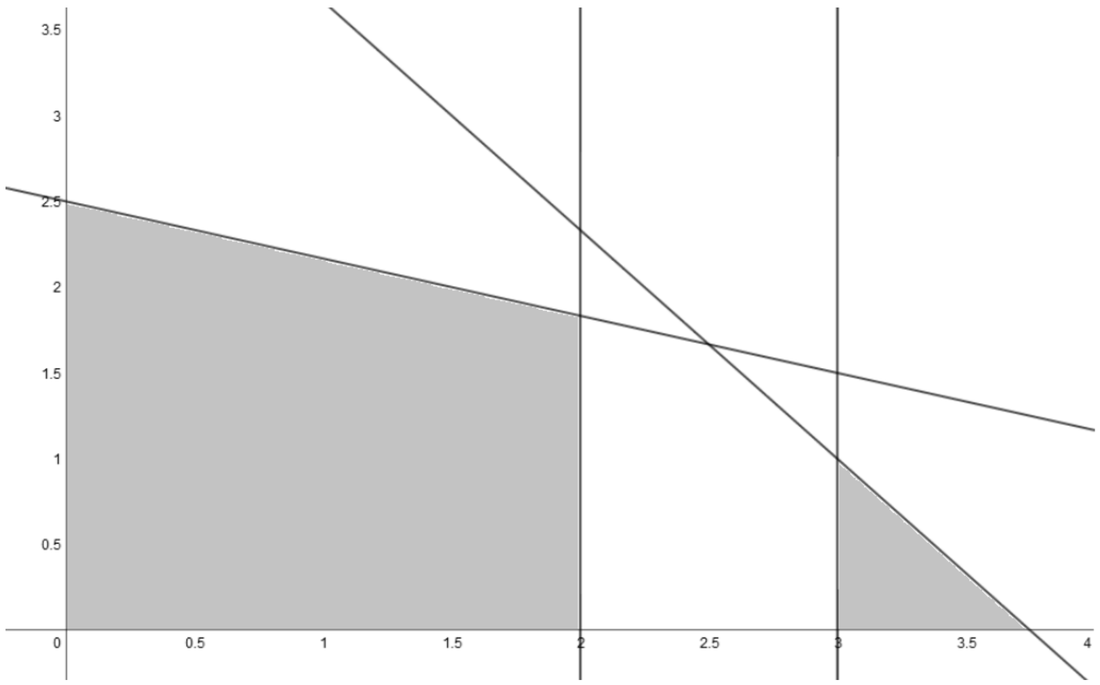


Figure 2.3: *Branching on $x_1 \leq 2$ and $x_1 \geq 3$ for Example 2.2.1*

first subproblem is denoted T_2 with the following constraints: $2x_1+6x_2 \leq 15$, $2x_1+6x_2 \leq 15$, and the branching constraint $x_1 \leq 2$. The linear relation of T_2 is evaulated yielding a solution of $z^{*T_2} = \frac{51}{3}$ and $x^{*T_2} = (2, \frac{11}{6})$. The solution is noninteger and requires another iteration of BB. Since x_2 is the only noninteger value, the branching constraints are $x_2 \leq \lfloor \frac{11}{6} \rfloor = 1$ and $x_2 \geq \lfloor \frac{11}{6} \rfloor + 1 = 2$. Two new nodes are generated and are labled $T_2^L = T_4$ and $T_2^G = T_5$.

The next node to be evaluated is T_4 which includes the constraints $2x_1 + 6x_2 \leq 15$, $2x_1 + 6x_2 \leq 15$, and the branching constraints $x_1 \leq 2$ and $x_2 \leq 1$. Note that these are the constraints for node T_2 with the additional branching constraint. The solution to the LR is given by $z^{*T_4} = 14$ and $x^{*T_4} = (2, 1)$ which is integer. The node T_4 is fathomed since its solution is integer. Since there is no current best integer solution, z^* is set to $z^{*T_4} = 14$ and x^* is set to $x^{*T_4} = (2, 1)$.

When a node is fathomed, it is no longer a candidate for branching. Once fathomed, the algorithm finds another unfathomed node and continues enumerartion according to the search strategy given.

BB moves to the next unfathomed node (T_5) and continues with a depth first left search. To evaluate T_5, BB uses all constraints from its parent, T_3, along with the branching constraint $x_2 \geq 2$. The LR of T_5 yields a solution of $z^{*T_5} = \frac{31}{2}$ and $x^{*T_5} = (\frac{3}{2}, 2)$. To create the new nodes $T_5^L = T_6$ and $T_5^G = T_7$, BB uses the branching constraints $x_1 \leq \lfloor \frac{3}{2} \rfloor = 1$ and $x_1 \geq \lfloor \frac{3}{2} \rfloor + 1 = 2$ respectively.

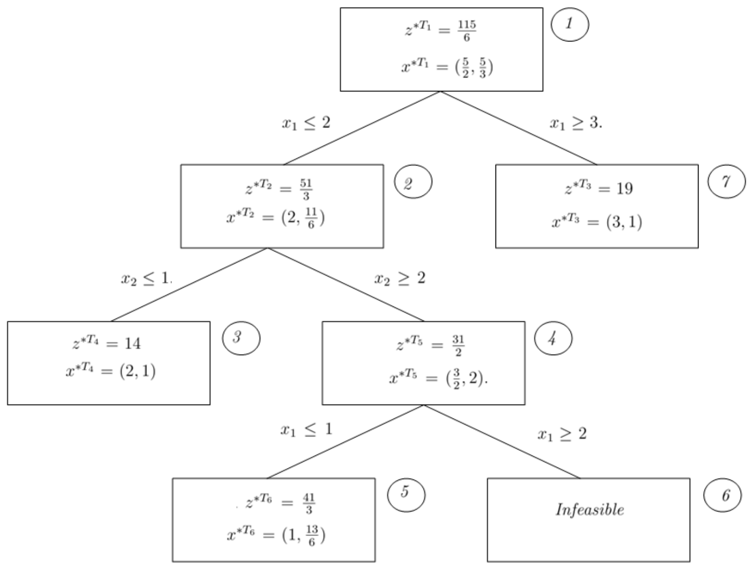
The next node to be evaluated is T_6 which includes the constraints from T_5 and the branching constraint $x_1 \leq 1$. Evaluating the LR of T_6 yields the solution $z^{*T_6} = \frac{41}{3}$ and $x^{*T_6} = (1, \frac{13}{6})$. Since $z^{*T_6} \leq z^*$, T_6 is fathomed as it is inferior to the best existing integer solution.

BB moves to the next unfathomed node T_7. The algorithm uses all constraints from T_5 along with the branching constraint $x_1 \geq 2$ to evaluate T_7. There does not exist a feasible solution to this linear relaxation. As a result, T_7 is fathomed.

Finally, BB evaulates the only remaining node in the tree, T_3. To evaluate this node, BB uses all of the constraints from T_1 along with the branching constraint $x_1 \geq 3$. Solving this linear relaxation yields a solution of $z^{*T_3} = 19$ and $x^{*T_3} = (3, 1)$. Since $z^{*T_3} \geq z^* = 14$, z^* is set to $z^{*T_3} = 19$ and x^* is set to $x^{*T_3} = (3, 1)$. The node T_3 is fathomed since x^{*T_3} is integer. As there are no remaining unfathomed nodes in the branching tree, BB terminates and reports the solution $z^* = 19$ and $x^* = (3, 1)$. The full enumeration tree can be seen in Figure 2.4.

2.2.4 Hyperplane Branching

Hyperplane branching is a modification on the branching constraints in BB. For BB, branching constraints are based on single variables. Hyperplane branches, however, attempt to increase the efficiency of the enumeration tree by branching on constraints based on multiple variables. The branching constraints would follow a similar pattern to BB. One child includes the constraint $\sum_{i=1}^{n} \alpha_i x_i \leq \beta$ and the other child has the constraint $\sum_{i=1}^{n} \alpha_i x_i \geq \beta + 1$ where $\beta \in \mathbb{Z}$.



$z^{*T_1} = \frac{115}{6}$
$x^{*T_1} = (\frac{5}{2}, \frac{5}{3})$
1
$x_1 \leq 2$
$x_1 \geq 3.$
$z^{*T_2} = \frac{51}{3}$
$x^{*T_2} = (2, \frac{11}{6})$
2
$z^{*T_3} = 19$
$x^{*T_3} = (3, 1)$
7
$x_2 \leq 1.$
$x_2 \geq 2$
$z^{*T_4} = 14$
$x^{*T_4} = (2, 1)$
3
$z^{*T_5} = \frac{31}{2}$
$x^{*T_5} = (\frac{3}{2}, 2).$
4
$x_1 \leq 1$
$x_1 \geq 2$
$z^{*T_6} = \frac{41}{3}$
$x^{*T_6} = (1, \frac{13}{6})$
5
Infeasible
6

Figure 2.4: *BB Enumeration Tree for Example 2.2.1*

While hyperplane branching has shown good results[9 20 24], it is limited by several factors. Unlike traditional BB which only stores the index of the variable branched on, hyperplane branching must store information on each variable in each node. In addition, by adding constraints that do not simply act as lower and upper bounds on a single variable, hyperplane branching causes the size of the basis to increase linearly with the depth of the node in the tree. This results in an increase in the time to solve each linear relaxtion and the amount of memory required to store the basis.

2.2.5 Quaternary Hyperplane Branching Algorithm

The Quaternary Hyperplane Branching Algorithm (QHBA)[15] is a modified version of BB and is the most related work to the algorithm presented in this thesis. QHBA utilizes hyperplane branching constraints and internal cutting planes to generate an efficient quaternary branching scheme. Implementation of this scheme theoretically improves QHBA in comparison to BB. It can also be shown that QHBA guarantees an optimal solution in finite steps for a bounded IP. A short computational study shows that using QHBA to evaluate the first 2000 nodes results in a 26.7% decrease when compared to CPLEX alone.

QHBA begins by solving the linear relaxation of the IP, denoted by T_1 and stores the solution as x^{*LR} with an objective value of z^{*LR}. These two values represent the root node, T_1, of the tree generated by QHBA. Denote the node being evaluated as the parent node T_p. If x^{*T_p} is noninteger, then select α^{1T_p} and $\alpha^{2T_p} \in \{-1, 0, 1\}^n$. For $k = 1$ and 2, if $\sum_{i=1}^{n} \alpha^{kT_p} x_i^{*T_p} \in \mathbb{Z}$, then select a $j \in \{1, ..., n\}$ such that $x_j^{*T_p} \notin \mathbb{Z}$, and if $\alpha_j^{kT_p} = 0$, then set $\alpha_j^{kT_p}$ to 1, else set $\alpha_j^{kT_p}$ to 0. Set β^{kT_p} to $\lfloor \sum_{i=1}^{n} \alpha^{kT_p} x_i^{*T_p} \rfloor$ for $k = 1$ and 2. Four new nodes for QHBA are generated and these nodes are referred to as children of the node T_p. All children begin with the same linear relaxation as the parent node T_p. The "less than less than," "less than greater than," "greater than less than," and "greater than greater than," denoted T_p^{LL}, T_p^{LG}, T_p^{GL} and T_p^{GG} repectively, have constraints as follows and can be seen in Figure 2.5:

T_p^{LL}) $\sum_{i=1}^{n} \alpha_i^{1T_p} x_i \leq \beta^{1T_p}$, $\sum_{i=1}^{n} \alpha_i^{2T_p} x_i \leq \beta^{2T_p}$, $\sum_{i=1}^{n} \lfloor \frac{\alpha_i^{1T_p} + \alpha_i^{2T_p}}{2} \rfloor x_i \leq \lfloor \frac{\beta^{1T_p} + \beta^{2T_p}}{2} \rfloor$.

T_p^{LG}) $\sum_{i=1}^{n} \alpha_i^{1T_p} x_i \leq \beta^{1T_p}$, $\sum_{i=1}^{n} \alpha_i^{2T_p} x_i \geq \beta^{2T_p} + 1$, $\sum_{i=1}^{n} \lfloor \frac{\alpha_i^{1T_p} - \alpha_i^{2T_p}}{2} \rfloor x_i \leq \lfloor \frac{\beta^{1T_p} - \beta^{2T_p} - 1}{2} \rfloor$.

T_p^{GL}) $\sum_{i=1}^{n} \alpha_i^{1T_p} x_i \geq \beta^{1T_p} + 1$, $\sum_{i=1}^{n} \alpha_i^{2T_p} x_i \leq \beta^{2T_p}$, $\sum_{i=1}^{n} \lfloor \frac{-\alpha_i^{1T_p} + \alpha_i^{2T_p}}{2} \rfloor x_i \leq \lfloor \frac{-\beta^{1T_p} + \beta^{2T_p} - 1}{2} \rfloor$.

T_p^{GG}) $\sum_{i=1}^{n} \alpha_i^{1T_p} x_i \geq \beta^{1T_p} + 1$,, $\sum_{i=1}^{n} \alpha_i^{2T_p} x_i \geq \beta^{2T_p} + 1$, $\sum_{i=1}^{n} \lfloor \frac{-\alpha_i^{1T_p} - \alpha_i^{2T_p}}{2} \rfloor x_i \leq \lfloor \frac{-\beta^{1T_p} - \beta^{2T_p} - 2}{2} \rfloor$.

QHBA continues until all pendant nodes are fathomed. A node is fathomed if its LR is infeasible, if the LR is integer, or if z^{*LR} is inferior to the best current integer solution. QHBA stores the best solution, x^* and z^*. If x^{*LR} is integer and $z^{*LR} > z^*$, then x^* and z^* are replaced with x^{*LR} and z^{*LR} respectively. If there are no more unfathomed pendant nodes, then QHBA terminates and reports an optimal solution if it exists. If there is no solution, then QHBA reports that the IP is infeasible. Formally,

Quaternary Hyperplane Branching Algorithm

Initialization

Let $T = \{T_1\}$ be the enumeration tree and T_1 has the IP's linear relaxation.

Set $z^* \leftarrow -\infty$ where z^* is the current best integer solution.

Set $totalnodes \leftarrow 1$.

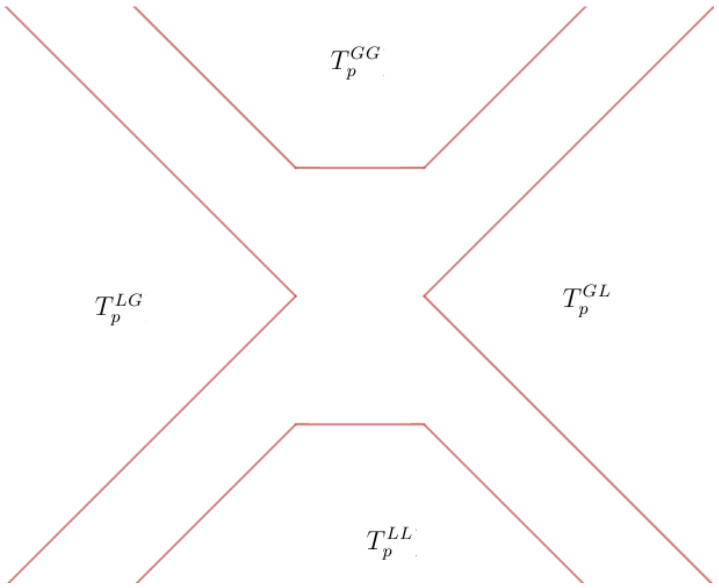


Figure 2.5: *QHBA Branching Structure*

Main Step

```
While there exists an unfathomed pendant node.
    Let $T_p$ be any unfathomed pendant node of $T$.
    Solve the linear relaxation for $T_p$ and denote it as $z^{*T_p}$ and $x^{*T_p}$.
    If $T_p$ is infeasible, then mark $T_p$ as fathomed.
    If $x^{*T_p} \in \mathbb{Z}^n$, then mark $T_p$ as fathomed and if $z^{*T_p} > z^*$, then set $z^* \leftarrow z^{*T_p}$
            and set $x^* \leftarrow x^{*T_p}$.
    If $z^{*T_p} \leq z^*$, then mark $T_p$ as fathomed.
    If $T_p$ is unfathomed, then Begin.
        Select an $\alpha^{1T_p}$ and $\alpha^{2T_p} \in \{-1, 0, 1\}^n$.
        For $k = 1$ and 2, Begin
            If $\sum_{i=1}^n \alpha^{kT_p} x_i^{*T_p} \in \mathbb{Z}$, then select a $j \in \{1, ..., n\}$ such that $x_j^{*T_p} \notin \mathbb{Z}$,
                    and if $\alpha_j^{kT_p} = 0$, then set $\alpha_j^{kT_p} \leftarrow 1$, else set $\alpha^{kT_p} \leftarrow 0$
            Set $\beta^{kT_p} \leftarrow \lfloor \sum_{i=1}^n \alpha_i^{kT_p} x_i^{*T_p} \rfloor$.
        Create the following four new nodes with $T_p$ as the parent by adding the
```

```
            following constraints to $T_p$.
        Set $T_p^{LL} \leftarrow T_{totalnodes+1}$ with the following constraints appended to $T_p$.
            $\sum_{i=1}^{n} \alpha_i^{1T_p} x_i \leq \beta^{1T_p}$.
            $\sum_{i=1}^{n} \alpha_i^{2T_p} x_i \leq \beta^{2T_p}$.
            $\sum_{i=1}^{n} \lfloor \frac{\alpha_i^{1T_p} + \alpha_i^{2T_p}}{2} \rfloor x_i \leq \lfloor \frac{\beta^{1T_p} + \beta^{2T_p}}{2} \rfloor$.
        Set $T_p^{LG} \leftarrow T_{totalnodes+2}$ with the following constraints appended to $T_p$.
            $\sum_{i=1}^{n} \alpha_i^{1T_p} x_i \leq \beta^{1T_p}$.
            $\sum_{i=1}^{n} \alpha_i^{2T_p} x_i \geq \beta^{2T_p} + 1$.
            $\sum_{i=1}^{n} \lfloor \frac{\alpha_i^{1T_p} - \alpha_i^{2T_p}}{2} \rfloor x_i \leq \lfloor \frac{\beta^{1T_p} - \beta^{2T_p} - 1}{2} \rfloor$.
        Set $T_p^{GL} \leftarrow T_{totalnodes+3}$ with the following constraints appended to $T_p$.
            $\sum_{i=1}^{n} \alpha_i^{1T_p} x_i \geq \beta^{1T_p} + 1$.
            $\sum_{i=1}^{n} \alpha_i^{2T_p} x_i \leq \beta^{2T_p}$.
            $\sum_{i=1}^{n} \lfloor \frac{-\alpha_i^{1T_p} + \alpha_i^{2T_p}}{2} \rfloor x_i \leq \lfloor \frac{-\beta^{1T_p} + \beta^{2T_p} - 1}{2} \rfloor$.
        Set $T_p^{GG} \leftarrow T_{totalnodes+4}$ with the following constraints appended to $T_p$.
            $\sum_{i=1}^{n} \alpha_i^{1T_p} x_i \geq \beta^{1T_p} + 1$.
            $\sum_{i=1}^{n} \alpha_i^{2T_p} x_i \geq \beta^{2T_p} + 1$.
            $\sum_{i=1}^{n} \lfloor \frac{-\alpha_i^{1T_p} - \alpha_i^{2T_p}}{2} \rfloor x_i \leq \lfloor \frac{-\beta^{1T_p} - \beta^{2T_p} - 2}{2} \rfloor$.
        Set $totalnodes \leftarrow totalnodes + 4$
    End if
End while
```

Output

```
If $z^* = -\infty$, then report the problem as infeasible; else report $z^*$ and $x^*$ as an
    optimal solution.
```

While the computational study of QHBA showed promising results, it was limited by the insufficient amount of memory required to evaluate the entire enumeration tree. By generating 2 or 3 hyperplane branching constraints, QHBA required up to $3(n+1)$ coefficients to be stored for each node generated. For each child generated, the size of the basis for a child would increase by up to 3. As a result, the memory of the computer was exceedd and QHBA was unable to solve to termination.

2.2.6 A Quaternary Hyperplane Branching Algorithm Example

Now that a formal definition of QHBA has been established, an example of how to solve an IP using QHBA is given. For this example, no priority for branching on variables is given and depth first left search is used to evaluate the enumeration tree.

Example 2.2.2.

Reconsider the problem from Example 2.2.1:

$$\begin{array}{ll} \text{Maximize} & 5x_1 + 4x_2 \\ \text{subject to} & 2x_1 + 6x_2 \leq 15 \\ & 4x_1 + 3x_2 \leq 15 \\ & x_1, x_2 \geq 0,\ x_1, x_2 \in \mathbb{Z}. \end{array}$$

Again depth first left search and no priorities for selecting branching variables are used. The first iteration of QHBA finds the solution to the root node, T_1, which is $z^{*T_1} = \frac{115}{6}$ and $x^{*T_1} = (\frac{5}{2}, \frac{5}{3})$. Let $\alpha^{1T_1} = \{1, 1\}$ and $\alpha^{2T_1} = \{1, -1\}$. Now β_1 and β_2 can be calculated with $\beta_1 = \lfloor \frac{\alpha_1^{1T_1} x_1^{*T_1} + \alpha_2^{1T_1} x_2^{*T_1}}{2} \rfloor = 4$ and $\beta_2 = \lfloor \frac{\alpha_1^{2T_1} x_1^{*T_1} + \alpha_2^{2T_1} x_2^{*T_1}}{2} \rfloor = 0$. Branching constraints are added to the root node to create 4 children as follows with the ordered determined by the depth first left search:

$T_1^{LL} = T_2$ is given the additional constraints $x_1 + x_2 \leq \beta_1 = 4$ and $x_1 - x_2 \leq \beta_2 = 0$.

$T_1^{LG} = T_3$ is given the additional constraints $x_1 + x_2 \leq \beta_1 = 4$, $x_1 - x_2 \geq \beta_2 + 1 = 1$ and $x_2 \leq 1$.

$T_1^{GL} = T_4$ is given the additional constraints $x_1 + x_2 \geq \beta_1 + 1 = 5$, $x_1 - x_2 \leq \beta_2 = 0$ and $x_2 \geq 3$.

$T_1^{GG} = T_5$ is given the additional constraints $x_1 + x_2 \geq \beta_1 + 1 = 5$ and $x_1 - x_2 \geq \beta_2 + 1 = 1$.

As shown in Figure 2.6, these 4 subproblems partition the solution space without eliminating any integer points. Since depth first left search is used, the first node to be evaluated is T_2. Evaluating this node yields a solution of $z^{*T_2} = \frac{135}{8}$ and $x^{*T_2} = (\frac{15}{8}, \frac{15}{8})$. Since x^{*T_2} is noninteger, four additional children must be created. Let $\alpha^{1T_2} = \{1, 1\}$ and $\alpha^{2T_2} = \{1, 0\}$. Now β_1 and β_2 can be calculated with $\beta_1 = \lfloor \frac{\alpha_1^{1T_2} x_1^{*T_2} + \alpha_2^{1T_2} x_2^{*T_2}}{2} \rfloor = 3$ and $\beta_2 = \lfloor \frac{\alpha_1^{2T_2} x_1^{*T_2} + \alpha_2^{2T_2} x_2^{*T_2}}{2} \rfloor = 1$. Branching constraints are added to the T_2 to create 4 children as follows:

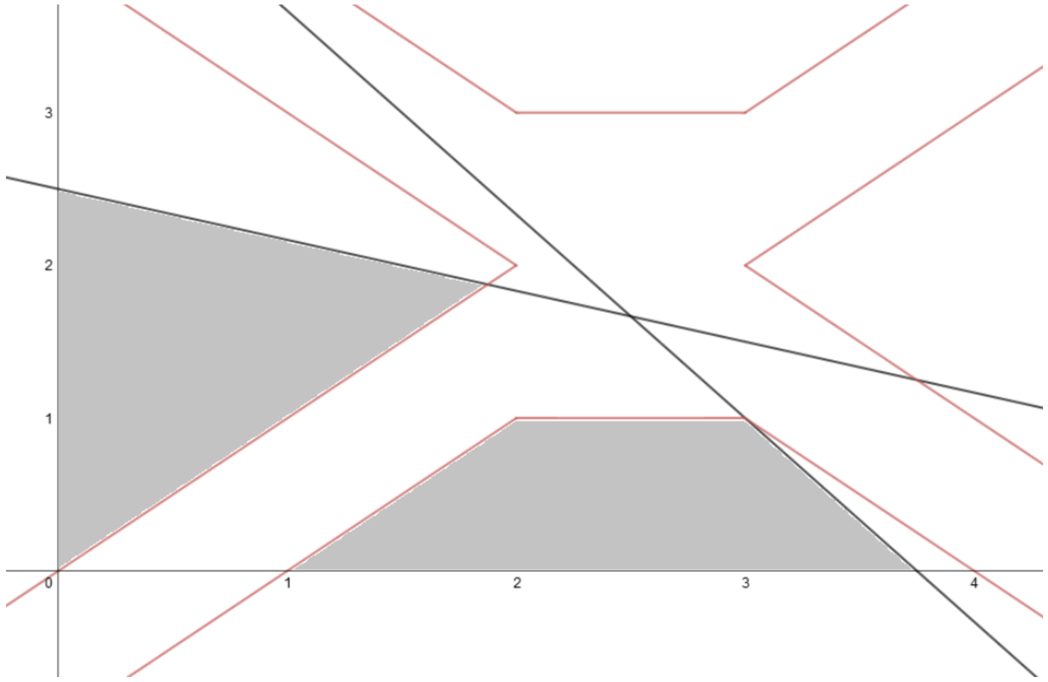


Figure 2.6: *QHBA Branching Structure for Example 2.2.1*

$T_1^{LL} = T_6$ is given the additional constraints $x_1 + x_2 \leq \beta_1 = 3$ and $x_1 \leq \beta_2 = 1$.

$T_1^{LG} = T_7$ is given the additional constraints $x_1 + x_2 \leq \beta_1 = 3$ and $x_1 \geq \beta_2 + 1 = 2$.

$T_1^{GL} = T_8$ is given the additional constraints $x_1 + x_2 \geq \beta_1 + 1 = 4$, $x_1 \leq \beta_2 = 1$ and $x_2 \geq 2$.

$T_1^{GG} = T_9$ is given the additional constraints $x_1 + x_2 \geq \beta_1 + 1 = 4$ and $x_1 \geq \beta_2 + 1 = 2$.

Evaluating T_6 yields the solution $z^{*T_6} = 13$ and $x^{*T_6} = (1, 2)$. Since the solution is integer, T_6 is fathomed and z^* is set to $z^{*T_6} = 13$ and x^* is set to x^{*T_6}. The nodes T_7, T_8

and T_9 are all found to be infeasible.

The next node to evaluate is T_3. This yields a solution of $z^{*T_3} = 19$ and $x^{*T_3} = (3,1)$. Since the solution is integer and $z^{*T_3} > z^*$, z^* is set to $z^{*T_3} = 19$ and x^* is set to x^{*T_3}. The nodes T_4 and T_5 are found to be infeasible and QHBA reports the optimal solution z^* and x^*. The full enumeration tree can be seen in Figure 2.7.

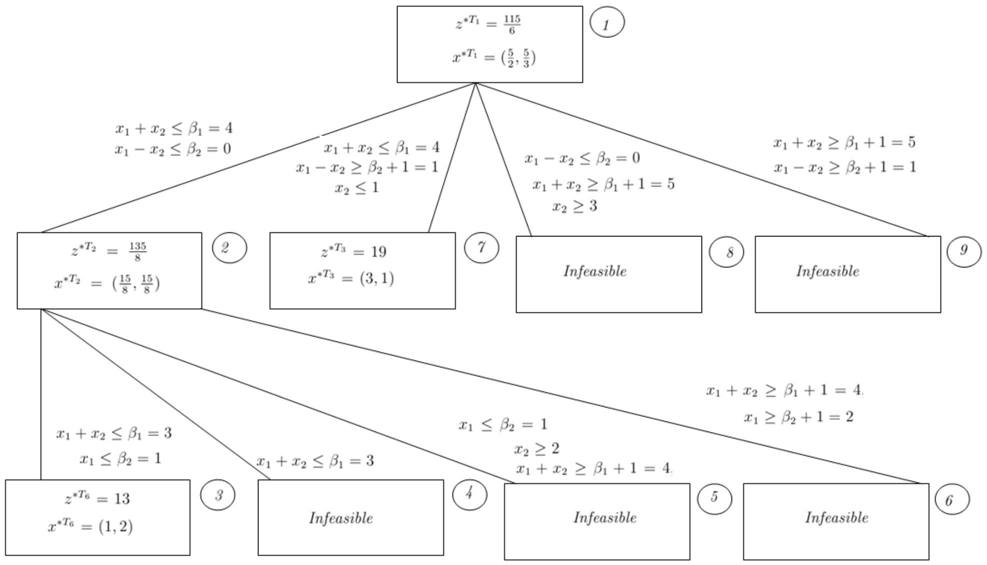

$z^{*T_1} = \frac{115}{6}$
$x^{*T_1} = (\frac{5}{2}, \frac{5}{3})$
1
$x_1 + x_2 \leq \beta_1 = 4$
$x_1 - x_2 \leq \beta_2 = 0$
$x_1 + x_2 \leq \beta_1 = 4$
$x_1 - x_2 \geq \beta_2 + 1 = 1$
$x_2 \leq 1$
$x_1 - x_2 \leq \beta_2 = 0$
$x_1 + x_2 \geq \beta_1 + 1 = 5$
$x_2 \geq 3$
$x_1 + x_2 \geq \beta_1 + 1 = 5$
$x_1 - x_2 \geq \beta_2 + 1 = 1$
$z^{*T_2} = \frac{135}{8}$
$x^{*T_2} = (\frac{15}{8}, \frac{15}{8})$
2
$z^{*T_3} = 19$
$x^{*T_3} = (3,1)$
7
Infeasible
8
Infeasible
9
$x_1 + x_2 \geq \beta_1 + 1 = 4$
$x_1 \geq \beta_2 + 1 = 2$
$x_1 + x_2 \leq \beta_1 = 3$
$x_1 \leq \beta_2 = 1$
$x_1 \leq \beta_2 = 1$
$x_2 \geq 2$
$x_1 + x_2 \geq \beta_1 + 1 = 4$
$x_1 + x_2 \leq \beta_1 = 3$
$z^{*T_6} = 13$
$x^{*T_6} = (1,2)$
3
Infeasible
4
Infeasible
5
Infeasible
6

Figure 2.7: *QHBA Enumeration Tree for Example 2.2.1*

Chapter 3

The Octanary Branching Algorithm (OBA)

This chapter describes the Octanary Branching Algorithm (OBA) and gives an example to demonstrate its implementation. In addition, all theory necessary to show that OBA correctly solves a bounded IP in finite time is provided. Theoretical benefits are also discussed in this chapter.

OBA, like BB, begins by solving the linear relaxation of the IP, represented by T_1, and stores the solution as x^{*LR} with an objective value of z^{*LR}. These two values represent the root node, T_1 of the tree generated by BB. The primary difference between BB and OBA is the branching step of the algorithm. Denote the node being evaluated as the parent node T_p. If x^{*T_p} is noninteger, then find an x_i and x_j such that $x_i^{*T_p}, x_j^{*T_p} \notin \mathbb{Z}_+$. If no x_j exists such that $i \neq j$, then x_j can be chosen arbitrarily. OBA allows for $i = j$ but operates more effictively when i and j are unique. Eight new nodes for OBA are generated and these nodes are referred to as children of the node T_p. The children begin with the same feasible region as the parent node T_p. The eight children shall be denoted as $T_p^{ab_k}$ where a or b specify less than or greater than constraints for x_i and x_j respectively and k specifies if the child has equality or inequality constraints. The eight children are defined as follows and are depicted graphically in Figure 3.1:

$T_p^{LL_e} = T_p \cap \{x \in \mathbb{R}_+^n | x_i = \lfloor x_i^{*T_p} \rfloor, x_j = \lfloor x_j^{*T_p} \rfloor\}$.

$$T_p^{GL_e} = T_p \cap \{x \in \mathbb{R}_+^n | x_i = \lfloor x_i^{*T_p} \rfloor + 1, x_j = \lfloor x_j^{*T_p} \rfloor\}$$
$$T_p^{LG_e} = T_p \cap \{x \in \mathbb{R}_+^n | x_i = \lfloor x_i^{*T_p} \rfloor, x_j = \lfloor x_j^{*T_p} \rfloor + 1\}$$
$$T_p^{GG_e} = T_p \cap \{x \in \mathbb{R}_+^n | x_i = \lfloor x_i^{*T_p} \rfloor + 1, x_j = \lfloor x_j^{*T_p} \rfloor + 1\}$$
$$T_p^{LL_i} = T_p \cap \{x \in \mathbb{R}_+^n | x_i \leq \lfloor x_i^{*T_p} \rfloor, x_j \leq \lfloor x_j^{*T_p} \rfloor, x_i + x_j \leq \lfloor x_i^{*T_p} \rfloor + \lfloor x_j^{*T_p} \rfloor - 1\}.$$
$$T_p^{GL_i} = T_p \cap \{x \in \mathbb{R}_+^n | x_i \geq \lfloor x_i^{*T_p} \rfloor + 1, x_j \leq \lfloor x_j^{*T_p} \rfloor, -x_i + x_j \leq -\lfloor x_i^{*T_p} \rfloor + \lfloor x_j^{*T_p} \rfloor - 2\}.$$
$$T_p^{LG_i} = T_p \cap \{x \in \mathbb{R}_+^n | x_i \leq \lfloor x_i^{*T_p} \rfloor, x_j \geq \lfloor x_j^{*T_p} \rfloor + 1, x_i - x_j \leq \lfloor x_i^{*T_p} \rfloor - \lfloor x_j^{*T_p} \rfloor - 2\}.$$
$$T_p^{GG_i} = T_p \cap \{x \in \mathbb{R}_+^n | x_i \geq \lfloor x_i^{*T_p} \rfloor + 1, x_j \geq \lfloor x_j^{*T_p} \rfloor + 1, -x_i - x_j \leq -\lfloor x_i^{*T_p} \rfloor - \lfloor x_j^{*T_p} \rfloor - 3\}.$$

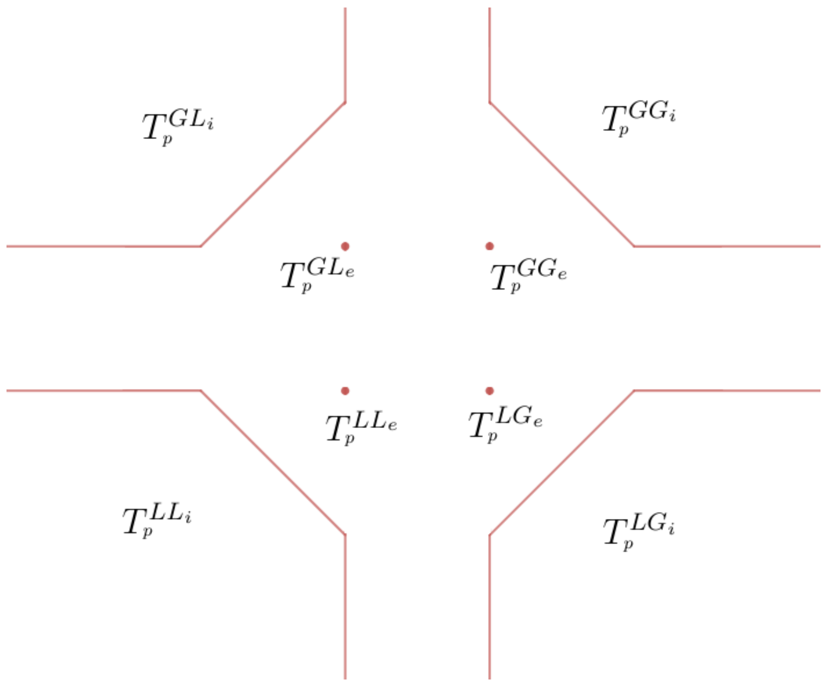


Figure 3.1: *OBA Branching Structure*

Observe that the four children closest to the parent's optimal solution have two variables set to integer values. The outer four children, which only create bounds for the variables, are pushed further from the parent's optimal solution and are likely to have poor objective values.

3.1 OBA

OBA is similar to BB in that it iteratively generates a set of new constraints to eliminate noninteger solutions. OBA garauntees to find an optimal solution, if it exists, in finite time for a bounded IP. OBA generates a tree structure of bounds and constraints to find an optimal solution.

OBA continues until all pendant nodes are fathomed. A node is fathomed if its LR is infeasible, if the LR is integer, or if z^{*LR} is inferior to the best current integer solution. OBA stores the best solution, x^* and z^*. If x^{*LR} is integer and $z^{*LR} > z^*$, then x^* and z^* are replaced with x^{*LR} and z^{*LR} respectively. If there are no more unfathomed pendant nodes, then OBA terminates and reports an optimal solution if it exists. If there is no solution, then OBA reports that the IP is infeasible. Formally,

```
The Octanary Branching Algorithm
 Initialization
     Let $T = \{T_1\}$ be the enumeration tree and $T_1$ has the IP's linear relaxation.
     Set $z^* \leftarrow -\infty$ where $z^*$ is the current best integer solution.
     Set $totalnodes \leftarrow 1$.
 Main Step
     While there exists an unfathomed node pendant node.
         Let $T_p$ be any unfathomed pendant node of $T$.
         Solve the linear relaxation for $T_p$ and denote it as $z^{*T_p}$ and $x^{*T_p}$.
         If $T_p$ is infeasible, then mark $T_p$ as fathomed.
         If $x^{*T_p} \in \mathbb{Z}^n$, then mark $T_p$ as fathomed and if $z^{*T_p} > z^*$, then set $z^* \leftarrow z^{*T_p}$
                 and set $x^* \leftarrow x^{*T_p}$.
         If $z^{*T_p} \leq z^*$, then mark $T_p$ as fathomed.
         If $T_p$ is unfathomed, then Begin.
             Select an $x_i$ and $x_j$ such that $x_i^{*T_p}, x_j^{*T_p} \notin \mathbb{Z}$. If there does not exist such
                 an $x_j$ where $i \neq j$, then $x_j$ can be chosen arbitrarily.
```

```
Set $\beta_1^{T_p} \leftarrow \lfloor x_i^{*T_p} \rfloor$ and set $\beta_2^{T_p} \leftarrow \lfloor x_j^{*T_p} \rfloor$.
Create the following eight new nodes with $T_p$ as the parent by adding the
    following constraints to $T_p$.
Set $T_p^{LL_e} \leftarrow T_{totalnodes+1}$ with the following constraints appended to $T_p$.
    $x_i = \beta_1^{T_p}$.
    $x_j = \beta_2^{T_p}$.
Set $T_p^{GL_e} \leftarrow T_{totalnodes+2}$ with the following constraints appended to $T_p$.
    $x_i = \beta_1^{T_p} + 1$.
    $x_j = \beta_2^{T_p}$
Set $T_p^{LG_e} \leftarrow T_{totalnodes+3}$ with the following constraints appended to $T_p$.
    $x_i = \beta_1^{T_p}$.
    $x_j = \beta_2^{T_p} + 1$
Set $T_p^{GG_e} \leftarrow T_{totalnodes+4}$ with the following constraints appended to $T_p$.
    $x_i = \beta_1^{T_p} + 1$.
    $x_j = \beta_2^{T_p} + 1$
Set $T_p^{LL_i} \leftarrow T_{totalnodes+5}$ with the following constraints appended to $T_p$.
    $x_i \leq \beta_1^{T_p}$.
    $x_j \leq \beta_2^{T_p}$.
    $x_i + x_j \leq \beta_1^{T_p} + \beta_2^{T_p} - 1$
Set $T_p^{GL_i} \leftarrow T_{totalnodes+6}$ with the following constraints appended to $T_p$.
    $x_i \geq \beta_1^{T_p} + 1$.
    $x_j \leq \beta_2^{T_p}$
    $-x_i + x_j \leq -\beta_1^{T_p} + \beta_2^{T_p} - 2$
Set $T_p^{LG_i} \leftarrow T_{totalnodes+7}$ with the following constraints appended to $T_p$.
    $x_i \leq \beta_1^{T_p}$.
    $x_j \geq \beta_2^{T_p} + 1$
    $x_i - x_j \leq \beta_1^{T_p} - \beta_2^{T_p} - 2$
```

```
            Set $T_p^{GG_i} \leftarrow T_{totalnodes+8}$ with the following constraints appended to $T_p$.
                $x_i \geq \beta_1^{T_p} + 1.$
                $x_j \geq \beta_2^{T_p} + 1$
                $-x_i - x_j \leq -\beta_1^{T_p} - \beta_2^{T_p} - 3$
            Set $totalnodes \leftarrow totalnodes + 8$
        End if
    End while
Output
    If $z^* = -\infty$, then report the problem as infeasible; else report $z^*$ and $x^*$ as an
        optimal solution.
```

3.1.1 An Octanary Branching Example

Now that a formal definition of OBA has been established, an example of how to solve an IP using OBA is given. For this example, no priority for branching on variables is given and depth first left search is used to evaluate the enumeration tree.

Example 3.1.1.

Reconsider the problem from Example 2.2.1:

$$\begin{array}{ll} \text{Maximize} & 5x_1 + 4x_2 \\ \text{subject to} & 2x_1 + 6x_2 \leq 15 \\ & 4x_1 + 3x_2 \leq 15 \\ & x_1, x_2 \geq 0,\ x_1, x_2 \in \mathbb{Z}. \end{array}$$

Again depth first left search and no priorities for selecting branching variables are used. The first iteration of OBA finds the solution to the root node, T_1, which is $z^{*T_1} = \frac{115}{6}$ and $x^{*T_1} = (\frac{5}{2}, \frac{5}{3})$. Both values are noninteger and shall be used in the branching step. Now β_1 and β_2 can be calculated with $\beta_1 = \lfloor \frac{5}{2} \rfloor = 2$ and $\beta_2 = \lfloor \frac{5}{3} \rfloor = 1$. Branching constraints are added to the root node to create 8 children as follows with the ordered determined by the depth first left search:

$T_1^{LL_e} = T_2$ is given the additional constraints $x_1 = \beta_1 = 2$ and $x_2 = \beta_2 = 1$.

$T_1^{GL_e} = T_3$ is given the additional constraints $x_1 = \beta_1 + 1 = 3$ and $x_2 = \beta_2 = 1$.

$T_1^{LG_e} = T_4$ is given the additional constraints $x_1 = \beta_1 = 2$ and $x_2 = \beta_2 + 1 = 2$.

$T_1^{GG_e} = T_5$ is given the additional constraints $x_1 = \beta_1 + 1 = 3$ and $x_2 = \beta_2 + 1 = 2$.

$T_1^{LL_i} = T_6$ is given the additional constraints $x_1 \leq \beta_1 = 2$, $x_2 \leq \beta_2 = 1$ and $x_1 + x_2 \leq \beta_1 + \beta_2 - 1 = 2$.

$T_1^{GL_i} = T_7$ is given the additional constraints $x_1 \geq \beta_1 + 1 = 3$, $x_2 \leq \beta_2 = 1$ and $-x_1 + x_2 \leq -\beta_1 + \beta_2 - 2 = -3$.

$T_1^{LG_i} = T_8$ is given the additional constraints $x_1 \leq \beta_1 = 2$, $x_2 \geq \beta_2 + 1 = 2$ and $x_1 - x_2 \leq \beta_1 - \beta_2 - 2 = -1$.

$T_1^{GG_i} = T_9$ is given the additional constraints $x_1 \geq \beta_1 + 1 = 3$, $x_2 \geq \beta_2 + 1 = 2$ and $-x_1 - x_2 \leq -\beta_1 - \beta_2 - 3 = -6$.

As shown in Figure 3.2, these 8 subproblems partition the solution space without eliminating any integer points. Since depth first left search is used, the first node to be evaluated is T_2. Evaluating this node yields a solution of $z^{*T_2} = 14$ and $x^{*T_2} = (2, 1)$. Since x^{*T_2} is integer, T_2 is fathomed and z^* is set to $z^{*T_2} = 14$ and x^* is set to x^{*T_2}. T_3 is the next node to be evaluated. Evaluating T_3 gives the solution $z^{*T_3} = 19$ and $x^{*T_3} = (3, 1)$. Since x^{*T_3} is integer, T_3 is fathomed and since $z^{*T_3} \geq z^*$, z^* is set to $z^{*T_3} = 19$ and x^* is set to x^{*T_3}.

Nodes T_4 and T_5 are to be evaluated next. However, T_4 and T_5 are fathomed as there does not exist a feasible solution for either node. The node T_6 is evaluated next yielding a solution of $z^{*T_6} = 10$ and $x^{*T_6} = (2, 0)$ and is fathomed as $z^{*T_6} \leq z^*$ and since x^{*T_6}. T_7 generates a solution with objective $z^{*T_7} = \frac{132}{7}$. Since $z^{*T_7} \leq z^*$, T_7 is fathomed. When evaluated, T_8 yields a solution with objective $z^{*T_8} = \frac{113}{8}$. Since $z^{*T_8} \leq z^*$, T_8 is fathomed. T_9 is the final node in the tree to be evaluated. This node is fathomed as there does not exist a feasible solution. As there are no more nodes to be evaluated, OBA terminates and

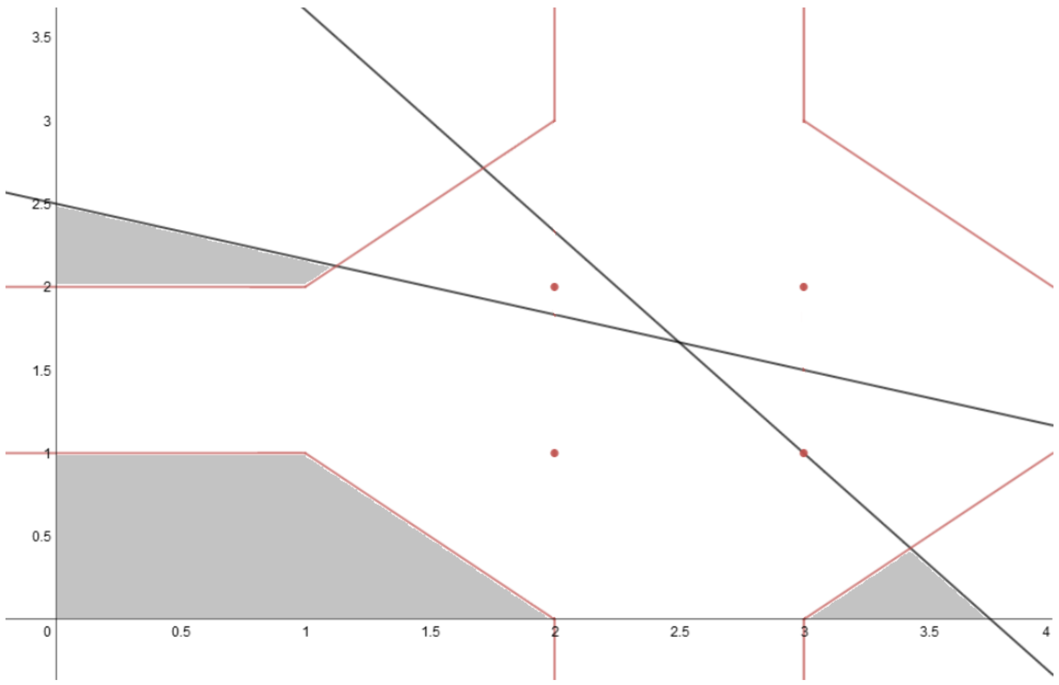


Figure 3.2: *OBA Branching Structure for Example 2.2.1*

reports an optimal solution of $z^* = 19$ and $x^* = (3, 1)$. The full enumeration tree can be seen in Figure 3.3.

3.2 Theory of OBA

The purpose of this section is to show that OBA solves integer programs to optimality in finite time. First a lemma is presented showing that the branching constraints of the children generated by OBA have integers as extreme points. A proof follows that demonstrates that no integer solutions are removed in the branching step of OBA. Finally, these lemmas are brought together to prove that OBA solves to optimality in finite time.

First it is shown that the regions described by the constraints all have integer extreme points when $i \neq j$. Define the 8 regions as follows:

$P^{LR}_{LL_e} = \{x \in \mathbb{R}^n_+ | x_i = \beta_1, x_j = \beta_2, x_k \leq u_k \ \forall \ k \in N\}$.

$P^{LR}_{GL_e} = \{x \in \mathbb{R}^n_+ | x_i = \beta_1 + 1, x_j = \beta_2, x_k \leq u_k \ \forall \ k \in N\}$.

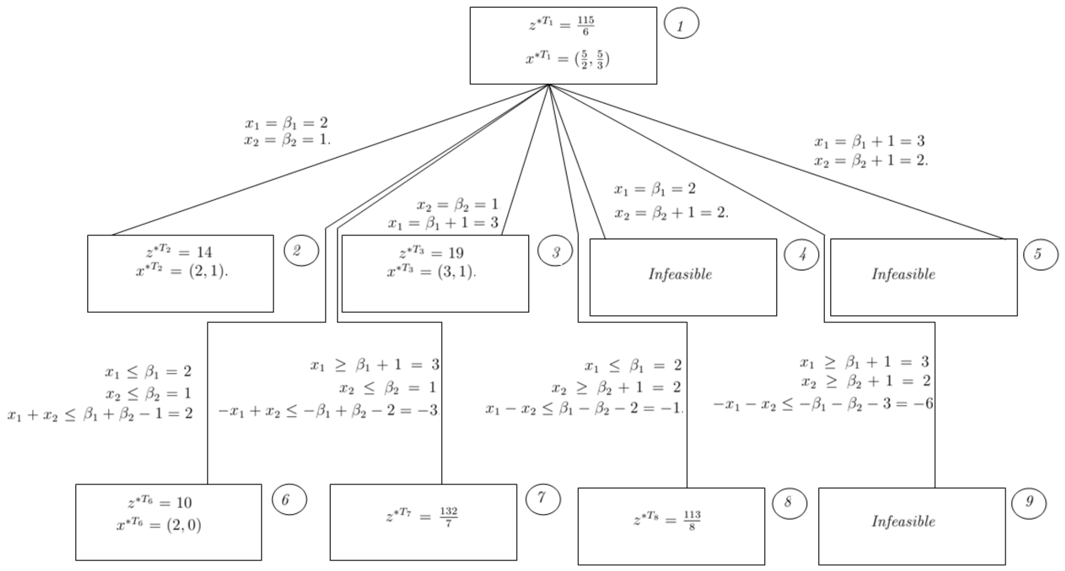


Figure 3.3: *OBA Enumeration Tree for Example 2.2.1 with the First Four Children in the First Row*

$P^{LR}_{LG_e} = \{x \in \mathbb{R}^n_+ | x_i = \beta_1, x_j = \beta_2 + 1, x_k \le u_k \ \forall \ k \in N\}.$

$P^{LR}_{GG_e} = \{x \in \mathbb{R}^n_+ | x_i = \beta_1 + 1, x_j = \beta_2 + 1, x_k \le u_k \ \forall \ k \in N\}.$

$P^{LR}_{LL_i} = \{x \in \mathbb{R}^n_+ | x_i \le \beta_1, x_j \le \beta_2, x_i + x_j \le \beta_1 + \beta_2 - 1, x_k \le u_k \ \forall \ k \in N\}.$

$P^{LR}_{GL_i} = \{x \in \mathbb{R}^n_+ | x_i \ge \beta_1 + 1, x_j \le \beta_2, -x_i + x_j \le -\beta_1 + \beta_2 - 2, x_k \le u_k \ \forall \ k \in N\}.$

$P^{LR}_{LG_i} = \{x \in \mathbb{R}^n_+ | x_i \le \beta_1, x_j \ge \beta_2 + 1, x_i - x_j \le \beta_1 - \beta_2 - 2, x_k \le u_k \ \forall \ k \in N\}.$

$P^{LR}_{GG_i} = \{x \in \mathbb{R}^n_+ | x_i \ge \beta_1 + 1, x_j \ge \beta_2 + 1, -x_i - x_j \le -\beta_1 - \beta_2 - 3, x_k \le u_k \ \forall \ k \in N\}.$

Let e_k denote the vector with all zeroes with a one in the k^{th} column. Let $\delta_k = 0$ if $x_k = 0$ and $\delta_k = 1$ otherwise.

Lemma 3.2.1. *The extreme points given by $P^{LR}_{LL_e}$, $P^{LR}_{LL_i}$, $P^{LR}_{GL_e}$, $P^{LR}_{GL_i}$, $P^{LR}_{LG_e}$, $P^{LR}_{LG_i}$, $P^{LR}_{GG_e}$, and $P^{LR}_{GG_i}$ are integer if $i \neq j$.*

Proof. A well known result is that the extreme points of a linear program are the basic feasible solutions. The nonbasic variables must be either at the lower or upper bound. The basic variables occur at the intersection of a subset of the constraints that define the feasible region. It is now straightforward to argue that the following points are the only extreme points in OBA's branching structure.

$P^{LR}_{LL_e}$) $x = \beta_1 e_i + \beta_2 e_j + \sum_{k\in N\setminus\{i,j\}} \delta_k u_k e_k$.

$P^{LR}_{GL_e}$) $x = (\beta_1 + 1)e_i + \beta_2 e_j + \sum_{k\in N\setminus\{i,j\}} \delta_k u_k e_k$.

$P^{LR}_{LG_e}$) $x = \beta_1 e_i + (\beta_2 + 1)e_j + \sum_{k\in N\setminus\{i,j\}} \delta_k u_k e_k$.

$P^{LR}_{GG_e}$) $x = (\beta_1 + 1)e_i + (\beta_2 + 1)e_j + \sum_{k\in N\setminus\{i,j\}} \delta_k u_k e_k$.

$P^{LR}_{LL_i}$) $x = (\beta_1 - 1)e_i + \beta_2 e_j + \sum_{k\in N\setminus\{i,j\}} \delta_k u_k e_k$ and also

$x = \beta_1 e_i + (\beta_2 - 1)e_j + \sum_{k\in N\setminus\{i,j\}} \delta_k u_k e_k$.

$P^{LR}_{GL_i}$) $x = (\beta_1 + 2)e_i + \beta_2 e_j + \sum_{k\in N\setminus\{i,j\}} \delta_k u_k e_k$ and also

$x = (\beta_1 + 1)e_i + (\beta_2 - 1)e_j + \sum_{k\in N\setminus\{i,j\}} \delta_k u_k e_k$.

$P^{LR}_{LG_i}$) $x = (\beta_1 - 1)e_i + (\beta_2 + 1)e_j + \sum_{k\in N\setminus\{i,j\}} \delta_k u_k e_k$ and also

$x = \beta_1 e_i + (\beta_2 + 2)e_j + \sum_{k\in N\setminus\{i,j\}} \delta_k u_k e_k$.

$P^{LR}_{GG_i}$) $x = (\beta_1 + 2)e_i + (\beta_2 + 1)e_j + \sum_{k\in N\setminus\{i,j\}} \delta_k u_k e_k$ and also

$x = (\beta_1 + 1)e_i + (\beta_2 + 2)e_j + \sum_{k\in N\setminus\{i,j\}} \delta_k u_k e_k$.

Trivially all of these points are integer.

□

Next it is shown that no integer points are removed from the feasible region during the branching step. This is done by first breaking the feasible region into four quadrants. Due to the symmetry of the quadrants, only one is examined. It is shown that all integer points in the quadrant are also contained in the union its two children.

Lemma 3.2.2. *Let T_p be any node in OBA's branching tree, then $\mathbb{Z}^n \cap T_p = \mathbb{Z}^n \cap (T_p^{LL_e} \cup T_p^{LL_i} \cup T_p^{LG_e} \cup T_p^{LG_i} \cup T_p^{GL_e} \cup T_p^{GL_i} \cup T_p^{GG_e} \cup T_p^{GG_i})$.*

Proof. Trivially, $\mathbb{Z}^n \cap (T_p^{LL_e} \cup T_p^{LL_i} \cup T_p^{LG_e} \cup T_p^{LG_i} \cup T_p^{GL_e} \cup T_p^{GL_i} \cup T_p^{GG_e} \cup T_p^{GG_i}) \subseteq \mathbb{Z}^n \cap T_p$, because each of the eight regions is a subset of T_p.

To show containment in the opposite direction, begin by examining four regions defined as T_p^{LL}, T_p^{GL}, T_p^{LG}, and T_p^{GG}.

$T_p^{LL} = T_p \cap \{x \in \mathbb{R}^n_+ | x_i \le \beta_1^{T_p}, x_j \le \beta_2^{T_p}\}$.

$T_p^{GL} = T_p \cap \{x \in \mathbb{R}^n_+ | x_i \ge \beta_1^{T_p} + 1, x_j \le \beta_2^{T_p}\}$.

$T_p^{LG} = T_p \cap \{x \in \mathbb{R}_+^n | x_i \leq \beta_1^{T_p}, x_j \geq \beta_2^{T_p} + 1\}$.

$T_p^{GG} = T_p \cap \{x \in \mathbb{R}_+^n | x_i \geq \beta_1^{T_p} + 1, x_j \geq \beta_2^{T_p} + 1\}$.

Trivially, every $x \in \mathbb{Z}^n$ satisfies either $x_i \leq \beta$ or $x_i \geq \beta + 1$. As a result, $\mathbb{Z}^n \cap T_p \subseteq \mathbb{Z}^n \cap (T_p^{LL} \cup T_p^{GL} \cup T_p^{LG} \cup T_p^{GG})$. Because each of the four regions is a subset of T_p, $\mathbb{Z}^n \cap (T_p^{LL} \cup T_p^{GL} \cup T_p^{LG} \cup T_p^{GG}) \subseteq \mathbb{Z}^n \cap T_p$. Thus, $\mathbb{Z}^n \cap T_p = \mathbb{Z}^n \cap (T_p^{LL} \cup T_p^{GL} \cup T_p^{LG} \cup T_p^{GG})$.

Without loss of generality only the region T_p^{LL} shall be considered since the additional constraints added to define the children are symmetric around $x_i = \beta_1^{T_p} + .5$ and $x_j = \beta_2^{T_p} + .5$.

For contradiction, assume there exists $x' \in (\mathbb{Z}^n \cap T_p^{LL})$ where $x' \notin (T_p^{LL_e} \cup T_p^{LL_i})$. By De Morgan's law, $x' \in (\neg T_p^{LL_e} \cap \neg T_p^{LL_i}) \cap (\mathbb{Z}^n \cap T_p^{LL})$. Since $T_p^{LL_i} = T_p^{LL} \cap \{x \in \mathbb{R}^n | x_i + x_b \leq \beta_1 + \beta_2 - 1\}$, it follows that $x' \in T_p^{LL} \cap \{x \in \mathbb{R}^n | x_i + x_b \geq \beta_1 + \beta_2 - 1\}$. But this region describes the space described by the points $\{x \in \mathbb{R}^n | x_i = \beta_1 \text{ and } x_j = \beta_2\}$, $\{x \in \mathbb{R}^n | x_i = \beta_1 - 1 \text{ and } x_j = \beta_2\}$, and $\{x \in \mathbb{R}^n | x_i = \beta_1 \text{ and } x_j = \beta_2 - 1\}$. Trivially, the only integers in the space are given by $\{x \in \mathbb{Z}^n | x_i = \beta_1 \text{ and } x_j = \beta_2\}$, $\{x \in \mathbb{Z}^n | x_i = \beta_1 - 1 \text{ and } x_j = \beta_2\}$, and $\{x \in \mathbb{Z}^n | x_i = \beta_1 \text{ and } x_j = \beta_2 - 1\}$. But this implies that $x' \in (T_p^{LL_e} \cup T_p^{LL_i})$ and thus a contradiction is formed therby confirming that $(\mathbb{Z}^n \cap T_p^{LL}) \subseteq (\mathbb{Z} \cap (T_p^{LL_e} \cup T_p^{LL_i}))$. As a result, this lemma is shown. $\square$

Lemma 3.2.2 shows that any integer solution that satisfies the parent node is contained in exactly one of the children. However, this does not show that OBA terminates. This lemma only ensures that OBA does not eliminate any integer points and that the solution is optimal, if OBA terminates.

Next it is shown that OBA terminates and correctly solves any bounded integer program. This is done through a proof by induction on the number of variables in the problem. First it is shown that OBA correctly solves a problem with one variable. In the inductive step, it is shown that OBA either terminates because an optimal integer solution is found, there does not exist a feasible solution or at least one variable is forced to equality thus reducing the number of decision variables in the problem by at least one.

Theorem 3.2.3. *OBA correctly solves any bounded integer program in a finite number of* *steps.*

Proof. For all variables x_i, let the lower and upper bound be denoted by lb_i and ub_i respectively. Note that initially $lb_i = 0$ and $ub_i = u_i \ \forall \ i \in [1, n] \cap \mathbb{Z}$.

Base Case: OBA correctly solves a bounded IP with one variable.

Consider the problem:

$$\begin{aligned} &\text{Maximize} && c_1 x_1 \\ &\text{subject to} && A x_1 \leq b \\ &&& x_1 \in \mathbb{Z}^+. \end{aligned}$$

Trivially, since there is only one variable, this problem can be reduced to the following:

$$\begin{aligned} &\text{Maximize} && c_1 x_1 \\ &\text{subject to} && b_1 \leq x_1 \leq b_2 \\ &&& x_1 \in \mathbb{Z}^+. \end{aligned}$$

Without loss of generality, assume that $c_1 > 0$. If c_1 is negative, then an upper bound substitution could be performed to generate the same problem with new b_1 and b_2. Also assume $c_1 = 1$ since maximizing x_1 and maximizing $c_1 x_1$ are effectively the same problem. It can also be assumed that $lb_1 \leq b_1$ and $ub_1 \geq b_2$ since the problem would otherwise be simplified.

If $b_2 < b_1$, then there trivially is no integer solution and OBA reports such as there is no feasible solution when the first LR is solved which terminates this case.

If $b_2 \geq b_1$ and $[b_1, b_2] \cap \mathbb{Z} = \emptyset$, then there is no integer solution. OBA solves the first LR and yields the solution $x_1 = b_2$. OBA then generates the following eight children:

$T^{LL_e} = T_p \cap \{x \in \mathbb{R} | x_1 = \lfloor b_2 \rfloor, x_1 = \lfloor b_2 \rfloor\}$ which yields no solution.

$T^{GL_e} = T_p \cap \{x \in \mathbb{R} | x_1 = \lfloor b_2 \rfloor + 1, x_1 = \lfloor b_2 \rfloor\}$ which yields no solution.

$T^{LG_e} = T_p \cap \{x \in \mathbb{R} | x_1 = \lfloor b_2 \rfloor, x_1 = \lfloor b_2 \rfloor + 1\}$ which yields no solution.

$T^{GG_e} = T_p \cap \{x \in \mathbb{R} | x_1 = \lfloor b_2 \rfloor + 1, x_1 = \lfloor b_2 \rfloor + 1\}$ which yields no solution.

$T^{LL_i} = T_p \cap \{x \in \mathbb{R} | x_1 \leq \lfloor b_2 \rfloor, x_1 \leq \lfloor b_2 \rfloor, 2x_1 \leq 2\lfloor b_2 \rfloor - 1\}$ which yields no solution.

$T^{GL_i} = T_p \cap \{x \in \mathbb{R} | x_1 \geq \lfloor b_2 \rfloor + 1, x_1 \leq \lfloor b_2 \rfloor, 0 \leq -2\}$ which yields no solution.

$T^{LG_i} = T_p \cap \{x \in \mathbb{R} | x_1 \leq \lfloor b_2 \rfloor, x_1 \geq \lfloor b_2 \rfloor + 1, 0 \leq -2\}$ which yields no solution.

$T^{GG_i} = T_p \cap \{x \in \mathbb{R} | x_1 \geq \lfloor b_2 \rfloor + 1, x_1 \geq \lfloor b_2 \rfloor + 1, -2x_1 \leq -2\lfloor b_2 \rfloor - 3\}$ which yields no solution.

All of these children are infeasible and OBA correctly reports that there does not exist a solution, which concludes this case.

If $b_2 \geq b_1$ and $[b_1, b_2] \cap \mathbb{Z} \neq \emptyset$. Trivially, the solution to this IP is $x_1 = \lfloor b_2 \rfloor$. If $b_2 \in \mathbb{Z}$, then OBA correctly reports the optimal solution in the first iteration. Otherwise OBA finds the noninteger solution $x_1 = b_2$ and generates the following eight children:

$T^{LL_e} = T_p \cap \{x \in \mathbb{R} | x_1 = \lfloor b_2 \rfloor, x_1 = \lfloor b_2 \rfloor\}$ which yields the optimal solution of $z^* = x_1 = \lfloor b_2 \rfloor$.

$T^{GL_e} = T_p \cap \{x \in \mathbb{R} | x_1 = \lfloor b_2 \rfloor + 1, x_1 = \lfloor b_2 \rfloor\}$ which yields no solution.

$T^{LG_e} = T_p \cap \{x \in \mathbb{R} | x_1 = \lfloor b_2 \rfloor, x_1 = \lfloor b_2 \rfloor + 1\}$ which yields no solution.

$T^{GG_e} = T_p \cap \{x \in \mathbb{R} | x_1 = \lfloor b_2 \rfloor + 1, x_1 = \lfloor b_2 \rfloor + 1\}$ which yields no solution.

$T^{LL_i} = T_p \cap \{x \in \mathbb{R} | x_1 \leq \lfloor b_2 \rfloor, x_1 \leq \lfloor b_2 \rfloor, 2x_1 \leq 2\lfloor b_2 \rfloor - 1\}$ which is fathomed by T^{LL_e}.

$T^{GL_i} = T_p \cap \{x \in \mathbb{R} | x_1 \geq \lfloor b_2 \rfloor + 1, x_1 \leq \lfloor b_2 \rfloor, 0 \leq -2\}$ which yields no solution.

$T^{LG_i} = T_p \cap \{x \in \mathbb{R} | x_1 \leq \lfloor b_2 \rfloor, x_1 \geq \lfloor b_2 \rfloor + 1, 0 \leq -2\}$ which yields no solution.

$T^{GG_i} = T_p \cap \{x \in \mathbb{R} | x_1 \geq \lfloor b_2 \rfloor + 1, x_1 \geq \lfloor b_2 \rfloor + 1, -2x_1 \leq -2\lfloor b_2 \rfloor - 3\}$ which yields no solution.

All of these children are evaluated and OBA reports the optimal solution with $z^* = x_1 = \lfloor b_2 \rfloor$ and this concludes this case. All possible case have been examined so OBA correctly solves a bounded one dimensional IP.

Inductive Step: By strong induction, assume OBA correctly solves bounded IPs with $i \in [1, n-1] \cap \mathbb{Z}$ variables. It is shown that OBA correctly solves bounded IPs with n variables.

For all variables, let $ub_i - lb_i = u_i$. For each non-fathomed node, there exists an $x_i^{*LR} \notin \mathbb{Z}$ and eight children are generated. For each child, x_i satisfies either $x_i \leq \lfloor x_i^{*LR} \rfloor < ub_i$ or $x_i \geq \lfloor x_i^{*LR} \rfloor + 1 > lb_i$ as shown in Lemma 3.2.1. In the first case, ub_i can be updated to $\lfloor x_i^{*LR} \rfloor$ while in the second case lb_i is updated to $\lfloor x_i^{*LR} \rfloor + 1$. In either case, when u_i is recalculated, it is reduced by at least one. As a result, x_i can be branched on by at most $u_i - 1$ before $lb_i = ub_i$. This implies that $lb_i = ub_i$ for some $i \in [1, n] \cap \mathbb{Z}$ when the enumeration tree reaches a depth of $\sum_{i=1}^{n} u_i - n + 1$.

Once $lb_i = ub_i$, x_i is constant and the number of variables is reduced by at least one. Thus, every pendant node in the enumeration tree is eventually fathomed at a depth of $\sum_{i=1}^{n} u_i - n + 1$ or the total number of variables is reduced by at least one. By the inductive hypothesis, all nodes where the number of variables is less than n terminate and solve correctly. As a result, OBA terminates when solving a bounded IP with n variables. For every pendant node in the fully expanded enumeration tree, there is an integer solution, the region is infeasible, or the LR is inferior to an existing integer solution. Since there are no feasible integer points removed as shown in Lemma 3.2.2, OBA correctly reports an optimal solution, if it exists.

□

Theorem 3.2.3 gives an inductive proof that that OBA correctly solves a bounded IP. Alternatively, it can be shown that OBA solves correctly by recognizing that all variables will be forced to integer at a depth of $\sum_{i=1}^{n} u_i$.

OBA correctly reports an optimal solution, if it exists, for any bounded IP. However, there must be a benefit gained by OBA in order for its implementation to be practical. The next section focuses on some theoretical benefits of OBA.

3.3 Theoretical Benefits of OBA

OBA generates a variety of benefits when compared to the traditional implementation of BB. For half of the children generated by OBA, the dimension of the LR is reduced by at least one. Whenever there are two noninteger values selected for branching, OBA reduces the dimension of the LR by two in four of the children generated. For the children where the dimension is not reduced, it is shown that the upper bound for the objective value is less than or equal to the upper bound for the objective bound for children that reduce the dimension of the IP.

Finally, it is shown that OBA can evaluate at most n of the four equality children during a single dive before fathoming a node. This bound can be reduced to $\lfloor \frac{n}{2} \rfloor$ when there are two noninteger values for every iteration (with one interation containing only one noninteger value if n is odd).

Theorem 3.3.1. *If OBA creates children, $T_p^{LL_e}$, $T_p^{GL_e}$, $T_p^{LG_e}$, or $T_p^{GG_e}$, then the dimension of these four children's spaces are at least one less than the dimension of the parent's feasible space.*

Proof. At an iteration of OBA, assume that node T_p is branched upon creating the four children $T_p^{LL_e}$, $T_p^{GL_e}$, $T_p^{LG_e}$, and $T_p^{GG_e}$. Since T_p is branched, there exists an $x_i^{*T_p} \in \mathbb{R}\backslash\mathbb{Z}$. Thus, $T_p^{LL_e}$, $T_p^{GL_e}$, $T_p^{LG_e}$, and $T_p^{GG_e}$ set x_i to either β_1 or $\beta_1 + 1$ during the branching step. The four children are similar, so here only $T_p^{LL_e}$ is considered.

Assume $T_p^{LL_e}$ has a feasible solution. Clearly, $x_i^{*T_p^{LL_e}} = \beta_1$. Thus, T_p's feasible space includes a vector between $x^{*T_p^{LL_e}}$ and x^{*T_p}. Clearly, $T_p^{LL_e}$ no longer has this vector as a feasible vector and so the dimension must have decreased by at least one.

Clearly if there is no feasible solution to the $T_p^{LL_e}$, then the dimension of $T_p^{LL_e}$ is -1. Since there exists a feasible solution to T_p, its dimension must be at least 0 and the result follows. $\square$

Theorem 3.3.1 shows that the four leftmost children of OBA reduce the dimension of the problem by at least one. The four leftmost children actually reduce the dimension of the problem by at least two whenever there are two branching noninteger variables. By reducing the dimension of the problem, the complexity of each child is reduced which often times leads in a reduction in the amount of time to solve each node. In addition, forcing a variable to be constant creates a simplified IP that would take fewer iterations of BB to evaluate than constraints that only create new upper and lower bounds.

Let $ub(z^{T_p^{ab_q}})$ denote the upper bound for the objective value $z^{*T_p^{ab_k}}$ for $a, b \in \{L, G\}$ and $k \in \{i, e\}$. Theorem 3.3.2 shows that $ub(z^{T_p^{ab_e}}) \geq ub(z^{T_p^{ab_i}})$ whenever branching on two distinct variables with noninteger values. This is accomplished by generating $ub(z^{T_p^{ab_e}})$ and an upper bound for $ub(z^{T_p^{ab_i}})$ for each of the eight children generated by OBA. It is assumed that the reader has a basic understanding of linear programming including the subjects of tableaus, basic variables, dual feasibility and reduced costs. For further reading on the subject, see[4].

Theorem 3.3.2. *When branching on two distinct variables with noninteger values,* $ub(z^{T_p^{ab_e}}) \geq ub(z^{T_p^{ab_i}})$.

Proof. Let (z^{*T_p}, x^{*T_p}) be an optimal solution to T_p's linear program. Let $c^\pi = \{c_1^\pi, c_2^\pi, ..., c_{n+m}^\pi, z^{*T_p}\}$ denote the objective row in the simplex tableau corresponding to the solution for T_p where indices $n+1$ to $n+m$ represent the slack variables for the constraints. Due to optimality $c_k^\pi \geq 0 \ \forall \ k = 1, ..., m+n$ and $c_k^\pi = 0$, if x_k is basic in x^{*T_p}. Let $a'_k = \{a'_{k,1}, a'_{k,2}, ..., a'_{k,n+m}, x_k^{*T_p}\}$ denote the row in T^p's optimal tableau corresponding to the basic variable x_k. Clearly, $a'_{k,l} = 0$ if $l \neq k$ and x_l is basic. When branching on x_i and x_j, there exists an a'_i and a'_j since both values are noninteger and therefore basic.

The proof continues by finding a bound on the equality children. Without loss of generality, it is sufficient to only consider $T_p^{LL_e}$ and $T_p^{LL_i}$. The other three cases are identical under some straightforward substitution. To evaluate the child $T_p^{LL_e}$, OBA forces the variables x_i and x_j to be $\lfloor x_i^{*T_p} \rfloor$ and $\lfloor x_j^{*T_p} \rfloor$, respectively.

To evaluate $T_p^{LL_e}$, it suffices to solve the following LP:

$$\begin{array}{ll} \text{Maximize} & z^{*T_p^{LL_e}} = \sum_{k=1}^{n+m} -c_k^{\pi} x'_k \\ \text{subject to} & \sum_{k=1}^{n+m} a'_{l_k} x'_k = x_l^{*T_p} \ \forall \ l \in [1,m] \cap \mathbb{Z} \\ & x'_i = \lfloor x_i^{*T_p} \rfloor \\ & x'_j = \lfloor x_j^{*T_p} \rfloor \\ & x \geq 0. \end{array}$$

To generate an upper bound $T_p^{LL_e}$, the above LP is relaxed to form the following LP:

$$\begin{array}{ll} \text{Maximize} & z'' = \sum_{k=1}^{n+m} -c_k^{\pi} x'_k \\ \text{subject to} & \sum_{k=1}^{n+m} a'_{i_k} x'_k = x_i^{*T_p} \\ & \sum_{k=1}^{n+m} a'_{j_k} x'_k = x_j^{*T_p} \\ & x'_i = \lfloor x_i^{*T_p} \rfloor \\ & x'_j = \lfloor x_j^{*T_p} \rfloor \\ & x \geq 0. \end{array}$$

Trivially, $z'' \geq z^{*T_p^{LL_e}}$ and $ub(z^{T_p^{LL_e}}) = z''$.

Next, $ub(z^{T_p^{LL_i}})$ is found. To evaluate the child $T_p^{LL_i}$, OBA forces the variables x_i and x_j to be less than or equal to $\lfloor x_i^{*T_p} \rfloor$ and $\lfloor x_j^{*T_p} \rfloor$, respectively.

To evaluate $T_p^{LL_i}$, the following LP must be solved:

$$\begin{array}{ll} \text{Maximize} & z^{*T_p^{LL_i}} = \sum_{k=1}^{n+m} -c_k^{\pi} x'_k \\ \text{subject to} & \sum_{k=1}^{n+m} a'_{l_k} x'_k = x_l^{*T_p} \ \forall \ l \in [1,m] \cap \mathbb{Z} \\ & x'_i \leq \lfloor x_i^{*T_p} \rfloor \\ & x'_j \leq \lfloor x_j^{*T_p} \rfloor \end{array}$$

$$x'_i + x'_j \leq x_i^{*T_p} + x_j^{*T_p} - 1$$
$$x \geq 0.$$

An upper bound is generated for $z^{*T_p^{LL_i}}$ by solving the following relaxed LP:

$$\begin{array}{ll}
\text{Maximize} & z' = \sum_{k=1}^{n+m-2} -c_k^{\pi} x'_k \\
\text{subject to} & \sum_{k=1}^{n+m} a'_{i_k} x'_k = x_i^{*T_p} \\
& \sum_{k=1}^{n+m} a'_{j_k} x'_k = x_j^{*T_p} \\
& x'_i \leq \lfloor x_i^{*T_p} \rfloor \\
& x'_j \leq \lfloor x_j^{*T_p} \rfloor \\
& x \geq 0.
\end{array}$$

Trivially, $z' \geq z^{*T_p^{LL_i}}$ and $ub(z^{T_p^{LL_i}}) = z'$. Consider relaxing this problem such that the third constraint does not have to be satisfied. Since α'_{i_i}, $\alpha'_{j_j} = 1$, the right hand side of the first two constraints are greater than zero, c_i^{π}, $c_j^{\pi} = 0$ and $c^{\pi} \geq 0$, then there trivially must be an optimal solution where x'_i and x'_j at their upper bounds. As a result the problem can be changed to the following:

$$\begin{array}{ll}
\text{Maximize} & z'' = \sum_{k=1}^{n+m-2} -c_k^{\pi} x'_k \\
\text{subject to} & \sum_{k=1}^{n+m} a'_{i_k} x'_k = x_i^{*T_p} \\
& \sum_{k=1}^{n+m} a'_{j_k} x'_k = x_j^{*T_p} \\
& x'_i = \lfloor x_i^{*T_p} \rfloor \\
& x'_j = \lfloor x_j^{*T_p} \rfloor \\
& x \geq 0.
\end{array}$$

Note that this is the same problem generated when finding $ub(z^{T_p^{LL_e}})$. Trivially, $z'' \geq z'$. As a result, $ub(z^{T_p^{LL_e}}) = z'' \geq z' = ub(z^{T_p^{LL_i}})$.

□

Theorem 3.3.2 suggests that the children with equality constraints are likely to have higher objective values than children with inequality constraints. In addition, by Theorem 3.3.1, children with equality constraints are typically easier to solve than children with inequailty constraints since children with equality constraints reduce the number of variables in the problem while children with inequality constraints often increase the basis by one due to adding an additional constraint. These two theorems combined suggest that children with higher objective values are easier to solve than children with lower objective values. By using a search strategy such as best bound, it is likely that many of the children that are more difficult to solve will be fathomed due to the existance of a superior integer solution.

Theorem 3.3.3. *OBA uses at most n of the children with superscripts LL_e, GL_e, LG_e, or GG_e consecutively before fathoming a pendant node.*

Proof. Each of the branching steps described by the superscripts LL_e, GL_e, LG_e, and GG_e set at least one of the variables to some constant not currently required by the constraints. This reduces the number of variables by at least one. It trivially follows that at most n of these branches can occur. $\square$

It is important to note that by adapting the algorithm that this theorem could be improved to say that at most $\lceil \frac{n}{2} \rceil$ of the children with superscripts LL_e, GL_e, LG_e, or GG_e consecutively before fathoming a pendant node. However, this would require that neither of the two new equality constraints added to the IP were redundant. To do this, any time there was only one noninteger variable, the branching tree would have to be checked to ensure the second variable selected for branching was not already forced to be constant by a prior branch. This also occurs when every iteration has at least two noninteger values. The next chapter provides computational evidence to support the thoretical benefits presented here.

Chapter 4

Computational Results for the Octanary Branching Algorithm

The Octanary Branching Algorithm (OBA) is a new branching algorithm that has shown theoretical benefits over traditional branch and bound. This chapter contains a computational study supporting these theoretical findings. OBA was able to find the first, second and third integer solution with 64.8%, 27.9% and 29.3% fewer nodes evaluated, respectively, than CPLEX. These integers were 44.9%, 54.7% and 58.2% closer to the optimal solution, respectively, when compared to CPLEX.

The C programming language was used to run OBA. Problems were solved using ILOG CPLEX 10.0[8]. Given that CPLEX has been developed and improved over the course of many years, it is unlikely to match the efficiency of the data retrieval and storage. As a result, only the number of nodes evaluated is considered in this computational study. This is reasonable as both OBA's and CPLEX's branching steps trivially take $O(n)$ time and constant space and $O(d)$ to load constraints where d is the depth of the node being evaluated.

The problems used for this study are multidimensional knapsack problems. Considerations were taken from Chvatal-Hard Problems[6] and a test bank[5] with multidimensional knapsack problems to generate problems for OBA. The IP's used for this study take the form of maximize $\sum_{i=1}^{n} c_i x_i$ subject to $\sum_{i=1}^{n} a_{ij} x_i \leq b_j \ \forall \ j \in [1, m] \cap \mathbb{Z}$ and $x_i \in [0, 20] \cap \mathbb{Z}$.

The a_{ij} are integer and are generated using a discrete uniform distribution with paramaters 0 and 1000. The right hand side b_j is calculated using $b_j = 10\sum_{i=1}^{n} a_{ij} \; \forall \; j \in [1, m] \cap \mathbb{Z}$. The objective function coefficients, c_i, are calculated using $c_i = u_i + \sum_{i=1}^{m} \frac{a_{ij}}{m} \; \forall \; i \in [1, n] \cap \mathbb{Z}$ where u_i is a random integer from a uniform discrete distribution with paramters 0 and 500. These problems are selected because the solution times are neither trivial nor prohibitively high.

First, OBA was compared to CPLEX's branching algorithm to determine the ability to quickly find quality integer solutions. In order to strictly test the quality of the branching structures, CPLEX's abilities to presolve nodes and apply various cuts and heuristics were disabled and no priority was used for variable selection. For both algorithms, depth first left was specified. Three sets of problems with 100 variables and 10 constraints, 250 variables and 25 constraints, and 500 variables and 50 constraints were considered.

When evaluating problems with 100 variables and 10 constraints, OBA outperforms CPLEX in nearly every instance. Only in problem j was CPLEX's first integer solution (the 118^{th} node was .3150% from optimal) superior to OBA's first integer solution (the 39^{th} node was .3322% from optimal). However, OBA's second integer found (the 40^{th} node was .2328% from optimal), exceeded CPLEX's best early solutions. The full table of results can be seen in Table 4.1.

When evaluating problems with 250 variables and 25 constraints, OBA finds the first integer with 58.4% fewer nodes with an average value 45.0% closer to the actual optimal value than CPLEX. In the only instance in which CPLEX finds a superior first integer solution (problem i), OBA manages to find a better integer value in fewer nodes.

OBA again outperforms CPLEX when evaluating problems with 500 variables and 50 constraints. For this case, OBA was superior to CPLEX for every instance. There were some instances, however, in which OBA was able to find a second or third integer solution prior to evaluating 100,000 nodes.

For the first three problems in each set, information on the first 50,000 nodes evaluated was collected. For information on all nine problems plotted, see the appendix. Figure 4.1 suggests that OBA finds quality integer solutions more quickly than BB.

Although OBA typically finds integer solutions with fewer nodes than BB, BB tends to solve the problems faster. It is believed that this is due to OBA's massive enumeration tree. At a depth of d, OBA has up to 8^d unevaluated nodes (compared to BB's 2^d). When the tree is sufficiently deep, OBA must fathom 8 nodes per parent, but BB only needs to fathom 2. Thus, OBA spends a substantial amount of time generating infeasible and inferior nodes. It is for this reason that OBA is suggested to only be used as a warm start or for random diving.

Overall, OBA shows to be computationally superior than CPLEX at quickly finding quality integer solutions. OBA was able to find the first, second and third integer solutions with 64.8%, 27.9% and 29.3% fewer nodes than CPLEX. In addition, these integers were 44.9%, 54.7% and 58.2% closer to the optimal solution, respectively, when compared to CPLEX.

Table 4.1: *1^{st}, 2^{nd} and 3^{rd} Integers Found with 100 Variables and 10 Constraints*

		1^{st} Int Found		2^{nd} Int Found		3^{rd} Int Found		
Prob		Node	Gap	Node	Gap	Node	Gap	Optimal
a	oba	35	0.2530%	36	0.1437%	248	0.1373%	1879428
a	cplex	126	0.4926%	506	0.4776%	514	0.4719%	1879428
b	oba	33	0.1474%	37	0.1447%	49	0.1428%	1861953
b	cplex	95	0.1706%	115	0.1697%	118	0.1633%	1861953
c	oba	69	0.2239%	90	0.2138%	139	0.2093%	1843446
c	cplex	97	0.4114%	101	0.4109%	108	0.4104%	1843446
d	oba	32	0.1703%	3591	0.1669%	5103	0.1635%	1864303
d	cplex	126	0.7193%	3100	0.7173%	3332	0.7151%	1864303
e	oba	31	0.1779%	32	0.0956%	593	0.0876%	1857056
e	cplex	104	0.2351%	107	0.2292%	113	0.2256%	1857056
f	oba	32	0.1637%	573	0.0976%	749	0.0959%	1880600
f	cplex	139	0.5602%	183	0.5559%	265	0.5425%	1880600
g	oba	35	0.2649%	36	0.1660%	37	0.1614%	1857644
g	cplex	127	0.3072%	192	0.3045%	268	0.3025%	1857644
h	oba	65	0.2221%	72	0.2199%	79	0.2034%	1863576
h	cplex	123	0.2411%	154	0.2366%	158	0.2363%	1863576
i	oba	34	0.2410%	35	0.1450%	44	0.1439%	1857345
i	cplex	90	0.2535%	447	0.2495%	1541	0.2382%	1857345
j	oba	39	0.3322%	40	0.2328%	234	0.2238%	1869292
j	cplex	118	0.3150%	122	0.3089%	176	0.2995%	1869292
Avg	**oba**	**40.5**	**0.2196%**	**454.2**	**0.1626%**	**727.5**	**0.1569%**	**1863464.3**
Avg	**cplex**	**114.5**	**0.3706%**	**502.7**	**0.3660%**	**659.3**	**0.3605%**	**1863464.3**

Table 4.2: *1st, 2nd and 3rd Integers Found with 250 Variables and 25 Constraints*

		1st Integer Found		2nd Integer Found		3rd Integer Found		
Prob		Node	Gap	Node	Gap	Node	Gap	Optimal
a	oba	82	0.1096%	102153	0.1090%	119845	0.1080%	4694496
a	cplex	251	0.1956%	1572	0.1935%	1621	0.1920%	4694496
b	oba	81	0.1095%	82	0.0775%	83	0.0745%	4655554
b	cplex	291	0.2382%	113838	0.2325%	113863	0.2319%	4655554
c	oba	159	0.1457%	166	0.1444%	187	0.1437%	4637609
c	cplex	337	0.2889%	342	0.2867%	351	0.2864%	4637609
d	oba	158	0.1884%	165	0.1849%	179	0.1829%	4649944
d	cplex	336	0.3637%	444	0.3631%	547	0.3630%	4649944
e	oba	83	0.1395%	84	0.0998%	85	0.0987%	4674379
e	cplex	288	0.2014%	1560	0.2002%	1979	0.1987%	4674379
f	oba	82	0.1583%	83	0.1209%	205	0.1196%	4644373
f	cplex	250	0.3209%	5171	0.3208%	5520	0.3207%	4644373
g	oba	79	0.1143%	80	0.0740%	124	0.0736%	4669568
g	cplex	306	0.2952%	362	0.2950%	423	0.2946%	4669568
h	oba	163	0.1570%	170	0.1483%	476	0.1474%	4669567
h	cplex	315	0.2511%	317	0.2497%	331	0.2481%	4669567
i	oba	172	0.1571%	179	0.1478%	202	0.0976%	4658365
i	cplex	265	0.1429%	812	0.1416%	4882	0.1412%	4658365
j	oba	159	0.1017%	166	0.1004%	173	0.0999%	4689443
j	cplex	290	0.2155%	698	0.2154%	842	0.2145%	4689443
Avg	**oba**	**121.8**	**0.1381%**	**10332.8**	**0.1207%**	**12155.9**	**0.1146%**	**4664329.8**
Avg	**cplex**	**292.9**	**0.2513%**	**12511.6**	**0.2498%**	**13035.9**	**0.2491%**	**4664329.8**

Table 4.3: *1st, 2nd and 3rd Integers Found with 500 Variables and 50 Constraints*

		1st Integer Found		2nd Integer Found		3rd Integer Found		
Prob		Node	Gap	Node	Gap	Node	Gap	Optimal
a	oba	164	0.0987%	327	0.0984%	334	0.0983%	9282051
a	cplex	518	0.2109%	540	0.2104%	548	0.2098%	9282051
b	oba	164	0.0814%	408	0.0806%	6792	0.0805%	9297789
b	cplex	701	0.3071%	1186	0.3070%	1865	0.3070%	9297789
c	oba	162	0.1002%	374	0.0997%	444	0.0991%	9343215
c	cplex	513	0.1939%	576	0.1934%	829	0.1930%	9343215
d	oba	167	0.1025%	———		———		9286651
d	cplex	591	0.2486%	595	0.2479%	639	0.2479%	9286651
e	oba	168	0.1098%	169	0.0908%	1813	0.0894%	9266250
e	cplex	601	0.2024%	814	0.2019%	32307	0.2018%	9266250
f	oba	175	0.1345%	176	0.1150%	177	0.1141%	9336665
f	cplex	568	0.2324%	594	0.2322%	1511	0.2322%	9336665
g	oba	176	0.1267%	177	0.1102%	185	0.1077%	9323660
g	cplex	636	0.2428%	639	0.2428%	718	0.2411%	9323660
h	oba	157	0.0740%	158	0.0548%	———		9375043
h	cplex	531	0.1585%	543	0.1582%	624	0.7337%	9375043
i	oba	168	0.0906%	169	0.0705%	173	0.0704%	9303255
i	cplex	523	0.2199%	3059	0.2198%	3153	0.2192%	9303255
j	oba	314	0.1247%	321	0.1244%	328	0.1239%	9319619
j	cplex	521	0.1527%	14359	0.1527%	21261	0.1526%	9319619
Avg	**oba**	**181.5**	**0.1043%**	**253.2**	**0.0938%**	**1280.8**	**0.0979%**	**9313419.8**
Avg	**cplex**	**570.3**	**0.2169%**	**2290.5**	**0.2166%**	**6345.5**	**0.2738%**	**9313419.8**

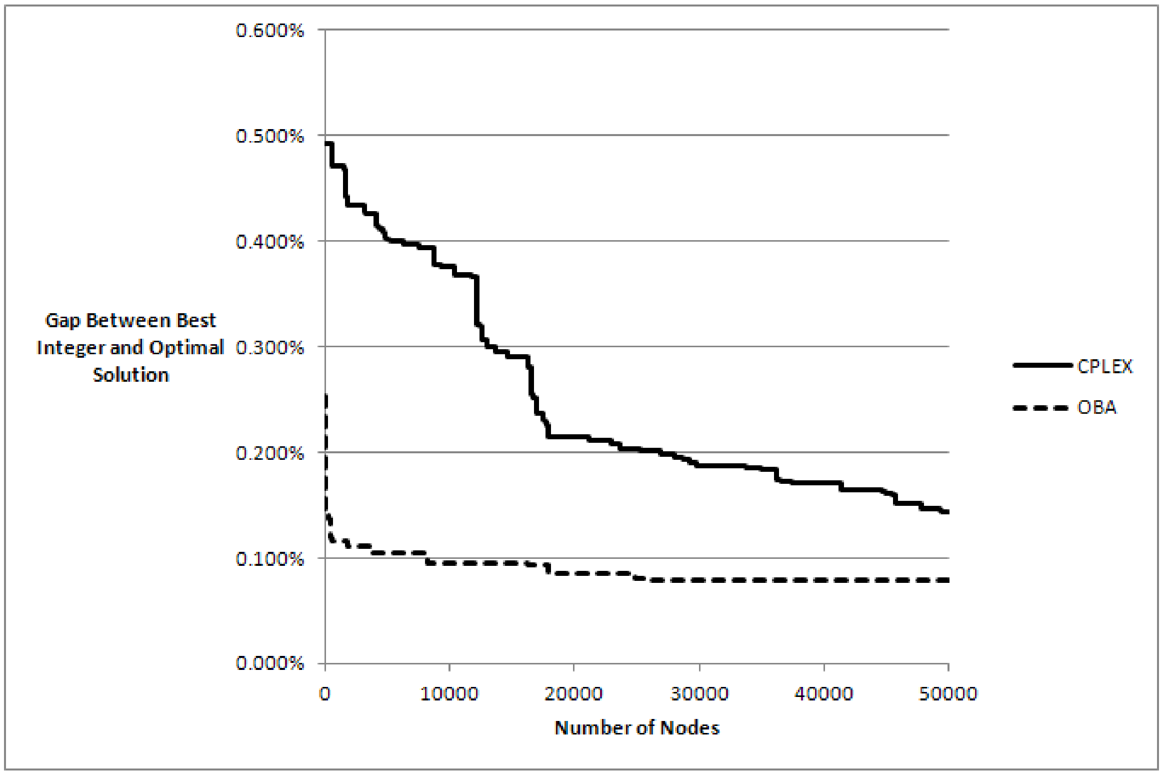


Figure 4.1: *Gap Between Best Integer Solution and Optimal Solution as a Function of Nodes Evaluated for prob.100.10.a.txt*

Table 4.4: *Summary of 1st, 2nd and 3rd Integers Found*

		1st Integer Found		2nd Integer Found		3rd Integer Found	
Prob		Node	Gap	Node	Gap	Node	Gap
Avg for 100.10	oba	40.5	0.2196%	454.2	0.1626%	727.5	0.1569%
Avg for 100.10	cplex	114.5	0.3706%	502.7	0.3660%	659.3	0.3605%
Avg for 250.25	oba	121.8	0.1381%	10332.8	0.1207%	12155.9	0.1146%
Avg for 250.25	cplex	292.9	0.2513%	12511.6	0.2498%	13035.9	0.2491%
Avg for 500.50	oba	181.5	0.1043%	253.2222	0.0938%	1280.75	0.0979%
Avg for 500.50	cplex	570.3	0.2169%	2290.5	0.2166%	6345.5	0.2738%
Average	oba	114.6	0.1540%	3680.074	0.1257%	4721.383	0.1231%
Average	cplex	325.9	0.2796%	5101.6	0.2775%	6680.233	0.2945%
Improvement Over cplex		**64.8%**	**44.9%**	**27.9%**	**54.7%**	**29.3%**	**58.2%**

Chapter 5

Conclusion

This thesis introduced a new branching algorithm called the Octanary Branching Algorithm. The primary advancement of OBA is the introduction of equality constraints close to the parent's solution. OBA has a variety of theoretical benefits and computational benefits over traditional branch and bound.

Theoretically, OBA causes half of the children to have a dimension at least one less than the dimension of its parent. Additionally, for children that do not reduce the dimension of the problem, it has been shown that the upper bound for the objective value is lower than the upper bound generated for children that reduce the dimension of the linear relaxation. OBA also ensures that only n of the four left most children can be used before fathoming a node.

OBA was able to find the first, second and third integer solution with 64.8%, 27.9% and 29.3% fewer nodes evaluated, respectively, than CPLEX. These integers were 44.9%, 54.7% and 58.2% closer to the optimal solution, respectively.

5.1 Future Research

The work presented in this thesis generates a variety of new research questions. As a result, there is future work that should be pursued to further advance branch and bound.

To date, selecting variables during the branching step by generating psuedo reduced costs has been observed to increase the time it takes to solve an IP[21]. However, OBA partitions the space such that the four right most children are likely to have significantly worse objective values with the proper selection of branching variables. This raises the question of whether it would be valuable to spend additional time in the branching step to find the "best" variables to branch on.

Additionally, OBA demonstrates a superior ability to find integer solutions in spite of the fact that it often takes more time to evaluate the entire tree. For these reason, it seems highly likely that OBA would be extremely useful for the random diving process. By using OBA strictly for random diving, users would be able to gain the benefit of being able to find integer solutions quickly without having a tree that grows so rapidly.

Other branching structures with similar properties to OBA should also be explored. OBA forces variables to be constant during the branching step in hopes of quickly finding integer solutions. The branching structure generated is just one of an infinite number of possibilities that could be used to force variables to become constant after a single interation. The question of whether some other structure with potentially more or less variables could generate better results still remains.

Another property of OBA is that the four children that create equality constraints represent an "inner layer" of the feasible region close to the parent's LR while the four children with inequality constraints represent the "outer layer." Any additional layers would be further from the parent's solution and would be likely to have significantly worse objective values. Could a branching scheme that creates more than two layers perform better than OBA?

Bibliography

[1] T. Achterberg and A. Martin. Miplib 2003. *Operations Research Letters*, 34(4), 2006.

[2] N. Agin. Optimum seeking with branch and bound. *Management Science*, 13(4), 1966.

[3] J. Bailey, M. Iwen, and C. Spencer. On the design of deterministic matrices for fast recovery of fourier compressible functions. *SIAM Journal on Matrix Analysis and Applications (SIMAX)*, 33, 2012.

[4] M. S Bazaraa, J. J. Jarvis, and H. D. Sherali. *Linear Programming and Network Flows.* New Jersey: John Wiley & Sons Inc., 2005.

[5] P. Chu and J. Beasley. A genetic algorithm for the multidimensinoal knapsack problem. *Journal of Heuristics*, 4, 1998.

[6] V. Chvátal. Hard knapsack problems. *Operations Research*, 28(6), 1980.

[7] W. Hamilton and M. Moses. An optimization model for corporate financial planning. *Operations Research*, 21(3), 1973.

[8] IBM ILOG CPLEX Inc. *Using CPLEX Callable Library.* 2006. Version 10.0.

[9] K. O. Jörnsten and P. Värbrand. A hybrid algorithm for the generalized assignment problem. *Optimization*, 22(2), 1991.

[10] R. Karp. *Complexity of Computer Computations.* R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972.

[11] L. Khachiyan. A polynomial time algorithm in linear programming. *Soviet Math. Dokl.*, 20, 1979.

[12] P. Krokhmal, S. Uryasev, and G. Zrazhevsky. Risk management for hedge fund portolios: A comparative analysis of linear rebalancing strategies. *The Journal of Alternative Investments*, 5(1), 2002.

[13] A. Land and A. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28, 1960.

[14] E. Lee, T. Fox, and I. Crocker. Integer programming applied to intensitymodulated radiation therapy treatment planning. *Annals of Operations Research*, 119, 2003.

[15] J. H. Lee. Theoretically and computationally improving branch and bround through multivariate branching with internal cutting planes. Master's thesis, Kansas State University, 2010.

[16] M. Lustig, D. L. Donoho, and J. M. Pauly. Sparse mri: The application of compressed sensing for rapid mr imaging. *Magnetic Resonance in Medicine*, 58(6), 2007.

[17] M. Lustig, D. L. Donoho, J. M. Santos, and J. M. Pauly. Compressed sensing mri. *IEEE Signal Processing Magazine*, 25(2), 2008.

[18] J. Ma. Single-pixel remote sensing. *IEEE Geoscience and Remote Sensing Letters*, 6(2), 2009.

[19] J. Ma. Compressed sensing for surface characterization and metrology. *IEEE Transactions on Instrumentation and Measurement*, 69(6), 2010.

[20] Mehrotra and S. Z. Li. Branching on hyperplane methods for mixed linear and convex programming using adjoin lattices. *Journal on Global Optimization*, 2010.

[21] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* New York: John Wiley & Sons Inc., 1999.

[22] P. Pendharkar and J. Rodger. Information tecnology capital budgeting using a knapsack problem. *International Transactions in Operations Research*, 13, 2006.

[23] R. L. Magananti R. K. Ahuja and J B. Orlin. *Network Flows: Theory, Algorithms and Applications*. New Jersey: Prentice-Hall Inc., 1993.

[24] B. Ryan and D. Foster. *Computer Scheduling of Public Transport*, chapter An integer programming approach to scheduling, pages 269–280. A. Wren eds., North-Holland Publishing Company, 1981.

[25] N. Shental, A. Amir, and O. Zuk. Identification of rare alleles and their carriers using compressed se(que)nsing. *Nucleic Acids Research*, 38(19), 2010.

[26] G. Singh, D. Sier, A. T. Ernst, O. Gavrillouk, R. Oyston, and T. Giles andP. Welgama. A mixed integer programming model for long term capacity expansion planning: A case study from the hunter valley coal chain. *European Journal of Operational Research*, 220(1), 2012.

[27] J. Stahl, N. Kong, S. Shechter, S. Shaefer, and S. Roberts. A methodological framework for optimally reorganizing liver transplant regions. *Medical Decision Making*, 2535, 2005.

[28] Subramanian, R. Scheff, R. Quillinan, J. Wiper, and R. Marsten. Cold-start: fleet assignment at delta airlines. *Interfaces*, 24(1), 1994.

[29] R. J. Walker. An enumerative technique for a class of combinatorial problems. *Proceedings of Symposia in Applied Mathematics*, Math 10, 1960.

Appendix A

Appendix

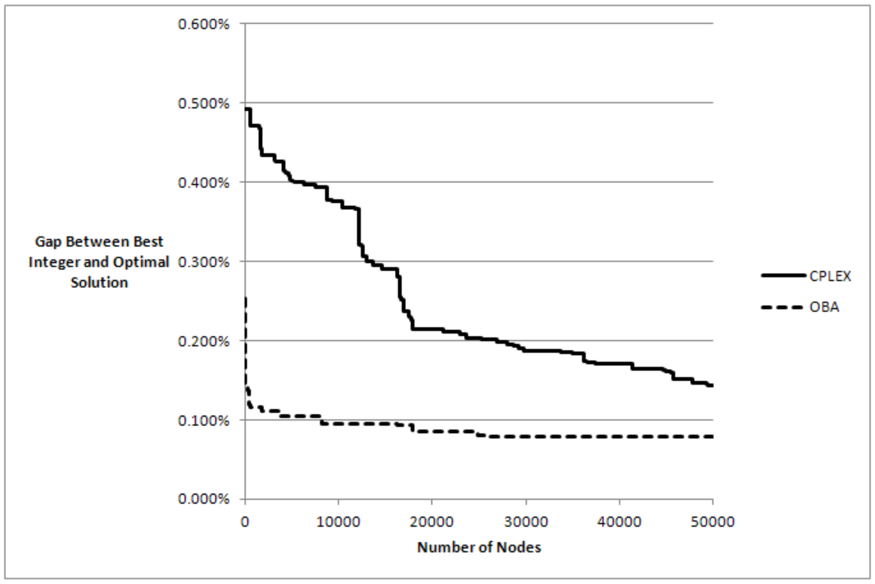


Figure A.1: *Gap Between Best Integer Solution and Optimal Solution as a Function of Nodes Evaluated for prob.100.10.a.txt*

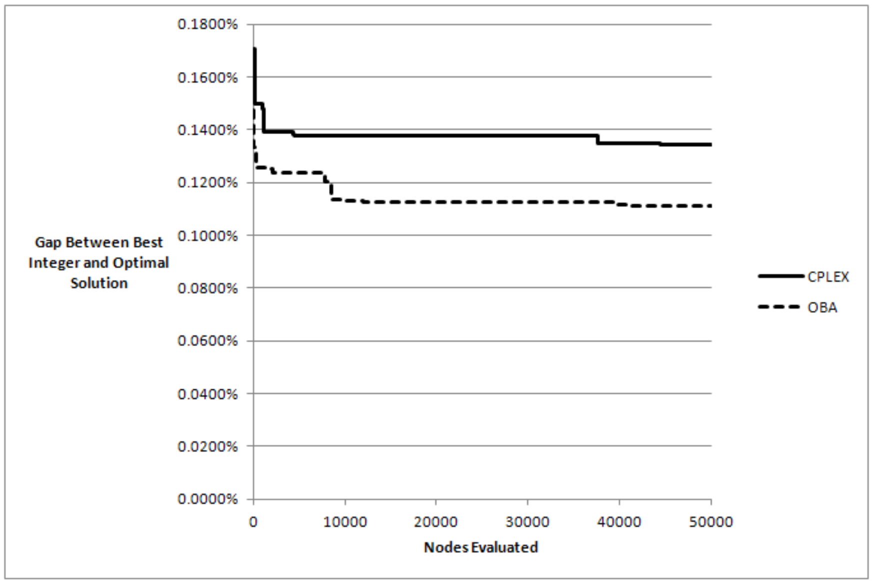


Figure A.2: *Gap Between Best Integer Solution and Optimal Solution as a Function of Nodes Evaluated for prob.100.10.b.txt*

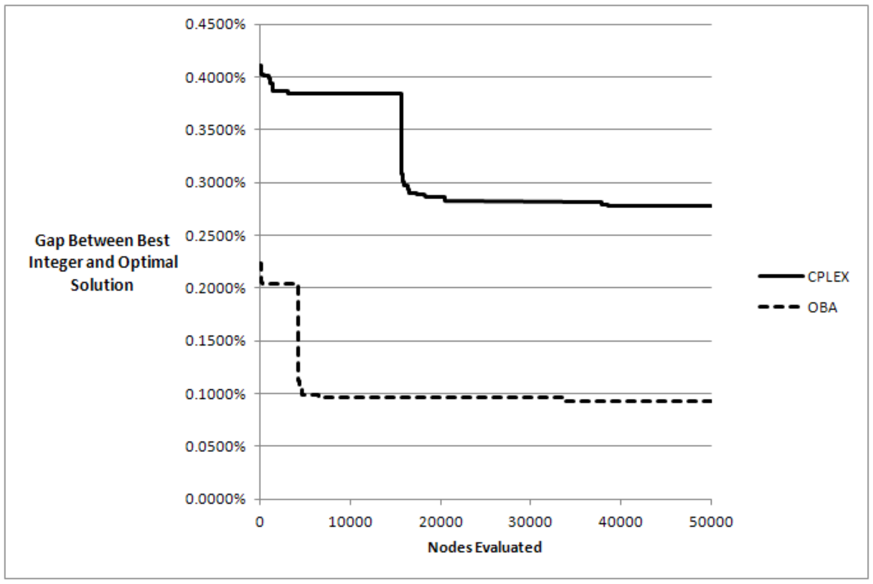


Figure A.3: *Gap Between Best Integer Solution and Optimal Solution as a Function of Nodes Evaluated for prob.100.10.c.txt*

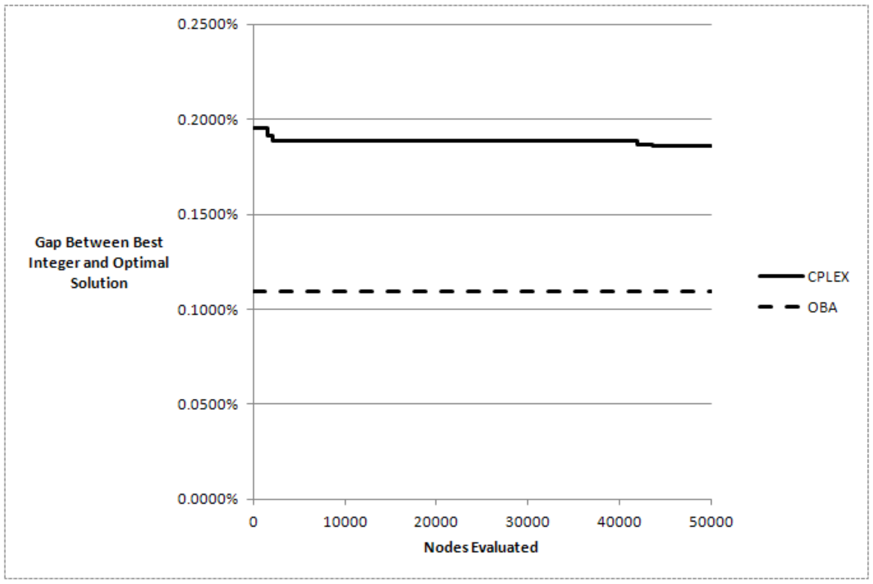


Figure A.4: *Gap Between Best Integer Solution and Optimal Solution as a Function of Nodes Evaluated for prob.250.25.a.txt*

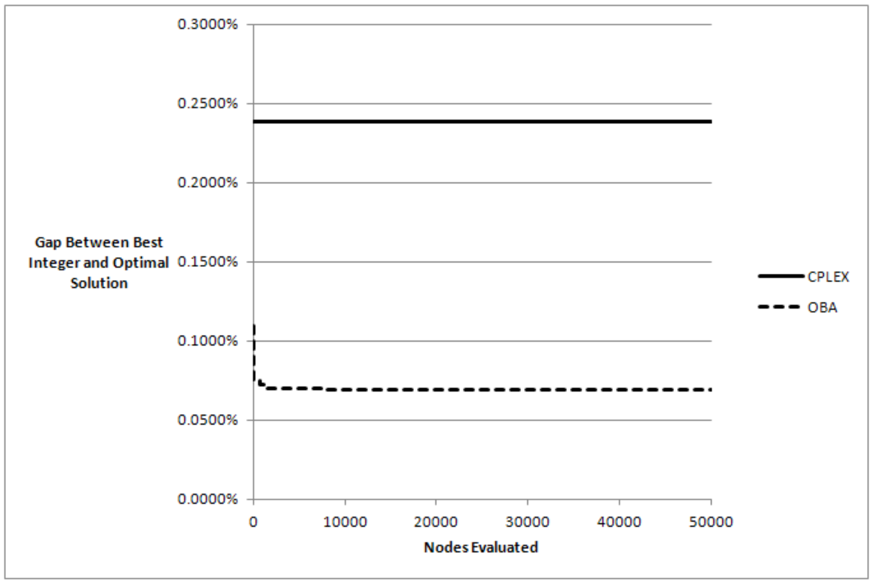


Figure A.5: *Gap Between Best Integer Solution and Optimal Solution as a Function of Nodes Evaluated for prob.250.25.b.txt*

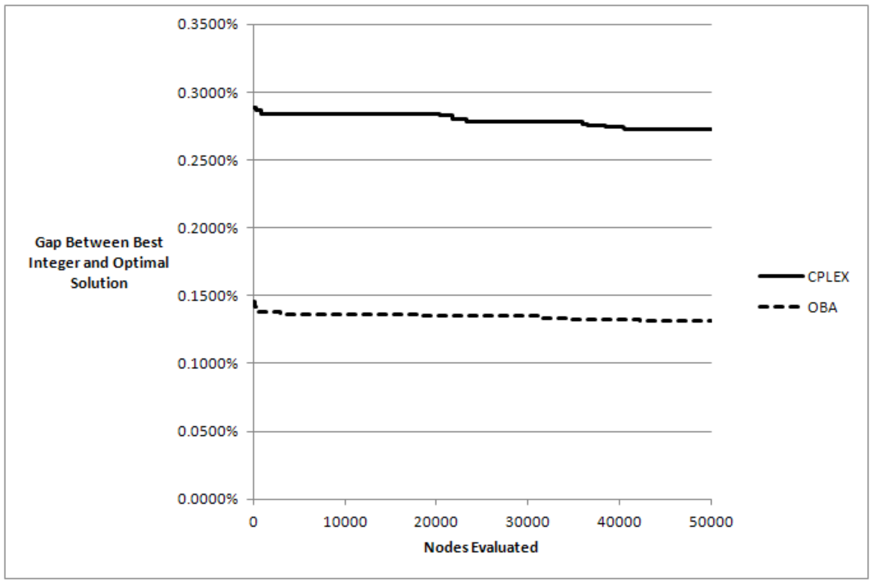


Figure A.6: *Gap Between Best Integer Solution and Optimal Solution as a Function of Nodes Evaluated for prob.250.25.c.txt*

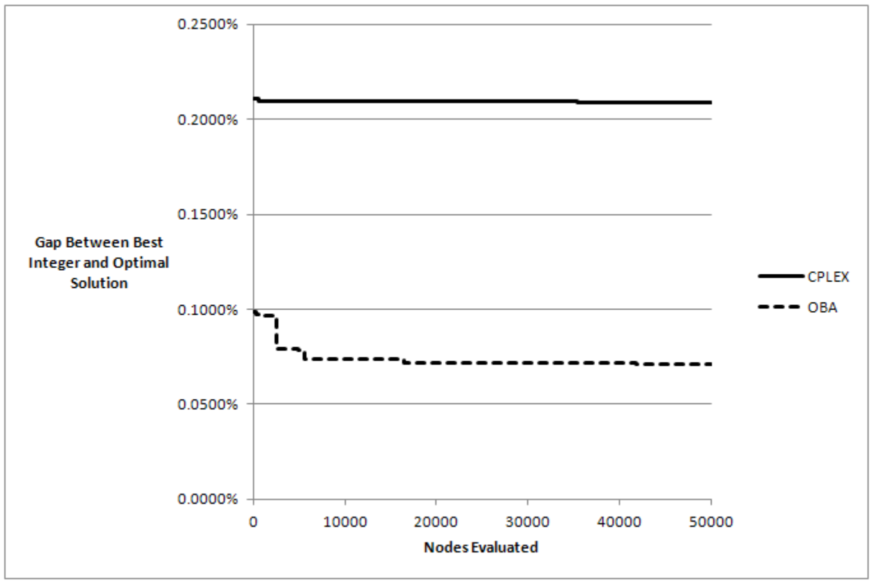


Figure A.7: *Gap Between Best Integer Solution and Optimal Solution as a Function of Nodes Evaluated for prob.500.50.a.txt*

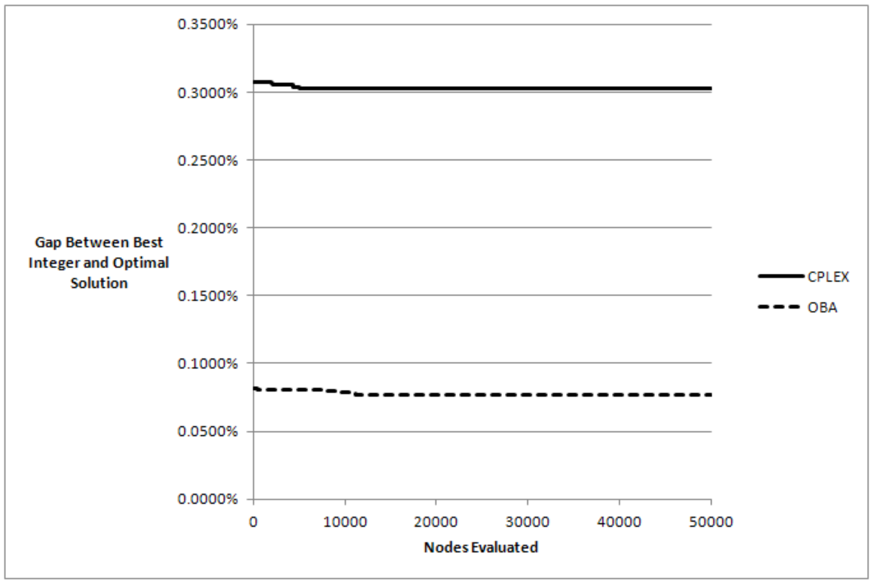


Figure A.8: *Gap Between Best Integer Solution and Optimal Solution as a Function of Nodes Evaluated for prob.500.50.b.txt*

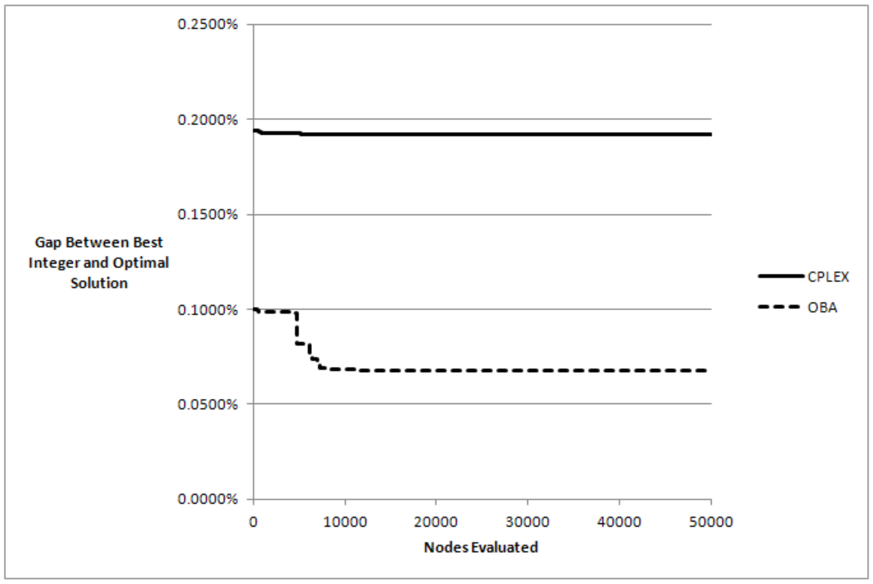


Figure A.9: *Gap Between Best Integer Solution and Optimal Solution as a Function of Nodes Evaluated for prob.500.50.c.txt*