IMPROVING THE SOLUTION TIME OF INTEGER PROGRAMS BY MERGING KNAPSACK CONSTRAINTS WITH COVER INEQUALITIES

by

FABIO TORRES VITOR

B.S., Maua Institute of Technology, Brazil, 2013

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

2015

Approved by:

Major Professor
Dr. Todd Easton

Abstract

Integer Programming is used to solve numerous optimization problems. This class of mathematical models aims to maximize or minimize a cost function restricted to some constraints and the solution must be integer. One class of widely studied Integer Program (IP) is the Multiple Knapsack Problem (MKP). Unfortunately, both IPs and MKPs are $\mathcal{NP}$-hard, potentially requiring an exponential time to solve these problems.

Utilization of cutting planes is one common method to improve the solution time of IPs. A cutting plane is a valid inequality that cuts off a portion of the linear relaxation space. This thesis presents a new class of cutting planes referred to as merged knapsack cover inequalities (MKCI). These valid inequalities combine information from a cover inequality with a knapsack constraint to generate stronger inequalities.

Merged knapsack cover inequalities are generated by the Merging Knapsack Cover Algorithm (MKCA), which runs in linear time. These inequalities may be improved by the Exact Improvement Through Dynamic Programming Algorithm (EITDPA) in order to make them stronger inequalities. Theoretical results have demonstrated that this new class of cutting planes may cut off some space of the linear relaxation region.

A computational study was performed to determine whether implementation of merged knapsack cover inequalities is computationally effective. Results demonstrated that MKCIs decrease solution time an average of 8% and decrease the number of ticks in CPLEX, a commercial IP solver, approximately 4% when implemented in appropriate instances.

Contents

List of Figures

List of Tables

Acknowledgments

I would first like to thank Dr. Todd Easton for his support, understanding, and immeasurable patience while guiding me through this thesis. Not only has he made this research possible due to his technical knowledge and experience, but he has inspired me through his exceptional teaching ability and smartness. I would also like to thank Dr. Jessica Heier Stamm and Dr. Christopher Pinner in their work on my review committee.

Lastly, I would like to thank the faculty, staff, and students of the Department of Industrial and Manufacturing Systems Engineering at Kansas State University, who have significantly contributed to my ability to perform research and also in my career with their encouragement, support, and willingness.

Dedication

This work is dedicated to my parents, Jose Marcos and Maria Helena, and my brother, William, for their unconditional love, encouragement, and continuing support of my education from 5,420 miles away.

Chapter 1

Introduction

Operations research is an important field of study for academics and practitioners. Winston [67] defines operations research as "a scientific approach to decision making that seeks to best design and operate a system, usually under conditions requiring the allocation of scarce resources." Optimization problems can be modeled in various ways, including use of linear programming, integer programming, dynamic programming, and simulation. This thesis discusses integer programming problems and develops a new technique to more quickly solve this class of problems.

Integer Program (IP) is a class of mathematical models defined as the maximization or minimization of $c^T x$ subject to $Ax \leq b$, where $x \in Z_+^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$. IPs are $\mathcal{NP}$-hard as proved by Karp [43] and no known polynomial time algorithm exists to optimally solve this class of problems.

Application of IPs is relevant in public and private sectors. For example, IPs have helped improve decision making in project/portfolio management (Bertsimas *et al.* [12] and

Pinto and Rustem [57]), capital budgeting problems (Finn [28] and Iwamura and Liu [41]), transportation of goods (Arunapuram *et al.* [4], Kaufman *et al.* [44], Ruiz *et al.* [59], and Toth and Vigo [64]), airline industry applications (Anbil *et al.* [3] and Subramanian *et al.* [62]), sports competitions (Easton *et al.* [25], Trick [65], and Urban and Russell [66]), and in the medical industry for areas such as genetic research (Brown and Harrower [15] and Ferreira *et al.* [27]) and radiation treatments (Lee and Zaider [52] and Lee *et al.* [51]).

One of the most widely studied class of IPs is the Knapsack Problem (KP). The concept of KP is similar to a hiker who must decide which items to pack in a knapsack for a camping trip. Each item has an associated benefit and weight, but the hiker can only carry a specific maximum weight. A Multiple Knapsack Problem (MKP) follows the same concept but it has various knapsack constraints. Due to the complexity of MKP when compared to KP, the multiple knapsack problems are preferable in this thesis. MKP applications have been researched by Chang and Lee [16], Szeto and Lo [63], and Kolliopoulos and Steiner [47]. Notice that both KP and MKP are $\mathcal{NP}$-hard as proved by Karp [43].

One of the main techniques used to solve IPs is the branch and bound algorithm first proposed by Land and Doig [50]. This algorithm begins with the linear relaxation solution, which is the solution of the integer program without the integer constraint. Thus, it takes one variable with a fractional solution and creates two child nodes. The first node adds a constraint, making the fractional variable less than or equal to the floor of its fractional value; the second node adds another constraint, forcing the variable to be greater than or equal to the ceiling of its fractional value. This process is repeated until all nodes have been fathomed and the algorithm terminates. A node is fathomed when an integer or infeasible

solution is found or the solution of the evaluated node is worse than the best integer solution found so far. One can see that this enumeration process may require exponential time.

Another technique for solving IPs is the addition of cutting planes. A cutting plane is a valid inequality because it does not eliminate any integer points and may cut off some linear relaxation solution. The cutting planes that cut off a larger portion of the linear relaxation space are considered useful and may help solve IPs more quickly. When these cutting planes are facet defining, they are theoretically the strongest inequalities. Examples of research conducted to generate new classes of useful cutting planes are found in Balas [9], Zemel [70], and Gu *et al*. [34].

One of the most useful cutting planes used to solve integer programs is the cover cut. They are valid inequalities that may cut off some space of the linear relaxation solution and also can be strengthened through the lifting process. Lifting begins with a valid inequality and seeks to modify this inequality by adding more variables with different coefficients. The output of the lifting process is another valid inequality that is typically stronger than the original inequality.

The following sections provide the primary motivations for this thesis, potential contributions of a new class of useful cutting planes, and the outline of the next chapters.

1.1 Motivation

Each year researchers develop new theoretically and computationally useful strategies intended to solve IPs faster. Consequently, finding a new technique that has never been

discovered before and decreases the time required to solve integer programs is the primary motivation of this thesis.

Hickman [38] has recently conducted research on inequality merging, which combines two or more low dimension valid inequalities to generate a new valid inequality with higher dimension. This research provided conditions for validity and conditions for facet defining. In addition, a computational study demonstrated that implementation of merged inequalities decreases the average time required to solve IPs approximately 9%. Although Hickman's results are good, some restrictions are applied. For example, merged inequalities can only be generated by merging two cover inequalities and coefficients of the resulting merged inequality are constant.

Hickman's previous work motivated two questions in this research. Is it possible to generate valid inequalities by combining information from a cover inequality and a knapsack constraint? Can these inequalities be generated in such a way that the coefficients of the merging variables differ, are as strong as possible, and are any positive real number?

1.2 Contributions

This thesis' primary contribution is a new class of cutting planes that is generated when information from a cover inequality and a knapsack constraint are merged, resulting in merged knapsack cover inequalities. Merging a knapsack constraint into a cover inequality requires a cover and its valid inequality from a knapsack constraint of the problem and a set of merging variables from the same knapsack constraint. To create a merged knapsack cover

inequality, the right-hand side of this new valid inequality is the size of the cover minus 1. The coefficients of the merged variables are linearly scaled (α) multiples of their respective coefficients in the knapsack constraint. The unmerged variables from the cover inequality have a coefficient with value equal to 1. If α is correctly selected, the resulting inequality has a higher dimensional face than the cover inequality in some cases. In certain instances, this inequality may also be facet defining.

This thesis presents the Merging Knapsack Cover Algorithm (MKCA) in order to generate this new class of cutting planes. This algorithm is able to report a valid merged knapsack cover inequality by identifying which variables in the knapsack constraint can be merged into a chosen cover inequality. The coefficients are determined in such a way that the validity conditions are met. The MKCA runs extremely fast since it is a linear time algorithm.

MKCA quickly guarantees a valid merged knapsack cover inequality, but these inequalities can sometimes be strengthened. Another contribution of this thesis is the Exact Improvement Through Dynamic Programming Algorithm (EITDPA), which determines the strongest possible merged knapsack cover inequality. EITDPA is a pseudo-polynomial time algorithm that runs reasonably fast in practice. In this computational study, it runs in less than 0.1 seconds.

A computational study demonstrates that merged knapsack cover inequalities are on average 8.0% better in time and 4.0% better on ticks in CPLEX. In addition, these cutting planes are worth implementing when an IP has a knapsack constraint such that the minimum coefficient is approximately 90% of the maximum coefficient.

1.3 Outline

Chapter 2 introduces the main theoretical background needed to understand this thesis, including integer programming and polyhedral theory. Theoretical information about knapsack and multiple knapsack problems, which are the class of problems studied in this research, cover inequalities, and lifting are also shown. Examples demonstrate the principles presented.

Chapter 3 describes merged knapsack cover inequalities generated by MKCA, which has two subroutines. The first subroutine defines the set of merging variables that guarantees the validity of the inequality; the second subroutine calculates the appropriate coefficient. The proof of validity is demonstrated for both subroutines. A second algorithm is also presented to improve the value of the coefficient and increase the strength of the inequality. Finally, two examples are shown to illustrate the generation, improvement, and theoretical usefulness of this new class of cutting planes.

The computational study of this thesis is presented in Chapter 4. Merged knapsack cover inequalities are generated for several instances, and their computational results are compared to results of CPLEX [40] when the same problem is solved without these inequalities. In addition, instances that demonstrate the relevance of this thesis are described in order to determine computational effectiveness of this new class of cutting planes based on computational results provided.

Lastly, Chapter 5 concludes the thesis with a presentation of theoretical and computational results achieved. Ideas for future computational studies and future theoretical extensions of merged knapsack cover inequalities are presented.

Chapter 2

Background Information

This chapter introduces the necessary background information about integer programming and a review of theoretical topics and examples to allow increased understanding of this thesis. The first section defines integer programs, discusses aspects of polyhedral theory, and describes how this is strongly related to solution techniques of IPs. An integer program example is presented to demonstrate how IPs are solved through cutting planes. For those interested in deeper details, Nemhauser and Wolsey [56] is suggested for reading.

The second section approaches knapsack and multiple knapsack problems, since these are critical to this research. A multiple knapsack example is presented along with its solution. The third section explains the idea of cover inequalities and use of these inequalities as cutting planes to solve integer programs. The fourth section defines lifting and clarifies how valid inequalities can be strengthened through this process. The last section presents one of the most relevant prior works on inequality merging, including an example to clearly demonstrate the concepts and usefulness of this class of cutting planes.

2.1 Integer Programming

Integer Programs (IP) are defined as a class of mathematical models to solve optimization problems. Let $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^m$. Thus, integer programs are formulated as the following:

$$\begin{aligned} \text{Maximize} \quad & c^T x \\ \text{Subject to:} \quad & Ax \leq b \\ & x \in Z_+^n. \end{aligned}$$

The linear relaxation space is the feasible region of the IP without the integer restrictions. Formally, the linear relaxation space is defined as $P^{LR} = \{x \in \mathbb{R}_+^n : Ax \leq b\}$. The linear relaxation solution is defined as (z_{LR}, x_{LR}) with (z_{LR}^*, x_{LR}^*) as the optimal solution where $x_{LR}^* \in P^{LR}$.

The feasible points of integer programs are denoted as P where $P = \{x \in \mathbb{Z}_+^n : Ax \leq b\}$ with the set of indices being $N = \{1, ..., n\}$. The integer solution is given as (z_{IP}, x_{IP}) with (z_{IP}^*, x_{IP}^*) as the optimal solution of the integer program where $x_{IP}^* \in P$. Clearly, $z_{LR}^* \geq z_{IP}^*$.

Integer Programs are considered $\mathcal{NP}$-hard as proved by Karp [43], meaning that no polynomial time algorithm is known to solve this class of problems. Therefore, a great computational effort may be required to find the optimal integer solution, even for some small instances. On the other hand, linear programs can be solved in polynomial time (Khachiyan [46] and Karmarkar [42]) and the optimal linear solution can be found quickly for most large instances.

The most used technique to solve IP problems is the branch and bound algorithm. First proposed by Land and Doig [50], the branch and bound algorithm is used by a majority of commercial solvers. This algorithm finds the optimal integer solution in some finite time considering that this optimal solution exists; however, it may require an exponential time.

The initialization of branch and bound algorithm is given by the optimal linear relaxation solution x^*_{LR} and z^*_{LR}. The algorithm terminates if the linear relaxation solution is integer, resulting in $x^*_{IP} = x^*_{LR}$ and $z^*_{IP} = z^*_{LR}$. Otherwise, this linear relaxation solution is considered to be the parent node and a non-integer variable is selected for branching. If $x_j = f$ is the variable selected, then two child nodes are generated with an added constraint $x_j \leq \lfloor f \rfloor$ and another added constraint $x_j \geq \lceil f \rceil$. The algorithm repeats this process until all nodes have been fathomed.

A node is fathomed if the node has an integer solution or is infeasible. If z^*_{LR} of the evaluated node is less than (greater than) the best integer solution found in previous steps for maximization (minimization) problems, the node is also fathomed.

This thesis does not include results on which non-integer variable should be branched first; however, Linderoth and Savelsbergh [54] and Achterberg *et al.* [1] have developed research in this field. In addition, various strategies to evaluate the nodes in the branch and bound tree are available, such as depth first, breadth first, and best child. Another relevant method to solve IP problems includes utilization of cutting planes to cut off some space of the linear relaxation solution. This topic is discussed next in this chapter.

2.1.1 Polyhedral Theory

One relevant field of research for IPs, polyhedral theory involves study of the feasible region of optimization problems. Convexity is a critical topic in this field and some definitions are presented in this thesis to increase understanding of the topic.

First, the set $S \subseteq R^n$ is said to be convex if and only if $\lambda x + (1-\lambda)x' \in S$ for all $x, x' \in S$ and all $\lambda \in [0,1]$. This definition states that a straight line can be drawn from any two points in S concluding all points on this line are in S^{ch}. The convex hull of S, S^{ch}, is the intersection of all convex sets that contain S. In this work, the covex hull of P is P^{ch}.

A hyperplane H in $\mathbb{R}^n$ is a set of the form $\{x \in \mathbb{R}^n : \alpha^T x = \beta\}$ where α is a non-zero vector in $\mathbb{R}^n$ and β is a scalar. In convex optimization, any inequality of the form $\leq$ or $\geq$ restricts the feasible region to either above or below the hyperplane. This is called a half-space, defined as $\{x \in \mathbb{R}^n : \sum_{i=1}^{n} \alpha_i x_i \leq \beta\}$. A polyhedron is defined as the intersection of a finite number of half-spaces. If this polyhedron is bounded, it is called a polytope. Both P^{ch} and P^{LR} are polyhedrons.

Integer programming is closely related to polyhedral theory. The extreme points of P^{LR} may be integer and non-integer, and the extreme points of P^{ch} are integer. An inequality of the form $\sum_{i=1}^{n} \alpha_i x_i \leq \beta$ is valid for P^{ch} if and only if $\sum_{i=1}^{n} \alpha_i x'_i \leq \beta$ is satisfied for every $x' \in P$. Also, a cutting plane is a valid inequality that removes some portion of P^{LR}. In other words, there exists an $x'' \in P^{LR}$ such that $\sum_{i=1}^{n} \alpha_i x''_i > \beta$.

Many valid inequalities are not considered useful because the portion of P^{LR} removed does not improve the solution time of the integer program. Theoretical usefulness of a valid

inequality is defined in terms of its induced dimension in P^{ch}. The dimension of a space is defined as the number of linearly independent vectors. Since the feasible region of an IP has no feasible vectors, the dimension of P^{ch} is defined as the maximum number of affinely independent points minus 1. The points $x^1, ..., x^p \in \mathbb{R}^n$ are affinely independent if and only if the unique solution to $\sum_{i=1}^{p} \lambda_i x_i = 0$ and $\sum_{i=1}^{p} \lambda_i = 0$ is $\lambda_i = 0$ for all $i \in \{1, ..., p\}$.

Another critical concept in polyhedral theory is faces. A face is defined as the induced points of an inequality in P^{ch}. Every valid inequality $\sum_{i=0}^{n} \alpha_i x_i \leq \beta$ defines a face $F \subseteq P^{ch}$ that takes the form $F = \{x \in P^{ch} : \sum_{i=0}^{n} \alpha_i x_i = \beta\}$. If $F \neq \emptyset$, then F supports P^{ch}. The strength of face F is closely related to the dimension of P^{ch}. A face is stronger as its dimension is closer to the dimension of P^{ch}, given that the dimension of the face must be at least one unit less than the dimension of P^{ch}. This is a crucial statement when trying to prove a face is facet defining.

When comparing the strength of faces, those faces that correspond to facet defining inequalities are preferable since the portion removed from P^{LR} is maximized. A facet defining inequality must have dimension equal to the dimension of P^{ch} minus 1. When all facet defining inequalities are included in the problem, P^{ch} is completely defined and, therefore, all extreme points are integers. In such a case, the integer program can be solved as a linear program. The example presented in the next section demonstrates these principles.

2.1.2 Integer Programming Example

Consider the following IP:

$$\begin{aligned} \text{Maximize} \quad & x_1 + 2x_2 \\ \text{Subject to} \quad & 3x_1 + 2x_2 \leq 12 \\ & 3x_1 + 4x_2 \leq 15 \\ & x_1, x_2 \in \mathbb{Z}_+. \end{aligned}$$

This example is a two-dimensional IP with two constraints and $x \in \mathbb{Z}_+^2$. A graphical representation is shown in Figure 2.1 with the large dots representing the set of feasible integer points P. The first constraint ($3x_1 + 2x_2 \leq 12$) crosses the points (0,6), C and D. The second constraint ($3x_1 + 4x_2 \leq 15$) crosses the points A, B, C and (5,0). In this example the point A is the optimal linear relaxation solution with $x_1^{*LR} = 0$, $x_2^{*LR} = \frac{15}{4}$, and $z^{*LR} = \frac{15}{2}$.

Clearly P^{ch} can be defined in this example with line segments that passes through points (0,0) and (0,3), points (0,3) and (1,3), points (1,3) and (4,0), and points (4,0) and (0,0). There is a portion of the linear relaxation space that is outside of P^{ch}, so adding cutting planes to remove this space seems a good strategy.

First, the inequality $x_1 + x_2 \leq 4$ is added. It does remove part of the linear relaxation space (triangle BCD) without removing any feasible integer points, thereby classifying it as a valid inequality. After adding this inequality and solving the linear program, the new optimal solution is still the point A with $x_1^{*LR} = 0$, $x_2^{*LR} = \frac{15}{4}$, and $z^{*LR} = \frac{15}{2}$.

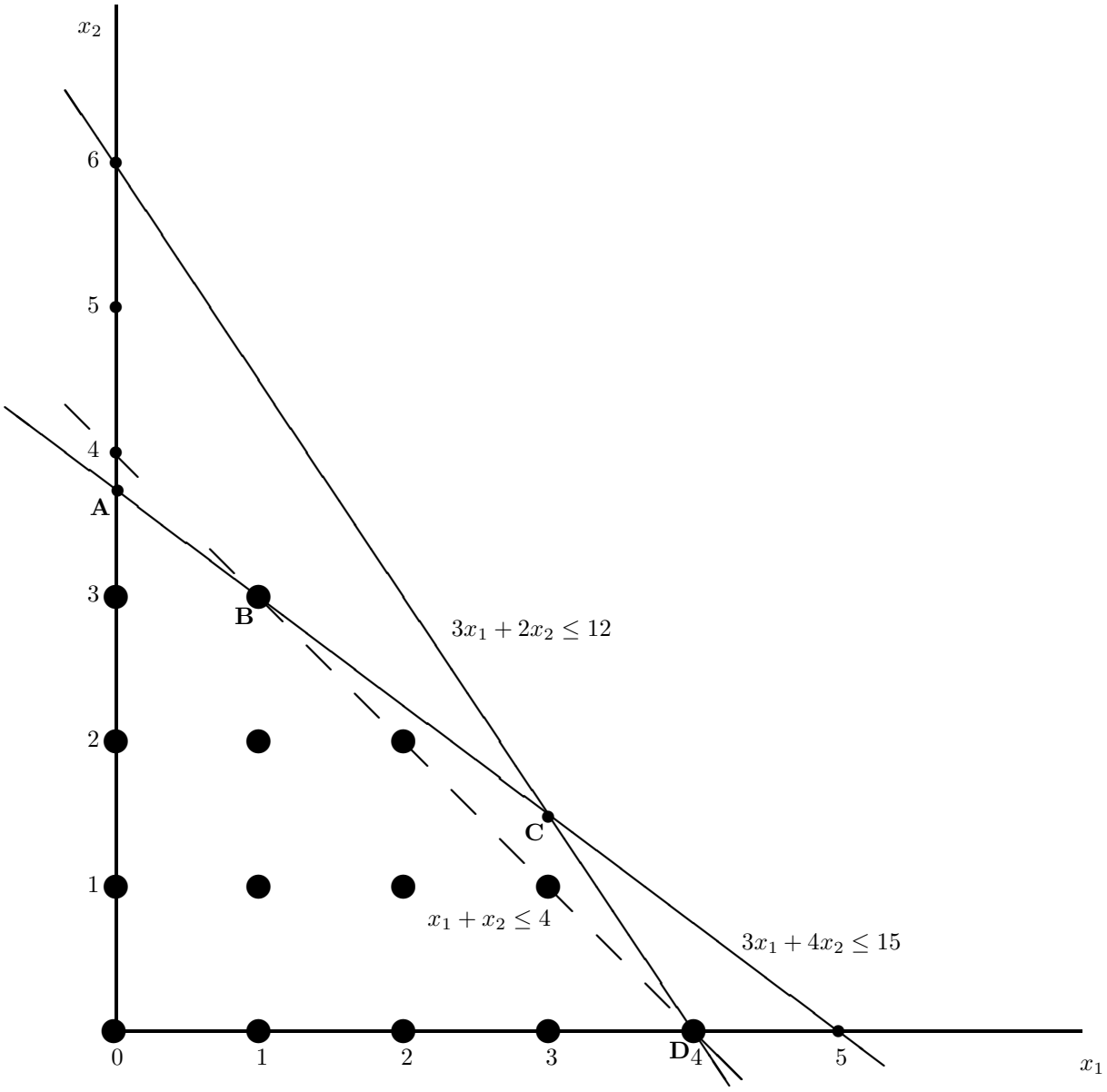

Figure 2.1: Integer Program Example

Adding inequality $x_2 \leq 3$ to this problem also removes part of the linear relaxation space (triangle A, B, and (0,3)) without removing any feasible integer points, thereby also classifying it as a valid inequality. After solving the linear program with this new constraint, the new optimal solution is the point B with $x_1^{*LR} = 1$, $x_2^{*LR} = 3$, and $z^{*LR} = 7$. Since the

linear relaxation solution is given by integer points, the solution is optimal with $x_1^{*IP} = 1$, $x_2^{*IP} = 3$, and $z^{*IP} = 7$.

Three conditions must be met in order to prove the inequality $x_1 + x_2 \leq 4$ is facet defining. First, this IP problem has two variables and $dim(P^{ch}) \leq 2$. Also, the dimension of P^{ch} must be greater than or equal to the maximum number of affinely independent points minus 1. The points $(0,0)$, $(1,0)$, $(0,1)$ are feasible and are affinely independent; therefore, $dim(P^{ch}) \geq 2$. Thus, $dim(P^{ch}) = 2$.

Second, $x_1 + x_2 \leq 4$ is valid, since there are no integer points that violate this inequality. Third, one must prove the dimension of this inequality's face F. In order to prove this is not the entire space, the point (0,0) $\in P^{ch}$, and $0 + 0 < 4$. Therefore, $dim(F) \leq dim(P^{ch}) - 1$, so $dim(F) \leq 1$. The points $(1,3)$ and $(4,0)$ are feasible, in F, and affinely independent; therefore, $dim(F) \geq 1$. Consequently, $x_1 + x_2 \leq 4$ is a facet defining inequality.

2.2 Knapsack and Multiple Knapsack Problems

The Knapsack Problem (KP) is a common category of integer programs widely studied by several researchers in this field such as Lai and Sahni [49] and Pisinger [58]. The idea of this problem may be explained by a hiker that is packing for a camping trip and has to decide which items to pack in a knapsack. There are n items available and each item i is associated with some benefit c_i and a non-negative weight a_i for each $i \in \{1, ..., n\}$. One seeks to maximize the amount of benefit, but the weight carried must be less than or equal to some limit b.

The knapsack problem can be modeled as a binary integer program, meaning that decision variables can be either one or zero. If $x_i = 1$, then item i is selected; on the other hand, if $x_i = 0$, item i is not selected. Let $c \in \mathbb{R}^n$, $a \in \mathbb{R}^n_+$, $b \in \mathbb{R}_+$, and a KP is formally defined as:

$$\begin{aligned} \text{Maximize} \quad & c^T x \\ \text{Subject to:} \quad & a^T x \leq b \\ & x \in \{0,1\}. \end{aligned}$$

Another class of integer programming problems related to this research is the Multiple Knapsack Problem (MKP). This problem also has n items available, and each item i is associated with some benefit c_i and some non-negative coefficient $a_{i,j}$ limited to some b_j for all $i \in \{1, 2, ..., n\}$ and for all $j \in \{1, 2, ..., m\}$ where m is the number of constraints. The multiple knapsack problem can also be formulated as a binary integer program where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}_+$, and $b \in \mathbb{R}^m_+$. Formally, an MKP is defined as:

$$\begin{aligned} \text{Maximize} \quad & c^T x \\ \text{Subject to:} \quad & Ax \leq b \\ & x \in \{0,1\}. \end{aligned}$$

Define P_{KP} as the set of feasible points of a knapsack problem with $P_{KP} = \{x \in \{0,1\} : a^T x \leq b\}$ and P_{MKP} as the set of feasible points of a multiple knapsack problem with $P_{MKP} = \{x \in \{0,1\} : Ax \leq b\}$. Every item i in the knapsack problem has its associated weight a_i such that $a_i \leq b$, otherwise no feasible point with $x_i = 1$ satisfies $a^T x \leq b$ and x_i can be removed from the problem. In addition, assume all items are sorted

by a_i in descending order. From this assumption, the convex hull P_{KP}^{ch} has dimension n because 0 and $\xi_i \in N$ are feasible where ξ_i is the i^{th} identity point.

Observe that any binary integer programming constraint can be transformed into a knapsack constraint using simple transformations. An equality constraint can be replaced by two other constraints: one less than or equal to constraint and one greater than or equal to constraint. If a greater than or equal to constraint exists, it is simply multiplied by negative 1. If $a_i < 0$ exists, it is replaced with $1 - x_i'$.

Applications of knapsack and multiple knapsack problems arise in many different research topics, such as production planning and inventory (Dawande *et al.*[20]), project/portfolio selection (Chang and Lee [16]), allocation of resources (Babaioff *et al.* [8]), profit maximization (Dizdar *et al.* [24] and Szeto and Lo [63]), machine scheduling techniques (Kellerer and Strusevich [45] and Kolliopoulos and Steiner [47]), and storage management/packing problems (Shachnai and Tamir [60]).

In addition to these applications, several other research topics have been developed in relation to solution techniques that primarily consider strategies to solve KP and MKP more quickly. For example, large neighborhood search techniques for multiple knapsack problems (Ahuja and Cunha [2]), exact synchronized simultaneous uplifting for the knapsack polytope (Beyer [13]), polyhedral study on knapsack problems with disjoint cardinality (Zeng and Richard [71]), synchronized simultaneous lifting in binary knapsack polyhedra (Bolton [14]), a genetic algorithm for the multidimensional knapsack problem (Chu and Beasley [18]), a cutting plane algorithm to lift variables in three sets (Harris [37]), and equality cuts applied to multi-demand multidimensional knapsack problem (Delissa [21]).

This research is mainly focused on multiple knapsack problems. Because applications in real world usually have multiple constraints, MKPs is of greater significance when compared to KPs. The subsequent section presents an example of an MKP.

2.2.1 Multiple Knapsack Problem Example

In order to illustrate a multiple knapsack problem, consider a well-known firm specializing in engineering projects with several project offers for the next year. The managers want to decide which projects to accept in order to maximize the profit. Each project requires a certain number of three different resources: engineers, administrative employees, and manpower. A maximum of 11 projects may be selected and 44 engineers, 52 administrative employees, and 61 laborers are available. Table 2.1 presents the annual profit for each project and the number of engineers, administrative employees, and laborers required. Profits are expressed in $100,000.

Project	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Profit	82	94	12	45	18	26	95	44	38	29	10
Engineers	30	25	20	15	12	11	11	10	10	5	1
Adm. Employees	18	26	28	21	13	17	29	34	31	19	10
Laborers	20	24	19	41	4	18	10	17	37	47	16

Table 2.1: Instance - Multiple Knapsack Example

This problem can be formulated as a multiple knapsack problem. The decision variable $x_i \in \{0, 1\}$ for $i \in \{1, ..., 11\}$ represents whether or not a project is selected. The objective function seeks to maximize the profit with $\sum_{i=1}^{11} c_i x_i$ where c_i represents the profit of each

project i. Constraints are defined as $\sum_{i=1}^{11} a_{i,j}x_i \leq b_j$ for $j \in \{1,2,3\}$ where $a_{i,j}$ represents the number of resource j needed for each project i and b_j represents the number of each resource j available. The formulation is:

$$\begin{aligned}
\text{Maximize} \quad & 82x_1 + 94x_2 + 12x_3 + 45x_4 + 18x_5 + 26x_6 + 95x_7 + 44x_8 + 38x_9 + 29x_{10} + 10x_{11} \\
\text{Subject to:} \quad & 30x_1 + 25x_2 + 20x_3 + 15x_4 + 12x_5 + 11x_6 + 11x_7 + 10x_8 + 10x_9 + 5x_{10} + x_{11} \leq 44 \\
& 18x_1 + 26x_2 + 28x_3 + 21x_4 + 13x_5 + 17x_6 + 29x_7 + 34x_8 + 31x_9 + 19x_{10} + 10x_{11} \leq 52 \\
& 20x_1 + 24x_2 + 19x_3 + 41x_4 + 4x_5 + 18x_6 + 10x_7 + 17x_8 + 37x_9 + 47x_{10} + 16x_{11} \leq 61 \\
& x_i \in \{0,1\}, \text{ for } i \in \{1, ..., 11\}.
\end{aligned}$$

The optimal solution of this multiple knapsack problem is $(1,0,0,0,0,0,1,0,0,0,0)$ with $z^*_{MKP} = 177$, meaning that the firm only works on Projects 1 and 7 since both projects return the maximum profit. The firm makes a profit of \$17,700,000 and uses 41 engineers, 47 administrative employees, and 30 laborers.

2.3 Cover Inequalities

Cover inequalities comprise a class of cutting planes for a knapsack constraint. In the problem presented in Section 2.2.1, a cover represents a number of selected projects that exceeds resource capacity. Equivalently, a cover represents an infeasible solution.

From a knapsack constraint, a cover is a set $C \subseteq N$ such that setting $x_i = 1$ for all $i \in C$ is infeasible. Formally, $C \subseteq N$ is a cover if and only if $\sum_{i \in C} a_i > b$. The corresponding valid cover inequality is $\sum_{i \in C} x_i \leq |C| - 1$.

A cover is said to be minimal if and only if any index i removed from set C implies the set is no longer a cover. A cover is minimal if and only if $\sum_{i \in C \setminus \{j\}} a_i \leq b$ for each $j \in C$. Also, an extended cover is defined as $E(C) = C \cup \{i \in N \setminus C : a_i \geq max_{j \in C}\{a_j\}\}$ with a corresponding valid inequality of the form $\sum_{i \in E(C)} x_i \leq |C| - 1$.

For the multiple knapsack problem presented in Section 2.2.1, the first knapsack constraint is $30x_1+25x_2+20x_3+15x_4+12x_5+11x_6+11x_7+10x_8+10x_9+5x_{10}+x_{11} \leq 44$. Consider the sets $C_1 = \{5, 6, 7, 8, 9\}$ and $C_2 = \{3, 4, 9, 10, 11\}$. Since $\sum_{i \in C_1} a_i = 12+11+11+10+10 = = 54 > 44$ and $\sum_{i \in C_2} a_i = 20+15+10+5+1 = 51 > 44$, both C_1 and C_2 are covers. The cover C_2 is not a minimal cover because the set $\{3, 4, 9, 10\}$ is a cover. On the other hand, C_1 is a minimal cover because $\sum_{i \in C_1 \setminus \{j\}} a_i \leq 44$ for any $j \in C_1$. The corresponding valid inequality of C_1 is $x_5+x_6+x_7+x_8+x_9 \leq 4$. This knapsack constraint is sorted, so $a_1 \geq a_2 \geq a_3 \geq a_4$ which are greater than $max_{i \in C_1}\{a_i\}$. Therefore, adding the first four variables to the original cover inequality gives the extended cover inequality $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 \leq 4$.

The valid extended cover inequality is not facet defining. However, the valid cover inequality is facet defining in the restricted space, but not in the full space. Lifting is one technique used to make this valid inequality facet defining. Section 2.4 explains the restricted space definition and the lifting process used to generate valid inequalities with higher dimension.

2.4 Lifting

Starting from a valid inequality, the lifting procedure, introduced by Gomory [31], is able to increase the dimension of an inequality, thereby creating stronger inequalities. The lifting procedure begins with a valid inequality of a restricted space and terminates with a valid inequality for the entire space.

A restricted space requires a subset of the variables $E \subseteq N$ and $k \in \mathbb{Z}^{|E|}$. The restricted space of P on E and K is defined as $P_{E,K} = \{x \in P : x_i = k_i \ \forall i \in E\}$. Lifting requires a valid inequality of $P^{ch}_{E,K}$ with the form $\sum_{i \in E} \alpha_i x_i + \sum_{i \in N \setminus E} \alpha_i x_i \leq \beta$. The lifted inequality is valid for P^{ch} and takes the form $\sum_{i \in E} \alpha_i{}' x_i + \sum_{i \in N \setminus E} \alpha_i x_i \leq \beta'$.

Several strategies of lifting, such as up, down or middle lifting, exact or approximate lifting, and sequential or simultaneous lifting are available. These strategies can all be applied independently creating 12 broad classes of lifting, such as exact sequential up lifting and approximate simultaneous down lifting. These categories depend on E, K, α' and β'.

Lifting procedures typically require solving an optimization problem. The solution to the optimization problem determines the values of α' and β'. If the optimization problem is solved optimally, then α' and β' are the strongest possible, and the procedure is considered an exact lifting procedure, as described in Zemel [70], Wolsey [68], Gutierrez [35] and Easton and Hooker [26]. If the optimization problem is not solved to optimality, then the lifting procedure is approximate, as described in Balas [9] and Gu *et al.* [34].

If $|E| = 1$, then the lifting procedure is sequential, as shown in Cho *et al.* [17], Gutierrez [35], Hammer *et al.* [36], and Wolsey [68]. In other words, only a single coefficient and

possibly the right-hand side of the inequality is modified. When multiple coefficients are modified, then it is simultaneous lifting. See Atamtürk [5], Gu *et al.* [32], [33], and [34], Shebalov and Klabjan [61], and Kubik [48].

Finally, if the values of K are at the lower bounds of the associated variables, then the method is called up lifting. If the K values are at the upper bounds, then it is down lifting. Middle lifting occurs if K is someplace in between. The following section provides an example of exact sequential up lifting to demonstrate the idea of the lifting procedure.

2.4.1 Lifting Example

In the first knapsack constraint $30x_1 + 25x_2 + 20x_3 + 15x_4 + 12x_5 + 11x_6 + 11x_7 + 10x_8 + +10x_9 + 5x_{10} + x_{11} \leq 44$ from the multiple knapsack problem presented in Section 2.2.1, a cover is $C_1 = \{5, 6, 7, 8, 9\}$ with corresponding valid inequality $x_5 + x_6 + x_7 + x_8 + +x_9 \leq 4$. The valid inequality generated by the lifting procedure depends on the starting lifting variable. In this example, assume the lifting order is x_4, x_3, x_2, x_1, x_{10}, and x_{11}. Beginning with variable x_4, the first step of exact sequential up lifting defines the inequality $\alpha' x_4 + x_5 + x_6 + x_7 + x_8 + x_9 \leq 4$. The value of α' is found by solving the following IP:

Maximize $x_5 + x_6 + x_7 + x_8 + x_9$

Subject to: $30x_1 + 25x_2 + 20x_3 + 15x_4 + 12x_5 + 11x_6 + 11x_7 + 10x_8 + 10x_9 + 5x_{10} + x_{11} \leq 44$

$x_4 = 1$

$x_i \in \{0, 1\}$, for $i \in \{1, ..., 11\}$.

The optimal solution of this integer program is given by $(0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1)$ with $z^*_{IP} = 2$. The value for α' is defined as $\alpha' = \beta - z^*_{IP}$, so $\alpha' = 4 - 2 = 2$. The valid inequality defined in this first step is $2x_4 + x_5 + x_6 + x_7 + x_8 + x_9 \leq 4$. The next step considers the lifting of variable x_3, and the exact sequential up lifting method defines a valid inequality $\alpha' x_3 + 2x_4 + x_5 + x_6 + x_7 + x_8 + x_9 \leq 4$ for this step. The new value of α' is defined by solving the following integer program:

$$\begin{aligned}
\text{Maximize} \quad & 2x_4 + x_5 + x_6 + x_7 + x_8 + x_9 \\
\text{Subject to:} \quad & 30x_1 + 25x_2 + 20x_3 + 15x_4 + 12x_5 + 11x_6 + 11x_7 + 10x_8 + 10x_9 + 5x_{10} + x_{11} \leq 44 \\
& x_3 = 1 \\
& x_i \in \{0, 1\}, \text{ for } i \in \{1, ..., 11\}.
\end{aligned}$$

The optimal solution of the integer program above is given by $(0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1)$ with $z^*_{IP} = 2$. Thus, α' is calculated by $\alpha' = 4 - 2 = 2$, and the valid inequality defined in this second step is $2x_3 + 2x_4 + x_5 + x_6 + x_7 + x_8 + x_9 \leq 4$. This algorithm terminates when all remaining variables are lifted and the exact sequential up lifted valid inequality is $3x_1 + 2x_2 + 2x_3 + 2x_4 + x_5 + x_6 + x_7 + x_8 + x_9 \leq 4$. This inequality, which is valid for the entire space, is stronger than the cover inequality. This inequality is also facet defining since 11 affinely independent points that meet the inequality at equality exist.

2.5 Inequality Merging

The most relevant prior work to this thesis involves the generation of valid inequalities by inequality merging. In this area, Dey and Wolsey [23] present a method to obtain a new class of cutting planes by combining information from two different rows of a simplex tableau. The coefficients of this new cutting plane are generated by lifting functions and integer coefficients are guaranteed in this process.

Additional research on inequality merging was conducted by Dey and Richard [22]. They present a new strategy to generate facet-defining inequalities for two-dimensional group problems by combining two facet-defining inequalities of one-dimensional group problems.

Recently, Hickman [38] presented different theoretical foundations to generate cutting planes by inequality merging for multiple knapsack problems. This research combines two or more low dimensional valid inequalities in order to generate a new valid inequality of higher dimension. Theoretical and computational studies are presented in which results demonstrate a 9% average decrease in computational time.

The research developed by Hickman [38] describes two low-dimensional valid inequalities with one being the host inequality and the other the donor inequality. The host inequality replaces one or more of its terms with a collection of terms attained from the donor inequality. Formally, generating a merged valid inequality on a binary variable x_p requires $C^1 \subset N$, $p \in C^1$ and $C^2 \subseteq (N \backslash C^1) \cup \{p\}$. The host valid inequality is defined as $\sum_{j \in C^1} \alpha_j^1 x_j \leq \beta^1$ where β^1 and α_j^1 are non-negative integers for all $j \in C^1$ and $\alpha_p^1 = 1$.

The valid donor inequality is defined as $\sum_{j \in C^2} \alpha_j^2 x_j \leq \beta^2$ and the merged inequality takes the form $\sum_{j \in C^1 \setminus \{p\}} \alpha_j^1 x_j + \sum_{j \in C^2} \frac{\alpha_j^2}{\beta^2} x_j \leq \beta^1$.

Hickman presents the following example to demonstrate the concept of inequality merging. Consider the next two knapsack constraints, where $x_i \in \{0, 1\}$ for $i \in \{1, 2, ..., 8\}$:

$$13x_1 + 12x_2 + 11x_3 + 5x_4 + 3x_5 + 2x_6 + 2x_7 + 1x_8 \leq 38$$

$$2x_1 + 4x_2 + 1x_3 + 7x_4 + 6x_5 + 8x_6 + 6x_7 + 5x_8 \leq 30.$$

Assume $C^1 = \{1, 2, 3, 4\}$, $C^2 = \{4, 5, 6, 7, 8\}$, and $p = 4$. Notice that C^1 is a cover in the first knapsack constraint and C^2 is a cover in the second knapsack constraint. Thus, $x_1 + x_2 + x_3 + x_4 \leq 3$ is a valid host inequality and $x_4 + x_5 + x_6 + x_7 + x_8 \leq 4$ is a valid donor inequality. The inequality generated by the inequality merging method is given by $x_1 + x_2 + x_3 + \frac{1}{4}\big(x_4 + x_5 + x_6 + x_7 + x_8\big) \leq 3$. This merged inequality is valid and also facet defining.

Merging two valid inequalities to generate another valid inequality is a new way to find useful cutting planes. This concept is the motivation to this thesis and the next chapter provides the theoretical foundations of merging a knapsack constraint into a cover inequality.

Chapter 3

Merging a Knapsack Constraint into a Cover Inequality

This chapter describes the theoretical advancement of this thesis. The first section introduces the concept of merging knapsack constraints with covers. This section also presents an algorithm developed in this research along with the proof of validity. The second section proposes two methods to exactly improve merged knapsack cover inequalities in order to make them stronger inequalities. The third section provides an example where the methodology of merging a knapsack constraint with covers is applied. In this example, the method produces an inequality that is shown to be facet defining. This demonstrates the power of merged knapsack cover inequalities when compared to exact sequential lifting and sequence independent lifting.

3.1 Merging Knapsacks with Covers Methodology

Given a knapsack constraint $\sum_{i \in N} a_i x_i \leq b$, let $C \subseteq N$ be a cover of the knapsack constraint and $M \subseteq N$ be a set of merging indices. A merged knapsack cover inequality, $MKC_{C,M,\alpha}$, takes the form $\sum_{i \in C \setminus M} x_i + \alpha \sum_{i \in M} a_i x_i \leq |C| - 1$.

Every $MKC_{C,M,\alpha}$ with $\alpha = 0$ is valid, but such an inequality is always dominated by the cover inequality. Additionally, if α is sufficiently large, then $MKC_{C,M,\alpha}$ is invalid. Furthermore, the larger the α value, the stronger the inequality.

This research presents the Merging Knapsack Cover Algorithm (MKCA), which determines both a set of merging indices $M \subseteq N$ and a value for α that guarantees the validity of the inequality. MKCA is based upon two subroutines: Determine the Set of Merging Indices (DSMI) and Calculate α Value (CAV). The DSMI subroutine identifies a set of merging variables that preserves the validity of the inequality, while the CAV subroutine finds a value for $\alpha > 0$.

The following DSMI subroutine identifies candidate indices to be considered for merging and guarantees that there exists a valid inequality with $\alpha > 0$. Input to DSMI is a knapsack constraint $\sum_{i \in N} a_i x_i \leq b$, a cover $C \subseteq N$ where $C = \{i_1, i_2, ..., i_{|C|}\}$, and a set of overlapping variables $M' \subseteq C$.

Determine the Set of Merging Indices Subroutine (DSMI)

$M \leftarrow \emptyset$

If $|M'| = 0$, *Then*

$\theta \leftarrow b - \sum_{j=2}^{|C|} a_{i_j}$

For $i \in N \setminus C$ *Do*

If $a_i > \theta$, Then

$M \leftarrow M \cup \{i\}$

End If

End For

End If

If $|M'| = 1$, *Then*

$\theta \leftarrow b - \sum_{j \in C \setminus M'} a_{i_j}$

For $i \in N \setminus (C \setminus M')$ *Do*

If $a_i > \theta$, *Then*

$M \leftarrow M \cup \{i\}$

End If

End For

End If

If $|M'| \geq 2$, *Then*

$M \leftarrow N \setminus (C \setminus M')$

End If

Report M

Calculating θ requires $O(|C|)$ whether $|M'| = 0$ or $|M'| = 1$. Assigning indices to the set M requires $O(n - |C|)$ regardless of whether the *If* condition is true. Thus, DSMI is a linear algorithm that requires $O(n)$ effort. Since the functionality of DSMI is not trivial, a proof is necessary. As shown in the next theorem, DSMI returns a set of indices M such that $MKC_{C,M,\alpha}$ is a valid inequality for some $\alpha > 0$.

Theorem 1. *Let* $\sum_{i \in N} a_i x_i \leq b$ *be a knapsack constraint,* $C \subseteq N$ *be a cover, and* $M' \subseteq C$ *be a set of overlapping variables. The DSMI subroutine returns a set of merging variables* M *such that* $MKC_{C,M,\alpha} = \sum_{i \in C \setminus M} x_i + \alpha \sum_{i \in M} a_i x_i \leq |C| - 1$ *is valid for* P^{ch}_{KP} *for some* $\alpha > 0$.

Proof: Given a knapsack constraint $\sum_{i \in N} a_i x_i \leq b$, a cover $C \subseteq N$, and a set of overlapping variables $M' \subseteq C$, let M be the set of merging variables returned from DSMI and $\alpha = \frac{1}{b}$. In order to show that $MKC_{C,M,\alpha}$ is valid for P^{ch}_{KP}, let x' be any point in P_{KP} and define $q = \sum_{i \in C \setminus M} x'_i$. Since C is a cover, $q \leq |C| - 1$.

First, assume $q \leq |C| - 2$. Applying x' to the left hand of the $MKC_{C,M,\alpha}$ inequality results in $q + \frac{1}{b}\left(\sum_{i \in N \setminus (C \setminus M)} a_i x'_i\right) \leq |C| - 2 + \frac{1}{b}\left(\sum_{i \in N \setminus (C \setminus M)} a_i x'_i\right)$. Since x' is feasible, $\sum_{i \in N \setminus (C \setminus M)} a_i x'_i \leq b$. Hence, $|C| - 2 + \frac{1}{b}\left(\sum_{i \in N \setminus (C \setminus M)} a_i x'_i\right) \leq |C| - 1$ and x' satisfies the $MKC_{C,M,\alpha}$ inequality.

Assume $q = |C| - 1$ and $|M'| = 1$. Thus, $x'_i = 1$ for every $i \in C \setminus M$. Since x' is feasible, $\sum_{i \in C \setminus M} a_i x'_i + \sum_{i \in M} a_i x'_i \leq b$. Therefore, $\sum_{i \in M} a_i x'_i \leq b - \sum_{i \in C \setminus M} a_i$. Since M is returned from DSMI, every $a_i > \theta$ where $\theta = b - \sum_{j \in C \setminus M'} a_{i_j}$. Thus, $x'_i = 0$ for every $i \in M$ and $\sum_{i \in C \setminus M} x'_i + \frac{1}{b}\left(\sum_{i \in M} a_i x'_i\right) = |C| - 1$.

Lastly, assume $q = |C| - 1$ and $|M'| = 0$. Since x' is feasible, $\sum_{i \in C} a_i x'_i + \sum_{i \in M} a_i x'_i \leq b$ and $\sum_{i \in M} a_i x'_i \leq b - \sum_{i \in C} a_i x'_i$. Due to the sorted order of C and $q = |C| - 1$, $\sum_{i \in C} a_i x'_i \geq \sum_{j=2}^{|C|} a_{i_j}$. Since M is returned from DSMI, every $a_i > \theta$ where $\theta = b - \sum_{j=2}^{|C|} a_{i_j}$. Thus, $x'_i = 0$ for every $i \in M$ and $\sum_{i \in C \setminus M} x'_i + \frac{1}{b}\left(\sum_{i \in M} a_i x'_i\right) = |C| - 1$. Hence, $MKC_{C,M,\alpha}$ is valid for P^{ch}_{KP} in this case. The cases are exhaustive and the result is shown.

□

One can see that a larger α leads to a stronger inequality. One of the challenges of this research is to select the right variables for merging such that $MKC_{C,M,\alpha}$ becomes a strong inequality, which would eliminate larger portions of the linear relaxation space.

The following CAV subroutine finds a value for α that creates a valid $MKC_{C,M,\alpha}$ inequality. The input to CAV subroutine is a knapsack constraint $\sum_{i \in N} a_i x_i \leq b$, a cover $C \subseteq N$ where $C = \{i_1, i_2, ..., i_{|C|}\}$, a set of merging indices $M \subseteq N$, and a set of overlapping variables $M' = C \cap M$. CAV does not assume that M comes from the DSMI subroutine, which guarantees $\alpha > 0$. The first few *If* conditions are included in case the only valid DMSI has $\alpha = 0$.

Calculate α Value Subroutine (CAV)

$\theta \leftarrow b$

$l \leftarrow |C| - |M'|$

$p \leftarrow |C| - |M'| + 1$

$\alpha \leftarrow \infty$

If $|M'| = 0$, *Then*

$p \leftarrow |C| - |M'| - 1$

If $min\{a_k : k \in M\} \leq b - \sum_{j=2}^{|C|} a_{i_j}$, *Then*

$\alpha \leftarrow 0$

End If

End If

If $|M'| = 1$, *Then*

$p \leftarrow |C| - |M'|$

If $min\{a_k : k \in M\} \leq b - \sum_{j \in C \setminus M'} a_{i_j}$, *Then*

$\alpha \leftarrow 0$

End If

End If

For $q = 1$ *to* p *Do*

$\alpha' \leftarrow \dfrac{|C| - q}{\theta}$

If $\alpha' < \alpha$, *Then*

$\alpha \leftarrow \alpha'$

End If

If $q < p$, *Then*

$\theta \leftarrow \theta - a_{i_l}$

$l \leftarrow l - 1$

End If

End For

Report α.

The first four operations of CAV require $O(1)$, and either of the next two *If* conditions require $O(n)$. The *For* loop repeats $q \leq (|C| - |M'|)$ times, and every step within the loop requires $O(1)$. Thus, the loop requires $O(|C|)$ effort, and CAV is on the order $O(n)$. The next theorem verifies that CAV returns a value for α such that $MKC_{C,M,\alpha}$ is a valid inequality.

Theorem 2. *Let* $\sum_{i \in N} a_i x_i \leq b$ *be a knapsack constraint,* $C \subseteq N$ *be a cover and* $M \subseteq N$ *be a set of merging variables. Then* $MKC_{C,M,\alpha'} = \sum_{i \in C \setminus M} x_i + \alpha' \sum_{i \in M} a_i x_i \leq |C| - 1$ *is a valid inequality of* P_{KP}^{ch} *for any* $\alpha' \leq \alpha$ *where* α *is returned from CAV.*

Proof: Given a knapsack constraint $\sum_{i \in N} a_i x_i \leq b$, a cover $C \subseteq N$ and a merging set $M \subseteq N$. For contradiction, assume that there exists an $\alpha' \leq \alpha$ such that $\sum_{i \in C \setminus M} x_i + \alpha' \sum_{i \in M} a_i x_i \leq |C| - 1$ is not a valid inequality of P_{KP}^{ch} where α is returned from CAV. Thus, there exists an $x' \in P_{KP}$ such that $\sum_{i \in C \setminus M} x'_i + \alpha' \sum_{i \in M} a_i x'_i > |C| - 1$.

Define $q = \sum_{i \in C \setminus M} x'_i$ and $C \setminus M = \{j_1, j_2, ..., j_{C \setminus M}\}$. Due to the feasibility of x', $\sum_{i \in C \setminus M} a_i x'_i + \sum_{i \in M} a_i x'_i \leq b$, and $\sum_{i \in M} a_i x'_i \leq b - \sum_{i \in C \setminus M} a_i x'_i$. Therefore, $\sum_{i \in M} a_i x'_i \leq b - \sum_{k=|C \setminus M|-q+1}^{|C \setminus M|} a_{j_k}$ due to the sets being sorted.

First, assume $q \leq |C| - 2$. Since $MKC_{C,M,\alpha'}$ is not a valid inequality, $\alpha' \sum_{i \in M} a_i x'_i > |C| - 1 - q$; therefore, $\alpha' > \dfrac{|C| - 1 - q}{\sum_{i \in M} a_i x'_i}$, resulting in $\alpha' > \dfrac{|C| - 1 - q}{\left(b - \sum_{k=|C \setminus M|-q+1}^{|C \setminus M|} a_{j_k}\right)}$. However, one can see that CAV requires $\alpha \leq \dfrac{|C| - 1 - q}{\left(b - \sum_{k=|C \setminus M|-q+1}^{|C \setminus M|} a_{j_k}\right)}$, which is a contradiction to α being returned from CAV.

Assume $q \geq |C| - 1$. If $q \geq |C|$, x' violates the cover inequality and contradicts C being a cover. If $q = |C| - 1$, then $\alpha' \sum_{i \in M} a_i x'_i > (|C| - 1) - (|C| - 1)$ implies that $\alpha' > 0$ and $x'_i = 1$ for some $i \in M$. Since $\alpha' > 0$, either $min\{a_k : k \in M\} > b - \sum_{j=2}^{|C|} a_{i_j}$ or $min\{a_k : k \in M\} > b - \sum_{j \in C \setminus M} a_{i_j}$ for $|M'| = 0$ and $|M'| = 1$, respectively. However, this contradicts the feasibility of x' and the result follows.

□

Based upon Theorem 2, CAV returns a value for α such that $MKC_{C,M,\alpha}$ is a valid inequality. In conclusion, both Theorem 1 and Theorem 2 proved that DSMI finds a set of merging variables M such that CAV is applied and finds a value for $\alpha > 0$ that generates a valid inequality $MKC_{C,M,\alpha} = \sum_{i \in C \setminus M} x_i + \alpha \sum_{i \in M} a_i x_i \leq |C| - 1$. As a result, the Merging Knapsack Cover Algorithm (MKCA) is presented. Input to MKCA is a knapsack constraint $\sum_{i \in N} a_i x_i \leq b$, a cover $C \subseteq N$, and a set of overlapping variables $M' \subseteq C$.

Merging Knapsack Cover Algorithm (MKCA)

$M \leftarrow$ Determine the Set of Merging Indices Subroutine (C, M')
$\alpha \leftarrow$ Calculate α Value Subroutine (C, M, M')
$MKC_{C,M,\alpha} \leftarrow \sum_{i \in C \setminus M} x_i + \alpha \sum_{i \in M} a_i x_i \leq |C| - 1$
Report $MKC_{C,M,\alpha}$

Clearly DSMI and CAV are both $O(n)$. Since reporting $MKC_{C,M,\alpha}$ is also $O(n)$, the MKCA runs in $O(n)$. Therefore, merged knapsack cover inequalities can be found in linear time.

Merged knapsack cover inequalities $MKC_{C,M,\alpha}$ generated by MKCA are valid inequalities that should be applied as cutting planes in order to cut off some linear relaxation solution in P^{LR}. However, depending on the value of α, merged knapsack cover inequalities can be strengthened. The next section provides two algorithms to improve α such that $MKC_{C,M,\alpha}$ are stronger inequalities.

3.2 Improving Merged Knapsack Cover Inequalities

This section introduces two algorithms to improve merged knapsack cover inequalities. This improvement is exact through both algorithms. The Exact Improvement Algorithm (EIA) guarantees the validity of the inequality by solving some knapsack problems, thereby potentially obtaining a larger value for α. The Exact Improvement Through Dynamic Programming Algorithm (EITDPA) uses dynamic programming to also potentially obtain a larger value for α. Both values for α are identical, but EITDPA typically runs with less computational effort.

The improvement process through EIA is briefly described in terms of θ. As demonstrated in the CAV subroutine, θ is calculated for each iteration of the *For* loop, and θ represents the maximum of $\sum_{i \in M} a_i$ that maintains feasibility. A lower θ leads to a larger α. Therefore, EIA maximizes the value of $z = \sum_{i \in M} a_i x_i$ such that $\sum_{i \in M} a_i x_i \leq \theta$ and $\sum_{i \in M} a_i x_i$. Thus, θ is replaced by z, potentially making $MKC_{C,M,\alpha}$ a stronger inequality.

The EIA provides an exact improvement for α and preserves the validity of $MKC_{C,M,\alpha}$. Input to EIA is a knapsack constraint $\sum_{i \in N} a_i x_i \leq b$, a cover $C \subseteq N$ where $C = \{i_1, i_2, ..., i_{|C|}\}$, a set of merging indices $M \subseteq N$, and a set of overlapping variables $M' = C \cap M$.

Exact Improvement Algorithm (EIA)

$\theta \leftarrow b$

$l \leftarrow |C| - |M'|$

$p \leftarrow |C| - |M'| + 1$

$\alpha \leftarrow \infty$

If $|M'| = 0$, *Then*

$p \leftarrow |C| - |M'| - 1$

If $min\{a_k : k \in M\} \leq b - \sum_{j=2}^{|C|} a_{i_j}$, *Then*

$\alpha \leftarrow 0$

End If

End If

If $|M'| = 1$, *Then*

$p \leftarrow |C| - |M'|$

If $min\{a_k : k \in M\} \leq b - \sum_{j \in C \setminus M'} a_{i_j}$, *Then*

$\alpha \leftarrow 0$

End If

End If

For $q = 1$ *to* p *Do*

Solve IP: Maximize $z = \sum_{i \in M} a_i x_i$

Subject to: $\sum_{i \in M} a_i x_i \leq \theta$

$x_i \in \{0, 1\} \forall i \in M$

$\alpha' \leftarrow \dfrac{|C| - q}{z}$

If $\alpha' < \alpha$, *Then*

$\alpha \leftarrow \alpha'$

End If

If $q < p$, *Then*

$\theta \leftarrow \theta - a_{i_l}$

$l \leftarrow l - 1$

End If

End For

Report α.

The first four operations of EIA require $O(1)$, and either of the next two *If* conditions require $O(n)$. The *For* loop repeats $q \leq (|C| - |M'|)$ times, and every step within the loop requires $O(1)$, with the exception of solving the knapsack problem that requires $O(S_{KP}(|M|))$ where $S_{KP}(n)$ is the time required to solve a knapsack problem with n variables. Therefore, EIA is on the order $O(n(S_{KP}(n)))$.

In order to prove the validity of EIA, the following theorem is presented with its proof. The theorem verifies that EIA returns a value for α that is at least as large as the α returned from CAV. In addition, it also proves that this is an exact algorithm because the α returned from EIA cannot be increased and still have the inequality remain valid.

Theorem 3. *Let* $\sum_{i \in N} a_i x_i \leq b$ *be a knapsack constraint,* $C \subseteq N$ *be a cover,* $M \subseteq N$ *be a set of merging variables and* α *returned from EIA. Then,* $MKC_{C,M,\alpha'} = \sum_{i \in C \setminus M} x_i + {} + \alpha' \sum_{i \in M} a_i x_i \leq |C| - 1$ *is a valid inequality of* P_{KP}^{ch} *for any* $\alpha' \leq \alpha$ *and is not a valid inequality of* P_{KP}^{ch} *for any* $\alpha' > \alpha$.

Proof: Given a knapsack constraint $\sum_{i \in N} a_i x_i \leq b$, a cover $C \subseteq N$, a set of merging variables $M \subseteq N$, and α returned from EIA. Define q' to be the value of q, θ' to be the value of θ, and z' to be the value of z when EIA calculates α reported by this algorithm. Also, consider the inequality $MKC_{C,M,\alpha'} = \sum_{i \in C \setminus M} x_i + \alpha' \sum_{i \in M} a_i x_i \leq |C| - 1$.

Assume $\alpha' > \alpha$ and let x^* be the optimal x solution of *Maximize* $z^* = \sum_{i \in M} a_i x_i$, *Subject to:* $\sum_{i \in M} a_i x_i \leq \theta'$, $x_i \in \{0, 1\}$ for all $i \in M$. Since $\theta' = b - \sum_{k=|C \setminus M|-q'+1}^{|C \setminus M|} a_{j_k}$ where $C \setminus M = \{j_1, j_2, ..., j_{C \setminus M}\}$, thus $x^* + \sum_{k=|C \setminus M|-q'+1}^{|C \setminus M|} \xi_{j_k} \in P_{KP}$. Applying this point to the left-hand side of $MKC_{C,M,\alpha'}$ inequality results in $\sum_{i \in C \setminus M} x_i^* + \alpha' \sum_{i \in M} a_i x_i^* = q' + \alpha' z^* > q' + \alpha z^* = = |C| - 1$. Therefore, the inequality $\sum_{i \in C \setminus M} x_i^* + \alpha' \sum_{i \in M} a_i x_i^* \leq |C| - 1$ is invalid.

Next, assume $\alpha' \leq \alpha$ and let $x'' \in P_{KP}$. Define $q'' = \sum_{i \in C \setminus M} x_i''$ and define z'' to be the value of z from EIA when $q = q''$. Due to the optimality of the integer program, $\sum_{k=|C \setminus M|-q''+1}^{|C \setminus M|} a_{j_k} x_{j_k}'' \leq z''$. Applying x'' into the left-hand side of $MKC_{C,M,\alpha'}$ inequality results in $\sum_{i \in C \setminus M} x_i'' + \alpha' \sum_{i \in M} a_i x_i'' \leq q'' + \alpha' z'' \leq q'' + \alpha z''$. Since α is returned from EIA, $\alpha \leq \dfrac{|C| - 1 - q''}{z''}$. Substituting results in $q'' + \alpha z'' \leq q'' + \left(\dfrac{|C| - 1 - q''}{z''} \right) z'' = |C| - 1$. Thus, every $x'' \in P_{KP}$ satisfies the inequality and the result is shown.

□

Solving n knapsack instances with the branch and bound algorithm is cumbersome. Use of dynamic programming allows these optimization problems be solved in pseudo-polynomial time. This improvement is formalized in the Exact Improvement Through Dynamic Programming Algorithm (EITDPA).

The primary difference between EITDPA and EIA is the method with which KP is solved. EITDPA uses dynamic programming to determine all possible integers that can be achieved by feasible points restricted to the indices in M. An array d is tracked and if the sum of the coefficients of any combination of feasible points with indices in M is equal to i, then d_i is set to 1. Once θ is determined, z is calculated as the index $i \leq \theta$ where $d_i = 1$.

Input to EITDPA is a knapsack constraint $\sum_{i \in N} a_i x_i \leq b$, a cover $C \subseteq N$ where $C = \{i_1, i_2, ..., i_{|C|}\}$, a set of merging indices $M \subseteq N$ where $M = \{y_1, y_2, ..., y_{|M|}\}$, and a set of overlapping variables $M' = C \cap M$.

Exact Improvement Through Dynamic Programming Algorithm (EITDPA)

$d_0 \leftarrow 1$
$\theta \leftarrow b$
$l \leftarrow |C| - |M'|$
$p \leftarrow |C| - |M'| + 1$
$\alpha \leftarrow \infty$
For $q = 1$ *to* $|M|$ *Do*
  For $r = b - a_{y_q}$ *to* 0 *Do*
    If $d_r = 1$, *Then*
      $d_{r+a_{y_q}} \leftarrow 1$
    End If
  End For
End For
If $|M'| = 0$, *Then*
  $p \leftarrow |C| - |M'| - 1$
  If $min\{a_k : k \in M\} \leq b - \sum_{j=2}^{|C|} a_{i_j}$, *Then*

$\alpha \leftarrow 0$

End If

End If

If $|M'| = 1$, *Then*

$p \leftarrow |C| - |M'|$

If $min\{a_k : k \in M\} \leq b - \sum_{j \in C \setminus M'} a_{i_j}$, *Then*

$\alpha \leftarrow 0$

End If

End If

For $q = 1$ *to* p *Do*

$flag \leftarrow 0$

$t \leftarrow \theta$

While $flag = 0$ *and* $t \geq 0$ *Do*

If $d_t = 1$, *Then*

$z \leftarrow t$

$flag \leftarrow 1$

End If

$t \leftarrow t - 1$

End While

$\alpha' \leftarrow \dfrac{|C| - q}{z}$

If $\alpha' < \alpha$, *Then*

$\alpha \leftarrow \alpha'$

End If

If $q < p$, *Then*

$\theta \leftarrow \theta - a_{i_l}$

$l \leftarrow l - 1$

End If

End For

Report α.

The first five operations of EITDPA require $O(1)$. The first *For* loop repeats $q \leq |M|$ times, the next *For* loop repeats $r \leq b$ times, and every step within the loop requires $O(1)$. Thus, this first part of the algorithm requires $O(b|M|)$. Either of the next two *If* conditions require $O(n)$. The next sequence of loops run in $O(bn)$ since p is bounded by n. Therefore, EITDPA is on the order $O(bn)$, its running time is a function of the input data, and it is a pseudo-polynomial time algorithm. Thus, EITDPA can be solved in polynomial time if the right-hand side b is also polynomial.

EITDPA solves the IP from EIA by determining all possible integers that can be achieved by any combination of points restricted to the indices in M. The array d tracks these integers by marking the appropriate index with a 1. EITDPA obtains z from EIA by finding the maximum index $i \leq \theta$ where $d_i = 1$. Since there does not exist a point in P_{KP} with a value larger than z, Theorem 3 is trivially extended to EITDPA. The following corollary formally presents this argument, thereby making EIA and EITDPA equivalent in terms of final result.

Corollary 1. *Let* $\sum_{i \in N} a_i x_i \leq b$ *be a knapsack constraint,* $C \subseteq N$ *be a cover, and* $M \subseteq N$ *be a set of merging variables. EITDPA returns the same value for* α *as EIA and* $MKC_{C,M,\alpha} = \sum_{i \in C \setminus M} x_i + \alpha \sum_{i \in M} a_i x_i \leq |C| - 1$ *is an exact valid merged knapsack cover inequality.*

□

From a theoretical standpoint, EIA is essential to understand the concept of improving merged knapsack cover inequalities and the benefits $MKC_{C,M,\alpha}$ inequalities bring to IP

research. However, less computational effort required by EITDPA makes this algorithm more appropriate. The next section describes an example of how merged knapsack cover inequalities are generated by MKCA and potentially improved through EITDPA.

3.3 Merging Knapsacks with Covers Example

Consider the first knapsack constraint from the multiple knapsack problem presented in Section 2.2.1. Let $x_i \in \{0, 1\}$ for $i \in \{1, 2, ..., 11\}$ and the knapsack constraint is presented in Equation 3.1.

$$30x_1 + 25x_2 + 20x_3 + 15x_4 + 12x_5 + 11x_6 + 11x_7 + 10x_8 + 10x_9 + 5x_{10} + x_{11} \leq 44. \quad (3.1)$$

Define $C = \{5, 6, 7, 8, 9\}$ as a minimal cover and $M' = \emptyset$. MKCA initially determines the set of merging variables through the DSMI subroutine. The first step calculates $\theta = b - \sum_{j=2}^{|C|} a_{i_j}$ where $C = \{i_1, i_2, ..., i_{|C|}\}$. Thus, $\theta = b - a_6 - a_7 - a_8 - a_9 = 2$. In order to determine the set of merging indices, $a_i > \theta$ for all $i \in N \setminus C$. As a result, all remaining variables are selected for merging except x_{11}. Therefore, the set of merging indices becomes $M = \{1, 2, 3, 4, 10\}$ and the $MKC_{C,M,\alpha}$ inequality is shown in Equation 3.2.

$$\alpha(30x_1 + 25x_2 + 20x_3 + 15x_4 + 5x_{10}) + x_5 + x_6 + x_7 + x_8 + x_9 \leq 4. \quad (3.2)$$

This $MKC_{C,M,\alpha}$ inequality follows Theorem 1, and the set $M = \{1, 2, 3, 4, 10\}$ guarantees that the inequality is valid for some $\alpha > 0$. A value for α is calculated using the CAV

subroutine. This subroutine begins with $\theta = 44$, $k = 5$, and $\alpha = \infty$. As $|M'| = 0$ and $min\{a_k : k \in M\} > b - \sum_{j=2}^{|C|} a_{i_j}$, the subroutine skips the two first *If* conditions. Thus, α' is defined by $\frac{|C| - q}{\theta}$, and $\alpha' = \frac{5-1}{44} = \frac{4}{44}$. Since $\alpha' < \alpha$, $\alpha = \frac{4}{44}$. Following the subroutine, $\theta = 44 - 10 = 34$ and $k = 5 - 1 = 4$ are updated. The new α' is calculated by $\alpha' = \frac{5-2}{34}$. Again, $\alpha' < \alpha$ and α is updated to $\frac{3}{34}$. Repeating this process $|C| - |M'| - 1 = 4$ times, the next two values for α' are $\frac{5-3}{34-10} = \frac{2}{24}$ and $\frac{5-4}{24-11} = \frac{1}{13}$. The CAV subroutine terminates with $\alpha = \frac{1}{13}$, and the merged knapsack cover inequality $MKC_{C,M,\frac{1}{13}}$ is described in Equation 3.3.

$$\frac{1}{13}(30x_1 + 25x_2 + 20x_3 + 15x_4 + 5x_{10}) + x_5 + x_6 + x_7 + x_8 + x_9 \leq 4. \tag{3.3}$$

Based upon Theorem 2, CAV returns $\alpha = \frac{1}{13}$, and $MKC_{C,M,\frac{1}{13}}$ is a valid inequality. The usefulness of this merged knapsack cover inequality is verified by determining whether $MKC_{C,M,\frac{1}{13}}$ cuts off some linear relaxation solution. Consider the linear relaxation solution given by the point (0,0,0,$\frac{2}{15}$,0,1,1,1,1,1,0,0). Applying this point to $MKC_{C,M,\frac{1}{13}}$ gives $\frac{30}{13}(0) + \frac{25}{13}(0) + \frac{20}{13}(0) + \frac{15}{13}\left(\frac{2}{15}\right) + 1(0) + 1(1) + 1(1) + 1(1) + 1(1) + \frac{5}{13}(0) = \frac{54}{13}$. Since $\frac{54}{13} > 4$, $MKC_{C,M,\frac{1}{13}}$ cuts off this linear relaxation solution and may be helpful in solving the integer program.

Potential improvements to this merged knapsack cover inequality can still be verified using EIA or EITDPA. Because both algorithms return the same result as stated in Corollary 1 and EITDPA runs in polynomial time, since $b = 44$, this algorithm is used to demonstrate how $MKC_{C,M,\frac{1}{13}}$ is improved.

The first part of EITDPA assumes the array d of size 44 where $d_i = 0$ for all $i \leq 44$. The algorithm begins with $d_0 = 1$, $\theta = 44$, $k = 5$, $p = 5$, and $\alpha = \infty$. The set M is $\{1, 2, 3, 4, 10\}$, and the variable x_1 has a coefficient equal to 30. Since $b - a_1 = 44 - 30 = 14$, going through all indexes of array d, starting at index 14 until index 0, the only index $d_i = 1$ for all $i \leq 14$ is index 0 and d_{0+30} is set to 1.

The next two iterations follow the same process, and d_{0+25} and d_{0+20} are both set to 1. At the fourth iteration the array d begins to change a little more. The fourth iteration considers the variable x_4 with coefficient equal to 15. Because $b - a_4 = 44 - 15 = 29$, going through all indexes of array d, starting at index 29 until index 0, then $d_i = 1$ for indexes 25, 20, and 0. Therefore, d_{25+15}, d_{20+15}, and d_{0+15} are set to 1. The next iteration assumes variable x_{10} with coefficient equal to 5, $b - a_{10} = 44 - 5 = 39$, and looking at all indexes of array d, starting at index 39 until index 0, $d_i = 1$ for indexes 35, 30, 25, 20, 15, and 0. Thus, d_{35+5}, d_{30+5}, d_{25+5}, d_{20+5}, and d_{15+5} are reset to 1, and d_{0+5} is set to 1 for the first time. This process is summarized in Table 3.1, which provides all indexes of array d.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
15	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**
1	0	0	0	0	1	0	0	0	0	1	0	0	0	0
30	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**
1	0	0	0	0	1	0	0	0	0	1	0	0	0	0

Table 3.1: Final d - Example $|M'| = 0$

The second part of EITDPA has $|M'| = 0$ and $min\{a_k : k \in M\} > b - \sum_{j=2}^{|C|} a_{i_j}$, so the algorithm skips the two first *If* conditions. Therefore, variables $flag = 0$ and $\theta = 44$.

Starting at index $\theta = 44$, $d_{44} = 0$, and the algorithm moves to index 43. Since $d_{43} = 0$, the algorithm jumps to the next index, continuing until it finds $d_{40} = 1$. Thus, $z = 40$, $flag = 1$, meaning that the loop is terminated, and $\alpha' = \frac{4}{40}$. Since $\alpha' < \alpha$, $\alpha = \frac{1}{10}$. The algorithm updates $\theta = 44 - 10 = 34$, $k = 5 - 1 = 4$, and variable $flag$ is reset to 0.

Beginning at index $\theta = 34$ of array d, $d_{34} = 0$ and the algorithm moves until it finds $d_{30} = 1$. The variable $z = 30$ and $flag = 1$ are updated, and the loop is terminated again. Therefore, $\alpha' = \frac{3}{30}$ and $\alpha = \frac{1}{10}$. The algorithm is updated with $\theta = 34 - 10 = 24$ and $k = 4 - 1 = 3$. The algorithm repeats $|C| - |M'| - 1 = 4$ times and the next two α' are $\frac{2}{20}$ and $\frac{1}{5}$. EITDPA terminates with $\alpha = \frac{1}{10}$. The merged knapsack cover inequality exactly improved $MKC_{C,M,\frac{1}{10}}$ is shown in Equation 3.4. A simplified $MKC_{C,M,\frac{1}{10}}$ inequality is presented in Equation 3.5.

$$\frac{1}{10}(30x_1 + 25x_2 + 20x_3 + 15x_4 + 5x_{10}) + x_5 + x_6 + x_7 + x_8 + x_9 \leq 4 \tag{3.4}$$

$$3x_1 + \frac{5}{2}x_2 + 2x_3 + \frac{3}{2}x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + \frac{1}{2}x_{10} \leq 4 \tag{3.5}$$

$MKC_{C,M,\frac{1}{10}}$ inequality dominates the $MKC_{C,M,\frac{1}{13}}$ inequality. Furthermore, $MKC_{C,M,\frac{1}{10}}$ is facet defining. To prove this, observe the $dim(P_{KP}^{ch}) = 11$ and $MKC_{C,M,\frac{1}{10}}$ is a valid inequality based on Corollary 1. Thus, one must prove the dimension of its face F is 10. First, this face is not the entire P_{KP}^{ch} space because the point $(0,0,0,0,0,0,0,0,0,0,0) \in P_{KP}$ and $0+0+0+0+0+0+0+0+0+0+0 < 4$. Therefore, $dim(F) \leq dim(P_{KP}^{ch}) - 1$, so $dim(F) \leq 10$. In addition, 11 affinely independent points that meet $MKC_{C,M,\frac{1}{10}}$ at equality are shown in Figure 3.1.

x_1	0	0	0	0	0	1	0	0	0	0	0
x_2	0	0	0	0	0	0	0	1	1	0	0
x_3	0	0	0	0	0	0	1	0	0	0	0
x_4	0	0	0	0	0	0	0	1	0	1	0
x_5	0	1	1	1	1	0	0	0	0	0	0
x_6	1	0	1	1	1	0	0	0	0	0	1
x_7	1	1	0	1	1	0	0	0	0	0	1
x_8	1	1	1	0	1	0	1	0	0	1	1
x_9	1	1	1	1	0	1	1	0	1	1	1
x_{10}	0	0	0	0	0	0	0	0	1	1	0
x_{11}	0	0	0	0	0	0	0	0	0	0	1

Figure 3.1: Affinely Independent Points - $MKC_{C,M,\frac{1}{10}}$

As shown in Figure 3.1, a cyclical permutation of the first five points allows the minimal cover to be facet defining in the restricted space. Clearly the points $(1,0,0,0,0,0,0,0,1,0,0)$, $(0,0,1,0,0,0,0,1,1,0,0)$, and $(0,0,0,0,0,1,1,1,1,0,1)$ are affinely independent since variables x_1, x_3, and x_{11} are set to 1, respectively, and all the other variables set to 1 are canceled by the previously described affinely independent points. The other three remaining points are also affinely independent because of the cyclical permutation between variables x_2, x_4, and x_{10}. Therefore, $dim(F) \geq 10$. Consequently, $MKC_{C,M,\frac{1}{10}}$ is a facet defining inequality and $dim(F) = 10$.

Usefulness of $MKC_{C,M,\frac{1}{10}}$ is verified by determining whether this cuts off some linear relaxation solution. The point (0,0,0,$\frac{2}{15}$,0,1,1,1,1,0,0) is in P^{LR}, and application of this point to $MKC_{C,M,\frac{1}{10}}$ gives $3(0)+\frac{5}{2}(0)+2(0)+\frac{3}{2}\left(\frac{2}{15}\right)+1(0)+1(1)+1(1)+1(1)+1(1)+\frac{1}{2}(0) = \frac{21}{5}$. Since $\frac{21}{5} > 4$, the inequality $MKC_{C,M,\frac{1}{10}}$ cuts off this linear relaxation point; therefore, it is a cutting plane.

In addition to providing a facet defining inequality, this example also demonstrates that $MKC_{C,M,\alpha}$ inequalities are a new class of previously unattainable inequalities. The two most similar methods to this thesis research are sequential and sequence independent lifting.

Consider the first knapsack constraint from Section 2.2.1 and the minimal cover $C = \{5, 6, 7, 8, 9\}$. The exact sequential up lifting inequality for this problem when variables are lifted in ascending order is given in Equation 3.6. For simple comparison, $MKC_{C,M,\frac{1}{13}}$ and $MKC_{C,M,\frac{1}{10}}$ are presented in Equations 3.7 and 3.8, respectively. Due to the sequential lifting method, the coefficients of exact sequential up lifting inequality are always integers. Thus, exact sequential up lifting could never create an $MKC_{C,M,\frac{1}{10}}$ inequality.

$$3x_1 + 3x_2 + 2x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 \leq 4 \tag{3.6}$$

$$\frac{30}{13}x_1 + \frac{25}{13}x_2 + \frac{20}{13}x_3 + \frac{15}{13}x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + \frac{5}{13}x_{10} \leq 4 \tag{3.7}$$

$$3x_1 + \frac{5}{2}x_2 + 2x_3 + \frac{3}{2}x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + \frac{1}{2}x_{10} \leq 4 \tag{3.8}$$

Sequence independent lifting, one of the most related techniques in this field, was introduced by Gu *et al.* [34]. This method provides a valid superadditive lifting function that obtains a good approximation to maximum lifting. The research of Gu *et al.* is closely related to merging knapsack constraints with covers because the α coefficients are related to the size of the knapsack coefficients.

For simplicity, the necessary algorithm to build a superadditive valid lifting function is not presented in this thesis and only the output is shown. Consider the first knapsack

constraint from Section 2.2.1 and the minimal cover $C = \{5, 6, 7, 8, 9\}$. The sequence independent lifting inequality is given in Equation 3.9, and $MKC_{C,M,\frac{1}{13}}$ and $MKC_{C,M,\frac{1}{10}}$ inequalities are presented in Equations 3.10 and 3.11, respectively, for comparison.

$$\frac{25}{9}x_1 + \frac{20}{9}x_2 + \frac{16}{9}x_3 + \frac{11}{9}x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + \frac{1}{3}x_{10} \leq 4 \tag{3.9}$$

$$\frac{30}{13}x_1 + \frac{25}{13}x_2 + \frac{20}{13}x_3 + \frac{15}{13}x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + \frac{5}{13}x_{10} \leq 4 \tag{3.10}$$

$$3x_1 + \frac{5}{2}x_2 + 2x_3 + \frac{3}{2}x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + \frac{1}{2}x_{10} \leq 4 \tag{3.11}$$

The sequence independent lifting inequality has coefficients of variables x_1, x_2, x_3, and x_4 larger than $MKC_{C,M,\frac{1}{13}}$ inequality and a coefficient of variable x_{10} smaller than $MKC_{C,M,\frac{1}{13}}$. However, with the exception of the cover inequality variables, all variables in $MKC_{C,M,\frac{1}{10}}$ inequality have coefficients larger than the sequence independent lifting inequality, meaning that $MKC_{C,M,\frac{1}{10}}$ dominates this inequality. Clearly, the sequence independent lifting inequality is not facet defining, and $MKC_{C,M,\frac{1}{10}}$ is a stronger inequality. Therefore, merging knapsack constraints with covers may be better than sequence independent lifting.

Various other methods are available to find valid inequalities. Arguments are not given in this thesis, but simple application of disjunctive cuts [10], Chvátal Gomory cuts [19], mixed integer rounding cuts [29], superadditive cuts [69], or modular arithmetic cuts [30] would not have generated merged knapsack cover inequalities. Consequently, $MKC_{C,M,\alpha}$ inequalities are a new class of valid inequalities for the knapsack polytope.

To further demonstrate $MKC_{C,M,\alpha}$ technology, consider a new problem with the first knapsack constraint from Section 2.2.1, the cover $C = \{5, 6, 7, 8, 9\}$, and $M' = \{9\}$. Following the DSMI subroutine, all remaining variables are merged, $M = \{1, 2, 3, 4, 9, 10, 11\}$, and the new merged knapsack cover inequality $MKC_{C,M,\alpha}$ is presented in Equation 3.12.

$$\alpha(30x_1 + 25x_2 + 20x_3 + 15x_4 + 10x_9 + 5x_{10} + x_{11}) + x_5 + x_6 + x_7 + x_8 \leq 4 \qquad (3.12)$$

A value for α is calculated to this new $MKC_{C,M,\alpha}$ inequality using the CAV subroutine. The subroutine can be followed $|C| - |M'| = 4$ times, and α' is $\frac{4}{44}$, $\frac{3}{34}$, $\frac{2}{23}$, and $\frac{1}{12}$ for each iteration. CAV terminates with $\alpha = \frac{1}{12}$. The merged knapsack cover inequality $MKC_{C,M,\frac{1}{12}}$ is shown in Equation 3.13.

$$\frac{1}{12}(30x_1 + 25x_2 + 20x_3 + 15x_4 + 10x_9 + 5x_{10} + x_{11}) + x_5 + x_6 + x_7 + x_8 \leq 4 \qquad (3.13)$$

This $MKC_{C,M,\frac{1}{12}}$ inequality can also be improved using EITDPA. For simplicity, only the final output is presented and Table 3.2 shows all indexes of array d. The EITDPA terminates with $\alpha = \frac{1}{11}$. The new $MKC_{C,M,\frac{1}{11}}$ inequality exactly improved is described in Equation 3.14.

$$\frac{1}{11}(30x_1 + 25x_2 + 20x_3 + 15x_4 + 10x_9 + 5x_{10} + x_{11}) + x_5 + x_6 + x_7 + x_8 \leq 4 \qquad (3.14)$$

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
1	1	0	0	0	1	1	0	0	0	1	1	0	0	0
15	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**
1	1	0	0	0	1	1	0	0	0	1	1	0	0	0
30	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**
1	1	0	0	0	1	1	0	0	0	1	1	0	0	0

Table 3.2: Final d - Example $|M'| = 1$

$MKC_{C,M,\frac{1}{11}}$ is a new valid inequality. Applying the point (0,0,0,$\frac{1}{15}$,0,1,1,1,1,0,1) to $MKC_{C,M,\frac{1}{11}}$ gives $\frac{30}{11}(0) + \frac{25}{11}(0) + \frac{20}{11}(0) + \frac{15}{11}\left(\frac{1}{15}\right) + 1(0) + 1(1) + 1(1) + 1(1) + \frac{10}{11}(1) + + \frac{5}{11}(0) + \frac{1}{11}(1) = \frac{45}{11}$. Since $\frac{45}{11} > 4$, $MKC_{C,M,\frac{1}{11}}$ cuts off the linear relaxation solution and is a cutting plane.

The preceding examples present two different valid and useful inequalities. The cover inequality is identical in both cases, but the number of merging variables, the initial α, and improved α values differ. In the example presented, $MKC_{C,M,\frac{1}{10}}$ is theoretically more useful than $MKC_{C,M,\frac{1}{11}}$ because it cuts off a larger space of the linear relaxation region. However, no conclusions regarding which inequality should be implemented can be made without a reasonable computational study. Chapter 4 describes a computational study to evaluate the effectiveness of merged knapsack cover inequalities.

Chapter 4

Computational Study

This chapter introduces a computational study performed to validate the usefulness of merged knapsack cover inequalities. The time required to solve multiple knapsack problems with and without these inequalities is compared. The chapter first presents a method used to generate random multiple knapsack instances. Computational implementation of merged knapsack cover inequalities is described along with the method used to generate cover inequalities. The importance of selecting strong cover inequalities to generate useful merged knapsack cover inequalities is also discussed. Two computational results are shown and analyzed.

This computational study was performed on an Intel® Core^{TM} i7-4770 3.4GHz processor with 8 GB of RAM. The study was completed using CPLEX 12.5 [40] and coded in C++ using Microsoft Visual Studio [55]. Two relevant measures were tracked in this study: the time required to solve the problem in seconds and the number of ticks in CPLEX. The number of ticks represents the number of operations used by CPLEX to solve the problem.

The number of ticks is independent of the computer on which the algorithm runs, thereby allowing a true comparison between various computational studies. However, the ticks depend on which CPLEX version is used. Many multiple knapsack problems require an intensive computational effort; therefore, the node files are stored in the hard drive instead of RAM in this computational study to avoid out of memory issues. The remainder of CPLEX's settings are at default.

4.1 Generating Random Instances

Multiple knapsack instances are randomly generated in this thesis. Let $c \in \mathbb{R}^n$, $A \in \mathbb{R}_+^{m \times n}$, and $b \in \mathbb{R}_+^m$. Thus, each instance takes the form:

$$\begin{aligned} \text{Maximize} \quad & c^T x \\ \text{Subject to:} \quad & Ax \leq b \\ & x \in \{0,1\}. \end{aligned}$$

The matrix A is integer and is generated according to a random uniform distribution between l and u. The right-hand side vector b is defined as $b_i = \left\lfloor \beta \sum_{j=1}^{n} a_{i,j} \right\rfloor$ for each $i \in \{1, ..., m\}$ where $\beta = 0.25$. The cost vector c is defined as $c_j = \sum_{i=1}^{m} a_{i,j} + \lfloor 500\gamma_j \rfloor$ where γ_j is a uniform random number between 0 and 1, for each $j \in \{1, ..., n\}$. All random numbers are generated through the Linear Congruential Generator algorithm (LCG) proposed by Lehmer [53] and the seed value is changed for each instance. Notice that each multiple knapsack instance is generated in polynomial time ($O(nm)$).

This computational study primarily focuses on problems with 5 and 10 constraints and 60, 80, and 100 variables. For each problem, 15 instances are tested to avoid random anomalies in any one instance.

The next section provides the relevant information about the computational implementation of merged knapsack cover inequalities. The section discusses the main technical aspects of this implementation and also includes a discussion on the importance of selecting strong cover inequalities.

4.2 Computational Implementation

The main goal of this computational study is to compare the solution time of multiple knapsack problems with and without implementation of merged knapsack cover inequalities. For each instance, the problem is first solved using CPLEX at default settings.

For the number of variables overlapped in this computational study, it is first assumed that at least half of the variables of the cover inequality should remain unmerged in order to maintain some properties of the cover inequality. Due to the intensive computational effort to solve some multiple knapsack instances, it is infeasible to overlap each possible set of variables and have all the results for comparison in a reasonable amount of time. Considering both constraints, the most reasonable experiment would identify the effectiveness of computational results when no variable is overlapped and when some few of them are overlapped. This results in overlapping a maximum of two variables in this computational study, concluding $|M'| = 0$, $|M'| = 1$, and $|M'| = 2$. It is assumed that the overlapping variables are those with the highest $a_{i,j}$ coefficients.

Adding multiple useful cutting planes to the problem may result in removing more of the linear relaxation space. Since these instances have so few constraints, increasing the size of matrix A may slow the computational processing and the results can be worse. Therefore, only one merged knapsack cover inequality is added to determine whether these inequalities are computationally effective.

Merged knapsack cover inequalities are generated by merging variables of one knapsack constraint into a cover inequality of the same knapsack constraint. Assume that only one cover inequality is considered in each knapsack constraint. Since there exist m constraints in every multiple knapsack problem, a variation of $3m$ merged knapsack cover inequalities can be implemented to each problem since overlapping zero, one, and two variables represents a different cutting plane. Because instances are randomly generated, there are no known theoretical foundations that prove merged knapsack cover inequalities of one knapsack constraint are better than merged knapsack cover inequalities of other knapsack constraints. Therefore, only merged knapsack cover inequalities generated from the first knapsack constraint are implemented and compared.

Identification of strong cover inequalities is critical to this research in terms of both theoretical and computational results. Strong cover inequalities are typically those that are violated or almost violated by the linear relaxation solution. Cover inequalities that are violated by the linear relaxation solution by the greatest extent are preferable. To better find a cover inequality in this computational study, the algorithm developed by Gu *et al.* [32] with some modifications is implemented.

Given a knapsack constraint $\sum_{i \in N} a_i x_i \leq b$, the optimal linear relaxation solution $x_i^* \in P^{LR}$, and the reduced cost d_i, define the ratio $r_i = \frac{d_i}{a_i} + x_i$ for all $i \in N$. Sort all variables x_i in non-increasing value of ratio r_i. Define the cover C by taking the indices in this sorted order until $\sum_{i \in C} a_i > b$. Determine whether the cover C is minimal. If not, remove indices from the cover C starting with the highest a_i until it is a minimal cover.

According to the brief method description, one can see that the minimal cover inequality reported has a reasonable probability of eliminating the optimal linear relaxation solution since the variables where $x_i^* = 1$ are selected first, followed by the variables where $0 < x_i^* < 1$, and finally $x_i^* = 0$. The next section describes computational results and analysis.

4.3 Computational Results and Discussions

This section presents and discusses the main computational results, which contains two computational studies. The first study examines random knapsack instances that mimic some standard benchmark instances. In these instances, EITDPA does not provide any improvement. A second computational study examines the types of multiple knapsack instances where EITDPA does improve the α coefficient. The section concludes with a final discussion that summarizes the results.

4.3.1 Computational Results Without EITDPA

A standard method to create difficult multiple knapsack instances is to have the $a_{i,j}$ be randomly distributed between 1 and $1,000$ as proposed by Chu and Beasley [18] and can be found in the OR-Library [11]. Thus, this computational study sets $l = 1$ and $u = 1,000$.

Table 4.1 provides the time in seconds and the number of ticks when the problems are solved using CPLEX at default settings and when merged knapsack cover inequalities are implemented with zero, one, and two overlapping variables. The problem column denotes the number of rows and variables. All times are listed in seconds and are the average of the 15 instances. A majority of the problems are solved in approximately 40 minutes on average. Table 4.2 shows the percent improvement.

	CPLEX		$\|M'\| = 0$		$\|M'\| = 1$		$\|M'\| = 2$	
Problem	**Time**	**# Ticks**	**Time**	**# Ticks**	**Time**	**# Ticks**	**Time**	**# Ticks**
5×60	95	10831	100	11511	100	11330	92	10810
5×80	162	10631	161	10778	164	10551	180	11788
5×100	984	137398	994	145297	1002	145623	1054	152146
10×60	4200	1297456	4171	1309254	4308	1336215	4254	1319230
10×80	4227	1445558	4456	1563998	4445	1509567	4461	1489726
10×100	5140	1645558	5253	1658704	5321	1660676	5268	1658310
Average	**2468**	**757905**	**2523**	**783257**	**2557**	**778993**	**2552**	**773668**

Table 4.1: Results - Time and Ticks with $l = 1$ and $u = 1,000$

The average time when no variable is overlapped is 2.1% worse and the average number of ticks is 3.9% worse. When one variable is overlapped, the average time is 3.9% worse and the average number of ticks is 3.0% worse. In addition, when two variables are overlapped, the average time is 4.0% worse and the average number of ticks is 4.5% worse.

	$\|M'\| = 0$		$\|M'\| = 1$		$\|M'\| = 2$	
Problem	**Time**	**# Ticks**	**Time**	**# Ticks**	**Time**	**# Ticks**
5×60	-5.3	-6.3	-5.3	-4.6	3.2	0.2
5×80	0.6	-1.4	-1.2	0.8	-11.1	-10.9
5×100	-1.0	-5.7	-1.8	-6.0	-7.1	-10.7
10×60	0.7	-0.9	-2.6	-3.0	-1.3	-1.7
10×80	-5.4	-8.2	-5.2	-4.4	-5.5	-3.1
10×100	-2.2	-0.8	-3.5	-0.9	-2.5	-0.8
Avg. Imp. %	**-2.1**	**-3.9**	**-3.9**	**-3.0**	**-4.0**	**-4.5**

Table 4.2: Results - Percent Improvement with $l = 1$ and $u = 1,000$

Most of the results presented in Table 4.2 are worse than CPLEX results. Regarding the 90 problems tested, when no variable is overlapped, 31.1% of them presents improvements in time and 23.3% presents improvements in number of ticks. When only one variable is overlapped, 31.1% are shown to be better in time and 28.9% are better in number of ticks. When two variables are overlapped, 27.7% of the problems presents improvements in time and 30.0% presents improvements in number of ticks.

One surprising result is that EITDPA did not ever improve the α coefficient, which means that α initially generated by MKCA remains the same. This statement is well explained by the research developed by Hunsaker and Tovey [39]. For numerous random knapsack instances, there is a probability approaching 1 that there exists a set of variables that sum exactly to b. Thus, in the majority of random knapsack instances, EITDPA does not improve α. One can confirm this by tracking the array d in EITDPA and see that almost all the indexes are marked, which virtually eliminates gaps for improvement.

In Chapter 3, the examples presented demonstrate the importance of improving α to generate stronger inequalities. Therefore, it is not likely that good computational results

are obtained if no α improvement is made. Thus, the next computational study describes instances where EITDPA does improve α and consequently shows a real computational improvement.

4.3.2 Computational Results With EITDPA

After concluding EITDPA does not improve α due to the structure of the random instances generated, an additional observation is also noticed in the computational study. The value of α occurs in the loop when $q = |C| \setminus |M|$ or $q = |C| \setminus |M| - 1$. Therefore, the primary goal of creating instances in which EITDPA provides some improvement is to create an improvement in these two cases.

If all $a_{i,j}$ are random between 1 and some max, then several small and one large number can be combined to create numerous combinations between 1 and $2 \times max$. Thus, $a_{i,j}$ should be between some min and max where min is at least 50% of max. After attempting a few problems, it appears as though something with min set to about 90% of the max appears to show substantial improvement in α through EITDPA.

A natural choice is to let all $a_{i,j}$ be uniformly distributed between 900 and $1,000$. Because of the small number of variables in this study, almost every integer in this range would be present and the problems lose the randomness. To alleviate this issue, the max has to be substantially larger than $1,000$. Successful results are found when $a_{i,j}$ are uniformly distributed integers between $30,000$ and $32,000$.

The multiple knapsack instances created in this computational study take the same form as presented in Section 4.1. The $a_{i,j}$ coefficients are integers and follow a random uniform distribution between $30,000$ and $32,000$. Thus $l = 30,000$ and $u = 32,000$. The right-hand side vector b is still defined as $b_i = \left\lfloor \beta \sum_{j=1}^{n} a_{i,j} \right\rfloor$ for each $i \in \{1, ..., m\}$ where $\beta = 0.25$. The cost vector c is also still defined as $c_j = \sum_{i=1}^{m} a_{i,j} + \lfloor 500\gamma_j \rfloor$ where γ_j is a uniform random number between 0 and 1, for each $j \in \{1, ..., n\}$.

The results of problems with 5 constraints and 60 variables are first presented in Table 4.3. This table provides the time in seconds and the number of ticks of 15 instances when the problems are solved using CPLEX at default settings and when merged knapsack cover inequalities are implemented with zero, one, and two overlapping variables. The majority of the problems are solved in approximately 10 minutes. Table 4.4 illustrates the percent improvement for each problem.

	CPLEX		$\|M'\| = 0$		$\|M'\| = 1$		$\|M'\| = 2$	
Instance	**Time**	**# Ticks**	**Time**	**# Ticks**	**Time**	**# Ticks**	**Time**	**# Ticks**
1	42	2622	24	2539	18	1734	18	1948
2	216	15075	152	14903	128	13367	133	13715
3	238	15240	134	12557	129	12980	134	13318
4	256	11568	154	14834	120	13169	134	14031
5	764	80458	706	70278	647	71372	653	72461
6	531	39742	359	33558	342	34612	326	35217
7	341	20287	212	18610	192	18176	187	17947
8	122	5174	48	4783	51	4704	51	4717
9	354	25049	287	21755	299	29144	318	28939
10	134	6015	74	7496	69	7067	66	7164
11	924	120657	761	105290	1090	108798	1066	108405
12	213	7674	118	5756	102	6435	103	7210
13	735	73721	775	61368	463	57761	466	57845
14	2840	480580	1996	384975	1801	376614	1800	380213
15	699	56696	501	58081	445	53564	458	58144
Average	**561**	**64037**	**420**	**54452**	**393**	**53966**	**394**	**54752**

Table 4.3: Results 5×60 - Time and Ticks with $l = 30,000$ and $u = 32,000$

	$\|M'\| = 0$		$\|M'\| = 1$		$\|M'\| = 2$	
Instance	**Time %**	**Ticks %**	**Time %**	**Ticks %**	**Time %**	**Ticks %**
1	43.9	3.2	57.7	33.9	57.7	25.7
2	29.7	1.1	40.8	11.3	38.6	9.0
3	43.9	17.6	45.7	14.8	43.6	12.6
4	40.1	-28.2	53.1	-13.8	47.8	-21.3
5	7.6	12.7	15.3	11.3	14.5	9.9
6	32.3	15.6	35.6	12.9	38.6	11.4
7	37.6	8.3	43.6	10.4	45.1	11.5
8	60.5	7.5	58.5	9.1	58.0	8.8
9	19.0	13.2	15.6	-16.3	10.4	-15.5
10	45.0	-24.6	48.3	-17.5	50.7	-19.1
11	17.6	12.7	-17.9	9.8	-15.4	10.2
12	44.8	25.0	52.2	16.2	51.6	6.1
13	-5.5	16.8	36.9	21.6	36.6	21.5
14	29.7	19.9	36.6	21.6	36.6	20.9
15	28.4	-2.4	36.3	5.5	34.4	-2.6
Avg. Imp. %	**31.6**	**6.5**	**37.2**	**8.7**	**36.6**	**5.9**

Table 4.4: Results 5×60 - Percent Improvement with $l = 30,000$ and $u = 32,000$

When no variables are overlapped, 14 out of 15 problems demonstrate improvements in time with an average improvement of approximately 31.6% and standard deviation of 204 seconds. For the number of ticks, 12 out of 15 problems present improvements and merged knapsack cover inequalities demonstrate an average improvement of 6.5% with a standard deviation of 24,396 ticks.

When only one variable is overlapped, 14 out of 15 problems are shown to be better in time with an average improvement of 37.2% and standard deviation of 262 seconds. In addition, 12 out of 15 problems are better in number of ticks than CPLEX with an overall improving average of approximately 8.7% and standard deviation of 26,508 ticks.

When two variables are overlapped, the same 14 out of 15 problems demonstrate better results in time with an improving average of 36.6% and standard deviation of 261 seconds. Also, 11 out of 15 instances present some improvement in the number of ticks, and it is on average 5.9% better with standard deviation of 25,778 ticks.

The average improvement of α through EITDPA is also tracked in this computational study. For simplicity, only the final output is discussed. When no variable is overlapped, EITDPA improves α an average of 20.1%. When one or two variables are overlapped, no improvement is found and α remains the same as initially generated by MKCA.

The next multiple knapsack problems have 5 constraints and 100 variables. Table 4.5 describes the results. These problems are harder than those presented so far and each problem is solved in approximately seven to nine hours. Table 4.6 provides the percent improvement for each of these problems following the same assumptions.

	CPLEX		$\|M'\| = 0$		$\|M'\| = 1$		$\|M'\| = 2$	
Instance	**Time**	**# Ticks**	**Time**	**# Ticks**	**Time**	**# Ticks**	**Time**	**# Ticks**
1	32037	6524350	36993	6446778	37730	6405108	38877	6511009
2	4866	583713	4106	463008	5391	581130	5183	549858
3	21316	4848565	24065	4515377	25983	4581434	26431	4589133
4	26083	4471644	30592	4546217	33013	4442404	33564	4374947
5	43911	5586931	43147	5575846	42278	5442210	43107	5586845
6	24142	5407826	28526	5102992	29522	4979355	29960	4963055
7	2495	551135	2662	522704	2304	419779	2782	490787
8	17426	2927893	18608	2735414	23144	3208621	21374	2915692
9	42027	5241233	38681	4740869	40231	4877226	40392	4941626
10	25617	3295996	24819	3235129	26010	3308366	25731	3309171
11	20629	4234824	24307	4094457	25442	4024260	25579	4039743
12	22951	4722795	26443	4637397	27635	4686955	28202	4658223
13	15877	3401043	20637	3270365	22523	3360554	23670	3485229
14	18161	2052490	18804	2129772	18969	2143884	19864	2169742
15	1950	435987	2185	402910	2267	394022	2297	400478
Average	**21299**	**3619095**	**22972**	**3494616**	**24163**	**3523687**	**24467**	**3532369**

Table 4.5: Results 5×100 - Time and Ticks with $l = 30,000$ and $u = 32,000$

	$\|M'\| = 0$		$\|M'\| = 1$		$\|M'\| = 2$	
Instance	**Time %**	**Ticks %**	**Time %**	**Ticks %**	**Time %**	**Ticks %**
1	-15.5	1.2	-17.8	1.8	-21.4	0.2
2	15.6	20.7	-10.8	0.4	-6.5	5.8
3	-12.9	6.9	-21.9	5.5	-24.0	5.4
4	-17.3	-1.7	-26.6	0.7	-28.7	2.2
5	1.7	0.2	3.7	2.6	1.8	0.0
6	-18.2	5.6	-22.3	7.9	-24.1	8.2
7	-6.7	5.2	7.6	23.8	-11.5	10.9
8	-6.8	6.6	-32.8	-9.6	-22.7	0.4
9	8.0	9.5	4.3	6.9	3.9	5.7
10	3.1	1.8	-1.5	-0.4	-0.4	-0.4
11	-17.8	3.3	-23.3	5.0	-24.0	4.6
12	-15.2	1.8	-20.4	0.8	-22.9	1.4
13	-30.0	3.8	-41.9	1.2	-49.1	-2.5
14	-3.5	-3.8	-4.4	-4.5	-9.4	-5.7
15	-12.1	7.6	-16.3	9.6	-17.8	8.1
Avg. Imp. %	**-8.5**	**4.6**	**-15.0**	**3.5**	**-17.1**	**3.0**

Table 4.6: Results 5×100 - Percent Improvement with $l = 30,000$ and $u = 32,000$

When no variable is overlapped, the time is improved in only 4 out of 15 problems and the average is about 8.5% worse than CPLEX with a standard deviation of 2,577 seconds. However, the number of ticks are improved in 13 out of 15 problems tested with an average improvement of 4.6% with a standard deviation of 156,033 ticks.

When only one variable is overlapped, there are a few problems that present improvements in time (3 out of 15), and the average time is 15.0% worse with a standard deviation of 3,131 seconds. However, the number of ticks are also improved in 12 out of 15 problems and the average improvement is about 3.5% with a standard deviation of 177,386 ticks.

Two variables overlapped demonstrate that only 2 out of 15 problems are improved in terms of time and the average time is 17.1% worse with a standard deviation of 3,242 seconds. The same occur with these merged knapsack cover inequalities; the number of ticks present an average improvement of 3.0% with a standard deviation of 151,429 ticks in 12 out of 15 problems tested.

One can see in these problems an example of how merged knapsack cover inequalities does decrease the number of operations to solve the integer programs but the addition of one extra constraint increases the time. This issue may be avoided in problems with a larger number of constraints. In addition, EITDPA improves α an average of 14.2% when no variable is overlapped. On the other hand, the average improvement when one or two variables are overlapped are close to 0.0%, approximately 0.03% and 0.01% respectively.

The following multiple knapsack instances have 10 constraints and 60 variables. Table 4.7 illustrates the results. These problems are also hard, requiring approximately 10 hours to solve each one. Table 4.8 summarizes the percent improvement results for each instance.

	CPLEX		$\|M'\| = 0$		$\|M'\| = 1$		$\|M'\| = 2$	
Instance	**Time**	**# Ticks**	**Time**	**# Ticks**	**Time**	**# Ticks**	**Time**	**# Ticks**
1	11141	3252027	11840	3395288	11310	3337736	11796	3355847
2	33806	9905552	25892	7495062	25001	7164011	22542	6659990
3	27818	7655830	25504	7271825	24551	7177520	25898	7490830
4	60314	16688886	64689	17764906	63446	17133070	65863	17725513
5	19617	6073236	20379	6084337	20249	6248183	20118	6177667
6	47313	13155649	48312	13578523	48433	13608359	51522	13485082
7	30738	8626026	31703	8927580	30542	8844204	30904	8865736
8	43138	12134831	49694	13774272	42431	11839776	40588	11734916
9	22248	6622808	24034	7155665	22983	6860463	22910	6792680
10	50688	14203626	46016	13014124	53207	14477743	45405	12715144
11	62675	13860815	59730	13838140	60021	13979173	52586	13527821
12	6152	1669192	6029	1538049	6207	1602970	6575	1722326
13	26302	7327517	23395	7083256	22843	6980367	23442	7035312
14	32905	8823231	30256	8298283	31559	8658225	29665	8279701
15	32386	8859666	33187	9139924	36026	9204253	31878	8608444
Average	**33186**	**9257260**	**33377**	**9223949**	**33254**	**9141070**	**32113**	**8945134**

Table 4.7: Results 10 × 60 - Time and Ticks with $l = 30,000$ and $u = 32,000$

	$\|M'\| = 0$		$\|M'\| = 1$		$\|M'\| = 2$	
Instance	**Time %**	**Ticks %**	**Time %**	**Ticks %**	**Time %**	**Ticks %**
1	-6.3	-4.4	-1.5	-2.6	-5.9	-3.2
2	23.4	24.3	26.0	27.7	33.3	32.8
3	8.3	5.0	11.7	6.2	6.9	2.2
4	-7.3	-6.4	-5.2	-2.7	-9.2	-6.2
5	-3.9	-0.2	-3.2	-2.9	-2.6	-1.7
6	-2.1	-3.2	-2.4	-3.4	-8.9	-2.5
7	-3.1	-3.5	0.6	-2.5	-0.5	-2.8
8	-15.2	-13.5	1.6	2.4	5.9	3.3
9	-8.0	-8.0	-3.3	-3.6	-3.0	-2.6
10	9.2	8.4	-5.0	-1.9	10.4	10.5
11	4.7	0.2	4.2	-0.9	16.1	2.4
12	2.0	7.9	-0.9	4.0	-6.9	-3.2
13	11.1	3.3	13.2	4.7	10.9	4.0
14	8.1	5.9	4.1	1.9	9.8	6.2
15	-2.5	-3.2	-11.2	-3.9	1.6	2.8
Avg. Imp. %	**1.2**	**0.8**	**1.9**	**1.5**	**3.9**	**2.8**

Table 4.8: Results 10 × 60 - Percent Improvement with $l = 30,000$ and $u = 32,000$

When no variable is overlapped, 7 out of 15 problems demonstrate some improvement in time and the overall average time is approximately 1.2% better with a standard deviation of 3,586 seconds. In addition, 7 out of 15 problems demonstrate some improvement in the number of ticks; the average improvement is about 0.8% with a standard deviation of 933,794 ticks.

When only one variable is overlapped, 7 out of 15 problems present improvements in time and the average time is 1.9% better with a standard deviation of 3,122 seconds. In addition, 6 out of 15 problems are shown to improve the number of ticks, and the average improvement is approximately 1.5% with a standard deviation of 780,736 ticks.

When two variables are overlapped, 8 out of 15 problems are shown to be better in time, with an improving average time equal to 3.9% and a standard deviation equal to 4,567 seconds. Also, 8 out of 15 problems present some improvement in the number of ticks and the average improvement is about 2.8% with a standard deviation of 975,552 ticks.

The average improvement of α with no overlapping variable is on the order of 18.1%. However, no improvement is made when one or two variables are overlapped, meaning that EITDPA has no significant impact on their final output and α remains the same as initially generated by MKCA.

Table 4.9 provides the time in seconds and the number of ticks of another class of multiple knapsack instances in which the matrix A is integer and is uniform randomly distributed between 30,000 and 33,000 ($l = 30,000$ and $u = 33,000$). Although these are not the main problems studied in this thesis and results from other variations of these problems are not presented, the intention is to provide an additional example of how impressive the results

can be when merged knapsack cover inequalities are applied to appropriate instances. Notice that these problems are solved in approximately 30 to 40 minutes each. Table 4.10 describes the percentage improvement.

	CPLEX		$\|M'\| = 0$		$\|M'\| = 1$		$\|M'\| = 2$	
Instance	**Time**	**# Ticks**	**Time**	**# Ticks**	**Time**	**# Ticks**	**Time**	**# Ticks**
1	78	4676	50	5843	57	4121	40	5165
2	864	101339	651	88649	598	90070	585	92207
3	1916	249122	1306	191856	1192	188237	1228	192318
4	751	70792	930	67035	726	64424	658	75472
5	3185	546378	2101	435749	1927	413326	1997	437151
6	4350	773773	2997	606826	3012	633751	2958	635126
7	1757	263959	1589	277313	1222	228468	1261	239509
8	1039	118967	671	79898	614	81830	594	79961
9	8517	1628007	2871	536918	2721	543107	2598	533182
10	2265	310689	1572	243933	1619	235566	1511	238042
11	7242	1239820	5040	1093867	5528	1147756	5416	1160282
12	615	42404	601	41028	573	38937	546	34887
13	1415	185760	1239	188439	987	155272	942	151192
14	2799	496375	1980	372357	1887	390688	1875	389877
15	4686	821523	3647	747937	3039	654648	3302	739262
Average	**2765**	**456905**	**1816**	**331843**	**1713**	**324680**	**1701**	**5003632**

Table 4.9: Results 5×60 - Time and Ticks with $l = 30,000$ and $u = 33,000$

In these problems, when no variable is overlapped, 14 out of 15 problems present some improvement in time and the average time is 24.8% better than CPLEX with a standard deviation of 1,442 seconds. The number of ticks also demonstrate some improvement in 12 out of 15 problems, and the average improvement is approximately 14.8% with a standard deviation of 273,561 ticks.

	$\|M'\| = 0$		$\|M'\| = 1$		$\|M'\| = 2$	
Instance	**Time %**	**Ticks %**	**Time %**	**Ticks %**	**Time %**	**Ticks %**
1	35.5	-25.0	27.4	11.9	48.9	-10.5
2	24.6	12.5	30.8	11.1	32.3	9.0
3	31.9	23.0	37.8	24.4	35.9	22.8
4	-23.8	5.3	3.4	9.0	12.4	-6.6
5	34.0	20.2	39.5	24.4	37.3	20.0
6	31.1	21.6	30.8	18.1	32.0	17.9
7	9.6	-5.1	30.4	13.4	28.2	9.3
8	35.4	32.8	40.9	31.2	42.8	32.8
9	66.3	67.0	68.1	66.6	69.5	67.2
10	30.6	21.5	28.5	24.2	33.3	23.4
11	30.4	11.8	23.7	7.4	25.2	6.4
12	2.2	3.2	6.8	8.2	11.2	17.7
13	12.4	-1.4	30.2	16.4	33.4	18.6
14	29.3	25.0	32.6	21.3	33.0	21.5
15	22.2	9.0	35.1	20.3	29.5	10.0
Avg. Imp. %	**24.8**	**14.8**	**31.1**	**20.5**	**33.7**	**17.3**

Table 4.10: Results 5×60 - Percent Improvement with $l = 30,000$ and $u = 33,000$

When only one variable is overlapped, all 15 problems are better in time and ticks. The average improvement time is about 31.1% with a standard deviation of 1,427 seconds, and the average improvement of ticks is 20.5% with a standard deviation of 268,947 ticks.

Considering two variables overlapped, all 15 problems are improved in terms of time; the average time is 33.7% better with a standard deviation of 1,446 seconds. In addition, 13 out of 15 problems present some improvement in the number of ticks with an overall average improvement of 17.3% and standard deviation of 272,299 ticks.

The average improvement of α is on the order of 19.1% when no variable is overlapped. However, EITDPA does not improve α when one or two variables are overlapped. In this case, α is maintained as initially generated in MKCA.

After the results are described, notice that multiple knapsack instances where $l = 30,000$ and $u = 32,000$ dominate the regular problems where $l = 1$ and $u = 1,000$. These problems advantageously utilize their structure in which gaps for improvement exist. Therefore, EITDPA may generate larger α values and consequently stronger inequalities.

Although α is mostly improved through EITDPA in problems where there is no overlapping variable, the results are shown to be satisfactory in all the cases. Table 4.11 summarizes the results of problems tested in this computational study.

	$\|M'\| = 0$		$\|M'\| = 1$		$\|M'\| = 2$	
Problem	**Time %**	**Ticks %**	**Time %**	**Ticks %**	**Time %**	**Ticks %**
5×60	31.6	6.5	37.2	8.7	36.6	5.9
5×100	-8.5	4.6	-15.0	3.5	-17.1	3.0
10×60	1.2	0.8	1.9	1.5	3.9	2.8
Avg. Imp. %	**8.1**	**4.0**	**8.0**	**4.6**	**7.8**	**3.9**

Table 4.11: Summary Results - Computational Study

When no variable is overlapped, the average improving time is 8.1% and the number of ticks are improved 4.0%. When only one variable is overlapped, the average improving time is 8.0% and the number of ticks are improved by 4.6%. Finally, when two variables are overlapped, the average improving time is 7.8% and the number of ticks can be improved in average 3.9%.

In conclusion, implementing merged knapsack cover inequalities are beneficial in this computational study when EITDPA makes real improvements in α. In this case, stronger inequalities are generated and the computational results are satisfactory. Therefore, this new class of cutting planes is worth implementing when an IP has a knapsack constraint

where the minimum coefficient $a_{i,j}$ is approximately 90% of the maximum coefficient. Based upon the results described in this thesis, $l = 30,000$ and $u = 32,000$ show improvements in α and good computational outcomes.

For the number of overlapping variables in this computational study, a recommendation is made to implement merged knapsack cover inequalities with no overlapping variables for two reasons. First, no overlapping variables has the least worst average improving time in problems with 5 constraints and 100 variables. Second, no overlapping variables has the smallest standard deviation, meaning the solution times are more consistent with less variability.

Chapter 5

Conclusions and Future Research

This research generated a new class of useful cutting planes by merging knapsack constraints with cover inequalities. Results showed that merged knapsack cover inequalities are valid inequalities, and an example demonstrated that these inequalities may be facet defining for some instances. This example also demonstrated that merged knapsack cover inequalities may cut off some portion of the linear relaxation space and help improve the solving time of integer programs.

The merged knapsack cover inequalities are generated using the Merging Knapsack Cover Algorithm (MKCA), which has two subroutines: Determine the Set of Merging Indices Subroutine (DSMI), and Calculate α Value Subroutine (CAV). The DSMI subroutine determines a set of merging variables that preserves the validity of the inequality while CAV finds a value for $\alpha > 0$. MKCA is a linear time algorithm and requires $O(n)$ effort. These inequalities can be improved using either the Exact Improvement Algorithm (EIA) or Exact Improvement Through Dynamic Programming Algorithm (EITDPA). Both algorithms pro-

vide identical output; however, EIA is an exponential time algorithm and EITDPA is a pseudo-polynomial time algorithm that runs in linear time when the instance is polynomial. The proof of correctness is shown for each algorithm.

A computational study is also developed in this thesis to determine whether implementation of merged knapsack cover inequalities is computationally effective. The multiple knapsack instances approached in this thesis are randomly generated. In order to more accurately find cover inequalities, the algorithm developed by Gu *et al.* [32] with some modifications is implemented to find the strongest covers.

The first multiple knapsack instances tested have their coefficients between 1 and $1,000$. The results are worse than CPLEX and surprisingly EITDPA does not improve α in any of the cases, which is well explained by the research developed by Hunsaker and Tovey [39]. Thus, some other class of problems are tested and improvement in α appears when the constraint coefficients are distributed between a minimum value that is approximately 90% of the maximum value. These multiple knapsack problems have their coefficients between $30,000$ and $32,000$ or $30,000$ and $33,000$. When compared to CPLEX, merged knapsack cover inequalities improve the time on average 8.0% and decrease the number of ticks on average 4.0%.

Additionally, three strategies on overlapping variables are tested in this thesis. The strategy that demonstrates better results and more consistency is the one with no overlapping variables. This approach has the highest average improving time and also the smallest standard deviation.

5.1 Future Research

During the development of this thesis, some questions arose on potential future research on both computational and theoretical areas. While other computational studies would help understand new classes of problems where these cutting planes are worth implementing, numerous theoretical extensions of merged knapsack cover inequalities can be approached. The next two sections present some future research ideas.

5.1.1 Future Computational Studies

A potential future computational study is to identify other classes of multiple knapsack instances where implementation of merged knapsack cover inequalities is also computationally effective. In this thesis, problems where the minimum constraint coefficient value is approximately 90% of the maximum value show substantial improvements in α and consequently in the final outcomes. However, other problems where the minimum value is at least 50% of the maximum value may be explored and possibly extend the usefulness of this class of cutting planes.

In addition, the study of merged knapsack cover inequalities applied to benchmark instances is another area that can be explored. For example, the OR-Library [11] has complex benchmark instances and is used by many researchers in the field. This library contains 270 different multiple knapsack instances that vary in terms of number of variables, number of constraints, and tightness ratio.

Variations on the number of overlapping variables is another potential future research area. In this thesis, at most two overlapping variables are considered. Although the strategy of overlapping no variables demonstrated better results and more consistency in the computational study performed in this thesis, it may be possible to find better results when the number of overlapping variables increases.

Another future research opportunity is extending merged knapsack cover inequalities to general integer programming problems. Because any binary integer program constraint can be converted into a knapsack constraint through a simple transformation, this new class of cutting planes can be implemented and help decrease the time required to solve these integer programs.

5.1.2 Future Theoretical Extensions

Topics for theoretical extensions in merging knapsack constraints with cover inequalities are described in this section. A potential future research direction is to merge multiple knapsack constraints into a cover inequality of the problem. This concept can be referred to as inequality lifting. As described in Chapter 3, the merging variable coefficients $a_{i,j}$ can be any positive real number as long as the validity conditions are met. Therefore, the information from one potential strong cover inequality of the problem can be combined with multiple knapsack constraints in order to generate a stronger inequality. This may generate another new class of useful cutting planes and also stronger computational results.

Since the merging variable coefficients $a_{i,j}$ can be literally any value, another potential future research is to develop a method that finds appropriate values of these coefficients in such a way that the validity conditions are still met and the EITDPA always produces real improvements in α. In this case, it is most likely that these inequalities have better computational results and can be applied in more classes of multiple knapsack problems.

Lastly, extending the concept of merged knapsack cover inequalities in order to merge multiple cover inequalities of the problem with either a knapsack or multiple knapsack constraints is a promising research direction. The idea is to combine as much useful information as possible from the cover inequalities with the knapsack constraints, which would probably discover an unknown class of cutting planes.

Bibliography

[1] Achterberg, T., Koch, T., and Martin, A. (2005). Branching rules revisited. *Operations Research Letters*, *33*, 42-54.

[2] Ahuja, R., and Cunha, C. (2005). Very large-scale neighborhood search for the k-constraint multiple knapsack problem. *Journal of Heuristics*, *11*(5-6), 465-481.

[3] Anbil, R., Gelman, E., Patty, B., and Tanga, R. (1989). Recent advances in crew-pairing optimization at American Airlines. *Interfaces*, *21*, 62-74.

[4] Arunapuram, S., Mathur, K., and Solow, D. (2003). Vehicle routing and scheduling with full truckloads. *Transportation Science*, *37*(2), 170-182.

[5] Atamtürk, A. (2003). On the facets of the mixed–integer knapsack polyhedron. *Mathematical Programming*, *98*, 145-175.

[6] Atamtürk, A., and Narayanan, V. (2011). Lifting for conic mixed-integer programming. *Mathematical Programming*, *126*, 351-363.

[7] Atamtürk, A., Nemhauser, G., and Savelsbergh, M. (2000). Conflict graphs in solving integer programming problems. *European Journal of Operational Research*, *121*, 40-55.

[8] Babaiof, M., Immorlica, N., Kempe, D., and Kleinberg, R. (2007). A knapsack secretary problem with applications. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, *4627*, 16-28.

[9] Balas, E. (1975). Facets of the knapsack polytope. *Mathematical Programming*, *8*(1), 146-164.

[10] Balas, E. (1979). Disjunctive programming. *Annals of Discrete Mathematics*, *5*, 3-51.

[11] Beasley, J. (2012). OR-Library. Retrieved from: http://people.brunel.ac.uk/m̃astjjb/jeb/info.html.

[12] Bertsimas, D., Darnell, C., and Soucy, R. (1999). Portfolio construction through mixed-integer programming at Grantham, Mayo, Van Otterloo and Company. *Interfaces*, *29*(1), 49-66.

[13] Beyer, A. (2011). *Exact synchronized simultaneous uplifting over arbitrary initial inequalities for the knapsack polytope*, MS Thesis, Department of Industrial and Manufacturing Systems Engineering, Kansas State University.

[14] Bolton, J. (2009). *Synchronized simultaneous lifting in binary knapsack polyhedra*, MS Thesis, Department of Industrial and Manufacturing Systems Engineering, Kansas State University.

[15] Brown, D., and Harrower, I. (2004). A new integer programming formulation for the pure parsimony problem in haplotype analysis. *Algorithms in Bioinformatics*, *3240*, 254-265.

[16] Chang, P., and Lee, J. (2012). A fuzzy DEA and knapsack formulation integrated model for project selection. *Computers and Operations Research*, *39*(1), 112-125.

[17] Cho, D., Padberg, M., and Rao, M. (1983). On the uncapacitated plant location problem II: Facets and lifting theorems. *Mathematics of Operations Research*, *8*(4), 590-612.

[18] Chu, P., and Beasley, J. (1998). A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, *4*(1), 63-86.

[19] Chvátal, V. (1973). Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, *4*(4), 305-337.

[20] Dawande, M., Kalagnanam, J., Keskinocak, P., Salman, F., and Ravi, R. (2000). Approximation algorithms for the multiple knapsack problem with assignment restrictions. *Journal of Combinatorial Optimization*, *4*(2), 171-186.

[21] DeLissa, L. (2014). *The existence and usefulness of equality cuts in the multi-demand multidimensional knapsack problem*, MS Thesis, Department of Industrial and Manufacturing Systems Engineering, Kansas State University.

[22] Dey, S., and Richard, J. (2007). Sequential-merge facets for two-dimensional group problems. *Integer Programming and Combinatorial Optimization*, *4513*, 30-42.

[23] Dey, S., and Wolsey, L. (2010). Two row mixed-integer cuts via lifting. *Mathematical Programming*, *124*(1-2), 143-174.

[24] Dizdar, D., Gershkov, A., and Moldovanu, B. (2011). Revenue maximization in the dynamic knapsack problem. *Theoretical Economics*, *6*(2), 157-184.

[25] Easton, K., Nemhauser, G., and Trick, M. (2003). Solving the traveling tournament problem: A combined integer programming and constraint programming approach. *Practice and Theory of Automated Timetabling IV*, *2740*, 100-109.

[26] Easton, T., and Hooker, K. (2008). Simultaneously lifting sets of binary variables into cover inequalities for knapsack polytopes. *Discrete Optimization*, *5*(2), 254-261.

[27] Ferreira, C., de Souza, C., and Wakabayashi, Y. (2002). Rearrangement of DNA fragments: A branch-and-cut algorithm. *Discrete Applied Mathematics*, *116*(1-2), 161-177.

[28] Finn, F. (1973). Integer programming, linear programming and capital budgeting. *Abacus*, *9*(2), 180-192.

[29] Gomory, R. (1958). Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, *64*(5), 275-278.

[30] Gomory, R. (1965). On the relation between integer and non-integer solutions to linear programs. *Proceedings of the National Academy of Science*, *53* 260-265.

[31] Gomory, R. (1969). Some polyhedra related to combinatorial problems. *Linear Algebra and its Applications*, *2*(4), 451-558.

[32] 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.

[33] Gu, Z., Nemhauser, G., and Savelsbergh, M. (1999). Lifted cover inequalities for 0-1 integer programs: Complexity. *INFORMS Journal on Computing*, *11*(1), 117-123.

[34] Gu, Z., Nemhauser, G., and Savelsbergh, M. (2000). Sequence independent lifting in mixed integer programming. *Journal of Combinatorial Optimization*, *4*(1), 109-129.

[35] Gutierrez, T. (2007). *Lifting general integer variables*, MS Thesis, Department of Industrial and Manufacturing Systems Engineering, Kansas State University.

[36] Hammer, P., Johnson, E., and Peled, U. (1975). Facets of regular 0-1 polytopes. *Mathematical Programming*, *8*(1), 179-206.

[37] Harris, A. (2010). *Generating an original cutting-plane algorithm in three sets (GO CATS)*, MS Thesis, Department of Industrial and Manufacturing Systems Engineering, Kansas State University.

[38] Hickman, R. (2014). *Generating cutting planes through inequality merging for integer programming problems*, PhD Dissertation, Department of Industrial and Manufacturing Systems Engineering, Kansas State University.

[39] Hunsaker, B., and Tovey, C. (2005). Simple lifted cover inequalities and hard knapsack problems. *Discrete Optimization*, *2*(3), 219-228.

[40] IBM. *ILOG CPLEX Optimization Studio*, (Version 12.5.1), Available from http://www-01.ibm.com/software/info/ilog/.

[41] Iwamura, K., and Liu, B. (1999). Dependent-chance integer programming applied to capital budgeting. *Journal of the Operations Research Society of Japan*, *42*(2), 117-127.

[42] Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. *Combinatorica*, *4*(4), 373-395.

[43] Karp, R. (1972). Reducibility among combinatorial problems. *Complexity of Computer Computations, The IBM Research Symposia Series*, 85-103.

[44] Kaufman, D., Nonis, J., and Smith, R. (1998). A mixed integer linear programming model for dynamic route guidance. *Transportation Research Part B: Methodological*, *32*(6), 431-440.

[45] Kellerer, H., and Strusevich, V. (2010). Fully polynomial approximation schemes for a symmetric quadratic knapsack problem and its scheduling applications. *Algorithmica*, *57*(4), 769-795.

[46] Khachiyan, L. (1979). A polynomial algorithm in linear programming. *Doklady Mathematics*, *20*(1), 191-194.

[47] Kolliopoulos, S., and Steiner, G. (2007). Partially ordered knapsack and applications to scheduling. *Discrete Applied Mathematics*, *155*(8), 889-897.

[48] Kubik, L. (2009). *Simultaneously lifting multiple sets in binary knapsack integer programs*, MS Thesis, Department of Industrial and Manufacturing Systems Engineering, Kansas State University.

[49] Lai, T., and Sahni, S. (1984, June). Anomalies in parallel branch-and-bound algorithms. *Communications of the ACM*, *27*, 594-602.

[50] Land, A., and Doig, A. (1960). An automatic method of solving discrete programming problems. *Econometrica*, *28*(3), 497-520.

[51] Lee, E., Fox, T., and Crocker, I. (2003). Integer programming applied to intensity-modulated radiation treatment planning optimization. *Annals of Operations Research*, *119*(1-4), 165-181.

[52] Lee, E., and Zaider, M. (2003). Mixed integer programming approaches to treatment planning for brachytherapy - application to permanent prostate implants. *Annals of Operations Research*, *119*(1-4), 147-163.

[53] Lehmer, D. (1951). Mathematical methods in large-scale computing units. *Annals of the Computation Laboratory of Harvard University*, *26*, 141-146.

[54] Linderoth J, and Savelsbergh, M. (1999). A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, *11*(2), 173-187.

[55] Microsoft Corporation. *Visual Studio*, (Version 2013), Available from https://www.visualstudio.com/

[56] Nemhauser, G., and Wolsey, L. (1999). *Integer and combinatorial optimization*. New York: John Wiley and Sons.

[57] Pinto, R., and Rustem, B. (1998). Solving a mixed-integer multiobjective bond portfolio model involving logical conditions. *Annals of Operations Research*, *81*(0), 497-514.

[58] Pisinger, D. (1995). A minimal algorithm for the multiple-choice knapsack problem. *European Journal of Operational Research*, *83*(2), 394-410.

[59] Ruiz, R., Maroto, C., and Alcaraz, J. (2004). A decision support system for a real vehicle routing problem. *European Journal of Operational Research*, *153*(3), 593-606.

[60] Shachnai, H., and Tamir, T. (2001). On two class-constrained versions of the multiple knapsack problem. *Algorithmica*, *29*(3), 442-467.

[61] Shebalov, S., and Klabjan, D. (2006). Sequence independent lifting for mixed integer programs with variable upper bounds. *Mathematical Programming*, *105*(2-3), 523-561.

[62] Subramanian, R., Scheff R., Quillinan J., Wiper D., and Marsten R. (1994). Coldstart: Fleet assignment at Delta Air Lines. *Interfaces*, *24*, 104-120.

[63] Szeto, K., and Lo, M. (2004). An application of adaptive genetic algorithm in financial knapsack problem. *Innovations in Applied Artificial Intelligence*, *3029*, 1220-1228.

[64] Toth, P., and Vigo, D. (1997). An exact algorithm for the vehicle routing problem with backhauls. *Transportation Science*, *31*(4), 372-385.

[65] Trick, M. (2004). Using sports scheduling to teach integer programming. *INFORMS Transactions on Education*, *5*(1), 10-17.

[66] Urban, T., and Russell, R. (2003). Scheduling sports competitions on multiple venues. *European Journal of Operational Research*, *148*(2), 302-311.

[67] Winston, W. (2004). *Operations research: Applications and algorithms* (4th ed.). Belmont, California: Duxbury Press.

[68] Wolsey, L. (1975). Faces for a linear inequality in 0-1 variables. *Mathematical Programming*, *8*(1), 165-178.

[69] Wolsey, L. (1977). Valid inequalities and superadditivity for 0-1 integer programs. *Mathematics of Operations Research*, *2*, 66-77.

[70] Zemel, E. (1978). Lifting the facets of 0-1 polytopes. *Mathematical Programming*, *15*(1), 268-277.

[71] Zeng B., and Richard, J. (2011). A polyhedral study on 01 knapsack problems with disjoint cardinality constraints: Strong valid inequalities by sequence-independent lifting. *Discrete Optimization*, *8*(2), 259-276.