Lifted Equality Cuts for the Multiple Knapsack Equality Problem

by

Alonso Talamantes

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
2017

Approved by:

Major Professor
Dr. Todd Easton

Copyright

Abstract

Integer programming is an important discipline in operation research that positively impacts society. Unfortunately, no algorithm currently exists to solve IP's in polynomial time. Researchers are constantly developing new techniques, such as cutting planes, to help solve IPs faster. For example, DeLissa discovered the existence of equality cuts limited to zero and one coefficients for the multiple knapsack equality problem (MKEP). An equality cut is an improper cut because every feasible point satisfies the equality. However, such a cut always reduces the dimension of the linear relaxation space by at least one.

This thesis introduces lifted equality cuts, which can have coefficients greater than or equal to two. Two main theorems provide the conditions for the existence of lifted equalities. These theorems provide the foundation for The Algorithm of Lifted Equality Cuts (ALEC), which finds lifted equality cuts in quadratic time.

The computational study verifies the benefit of lifted equality cuts in random MKEP instances. ALEC generated millions of lifted equality cuts and reduced the solution time by an average of 15%. To the best of the author's knowledge, ALEC is the first algorithm that has found over 30.7 million cuts on a single problem, while reducing the solving time by 18%.

Dedication

I dedicate my work to my family. Alonso Talamantes Guiterrez and Rosalia Talamantes Rey, my parents, courageously moved to a foreign country in hopes of providing a better life for my sisters and myself. I am extremely grateful for their unconditional support and would not be here without them.

My three sisters, Alma Henry, Ana Flores and Angelica Talamantes, and my two brothers Jose Flores, and Nathan Henry have also had a huge impact on my life. They continously give me great life advice and always remind me that family comes first. I would also like to mention my nieces and nephew, Annabel Flores, Olivia Flores, Mia Henry, and Dominic Henry. They always keep me laughing and teach me that it is the little things in life that matter the most.

Acknowledgments

I would like to acknowledge my K-State Industrial Engineering "family" and my K-State alto saxophone "family". I am blessed to have these terrific friends who have supported me. I would also like to acknowledge Dr. Jessica Heier Stamm for agreeing to be on my committee, but more importantly for being one of my role models. Her passion towards using her technical skills to help other people is inspiring. I would also like to acknowledge Dr. Frank Tracz for being on my committee, always pushing me, and always encouraging me to practice one more time. Lastly, I want to acknowledge Dr. Todd Easton. He has guided me throughout my college career and has made this research possible. From our chess games to real life talk and the occasional stern motivational discussions, he is a terrific mentor and I am happy to also call him my friend.

Table of Contents

List of Tables

Chapter 1

Introduction

Integer programming (IP) is a discipline in operations research that positively impacts society. Integer programs are mathematical optimization problems with integer decision variables. These decision variables are used to create models that optimize a linear objective function subject to resources or other limiting linear constraints. Integer programming is widely used for many real world applications.

Numerous IPs have been applied to sports [1, 2, 3, 4, 5]. One such application helps the National Collegiate Athletic Association (NCAA) host the national championship basketball tournament, which involves 68 collegiate basketball teams [6]. The location of the basketball games are spread throughout the U.S. and the NCAA pays for the teams' traveling expenses. In 2010, the NCAA basketball tournament had a significant increase in traveling cost and decrease in attendance because teams were traveling far from home. As a result, an integer program was developed to minimize the distance and cost. The decision variables are which teams should play at which locations while complying with NCAA regulations. The IP

created a schedule for the NCAA basketball tournament that saves 330,000 dollars a year. This IP had over 2,300 variables and 150 constraints [7].

There are variety of other IP applications. In manufacturing, IPs are used in inventory management [8, 9], production scheduling [10, 11], and supply chain networks [12, 13]. IPs are also often used for humanitarian relief disasters such as facility location [14, 15], supply distribution [16, 17], and pre-disaster polices [18, 19].

This research focuses on a special case of an IP called the knapsack problem (KP). The concept of a knapsack problem relates to a hiker who has a knapsack. The hiker must choose what hiking supplies to include in the knapsack. Each item has an associated non-negative weight and perceived benefit. The total weight of the items must be less than or equal to the hiker's carrying capacity while maximizing benefit. A knapsack problem with multiple constraints is called the multiple knapsack problem.

Other variants of the knapsack problem exist such as the knapsack equality problem (KEP) and the knapsack demand problem (KDP). KEP is a knapsack problem with an equality constraint and KDP has a greater than or equal to inequality. Multiple constraints in the KEP and KDP are called multiple knapsack equality problem (MKEP) and multiple knapsack demand problem (MKDP). Variants of the knapsack problem are used in many applications.

In health care, Wang and Hu [20] created a model for radiotherapy that minimized the exposure to healthy tissue while killing cancerous cells. Minimizing radiation exposure to healthy tissues increases a person's chance to live. Other applications in health care include diagnosis [21] and client preventive health care [22]. In all of the examples, a variant of

the knapsack problem exist. KP, KEP, and KDP also appear in scheduling for workforce [23, 24, 25] and routing networks [26, 27].

Unfortunately, IP, KP, KDP and KEP are all $\mathcal{NP}$-hard problems [28]. In other words, a polynomial time algorithm does not currently and may never exist. All known algorithms require an exponential amount of effort to solve an IP. Even a small IP with 100 binary decision variables has more than $2.5 * 10^{30}$ possible combinations. Attempting to enumerate and evaluate all possible solutions would require millions of years even on the fastest computers. Because of this, researchers must simplify or use advanced algorithm techniques to obtain a solution.

One of the most popular methods to solve IPs is branch and bound, which uses linear relaxations (LR). The linear relaxation of an IP is the IP without integer restriction. An IP's linear relaxation is called a linear program (LP), which can be solved in polynomial time [29, 30, 31, 32].

Branch and bound has an exponential run time. A popular technique to help branch and bound solve an IP quicker involves cutting planes. Traditionally, cutting planes are viewed as inequalities that are not originally present in the IP and do not violate any feasible solutions. Cutting planes create smaller linear relaxation spaces, which can be computationally advantageous.

Today, researchers [33, 34, 35, 38] have developed various classes of cutting planes to help IPs solve faster. Theoretically, the strongest cutting planes are called facet defining. If all facet defining valid inequalities are identified, then the IP can be solved as an LP. However, finding facet defining inequalities is challenging and there can be exponentially

many of them.

Lifting is a common technique to modify a weak valid inequality into a stronger and potentially facet defining inequality. There are 12 classes of lifting sorted into three categories. Lifting can occur sequentially or simultaneously [36], exact or approximate [37], and up, down or middle [38]. Selecting one from each of these three categories creates a type of lifting. For instance, there exists a sequential approximate uplifting technique [39].

In 2014, DeLissa [40] developed an entirely different concept of cutting planes. Rather than use inequalities, he used equalities to reduce the linear relaxation space, which he called equality cuts. These equality cuts were computationally advantageous in helping to solve knapsack related random IP instances. Although equality cuts, by definition, cannot be facet defining, they reduce the dimension of the linear relaxation space.

1.1 Research Motivation and Question

This research focuses on continuing the research of equality cuts. DeLissa's work required every coefficient to be zero or one in the equality cut. An obvious question is whether or not equality cuts exist with coefficients that are greater than or equal to two. Since lifting increases the coefficients of valid inequalities, this thesis will address the following questions: Does a lifted valid equality exist and under what conditions? Can lifted valid equalities be computationally advantageous in helping solve MKEPs?

1.2 Contribution

This research introduces a new class of equality cuts, called lifted equality cuts for MKEPs. This thesis discovers two theorems that provide conditions under which lifted equality cuts exist. These theorems are used to create ALEC (Algorithm of Lifted Equality Cuts). ALEC is a fast algorithm and requires only quadratic time. An example demonstrates the existence and usefulness of these lifted equality cuts. A computational study shows that implementing ALEC on some IPs improves the computational time required to solve these problems by 15% while generating millions of lifted equality cuts.

1.3 Outline

The remainder of this thesis will be covered in four chapters. Chapter 2 will cover background information needed to understand the contributions of this thesis. Topics discussed in Chapter 2 include integer programming, knapsack and related problems, cutting planes, polyhedral theory, lifting, and equality cuts.

Chapter 3 provides the existence of lifted equality cuts including definitions, theorems, and examples. In addition, Chapter 3 discusses an algorithm to identify lifted equality cuts, ALEC, and its properties including the theoretical run time. Chapter 3 illustrates how lifted equality cuts can reduce the dimension of the linear relaxation space through an example.

Chapter 4 discusses ALEC's impact on solving MKEPs through a computational study. ALEC's implementation is discussed as well as instances used. The results illustrate how often valid lifted equalities appear and the benefit of these equalities in terms of several

efficiency measures.

Chapter 5 summarizes the results and contributions of this thesis. In addition, Chapter 5 discusses future research in lifted equality cuts such as implementing ALEC on real world IPs, relationships between multiple equality constraints, and extending lifted equality cuts to mixed integer programming and nonlinear integer programming.

Chapter 2

Background Information

This chapter provides relevant background information in order to understand the remainder of this thesis. Topics such as integer programming, the knapsack and related problems, polyhedral theory, cutting planes, lifting, and equality cuts are all discussed. Additional information on these topics can be found in Nemhauser and Wolsey [41].

2.1 Integer Programming

Integer programming (IP) is a mathematical model of the form $\max c^T x$ subject to $Ax \leq b$, $x \in \mathbb{Z}_+^n$ where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$. The feasible solution space takes the form $P = \{x \in \mathbb{Z}_+^n : Ax \leq b\}$. The optimal solution to an IP is z^* and x^*, where $x^* \in P$ and $z^* = c^T x^* \geq c^T x$ for all $x \in P$.

As mentioned in Chapter 1, integer programming has many real world applications. An IP is an $\mathcal{NP}$-hard problem and all currently known algorithms theoretically require

an exponential amount of time to solve. Although IPs are useful in solving real world applications, some applications can take an unrealistic amount of time to solve. Due to these reasons, integer programming remains a prolific topic of researchers.

Branch and bound, the most common algorithm used to solve IPs, uses the IP's linear relaxation. An IP's linear relaxation takes the form $\max c^T x$ subject to $Ax \leq b$, $x \in \mathbb{R}^n_+$ where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$. The linear relaxation's feasible solution space is denoted as $P^{LR} = \{x \in \mathbb{R}^n_+ : Ax \leq b\}$. The optimal solution to a linear relaxation is z^{LR*} and x^{LR*}, where $x^{LR*} \in P^{LR}$ and $z^{LR*} = c^T x^{LR*} \geq c^T x$ for all $x \in P^{LR}$. Obviously, a linear relaxation is a linear program. Algorithms exist that solve LPs in polynomial time [29, 30, 31, 32].

Initially, branch and bound solves the IP's linear relaxation (z^{LR*}, x^{LR*}) and branches on a non integer variable, x_i^{LR*}. The algorithm then creates two children nodes by adding a constraint $x_i \leq p$ and $x_i \geq p + 1$ to the parent's linear relaxation, where $p = \lfloor x_i^{LR*} \rfloor$. Branch and bound successively resolves linear relaxations and creates nodes until all nodes are fathomed. Branch and bound has three rules to fathom nodes. Any node can be fathomed if the solution to the linear relaxation is an integer feasible solution, infeasible, or has a worse objective function than a known integer feasible solution. Once all nodes are fathomed, the algorithm terminates and reports the IP solution, if one exists.

The rules for which nodes are evaluated first is arbitrary. The most common rules are breadth first left, breadth first right, depth first left, depth first right, and best child. Much research has been done on the selection of variables to branch on and the best search strategy to evaluate the nodes [42, 43].

Table 2.1: Hiker's Items Associated Benefit and Weight

Items	1	2	3	4	5	6	7	8
Benefit	46	3	34	50	22	1	7	16
Weight	30	29	22	16	15	7	6	5

2.1.1 Knapsack and Related Problems

Recall from Chapter 1 the concept of a knapsack problem (KP). The knapsack problem can be formulated into an IP with the form $\max c^T x$ subject to $\sum_{j=1}^{n} a_j x_j \leq b$, $x \in \{0,1\}^n$, where b and $a_j \in \mathbb{R}_+$ for all $j = 1, ..., n$. The feasible space of a KP is defined by $P_{KP} = \{x \in \{0,1\}^n : \sum_{j=1}^{n} a_j x_j \leq b\}$ and the corresponding linear relaxation is P_{KP}^{LR}. Without loss of generality, the remainder of the paper assumes $a_{j-1} \geq a_j$ for all $j = 2, ..., n$.

For example, consider a hiker being able to carry up to fifty pounds and the hiker can select from eight items. The weight and benefit of these eight items is given in Table 2.1. The classical IP formulation of this knapsack problem is

$$\begin{aligned} \text{Maximize} \quad & 46x_1 + 3x_2 + 34x_3 + 50x_4 + 22x_5 + 1x_6 + 7x_7 + 16x_8 \\ \text{Subject to} \quad & 30x_1 + 29x_2 + 22x_3 + 16x_4 + 15x_5 + 7x_6 + 6x_7 + 5x_8 \leq 50 \\ & x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8 \in \{0,1\}. \end{aligned}$$

The branch and bound algorithm reports a solution of $x_3 = x_4 = x_7 = x_8 = 1$ with an objective function of 107. Thus, the hiker should pack items 3, 4, 7, and 8.

The knapsack problem is a common building block for many IPs. A variant of the knapsack problem is the multiple knapsack problem (MKP), which allows the hiker to consider

other constraints such as volume, budget, and safety restrictions. Thus, MKP has the form $\max c^T x$ subject to $Ax \leq b$, $x \in \{0,1\}^n$ where $A \in \mathbb{R}_+^n$ and $b \in \mathbb{R}_+^m$. Its feasible region is defined as P_{MKP}= $\{x \in \{0,1\}^n : Ax \leq b\}$ and the corresponding linear relaxation is P_{MKP}^{LR}. Chapter 1 illustrates many real world applications have been modeled as a KP or MKP.

Another variant of the KP is the knapsack equality problem (KEP), which requires the hiker to choose an exact knapsack weight for training purposes. The knapsack equality problem has the form $\max c^T x$ subject to $\sum_{j=1}^n a_j x_j = b$, $x \in \{0,1\}^n$, and b and $a_j \in \mathbb{R}_+$ for all $j = 1, ..., n$. The feasible space of a KEP can defined by $P_{KEP} = \{x \in \{0,1\}^n : \sum_{j=1}^n a_j x_j = b\}$ and the corresponding linear relaxation is P_{KEP}^{LR}.

Similar to the knapsack problem, the knapsack equality problem can be extended to allow multiple constraints known as the multiple knapsack equality problem (MKEP) allowing the hiker to consider other constraints such specifying an exact volume and budget. Thus, MKEP's formulation is $\max c^T x$ subject to $Ax = b$, $x \in \{0,1\}^n$ where $A \in \mathbb{R}_+^n$ and $b \in \mathbb{R}_+^m$. Its feasible region is defined as P_{MKEP}= $\{x \in \{0,1\}^n : Ax = b\}$ and the corresponding linear relaxation is P_{MKEP}^{LR}. KEP and MKEP have many useful applications [44, 45, 46, 47].

Chapter 1 briefly describes the concept of the knapsack demand problem (KDP). Suppose, the hiker wants to minimize the weight of the knapsack while still meeting the essentials for survival. The knapsack demand problem has the form $\min c^T x$ subject to $\sum_{j=1}^n a_j x_j \geq b$, $x \in \{0,1\}^n$, and b and $a_j \in \mathbb{R}_+$ for all $j = 1, ..., n$. The feasible space of a KDP can defined by $P_{KDP} = \{x \in \{0,1\}^n : \sum_{j=1}^n a_j x_j \geq b\}$ and the corresponding linear relaxation is P_{KDP}^{LR}.

KDP can be extended to allow multiple constraints known as the multiple knapsack demand problem (MKDP). This allows the hiker to consider other constraints such as volume,

budget, safety restrictions, and other hikers. Thus, MKDP $\min c^T x$ subject to $Ax \geq b$, $x \in \{0,1\}^n$ where $A \in \mathbb{R}_+^n$ and $b \in \mathbb{R}_+^m$. Its feasible region is defined as $P_{MKDP}= \{x \in \{0,1\}^n : Ax \geq b\}$ and the corresponding linear relaxation is P_{MKDP}^{LR}. MKDP also has useful applications [48, 49].

2.1.2 Polyhedral Theory

Polyhedral theory is a mathematical research area that helps describe the feasible and optimal solutions to LPs and IPs. By definition, a polyhedron is the intersection of finitely many half spaces $P = \{x \in \mathbb{R} : Ax \leq b\}$. A half space is the set of points that satisfy $\sum_{j=1}^{n} a_j x_j \leq b_i$. Since P^{LR} has a finite set of linear equalities, which are half spaces, it is obvious to see that P^{LR} is a polyhedron.

A set $S \subseteq \mathbb{R}^n$ is convex if and only if $\lambda x + (1-\lambda)x' \in S$ for all $x, x' \in S$ and $\lambda \in [0,1]$. The convex hull is the intersection of all convex sets containing S and is denoted by S^{CH}. A fundamental result of integer programming is that P^{CH} is convex and a polyhedron. Thus, every IP has two critical polyhedrons, P^{CH} and P^{LR}.

2.1.3 Cutting planes

One of the most common techniques to improve the solution time of IPs involves cutting planes [33, 34, 35, 38]. If every $x \in P$ satisfies $\sum_{j \in N} \alpha_j x_j \leq \beta$, then this inequality is valid for P^{CH}. Alternatively, let Q be the set of all points that do not satisfy $\sum_{j \in N} \alpha_j x_j \leq \beta$. If every $x \in Q$ is not in P, then the inequality is valid for P which is the contra-positive of the definition. This concept will be useful and discussed in more detail in Section 3.2. A

valid inequality is a cutting plane, if there exists an $x' \in P^{LR}$ such that $\sum_{j=1}^{n} a_j x'_j = b$ and $\sum_{j \in N} \alpha_j x'_j > \beta$. Similar definitions exist for P, P_{KP}, and P_{KDP}.

Every valid inequality invokes a face $F = \{x \in P^{CH} : \sum_{j \in N} \alpha_j x_j = \beta\}$, of P^{CH}. The dimension of F theoretically determines the usefulness of the inequality. The dimension of a convex space can be defined as the number of affinely independent points minus one. The points $v_1, ..., v_q$ are affinely independent if and only if the unique solution to $\sum_{i=1}^{q} \lambda_i v_0$, is $\lambda_i = 0$ for all $i = 1, ..., q$.

If a face has dimension one less than the dimension of P^{CH}, then it is a facet defining inequality. Theoretically, the strongest type of valid inequalities is called facet defining. Facet defining inequalities are not unique and can take on many forms. If one facet defining inequality is found for each facet, then $P^{CH} = P^{LR}$. Thus, the IP can be solved as a linear program. However, P^{CH} may have an exponential number of facet defining inequalities.

For the knapsack problem, some of the most widely used cutting planes involve cover cuts [39]. A set $C \subseteq N$ is a cover if $\sum_{j \in C} a_j > b$. A cover invokes a valid inequality of the form $\sum_{j \in C} x_j \leq |C| - 1$. A cover is a minimum cover if $\sum_{j \in C} a_j > b$ and $\sum_{j \in C \setminus k} a_j \leq b$ for each $k \in C$. A minimum cover invokes the same form of the inequality $\sum_{j \in C} x_j \leq |C| - 1$.

Recall the knapsack problem with the constraint $30x_1 + 29x_2 + 22x_3 + 16x_4 + 15x_5 + 7x_6 + 6x_7 + 5x_8 \leq 50$. Two examples of minimum covers for the knapsack constraint are $C = \{1, 2\}$ and $C = \{2, 3\}$. These two minimum covers invoke valid inequalities of the form $x_1 + x_2 \leq 1$ and $x_2 + x_3 \leq 1$, respectively. Two examples of non-minimum cover inequalities are $x_1 + x_2 + x_3 \leq 2$ and $x_2 + x_3 + x_4 \leq 3$. As illustrated, minimum covers invoke stronger valid inequalities than non-minimal covers.

An extension of a cover $E(C)$ uses the idea that swapping out an item in C for a heavier item results in a knapsack that is still too heavy to carry. Thus, $E(C) = C \cup \{j \in N : a_j \geq a_k$ for all $k \in C\}$. A cover's extension invokes a valid inequality of the form $\sum_{j \in E(C)} x_j \leq |C| - 1$. For example, the cover previously found $C = \{2, 3\}$ has an extended cover of $E(C) = \{1, 2, 3\}$, which invokes the valid inequality $x_1 + x_2 + x_3 \leq 1$. This extended cover inequality clearly dominates the cover inequality $x_2 + x_3 \leq 1$.

2.2 Lifting

Lifting is a common way to strengthen an inequality or make an invalid inequality valid [50, 51]. Let a valid cut have the form $\sum_{j \in N} \alpha_j x_j \leq \beta$. Lifting attempts to increase α_j and the dimension of the cutting plane. Increasing the dimension of the cutting planes can make the valid inequality facet defining.

Formally, lifting has a set $E \subseteq N$ and $K = (k_1, k_2, ..., k_{|E|}) \in \mathbb{Z}^{|E|}$, which defines a restricted feasible region, $P_{E,K} = \{x \in P : x_i = k_i$ for all $i \in E\}$. Given a valid inequality $\sum_{i \in E} \alpha_i x_i + \sum_{i \in N \setminus E} \alpha_i x_i \leq \beta$ of $P^{CH}_{E,K}$, a lifted valid inequality for P^{CH} is created with the form of $\sum_{i \in E} \alpha'_i x_i + \sum_{i \in N \setminus E} \alpha_i x_i \leq \beta'$.

Recall from Chapter 1, there are 12 classes of lifting depending on E, K, α, and β. These twelve classes are divided into three categories: up, middle or down lifting, sequential or simultaneous; and exact or approximate.

Uplifting is the most common type and occurs when $K = \{0\}$. Thus, each fixed variable is at its lower bound. Uplifting will not change the value of β. In contrast, down lifting

occurs when each fixed variable is at its upper bound. Down lifting typically changes the value of β. Middle lifting is a combination of both up and down lifting and may result in more than one inequality.

Sequential lifting occurs when $|E| = 1$. In sequential lifting, one variable is added at a time. To generate a single strong inequality many calls to a sequential lifting algorithm typically occur. Simultaneous lifting occurs when multiple variables are lifted in one algorithm, $|E| \geq 2$. Thus, a simultaneous lifting algorithm may only require one pass to create a strong inequality. Furthermore, sequentially lifted coefficients are typically integer, but simultaneous lifting may result in fractional coefficients.

Exact lifting requires the strongest possible coefficients for α and β. Thus, exact lifting typically leads to facet defining inequalities. Exact lifting algorithms require the exact solution to an optimization problem. Researchers use approximate lifting to avoid solving an optimization problem. Thus, the values of α and/or β could be strengthened and these types of inequalities are frequently not facet defining.

An exact sequential uplifting algorithm with binary variables will be explained to show the reader exact sequential uplifting. Thus, $|E| = 1$ and $K = \{0\}$. The variable selected for lifting is x_1. The algorithm takes a valid inequality, $\sum_{j \in N \setminus 1} \alpha_j x_j \leq \beta$, of the restricted space on $x_1 = 0$. The value of α_1 is determined by solving the IP of the following form:

$$
\begin{array}{lrcl}
\text{Maximize} & \sum_{j \in N \setminus \{E\}} \alpha_j x_j & & \\
\text{Subject to} & \sum_{j \in N} a_j x_j & \leq & b \\
& \sum_{j \in E} x_j & = & 1 \\
& x & \in & \{0,1\}^n
\end{array}
$$

If there is no solution, then $x_1 = 0$ for all $x \in P$ and x_1 should be removed from the problem. If not, the IP has an optimal solution value of z^* and $\alpha_1 = \beta - z^*$. Report the strengthened lifted inequality $\sum_{j \in N} \alpha_j x_j \leq \beta$ and terminate.

An example of this method is shown on the knapsack problem constraint $30x_1 + 29x_2 + 22x_3 + 16x_4 + 15x_5 + 7x_6 + 6x_7 + 5x_8 \leq 50$. Starting with the valid cover inequality $x_2 + x_3 + x_4 + x_5 \leq 3$, sequentially uplift x_1. Thus, begin by solving the following IP.

$$
\begin{array}{lrcl}
\text{Maximize} & x_2 + x_3 + x_4 + x_5 & & \\
\text{Subject to} & x_1 & = & 1 \\
& 30x_1 + 29x_2 + 22x_3 + 16x_4 + 15x_5 + 7x_6 + 6x_7 + 5x_8 & \leq & 50 \\
& x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8 & \in & \{0,1\}
\end{array}
$$

The optimal solution $z^* = 1$ so $\alpha_1 = 3 - 1 = 2$. Therefore, the new lifted inequality is $2x_1 + x_2 + x_3 + x_4 + x_5 \leq 3$. To lift x_6, one would change the IP's objective function to maximize $2x_1 + x_2 + x_3 + x_4 + x_5$ and set $x_6 = 1$. The optimal solution is 2 so $\alpha_6 = 3 - 2 = 1$ and the new inequality is now $2x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \leq 3$. If the process is repeated, the results illustrate that the coefficients of x_7 and x_8 are 0. Therefore, the algorithm terminates

and reports the final lifted inequality cut of $2x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \leq 3$. In this case, the inequality is not a facet defining, primarily due to the fact that $\{2, 3, 4, 5\}$ was not a minimal cover.

2.3 Equality Cuts

Recently, DeLissa discovered a new paradigm for cutting planes called equality cuts. Formally, if every $x \in P$ satisfies $\sum_{j \in N} \alpha_j x_j = \beta$, then this equality is valid for P^{CH}. A valid equality is a cutting hyperplane if there exists an $x' \in P^{LR}$ such that $\sum_{j \in N} \alpha_j x'_j \neq \beta$.

DeLissa proved the existence of equality cuts using anticovers. Given a constraint from a KDP of the form $\sum_{j \in N} a_j x_j \geq b$, a set $AC \subseteq N$ is an anticover of a KP constraint if $\sum_{j \in N \setminus AC} a_j < b$. If $x_j = 0$ for all $j \in AC$, then the KDP constraint can never be satisfied. Thus, anticovers also invoke valid inequalities. If a cover invokes a valid inequality $\sum_{j \in C} x_j \leq \beta$ and the anticover invokes a valid inequality $\sum_{j \in C} x_j \geq \beta$, then $\sum_{j \in C} x_j = \beta$ is a valid equality.

DeLissa showed several interesting and unsurprising facts. Valid equalities exist if and only if P^{CH} has dimension strictly less than n. Unlike a valid inequality, a valid equality is improper, because every point in P^{CH} satisfies the equality. Thus, it can never be facet defining. However, if a valid equality is an equality cut, then including this equality cut to the linear relaxation space reduces it dimension by at least one.

Surprisingly, DeLissa has found that valid equalities can be computationally advantageous. Furthermore, he demonstrated that over half a million equality cuts were found

and reduced the solution time by 25% for some classes of IPs. Thus, equality cuts can be computationally useful.

To help demonstrate equality cuts, consider the knapsack problem from Section 2.2 as a KEP $30x_1 + 29x_2 + 22x_3 + 16x_4 + 15x_5 + 7x_6 + 6x_7 + 5x_8 = 50$. An extended cover is $E(C) = \{1, 2, 3\}$, which invokes a valid inequality $x_1 + x_2 + x_3 \leq 1$. According to DeLissa, the extension's anticover invokes a valid inequality of the form $x_1 + x_2 + x_3 \geq 1$ for the KDP. The reader can easily confirm $x_1 + x_2 + x_3 \geq 1$ is a valid inequality. Since $x_1 + x_2 + x_3 \leq 1$ is valid for KP and $x_1 + x_2 + x_3 \geq 1$ is valid for the KDP, $x_1 + x_2 + x_3 = 1$ must be a valid equality for KEP.

In summary, this chapter has covered fundamental topics in integer programming such as the knapsack and related problems, polyhedral theory, and cutting planes. In addition, this chapter briefly discussed advanced topics such as lifting and equality cuts. From the knowledge presented, the reader knows that lifting strengthens inequalities, DeLissa's cuts have binary coefficients, and these cuts are powerful enough to reduce the solving time. This research shows that lifted equality cuts exists and how easily they can be found. These advancements are presented in the next chapter.

Chapter 3

Lifted Equality Cuts

This chapter extends the knowledge of equality cuts by researching the implementation of lifting to valid equality cuts. On a high level, this chapter provides the existence of lifted valid equality cuts including definitions, theorems, and examples. In addition, an algorithm that identifies lifted equality cuts, ALEC, is presented along with its characteristics such as the theoretical run time. Lastly, this chapter demonstrates how lifted equality cuts can reduce the dimension of the linear relaxation's space by at least one.

3.1 Existence of Lifted Equality Cuts

DeLissa illustrated that developing equality cuts not only exist, but can be found using a $O(n\ log(n))$ algorithm. In a small computational study, DeLissa found over half a million cuts on average for small instances with less than 90 nonzero coefficients. These cuts were also powerful enough that they reduced the solving time by 25% on average. However,

DeLissa's cuts were limited to binary coefficients for α_j. An intuitive question is, can these equality cuts have α_j strictly greater than one? Increasing α_j is often accomplished with lifting, which strengthens valid inequalities. Therefore, the research motivation is to obtain the conditions for which lifted valid equalities exist.

A lifted equality cut has the form $\sum_{j=1}^{n} \alpha_j x_j = b$ where one $\alpha_j > 1$. Recall, a non-lifted equality cut can be defined using a cover cut from KP and an anticover from KDP. Specifically, if a valid inequality exists in the form of $\sum_{j=1}^{n} \alpha_j x_j \leq \beta$ and its corresponding valid inequality exists in the form of $\sum_{j=1}^{n} \alpha_j x_j \geq \beta$, then $\sum_{j=1}^{n} \alpha_j x_j = \beta$ must be valid.

To formally define a lifted equality constraint, a valid inequality for the associated KDP is introduced. Prior to this result, define ψ_j as the origin translated one unit in the j^{th} dimension.

Theorem 3.1.1 *Given a sorted knapsack demand constraint,* $\sum_{j\in N} a_j x_j \geq b$, *a* β, $p \in \{1, ..., n\}$ *with* $\beta \leq p$ *and* $\sum_{j=p+1}^{n} a_j < b$, *then* $\sum_{j=1}^{p+\beta-1} \alpha_j x_j \geq \beta$ *is a valid inequality if and only if for every* $E \subseteq \{1, ..., p+\beta-1\}$ *such that* $\sum_{j\in E} a_j + \sum_{j=p+\beta}^{n} a_j \geq b$, *which implies* $\sum_{j\in E} \alpha_j \geq \beta$.

Proof: Assume a sorted knapsack demand constraint, $\sum_{j\in N} a_j x_j \geq b$, a β, p with $\beta \leq p$ and $\sum_{j=p+\beta-1}^{n} a_j < b$. For the first direction, assume $\sum_{j=1}^{p+\beta-1} \alpha_j x_j \geq \beta$ is a valid inequality. For contradiction, assume that there exists an $E \subseteq \{1, ..., p+\beta-1\}$ such that $\sum_{j\in E} a_j + \sum_{j=p+\beta}^{n} a_j \geq b$ and $\sum_{j\in E} \alpha_j < \beta$. The point $x' = \sum_{j\in E} \psi_j + \sum_{j=p+\beta}^{n} \psi_j$ is feasible. However, $\sum_{j=1}^{p+\beta-1} \alpha_j x'_j < \beta$, which contradicts the inequality being a valid inequality.

Conversely, assume that every $E \subseteq \{1, ..., p+\beta-1\}$ such that $\sum_{j\in E} a_j + \sum_{j=p+\beta-1}^{n} a_j \geq b$ has $\sum_{j\in E} \alpha_j \geq \beta$. For contradiction, assume $\sum_{j=1}^{p+\beta-1} \alpha_j x_j \geq \beta$ is not a valid inequality. Then

there exists an $x' \in P^{KDP}$ such that $\sum_{j=1}^{p+\beta-1} \alpha_j x'_j < \beta$. Let $E = \{j \in \{1, ..., p+\beta-1\} : x'_j = 1\}$. Consequently, $\sum_{j \in E} a_j + \sum_{j=p+1}^{n} a_i \geq b$, which contradicts $\sum_{j \in E} \alpha_j < \beta$.

□

Theorem 3.1.1 establishes the validity conditions for a lifted inequality in the form of $\sum_{j=1}^{n} \alpha_j x_j \geq \beta$. To convert this lifted valid inequality to an equality, its associated inequality, $\sum_{j=1}^{n} \alpha_j x_j \leq \beta$, must be valid. These conditions are established in the next theorem.

Theorem 3.1.2 *Given a knapsack equality constraint $\sum_{j \in N} a_j x_j = b$, a β, $p \in \{1, ..., n\}$ with $\beta \leq p$, $\sum_{j=p+1}^{n} a_j < b$ and an α that satisfies the conditions of Theorem 3.1.1, if for every $S \subseteq \{1, ..., p\}$ such that $\sum_{j \in S} \alpha_j > \beta$ implies $\sum_{j \in S} a_j > b$, then $\sum_{j \in N} \alpha_j x_j = \beta$ is a valid equality of P^{CH}_{KEP}.*

Proof: Given a knapsack equality constraint $\sum_{j \in N} a_j x_j = b$, a β, $p \in \{1, ..., n\}$ with $\beta \leq p$, $\sum_{j=p+1}^{n} a_j < b$ and an α that satisfies the conditions of Theorem 3.1.1. Assume that every $S \subseteq \{1, ..., p\}$ such that $\sum_{j \in S} \alpha_j > \beta$ implies $\sum_{j \in S} a_j > b$. For contradiction, assume that $\sum_{j \in N} \alpha_j x_j = \beta$ is not a valid equality of P^{CH}_{KEP}. Thus, there exists an $x' \in P_{KEP}$ such that $\sum_{j \in N} \alpha_j x'_j \neq \beta$. Since α satisfies Theorem 1, $\sum_{j \in N} \alpha_j x'_j > \beta$. Define $S = \{j \in N : x'_j = 1$ and $j \leq p\}$. Thus, $\sum_{j \in S} \alpha_j > \beta$, which implies $\sum_{j \in S} a_j > b$, which contradicts x' being a feasible point and the result follows.

□

As mentioned, if Theorem 3.1.1 is satisfied, then there exists a valid inequality with the form $\sum_{j=1}^{n} \alpha_j x_j \geq \beta$. Theorem 3.1.2 checks if this inequality has an associated valid inequality with the form $\sum_{j=1}^{n} \alpha_j x_j \leq \beta$. If Theorem 3.1.1 and 3.1.2 are satisfied, then a

lifted valid equality exists with the form $\sum_{j=1}^{n} \alpha_j x_j = \beta$. Theorem 3.1.1 and 3.1.2 were used as a foundation to build an algorithm that can quickly identify valid lifted equalities.

3.2 Algorithm of Lifted Equality Cuts

As stated, Theorems 3.1.1 and 3.1.2 give the conditions for which a valid lifted equality exists. The reader will see that ALEC has two subroutines, one for each theorem. The first subroutine satisfies the conditions of Theorem 3.1.1 to produce a valid inequality of the form $\sum_{j=1}^{n} \alpha_j x_j \geq \beta$. Given this valid inequality, the second subroutine attempts determines if the conditions of Theorem 3.1.2 are satisfied. If there exists an associated inequality, then ALEC reports a lifted valid equality $\sum_{j=1}^{n} \alpha_j x_j = \beta$. ALEC uses the parameters in the KEP constraint to rapidly produce lifted equalities that satisfy the previous two theorems, and may report multiple valid equalities.

The input to ALEC is a KEP constraint. The *Initialization* of ALEC starts by sorting the constraint's coefficients, a_j, in descending order. ALEC then calculates the sorted cumulative sum, *csum*, starting from the smallest coefficient a_n. Once *csum* is calculated, the starting index of the first candidate equality cut is found, denoted as p. ALEC calculates p by initially assigning it a value of n and assigns $\beta = 1$. ALEC decreases p by one until the cumulative sum is greater than or equal to the right hand side. ALEC uses p as the last index with a nonzero coefficient for potential lifted valid equalities.

The main step of ALEC is straightforward. While $\beta \leq 4$ and p is less than n, assign lifted coefficients α_j to create a lifted inequality of the form $\sum_{j=1}^{n} \alpha_j x_j \geq \beta$ according to the subroutine $AssignGreaterCoefficients$. ALEC uses sequential approximate uplifting

to avoid solving an IP. Note, $AssignGreaterCoefficients$ always produces valid lifted cuts by having $AssignGreaterCoefficients$ restrict $p \geq \beta$. This is explained in more detail in the descriptions of the subroutines.

After the α coefficients have been assigned, ALEC checks validity from the less than or equal to side in the lifted inequality using the subroutine $CheckLessCoefficients$. If the potential valid equality is valid, then the subroutine reports 2 and assigns 2 to $valid$. If $valid = 2$, then ALEC reports a valid lifted equality and increases β by one to keep searching for more equalities. If $CheckLessCoefficients$ reports $valid = 1$, then it implies that $\sum_{j=1}^{n} \alpha_j x_j \geq \beta$ does not have an associated valid less than or equal to inequality. Therefore, p is increased to create a different set of indices in hopes of finding a valid lifted equality. Increasing p will decrease α_j because $csum_{p+1}$ will decrease, which might create a valid less than or equal to inequality. If $valid$ is any other value, then a lifted cover cut never existed and therefore β should increase . For demonstration purposes, β's upper limit is set at four, but this can easily be modified to larger values of β. This idea is discussed in more detail at the end of this chapter.

Algorithm of Lifted Equality Cuts (ALEC)

Initialization

$a \leftarrow a$ sorted in descending order
$csum_n \leftarrow a_n$
For $i = 1$ to $n - 1$
    $csum_{n-i} \leftarrow csum_{n-i+1} + a_{n-i}$
$p \leftarrow n$, $\beta \leftarrow 1$
While $csum_p < b$ Do
    $p \leftarrow p - 1$
$firstp \leftarrow p$

Main Step

While $\beta \leq 4$ and $p \leq n$ Do
  $\alpha \leftarrow$ AssignGreaterCoefficients$(\beta, p, csum, a)$
  $valid \leftarrow$ CheckLessCoefficients(β, p, α)
  If $valid = 0$, Then
    $\beta \leftarrow \beta + 1$
  If $valid = 1$, Then
    $p \leftarrow p + 1$
  If $valid = 2$, Then
    Report $\sum_{i=1}^{n} \alpha_i x_i = \beta$ as a valid equality
    $\beta \leftarrow \beta + 1$
    If $p < firstp + \beta - 1$, Then
      $p \leftarrow firstp + \beta - 1$

The subroutine $AssignGreaterCoefficients$ calculates the lifted coefficients α_j for the indices set up by the $Initialization$ and the $MainStep$. The lifted coefficients primarily depend on β. If $\beta = 1$, then all the coefficients α_j in the indices are set to 1. Thus, if $\beta = 1$, DeLissa's anticover cuts are produced. If $\beta = 2$ and if any coefficient a_j plus the cumulative sum outside the indices, $csum_{p+1}$ is greater than or equal to the right hand side, then $\alpha_j = \beta = 2$. When $\beta = 3$, the previous condition is checked and sets $\alpha_j = \beta = 3$. In addition, if two coefficients, $a_j + a_{j+1}$, plus the $csum_{p+1}$ is greater than the right hand side b, then $\alpha_j = \beta - 1 = 2$. This same pattern repeats for all $\beta \leq 4$.

The combination of this technique with restricting $p \geq \beta$ allows $AssignGreaterCoefficients$ to always produced valid lifted inequalities. The idea behind it uses the alternative definition for a valid inequality. Chapter 2 defines an inequality to be valid if and only if every $x \in Q$ is not in P, where Q is the set of all points that do not satisfy the lifted inequality $\sum_{j \in N} \alpha_j x_j \geq \beta$. Since Q can have many permutations of x, one would think the algorithm would be inefficient at finding lifted, valid, and greater than or equal to inequalities.

However, ALEC does not need to check every element in Q due to the idea of the strongest index for some integer q. Formally, for a given β and a $q \leq \beta - 1$, define the strongest index for q to be the maximum r such that $\sum_{j=r}^{r+q-1} a_j + csum_{p+1} \geq b$. Due to the sorted nature of a, once a strongest index is known, q must increase and r will never decrease as q increases. This fact enables ALEC to efficiently determine the lifting coefficients.

To express the idea of the strongest index, consider the following constraint from the knapsack demand problem, $30x_1 + 29x_2 + 22x_3 + 16x_4 + 15x_5 + 7x_6 + 6x_7 + 5x_8 \geq 50$. When $p = 6$ and $\beta = 4$, the subroutine considers an inequality of the form $\alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_4 + \alpha_5 x_5 + \alpha_6 x_6 \geq 4$ with $\alpha_j \geq 1$ for all $j = 1, ..., 6$. Since $\beta = 4$ and $p = 6$, $csum_7 = a_7 + a_8 = 11$ and q can be 1, 2, or $3 = \beta - 1$ and there are three strongest indices that are calculated.

When $q = 1$, $a_1 + csum_7 = 30 + 11 < 50$. Since a_j is sorted, $a_1 \geq a_2 \geq a_3 \geq ... \geq a_6$. Therefore, there does not exist a strongest coefficient for $q = 1$. In other words, setting exactly one variable from x_1 to x_6 to a value of one can never result in a feasible point of the associated demand constraint.

If $q = 2$, $a_1 + a_2 + csum_7 = 30 + 29 + 11 > 50$, which implies a strongest index exists. Observe that $a_2 + a_3 + csum_{p+1} = 29 + 22 + 11 \geq 50$ and $a_3 + a_4 + csum_7 = 22 + 16 + 11 < 50$. Thus, the strongest index for $q = 2$ is 2. Thus, the coefficients of $\alpha_1 \geq 3 = \beta - q + 1$ and $\alpha_2 \geq 3$ or the inequality is not valid due to $(0, 1, 1, 0, 0, 0, 1, 1)$ being feasible for the associated demand constraint.

For $q = 3$, $a_3 + a_4 + a_5 + csum_7 \geq 50$ and $a_4 + a_5 + a_6 + csum_7 < 50$, so its strongest index is 3. Thus, the coefficients of $\alpha_1 \geq 2 = \beta - q + 1$, $\alpha_2 \geq 2$ and $\alpha_3 \geq 2$ or the inequality

is not valid due to $(0, 0, 1, 1, 1, 0, 1, 1)$ being feasible for the associated demand constraint.

The subroutine $AssignGreaterCoefficients$ incorporates the strongest index to lift α_j. Since this approach is an approximation of uplifting, ALEC will always choose the maximum α_j. For example, the previous paragraphs calculated $\alpha_1 \geq 3$ and $\alpha_1 \geq 2$. ALEC assigns $\alpha_1 = 3$, since it is the maximum. This concept allows ALEC to produce valid greater than or equal to inequalities quickly.

AssignGreaterCoefficients(β, p, $csum$, a, b)

For $i = p + 1$ to n
  $\alpha_i \leftarrow 0$
If $\beta = 1$
  For $j = 1$ to p
    $\alpha_j \leftarrow 1$
If $\beta = 2$, Then
  For $j = 1$ to p
    If $a_j + csum_p \geq b$, Then
      $\alpha_j \leftarrow 2$
If $\beta = 3$, Then
  For $j = 1$ to p
    If $a_j + csum_{p+1} \geq b$, Then
      $\alpha_j \leftarrow 3$
    If $a_j + a_{j+1} + csum_{p+1} \geq b$, Then
      $\alpha_j \leftarrow 2$
If $\beta = 4$, Then
  For $j = 1$ to $p + 1$
    If $a_j + csum_{p+1} \geq b$, Then
      $\alpha_j \leftarrow 4$
    If $a_j + a_{j+1} + csum_{p+1} \geq b$, Then
      $\alpha_j \leftarrow 3$
    If $a_j + a_{j+1} + a_{j+2} + csum_{p+1} \geq b$, Then
      $\alpha_j \leftarrow 2$
return α

Since $AssignGreaterCoefficients$ automatically produces valid, lifted, and greater than or equal to inequalities, the only component left is checking if it has an associated valid, lifted, and less than or equal to inequality, which is verified in the subroutine $CheckLessCoefficients$. To begin, the subroutine first checks if $\sum_{j=p-\beta+1}^{p} a_j > b$, or $\{p-\beta+1, ..., p\}$ is a cover. If it does not, then $valid = 0$ implying a lifted equality can never exist with the specified β. Therefore, β increases by one.

If $\beta = 1$, then $CheckLessCoefficients$ checks the strongest index. In this case, the condition is if $a_{p-1} + a_p > b$, then the $\sum_{j=1}^{n} \alpha_j x_j \geq \beta$ is not valid from the less than or equal to side and returns $valid = 1$. When $valid = 1$, then the $\sum_{j=1}^{n} \alpha_j x_j \geq \beta$ still might have an associated less than or equal to inequality, which is why $MainStep$ will increase p and not β. As β increases, the number of strongest indexes also increases requiring ALEC to check more cases for validity. This process is similarly repeated for all β. If $\sum_{j=1}^{n} \alpha_j x_j \geq \beta$ does have an associated inequality $\sum_{j=1}^{n} \alpha_j x_j \leq \beta$, then $valid = 2$ implying there exists a lifted valid equality $\sum_{j=1}^{n} \alpha_j x_j = \beta$. Therefore, a lifted valid equality is reported.

CheckLessCoefficients(β, p, α, $csum$, a, b)

$sum \leftarrow 0$
For $i = 0$ to β
    $sum \leftarrow sum + a_{p-i}$
If $sum \leq b$, Then
    return(0)
If $\beta = 1$ And $a_{p-1} + a_p > b$, Then
    return(2)
If $\beta = 2$, Then
    For $j = 1$ to p
        If $\alpha_j = 2$ And $a_p + a_j \leq b$, Then
            return(1)

If $\beta = 3$, Then
For $j = 1$ to p
If $\alpha_j = 3$ And $a_p + a_j \leq b$, Then
return(1)
If $\alpha_j = 2$ And $\alpha_{j+1} = 1$ And $a_j + a_{p-1} + a_p \leq b$, Then
return(1)
If $\alpha_j = 2$ And $\alpha_{j+1} = 2$ And $a_j + a_{j+1} \leq b$, Then
return(1)
If $\beta = 4$, Then
For $j = 1$ to p
If $\alpha_j = 4$ And $a_p + a_j \leq b$, Then
return(1)
If $\alpha_j = 3$ And $\alpha_{j+1} = 2$ And $a_j + a_{j+1} \leq b$, Then
return(1)
If $\alpha_j = 3$ And $\alpha_p = 1$ And $\alpha_{p-1} = 1$, Then
If $a_j + a_p + a_{p-1} \leq b$
return(1)
If $\alpha_j = 2$ And $\alpha_{j+1} = 2$ And $\alpha_p = 1$ And $a_j + a_{j+1} + a_p \leq b$, Then
return(1)
If $\alpha_j = 2$ And $\alpha_p = 1$ And $\alpha_{p-1} = 1$ And $\alpha_{p-2} = 1$
And $a_j + a_p + a_{p-1} + a_{p-2} \leq b$, Then
return(1)
return(2)

The argument that ALEC produces a valid equality is straightforward. Due to the sorted order of the coefficients, the reader can easily verify that the subroutine $AssignGreaterCoefficients$ is an application of Theorem 3.1.1 and $CheckLessCoefficients$ is an application of Theorem 3.1.2. Thus, any equality returned from ALEC is a valid lifted equality for KEP model.

To show that ALEC is a polynomial time algorithm when $\beta \leq 4$, the $Initialization$ sorts a, which requires $O(nlog(n))$ by merge sort. ALEC calculates $csum$ from $\{a_n, a_{n-1}, ..., a_1\}$ to find p, which requires $O(n)$ effort. Thus, the Initialization requires $O(nlog(n))$ effort.

The Main Step repeatedly calls two subroutines. $AssignGreaterCoefficients$ subroutine is called and its steps are dependent upon β. The initial assignments of α require $O(n)$. The number of conditions checked is $\beta - 1$ and each of these conditions is verifiable in $O(1)$, because $\beta \leq 4$. Thus, this subroutine requires $O(n)$ effort.

Next, the subroutine $CheckLessCoefficients$, with the given α, is applied to $a_1, a_2, .., a_p$ where it checks conditions to find an associated less than or equal to inequality. The number of cases to check depends on β and since $\beta \leq 4$, the checks are accomplished in $O(1)$. Since p is bounded by $O(n)$, $CheckLessCoefficients$ requires $O(n)$ effort.

In each iteration, the $MainStep$ increases either β or p. Since β is bounded by 4 and p is bounded by n, there are at most $O(n)$ iterations within Main Step. From the above arguments, each iteration requires at most $O(n)$ effort. Thus, the Main Step requires $O(n^2)$ effort. Combining this fact with the analysis of the $Initialization$ results in ALEC being a polynomial time algorithm that runs in quadratic time.

One may question the impact of increasing β. Fundamentally, as β increases, the cases increase and the time to check a condition may also increase. In addition, the contribution of additional equality cuts may be unnecessary due to generating non-independent equality cuts. More discussions of this are included in the remainder of this thesis and it is left as a future research topic.

Additionally, one may question why ALEC increases p. Let p' be the initial p in $\beta = 1$ and p'' be the p in $\beta = 2$. This implies that $p'' \geq p' + 1$. This phenomenon happens because of two properties of ALEC where $\sum_{j=p'}^{n} a_j \geq b$ and $\sum_{j=p'+1}^{n} a_j < b$. Therefore, $a_{p''} + \sum_{j=p''+1}^{n} a_j < b$ implying $a_{p''}$ will never be lifted. Since a_j is sorted for all j then $a''_p \geq a_{p''+1}$. Thus, only

$(\alpha_1, \alpha_2.., \alpha_{p'})$ will be lifted. As β increases so does p illustrating that for any $\beta \geq 2$, lifting will only occur in $(\alpha_1, \alpha_2.., \alpha_{p'})$. This is a unique and useful property of ALEC especially if used in practical applications. Lifting only through p' may decrease the practical run time of ALEC.

To further help the reader's understanding of lifted equalities and ALEC, consider applying this algorithm to the following sorted KEP constraint from Chapter 2.

$$30x_1 + 29x_2 + 22x_3 + 16x_4 + 15x_5 + 7x_6 + 6x_7 + 5x_8 = 50$$

ALEC's first step finds the cumulative sum of the coefficients starting from the variable with the least coefficient. The result produces $csum = [130, 100, 71, 49, 33, 18, 11, 5]$. The sorted cumulative sum first exceeds the right hand side 50 at $csum_3$ because $csum_4 = 49$ and $csum_3 = 71$. Therefore, $p = 3$. Next, ALEC assigns $\beta = 1$ and enters the $MainStep$. The subroutine $AssignGreaterCoefficients$ is then called which assigns $\alpha_j = 1$ for all $j = 1, ..., p$. Therefore, $x_1 + x_2 + x_3 \geq 1$ is a valid inequality. This valid inequality is used as input to the next subroutine $CheckLessCoefficients$. In this case, conditions for $\beta = 1$ are checked. According to the subroutine, if $a_{p-1} + a_p > b$, then return 2. Since $a_2 + a_3 = 51 > 50$, $CheckLessCoefficients$ returns 2. In the $MainStep$, 2 is assigned to $valid$ so ALEC reports the valid equality $x_1 + x_2 + x_3 = 1$. Additionally, β increases by one and becomes 2. Furthermore, since $p = 3$, $firstp = 3$, and $\beta = 2$, the if condition is satisfied. So p increases to $firstp + \beta - 1 = 4$.

The process repeats and enters $AssignGreaterCoefficients$ subroutine. Because $\beta > 1$, the algorithm has to adjust the coefficients for validity. $AssignGreaterCoefficients$ checks

$a_1 + csum_5$ for α_1, which sums to 63, which is greater than the right hand side 50. Similarly, ALEC checks $a_2 + csum_5$ for α_2 which sums to 62, which is also greater than 50. ALEC then checks $a_3 + csum_5$ for α_3. This sums to 56, which is still greater than 50. Thus, $\alpha_1 = \alpha_2 = \alpha_3 = 2$. Since $a_4 + csum_5$ is less than 50, ALEC assigns $\alpha_4 = 1$. The remaining coefficients $(\alpha_{p+1}, \alpha_{p+2}, ..., \alpha_n)$ are set to 0. The first subroutine produces a valid lifted inequality $2x_1 + 2x_2 + 2x_3 + x_4 \geq 2$.

ALEC then attempts to find an associated valid inequality $2x_1 + 2x_2 + 2x_3 + x_4 \leq 2$ using $CheckLessCoefficients$. Since $a_3 + a_4 = 48 \leq 50$, and $\alpha_3 + \alpha_4 = 3 > 2$, $S = \{3, 4\}$ violates Theorem 3.1.2. Thus, there is not an associated equality cut. Therefore, $CheckLessCoefficients$ reports 1 which is assigned to valid. In the $MainStep$, ALEC increases p to 5 and repeats the process.

Since $\beta = 2$, $AssignGreaterCoefficients$ checks $a_j + csum_6$ for all $j = 1, ..., p$. For $j = 1$, the subroutine checks $a_1 + csum_6$, which sums to 48. Thus, $\alpha_1 = 1$. Since a_j is sorted, $\alpha_2, \alpha_3, .., \alpha_p$ will also equal 1. Thus, ALEC generates $x_1 + x_2 + x_3 + x_4 + x_5 \geq 2$ as a valid inequality. This inequality passes through the next subroutine $CheckLessCoefficients$ where it checks if a cover exists. Because $a_3 + a_4 + a_5 > 50$, a cover does exist so $x_1 + x_2 + x_3 + x_4 + x_5 = 2$ is a valid lifted equality.

Next, ALEC increases $\beta = 3$. In this case, $AssignGreaterCoefficients$ checks whether or not $a_j + csum_6 \geq 50$ for all $j = 1, ..., p$. This is not true for any j and thus, no $\alpha_j = \beta$. To test whether or not an $\alpha_j = \beta - 1$, examine $a_j + a_{j+1} + csum_6 \geq 50$ for all $j = 1, ..., p - 1$. This statement is true when $j = 1, 2$, and 3. Thus, $2x_1 + 2x_2 + 2x_3 + x_4 + x_5 \geq 3$ is a valid lifted inequality.

Since $\beta = 3$, three subsets are evaluated in $CheckLessCoefficients$. The first is the existence of the cover $\{2, 3, 4, 5\}$, the second and third occur when $S = \{2, 3\}$ and $S = \{3, 4, 5\}$. Since $\sum_{j \in S} a_j$ equals 82, 51 and 53, respectively, $CheckLessCoefficients$ reports 2. Thus, $valid = 2$ and ALEC reports $2x_1 + 2x_2 + 2x_3 + x_4 + x_5 = 3$ as a valid lifted equality.

Both β and p increase to 4 and 6, respectively. Due to the conditions of the previous paragraph, no α has a coefficient of β. To test $\alpha_j = \beta - 1$, check $a_j + a_{j+1} + csum_7 \geq 50$ for all $j = 1, ..., p-1$. This is true for $j = 1$ and 2 so $\alpha_1 = \alpha_2 = 3$. Now $a_j + a_{j+1} + a_{j+2} + csum_7 \geq 50$ for all $j = 1, ..., p-2$. This is true when $j = 3$ and so $\alpha_3 = \beta - 2 = 2$. Thus, $3x_1 + 3x_2 + 2x_3 + x_4 + x_5 + x_6 \geq 4$ is a valid lifted inequality.

To test for an equality constraint, five subsets must be evaluated. The necessary cover is satisfied from the previous case. The remaining sets are $\{1, 2\}$, $\{2, 3\}$, $\{2, 5, 6\}$ and $\{3, 4, 5, 6\}$. Since $\sum_{j \in S} a_j$ equals 59, 51, 51, and 60, respectively. Thus $3x_1 + 3x_2 + 2x_3 + x_4 + x_5 + x_6 = 4$ is a valid equality. Next β increases to 5 and ALEC terminates. Recall, ALEC can easily be modified for additional values of β.

To prove $x_1 + x_2 + x_3 = 1$ is an equality cut, consider the following two feasible linear relaxation points $(0, 0.1, 0.8, 1, 1,\ 0, .15, 1)$ and $(0, 0.1, 1, 0, 0.9, 0.8, 0.167, 1)$. Both feasible linear relaxation points are removed from the linear relaxation space because $0.9 < 1$ and $1.1 > 1$. Since one point is strictly less than and the other is strictly greater than β, the equality cut passes through the center of the linear relaxation space.

Since there exists at least one linear relaxation point that violates $x_1 + x_2 + x_3 = 1$, adding this equality cut reduces the dimension of the linear relaxation space by at least one. To see this, observe that P_{KEP}^{LR} contains the linear relaxation point. Once the equality cut is added

to P_{KEP}^{LR}, every point satisfies this equality and thus no affine combination of points could ever attain the linear relaxation point. Thus, the number of affinely independent points has decreased by at least one. This is the same argument as formalized in a theorem by DeLissa.

The same conclusion can be made for $x_1 + x_2 + x_3 + x_4 + x_5 = 2$, which cuts the linear relaxation points $(0, 0.625, 0.375, 0, 0.375, 1, 1, 1)$ and $(0, 0, 1, 0.5, 1, 0.5, 0, 0.3)$. Both feasible linear relaxation points are removed from the linear relaxation space because $1.375 < 2$ and $2.5 > 2$. Another lifted valid equality cut is $3x_1 + 3x_2 + 2x_3 + x_4 + x_5 + x_6 = 4$, which cuts the linear relaxation points $(0.2, 0.2, 0.6, 0.1, 0.9, 0, 0.9, 0.9)$ and $(0.5, 0.2, 0.3, 0.5, 0.5, 1, 0, 0.02)$. Both feasible linear relaxation points are removed from the linear relaxation space because $3.4 < 4$ and $4.7 > 4$.

There is a key difference between lifted valid equality cuts and cutting planes. Cutting planes in general are inequalities that eliminate a linear relaxation point. With inequalities, only a portion of P_{KEP}^{LR} is eliminated from one side. However, lifted valid equality cuts eliminate portions of P_{KEP}^{LR} from both sides. Thus, P_{KEP}^{LR} with the equality cut is much smaller than applying a general cutting plane.

ALEC may produce linearly dependent cuts, which is additional work for no benefit. For instance the sum of the first and second equality cuts equals the third. Thus, any two of these cuts is sufficient to represent the other cut and any one of them can be removed from the model. This is one contributing reason why $\beta \leq 4$. However, the fourth cut is not linearly dependent of the first and second equality. Consequently, lifted equality cuts do exist and are not always combinations of other existing equality cuts.

The computational results will demonstrate how fast ALEC is and it is believed that

checking for linear independence would slow ALEC sufficiently as little improvement would be gained by knowing whether or not the equalities are linearly independent. If ALEC was modified to always increase p and β simultaneously, then it is believed that fewer linearly dependent constraints would be generated. Since ALEC runs in quadratic time, the cost of losing potential lifted equalities exceeds the benefit of guaranteeing linearly independent cuts.

To further examine the effects of ALEC's lifted and non-lifted equality cuts, the KP model from Chapter 2 was converted into a KEP model with an objective function of $z = \sum_{j=1}^{n} x_j$. This objective function is common in IP formulations. The model was solved using branch and bound with depth first left as a branching rule. The branch and bound tree required 61 nodes to solve. When including the four equality cuts, the branch and bound tree required three nodes to solve. Thus, equality cuts are anecdotally beneficial.

The results provide motivation to test the effects of valid lifted equalities on larger MKEP instances. Thus, the next chapter conducts a small computational study to see the impact of lifted equality cuts in a branch and bound environment on larger size MKEPs.

Chapter 4

Computational Studies

This chapter provides a computational study on valid lifted equalities. Specifically, ALEC is implemented into random MKEP instances that vary in size of variables and constraints. For each instance, the associated IP was solved in CPLEX [52], a commercial optimization software, with and without ALEC. The results show ALEC's capability to quickly find an enormous number of valid lifted equalities while still reducing the solution time.

4.1 Instances and Implementation

Although there exists many classes of benchmark problems, no MKEP instances exist on the OR Library [53]. However, the OR Library does have MKP benchmark instances. If one converted these MKP instances to MKEP instances, the problems become computationally intractable. Consequently, the first step was to determine MKEP random instances that were neither too difficult nor too easy to solve.

To create random MKEP instances, the format for the benchmark MKP instances was followed. Thus, $a_{i,j}$ are uniformly random distributed integers between 1 and 1,000, $b_i = \frac{1}{2}\sum_{j\in N} a_{i,j}$ and $c_j = u + \frac{\sum_{i\in M} a_{i,j}}{|M|}$ where u is a uniformly distributed random integer between 1 and 500 and M is the set of rows.

Recall, that MKEP is a $\mathcal{NP}$-hard problem. In practice, MKEP instances are either practically fast to solve or have an exponential number of nodes in the branch and bound tree that can require an unrealistic time to solve. This is because MKEP instances have a high chance of being infeasible due to their equality constraints. If an IP is infeasible, then branch and bound can never perform a bound or integer fathom and the only fathoming rule is an infeasible linear relaxation. Through computational experiments, not presented here, it was determined that problems ranging from 30 to 100 variables and 2 to 4 constraints were neither trivial nor too computationally challenging to solve. Note, two random instances were removed from the results because CPLEX could not solve them without running out of memory, which occurred with a branch and bound tree of approximately 16 gigabytes.

Overall, the computational study solved six classes of MKEP instances. The size of problems solved are in the first two columns of Table 1-3. To avoid anomalies, fifteen randomly generated MKEP instances of the same size are solved. This study solved 88 IPs. The data reported in the tables represent the average of the fifteen instances.

The primary goal of this computational study is to solve these instances and compare ALEC's impact on the solution time. The study was performed on an Intel(R) Core(TM) i7-6700 3.4 GHz processor with 16.0 GB of RAM. The code was written in C, which called CPLEX 12.6.2. CPLEX's parameters are identical with and without ALEC and are set at

the default settings with two exceptions, which are explained in the following paragraphs.

In some instances for 40 variables and 3 constraints, CPLEX could not solve the IP before running out of RAM memory. To keep consistency, the node file for the branch and bound tree was always stored in the hard drive after 8GB of RAM were used. CPLEX's variable selection rule for branching appears to change depending on whether or not the node files are stored in RAM.

CPLEX's callback functions enable the user to interface with CPLEX's branching tree. ALEC was implemented into these callback functions in CPLEX's branch and bound environment. Due to ALEC's location of implementation, the time to find and add the equality cuts is contained in the total solving time. Since CPLEX's branching rules are a black box, the instances solved by CPLEX were also solved with callback routines. These callback routines did nothing and exited immediately. Consequently, it is believed that CPLEX made the same variable selection when branching. The only impact is from ALEC's equality cuts and its processing time.

One other aspect occurred when implementing ALEC. At the root node, ALEC never found a single equality cut. This result is expected as finding valid equalities with a large number of indices is challenging in nature. Recall, ALEC limits $\beta \leq 4$, which helps control the size of the indices with the equality cuts. Increasing β will have diminishing returns on lifted equalities since the indices will increase. Thus, ALEC should not be implemented at every node in the branch and bound tree.

To help the reader see this, consider the following MKEP constraint.

$$40x_1 + 30x_2 + 29x_3 + 22x_4 + 20x_5 + 16x_6 + 15x_7 + 7x_8 + 6x_9 + 5x_{10} + 2x_11 = 70$$

ALEC cannot find any valid equalities with this MKEP constraint. Observe that ALEC begins by assigning $p = 5$ and $\beta = 1$. Since $a_4 + a_5 < 70$, then no equality exist when $\beta = 1$. When $\beta = 2$ and $p = 6$, $a_4 + a_5 + a_6 < 70$ so no equality exists and p increases to 7. Since a_j is sorted, increasing p will never result in a lifted equality. Therefore, ALEC reports no lifted equalities.

Now assume that a node at depth 4 in the branching tree has branches of $x_1 = 0$, $x_5 = 1$ and $x_{11} = 0$. Replacing x_5 with 1 results in a new knapsack equality constraint with $b = 70 - 20 = 50$. Since x_1 and x_{11} both equal 0, they can be removed from the problem. The resulting MKEP constraint is the exact same as the primary example used throughout this thesis. Consequently, ALEC finds multiple useful lifted equalities at this node in the branching tree.

Due to this fact, ALEC is only called after 50% of the variables have been branched. Examining nodes with 50% of the variables branched results in smaller MKEP instances where equality cuts are more likely to exist. An interesting phenomenon occurred when solving the 100 variable problems with two constraints. ALEC initially did not perform well because the reduced problems may have 50 unbranched variables. Therefore, ALEC spent time at nodes where there are no lifted equalities. This concept is confirmed by changing ALEC to be called after 75% of the variables were fixed, which is shown in Tables 1-3. As a result, ALEC reduced the solving time for the 100 variable problems with two constraints

by 3%. Thus, individuals should primarily implement ALEC only when there are a limited number of unbranched variables. Fortunately, in every binary branch and bound tree, the number of fathomed nodes is equal to half of the total number of nodes plus one. Because of this fact, the majority of all time spent in every branch and bound is consumed near the fathomed nodes.

4.2 Computational Results and Discussions

The computational results are in Tables 1-3 where each row represents the average of 15 randomly generated instances. Table 1 shows the average solving time with and without ALEC in CPLEX, the number of equality cuts, the number of lifted equality cuts, and the percent improvement for different types of random problems. Since the magnitude of equalities was so high, the averages provided in the tables are floored to the nearest thousand. Here, the term equality cuts refers to when $\alpha_j = 1$ for all $j = 1, 2, ..., p$. These are important because DeLissa's research would have already created these inequalities. Thus, the number of lifted equalities describes the number of cuts introduced by this research.

Table 4.1: CPLEX and CPLEX with ALEC Solution Times

		CPLEX	CPLEX+ALEC			
#Var	#Cts	Avg Time (sec)	Avg Time (sec)	Avg # Eq Cuts	Avg # Lifted Cuts	Time %Imprv
30	3	74	59	161,000	346,000	20%
35	3	1,605	1,356	3,000,000	6,200,000	16%
40	3	8,440	7,317	6,200,000	11,900,000	13%
50	2	28	24	54,000	59,000	17%
75	2	47	38	48,000	51,000	18%
100	2	60	58	51,000	62,000	3%
Average		**1700**	**1500**	**1,585,000**	**3,103,000**	**15%**

Table 1 illustrates the solution time with and without ALEC. Observe the magnitude of lifted equality cuts found in the study. In the 40 variable problem with 3 constraints, ALEC found over 18 million cuts on average while reducing the solution time by 13%. ALEC also solved them with an average time of 7,317 seconds. Therefore, ALEC found an equality cut on average every 0.46 millisecond.

In one particular random instance, ALEC identified over 30.7 million equality cuts. ALEC found an equality cut every 0.17 millisecond. Even with the time to generate 30.7 million cuts, ALEC still reduced the solution time by 18% in this instance. The results verify that ALEC quickly identifies useful valid equality cuts.

Table 4.2: CPLEX and CPLEX with ALEC # of Nodes

		CPLEX	CPLEX +ALEC		
# Var	#Cts	Avg # Nodes	Avg # Nodes	%Imprv	Avg Node/Cut
30	3	1,800,000	1,400,000	26%	3
35	3	38,600,000	29,500,000	24%	3
40	3	79,800,000	70,400,000	12%	4
50	2	840,000	590,000	30%	5
75	2	980,000	800,000	18%	8
100	2	1,100,000	1,000,000	4%	8
Average		**20,500,000**	**17,300,000**	**19%**	**5**

Table 2 demonstrates ALEC's impact on the number of nodes CPLEX solved. Since the solution time decreased on average, the number of nodes solved should also decrease. In the 35 variables with 3 constraints, ALEC reduced the number of solved nodes by over 9 million. Overall, ALEC reduced the number of nodes by 19%. In addition, the results show that on average, one in every 5 nodes in the branching tree, ALEC identifies an equality cut.

In a particular instance for the 50 variable and 2 constraint problem, ALEC decreased the number of solved nodes by 130,000. ALEC also found 10,000 equality cuts. Every equality created reduced the number of solved nodes by 13, reiterating that equality cuts reduce the linear relaxation space and are computationally strong.

Table 4.3: CPLEX and CPLEX with ALEC Avg Ticks

		CPLEX	CPLEX +ALEC	
# Var	#Cts	Avg # Ticks	Avg # Ticks	%Imprv
30	3	44,400	24,300	45%
35	3	910,000	540,000	40%
40	3	2,100,000	1,500,000	31%
50	2	14,000	8,000	42%
75	2	19,800	14,000	32%
100	2	26,500	19,000	30%
Average		**520,000**	**350,000**	**36%**

In each type of instance, the ticks are recorded in Table 3. Ticks refer to the main system clock on a personal computer, which generally run at 66MHz. The ticks allow a standard measure of time across different computers regardless of the computer's capability. ALEC reduced the average number of ticks by 170,000.

Although the average time, ticks and nodes were reduced, 35% of instances run with ALEC had a higher solution time. ALEC's cutting planes can potentially change the order in which nodes are evaluated. Even though ALEC theoretically reduces the dimension of P_{KEP}^{LR} by one, ALEC can negatively affect CPLEX's performance due to a different variable selection for branching. On average, ALEC is worth implementing.

The most surprising result of this study is the number of equality and lifted equality cuts found. Over the 88 instances solved, ALEC found over 380 million cuts. For the problems with 3 constraints and 40 variables, ALEC found on average over 11 million such cuts. The

branching tree was also large and on average, a cut was found in one out of four nodes. In every scenario, there are more lifted equalities being found than non-lifted equality cuts. Thus, lifted equality cuts are exceptionally plentiful in these MKEP instances.

Even though hundreds of millions of cuts were found, the total computational time decreased by an average of 15%. This result illustrates that ALEC is extremely fast at creating lifted valid equalities. In addition, it shows that ALEC produces extremely powerful cuts. Therefore, lifted equality cuts are computationally advantageous for the random knapsack equality instances studied in this research.

Chapter 5

Conclusions and Future Work

The goal of this thesis is to determine if valid equalities exist with coefficients greater than one. This thesis affirmatively answers this question by extending lifting to valid equalities. This section reiterates the major contributions of this work and provides a direction for future research.

5.1 Conclusions

This thesis is the first to introduce the concept of lifted equality cuts for the knapsack equality polytope. Two main theorems provide the conditions for which a lifted equality cut can exist. Theoretically, lifted equality cuts reduce the dimension of the linear relaxation's space by at least one. In addition, the algorithm of lifted equality cuts (ALEC) finds equalities that satisfy the conditions of the two theorems. ALEC's run time is $O(n^2)$. An example demonstrates how ALEC is implemented on a small KEP constraint and proves that lifted

equality cuts exist.

A computational study verifies that valid lifted equalities are computationally advantageous by implementing ALEC in a branch and bound tree. The results illustrate that not only do lifted equalities exist, but they are plentiful. In small instances with only three constraints and 40 non-zero coefficients, over 18 million equality cuts were found on average. In the same instances, the results show an approximate average reduction in the solving time of 20 minutes. In another type of instance, ALEC reduced the branch and bound tree by 30%, on average. The results also demonstrate that varying ALEC's implementation depth in the branching tree can change CPLEX's solution time. To the best of the author's knowledge, ALEC is the first algorithm that has found over 30.7 million equality cuts in a single instance while still reducing the solving time by 18%, which provides substantial optimism for future research.

5.2 Future Work

The research primarily focuses on the theory behind valid lifted equalities. A future research topic is to implement ALEC on benchmark and real world IPs. The expansion to real world IPs will provide insight if valid equalities are useful in practice. Implementing valid lifted equalities to real world IPs also goes hand in hand in expanding lifted equalities beyond MKEPs. For example, relaxing the knapsack condition and allowing negative coefficients is an important research topic.

ALEC finds lifted equalities by examining each constraint individually. A future research topic should study the relationships between equality constraints and their relative tightness.

Can ALEC be modified to include information from multiple constraints? Or can ALEC combine lifted equality constraints using a mod or floor function to produce stronger lifted equality cuts?

Lastly, integer programming is a subset of optimization problems. A general idea of interest is how lifted equality cuts can be expanded to optimization problems including mixed integer programming and nonlinear integer programming.

Bibliography

[1] Recalde D., Torres R., and Vaca P., 2013, "Scheduling the Professional Ecuadorian Football League by Integer Programming," *Computers and Operations Research* 40, 2478-2484.

[2] Ribeiro C., 2011, "Sports Scheduling: Problems and Applications," *International Transactions in Operational Research* 19, 201-226.

[3] Bhattacharjee D., and Saikia H., 2015, "An Objective Approach of Balanced Cricket Team Selection Using Binary Integer Programming Method," *Operational Research Society of India 53*, 225-247.

[4] Alarcon F., Duran G., and Guajardod M., 2013, "Referee Assignment in the Chilean Football League Using Integer Programming and Patterns," *International Transactions in Operational Research* 21, 415-438.

[5] Russell R., and Urban L., 2004, "A Constraint Programming Approach to Multiple-Venue, Sport-Scheduling Problem," *Computers and Operations Research* 33, 1895-1906.

[6] "March Madness Bracket: How the 68 teams are selected for the Division I Men's Basketball Tournament," NCAA, Retrieved Date on March 20, 2017,

http://www.ncaa.com/news/basketball-men/article/2017-03-12/march-madness-bracket-how-68-teams-are-selected-division-i.

[7] Melouk S., and Keskin B., 2011, "Team Assignments and Scheduling for the NCAA Basketball Tournament," *Operational Research Society* Vol. 63, No. 5, 620-630.

[8] Taleizadeha A.,Niakib S., and Seyedjavadic S., 2011, "Multi-Product Multi-Chance-Constraint Stochastic Inventory Control Problem with Dynamic Demand and Partial Back-Ordering: A Harmony Search Algorithm," *Journal of Manufacturing Systems* 31, 204-213.

[9] Goel V., Slusky M., van Hoevec W., and Shao Y., 2014, "Constraint Programming for LNG Ship Scheduling and Inventory Management," *European Journal of Operational Research* 241, 662-673.

[10] Mokhtarii H., Abadi I., and Amin-Naseri M., 2011, "Production Scheduling with Outsourcing Scenarios: a Mixed Integer Programming and Efficient Solution Procedure," *International Journal of Production Research* Vol. 50, No. 19.

[11] Sawik T., 2003, "Integer Programming Approach to Production Scheduling for Make-To-Order Manufacturing," *Mathmetical and Computer Modelling* 41, 99-118.

[12] Zceylan E., and Paksoy T., 2012, "A Mixed Integer Programming Model for a Closed-Loop Supply-Chain Network," *International Journal of Production Research* Vol. 51, No. 3.

[13] Jindal A., and Sangwan K., 2014, 'Closed Loop Supply Chain Network Design and Optimisation Using Fuzzy Mixed Integer Linear Programming Model," *International Journal of Production Research* Vol. 52, No. 14.

[14] Doyen A., Aras N., and Barbarosoglu G., 2011, "A Two-Echelon Stochastic Facility Location Model for Humanitarian Relief Logistics," *Springer-Verlag Berlin Heidelberg* 6, 1123-1145.

[15] Albareda-Sambola M., Hinojosa Y., Marin A., and Puerto J., 2015, "When Centers Can Fail: A Close Second Opportunity," *Computers and Operations Research* 62, 145-156.

[16] Bastian N., Griffin P., Spero E., and Fulton L., 2015, "Multi-criteria Logistics Modeling for Military Humanitarian Assistance and Disaster Relief Aerial Delivery Operations," *Springer-Verlag Berlin Heidelberg* 10, 921-953.

[17] Theeb N., and Murray C., 2015, "Vehicle Routing and Resource Distribution in Post-disaster Humanitarian Relief Operations," *International Transactions in Operational Research* 00 (2016), 1-32.

[18] Salas L., Cardenas M., and Zhang M., 2012, "Inventory Policies for Humanitarian Aid During Hurricanes," *Socio-Economic Planning Sciences* 46, 272-280.

[19] Akbari V., and Salman F., 2016, "Multi-Vehicle Synchronized Arc Routing Problem to Restore Post-Disaster Network Connectivity," *European Journal of Operational Research* 257, 625-640.

[20] Wang C., Dai J., and Hu., 2003, "Optimization of Beam Orientations and Beam Weights for Conformal Radiotherapy Using Mixed Integer Programming," *Physics in Medical and Biology* 48, 4065-4076.

[21] Lee, E., 2007, "Large-Scale Optimization-Based Classification Models in Medicine and Biology," *Annals of Biomedical Engineering* Vol. 35, No. 6, 1095-1109.

[22] Zhang Y., Berman O., and Verter V., 2011, "The Impact of Client Choice on Preventive Healthcare Facility Network Design," *Springer-Verlag Berlin Heidelberg* 34, 349-370.

[23] Seckiner S., Gokcen H., and Kurt M., 2006, "An Integer Programming Model for Hierarchical Workforce Scheduling Problem," *European Journal of Operational Research* 183, 694-699.

[24] Pastor R., and Corominas A., 2010, "A Bicriteria Integer Programming Model for the Hierarchical Workforce Scheduling Problem," *Journal of Modelling in Management* Vol. 5, No. 1, 54-62.

[25] Ozguyen C., and Sungur B., 2013, "Integer Programming Models for Hierarchical Workforce Scheduling Problems Including Excess Off-Days and Idle Labour Times," *Applied Mathematical Modelling* 37, 9117-9131.

[26] Koepke C., Armacost A., Barnhart C., and Kolitz S., 2006, "An Integer Programming Approach to Support the US Air Forces Air Mobility Network," *Computers and Operations Research* 35, 1771-1788.

[27] Kamra N., and Ayanian N., 2015, "A Mixed Integer Programming Model for Timed Deliveries in Multirobot Systems," IEEE International Conference on Automation Science and Engineering

[28] Karp, R., 1972, "Reducibility Among Combinatorial Problems," *Complexity of Computer Computations*, The IBM Research Symposia Series, 85-103.

[29] Dantzig, G., and Thapa M., 2003, *Linear Programming 2: Theory and Extensions* Springer-Verlag.

[30] Boyd, S., and Vandenberghe, L., 2004, *Convex Optimization*, Cambridge: Cambridge University Press. p. 143.

[31] Wright M., 2004, "The Interior-point Revolution in Optimization: History, Recent Developments, and Lasting Consequences," *Bulletin of the American Mathematical Society* 42, No. 1, 39-56.

[32] Potra F., and Wright S., 2000, "Interior-point Methods," *Journal of Computational and Applied Mathematics* 124, 281-302.

[33] Gomory, R., 1958, "Outline of an Algorithm for Integer Solutions to Linear Programs," *Bulletin of the American Mathematical Society*, 64(5), 275-278.

[34] Wolsey, L., 1977, "Valid Inequalities and Superadditivity for 0-1 Integer Programs," Mathematics of Operations Research, 2, 66-77.

[35] Balas, E., 1979, "Disjunctive Programming," *Annals of Discrete Mathematics*, 5, 3-51.

[36] Easton T., and Hooker K., 2007, "Simultaneously Lifting Sets of Binary Variables into Cover Inequalities for Knapsack Polytopes," *Discrete Optimization* 5, 254-261.

[37] Gu Z., Nemhauser G., and Savelsbergh M., 1999, "Sequence Independent Lifting in Mixed Integer Programming," *Journal of Combinatorial Optimization* 4, 109-129.

[38] Dey, S., and Wolsey., 2010, "Two Row Mixed-Integer Cuts Via Lifting," Mathematical Programming 124(1-2), 143-174.

[39] Balas, E., Zemel, E., 1978, "Facets of the Knapsack Polytope from Minimal Covers," *SIAM Journal on Applied Mathematics*, 34(1), 119-148.

[40] DeLissa, L., 2014. *The Existence and Usefulness of Equality Cuts in Multi-Demand Multi-Dimensional Knapsack Problem*, MS Thesis, Department of Industrial and Manufacturing Systems Engineering, Kansas State University.

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

[42] Morrisona D., Jacobsonb S., Sauppec J., and Sewelld E., 2014, "Branch-and-bound Algorithms: A Survey of Recent Advances in Searching, Branching, and Pruning," *Discrete Optimization* 19, 79-102.

[43] Kiziltan Z., Lodi A., Milano M., and Parisini F., 2011, "Bounding, Filtering and Diversification in CP-based Local Branching," *Business Media* 18, 353-374.

[44] Boland N., Kalinowski T., Waterer H., and Zheng L., 2012, "Mixed Integer Programming Based Maintenance Scheduling for the Hunter Valley Coal Chain," *Business Media* 16, 649-659.

[45] Fugenschuh A., 2007, "Solving a School Bus Scheduling Problem with Integer Programming," *European Journal of Operational Research 193*, 867-884.

[46] Wong J., Mason A., Neve M., and Sowerby K., 2006, "Base Station Placement in Indoor Wireless Systems Using Binary Integer Programming," *IEE Proceedings - Communications* Vol. 153, No. 5.

[47] Chen C., Kang M., Hwang J., and Huang C., 2000, "Application of Binary Integer Programming for Load Transfer of Distribution Systems," *Power System Technology*, 2000. Proceedings.

[48] Gunpinar S., and Centeno G., 2014, "Stochastic Integer Programming Models for Reducing Wastages and Shortage of Blood Products at Hospitals," *Computers and Operations Research* 54, 129-141.

[49] Hasegawa S., and Kosugi Y., 2006, "Solving Nurse Scheduling Problem by Integer-Programming-Based Local Search," *Systems, Man and Cybernetics*, 2006.

[50] Gu, Z., Nemhauser, G., and Savelsbergh, M., 1998, "Lifted Cover Inequalities for 0-1 Integer Programs: Computation," *INFORMS Journal on Computing*, 10(4), 427-437.

[51] Gu, Z., Nemhauser, G., and Savelsbergh, M., 2000, "Sequence Independent Lifting in Mixed Integer Programming," *Journal of Combinatorial Optimization*, 4(1), 109-129.

[52] IBM, "ILOG CPLEX Optimization Studio," Version 12.6.2, Retrieved Date On April 4, 2017, http://www-01.ibm.com/software/info/ilog/.

[53] Beasley, J., 1990, "OR-Library: Distributing Test Problems by Electronic Mail," *The Journal of the Operational Research Society*, 41(11), 1069-1072.