GENERATING CUTTING PLANES THROUGH INEQUALITY MERGING ON MULTIPLE VARIABLES IN KNAPSACK PROBLEMS

by

THOMAS CHARLES BOLTON

B.S., Kansas State University, 2015

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

Todd Easton

ABSTRACT

Integer programming is a field of mathematical optimization that has applications across a wide variety of industries and fields including business, government, health care and military. A commonly studied integer program is the knapsack problem, which has applications including project and portfolio selection, production planning, inventory problems, profit maximization applications and machine scheduling. Integer programs are computationally difficult and currently require exponential effort to solve.

Adding cutting planes is a way of reducing the solving time of integer programs. These cutting planes eliminate linear relaxation space. The theoretically strongest cutting planes are facet defining inequalities.

This thesis introduces a new class of cutting planes called multiple variable merging cover inequalities (MVMCI). The thesis presents the multiple variable merging cover algorithm (MVMCA), which runs in linear time and produces a valid MVMCI. Under certain conditions, an MVMCI can be shown to be a facet defining inequality. An example demonstrates these advancements and is used to prove that MVMCIs could not be identified by any existing techniques.

A small computational study compares the computational impact of including MVMCIs. The study shows that finding an MVMCI is extremely fast, less than .01 seconds. Furthermore, including an MVMCI improved the solution time required by CPLEX, a commercial integer programming solver, by 6.3% on average.

Contents

List of Figures

List of Tables

Dedication

This work is dedicated to my parents, John and Susan, and siblings, Jen, Ben, Amanda and James, for their unconditional love and support.

Acknowledgments

I would first thank Dr. Todd Easton. Without his excellent mentorship, scholarly support and patience, this thesis would not have been possible.

I am also grateful to Dr. Jessica Heier Stamm and Dr. Bette Grauer for their participation on my Supervisory Committee.

Lastly, I want to sincerely thank the people of Kansas State University, especially the faculty, staff and students of the Industrial and Manufacturing Systems Engineering Department. My personal growth, educational development and professional endeavors have greatly benefitted from my relationships with these individuals.

Chapter 1

Introduction

Integer programming is a field of mathematical optimization with great potential to transform the world and improve people's lives. Integer programming has applications in a wide variety of business, technical and personal situations where finding the optimal solution to a problem is the goal. It is easiest to explain the contributions of integer programming by looking at how it is applied in the world today.

One area where integer programs are widely used is in various types of vehicle routing problems. In Ankara, Turkey, there were problems stemming from the costs associated with busing students from rural regions to schools. Once the problem was formulated as an integer program, the researchers [3] were able to decrease the overall cost by 28% and reduce the total miles traveled by all buses by 14%. Other benefits obtained were an increased utilization of 27% and a decreased variation in capacity by 33%.

There are numerous other applications of integer programs (IPs). IPs have been used to assist with cancer location and treatment [33]. IPs has also been applied to the overseas

shipping industry [39] and facilities layout [16]. Other applications include military applications [37], sports scheduling [14], airline security [35] and criminal justice assignment [6], which have all benefitted society.

While IPs are very useful, solving IP problems is $\mathcal{NP}$ hard, which means that there does not exist a polynomial time algorithm to solve the problem to optimality, unless $\mathcal{P} = \mathcal{NP}$. In laymans terms it means that any algorithm used to solve an IP requires exponential time. Thus, certain concessions are typically made to the IP model, which leave open the chance that the solution that is found is suboptimal in the real world. This means that the solution needs to be evaluated to make sure it is implementable and that it provides benefit to the process.

There are a few different ways to solve an IP. The most widely used algorithm is the branch and bound algorithm. Branch and bound [32] works by solving the linear relaxation and then branching on one of the non-integer variables. Branch and bound is a good strategy for smaller problems, but as the problems get larger, the time to solve these problems increases exponentially. While computers are constantly getting faster and more advanced, there still exist IPs that cannot be solved to optimality.

One common method to reduce the IP solution time is through cutting planes. Cutting planes help to decrease the solving time of integer programming software by decreasing the space of the linear relaxation that the software might check. It is called a cutting plane because it cuts off these undesirable noninteger solutions. Thus, many cutting plane preprocessing techniques are used to help reduce IP's solution time.

Cutting planes are valid inequalities and the theoretically strongest valid inequalities are

called facet defining. If all facet defining inequalities are added to an IP, then its linear relaxation solution is integer. Thus, branch and bound only requires one iteration and not exponential effort.

This thesis focuses on a specific class of IPs called a knapsack problem. Knapsack problems are found in various settings related to pricing of items [4], defense [1], aviation security [28] and portfolio selection [38].

A commonly used cutting plane for a knapsack problem are cover inequalities. Occasionally, a cover inequality can be facet defining, but most cover inequalities can be strengthened. Fundamentally, this thesis takes a cover inequality and strengthens it.

1.1 Research Motivation and Questions

In 2014, Hickman and Easton [24] generated inequalities by merging two cover inequalities together. This resulted in a new class of cutting planes for the knapsack instances. Furthermore under certain conditions, these merged cover inequalities could be facet defining.

Hickman's results could only merge covers on a single variable. That is, the covers had to have exactly one element in common. In certain instances, Hickman and Easton's method would create inequalities that were weak and these inequalities could be strengthened.

The motivation for this thesis is to build on and stregthen their work. Thus, this work sought to answer the following reseasrch questions. Can cover inequalities be merged on multiple variables? Can merging be done so that the inequalities created cannot be dominated by an inequality of the similar form?

1.2 Research Contributions

The primary contribution of this thesis is the development of a new class of cutting planes for the knapsack problem that decreases the time to solve integer programs. This class is called multiple variable merging cover inequalities (MVMCI). These inequalities merge two cover inequalities from an existing constraint on multiple variables. They are merged with the aid of a merging coefficient. This coefficient, α, is also the theoretically strongest coefficient that can be obtained. Furthermore, this class of cutting planes cannot be obtained by current methods without the consultation of an oracle.

An algorithm is also presented, which generates MVMCI from a single knapsack constraint. This algorithm runs in linear time, which is the theoretically best run time possible. It also can generate facet defining inequalities, which are the theoretically strongest inequalities.

To show the usefulness of MVMCIs, a computational study was conducted. The multiple variable merging cover algorithm (MVMCA) was coded into C and added to CPLEX [10], a commercial integer program solver, for small and medium problems, and run to completion. The time required to solve these problems is referred to as run time. MVMCA has the ability to decrease run times of CPLEX solver by up to 40% in this computational study on certain instances. In addition, MVMCI was able to cut CPLEX run times by on average 6.3% on the medium sized problems, which reduced the run time by over an hour.

1.3 Outline

This thesis is organized as follows: Chapter 2 presents an overview of the background information that is necessary to understand the contributions of this thesis. It covers integer programming, polyhedral theory, knapsack problems, cover cuts, and lifting examples. This includes formal definitions, explanations and examples to aid in the understanding of the background material.

Chapter 3 provides the theoretical basis for inequality merging on multiple variables. It contains the theorems, explanations and conditions for validity of the merged inequality. An example where multiple variable merging cover algorithm creates a facet defining inequality demonstrates these concepts. The run time is discussed and it is shown that it is not dominated by previous methods. Chapter three ends with an argument that these inequalities are new and not simply a rehashing of previous work.

Chapter 4 is a computational study to show the usefulness of MVMCI. The results are interpreted and it is shown that MVMCA can be easily implemented and the inequalities generated can be useful in solving knapsack integer programs.

Finally, Chapter 5 is a summary of the important contributions from this thesis. It also offers up possible areas of future research.

Chapter 2

Background Information

Chapter 2 gives an overview of the background information necessary to grasp the contributions of this thesis. This includes an overview of integer programming as well as polyhedral theory. The use of cutting planes to shorten the process of solving an IP is also discussed along with lifting. Lastly an overview and example of Hickman's inequality merging technique is discussed.

2.1 Integer Programming

An integer program (IP) is a mathematical model of the form maximize $c^T x$ subject to $Ax \leq b, x \geq 0$ and $x \in \mathbb{Z}^n$ where $A \in \mathbb{R}^{mxn}, b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$. Define the set of feasible IP solutions as $P = \{x \in \mathbb{Z}_+^n : Ax \leq b\}$ and the set of indices to be $N = \{1, ..., n\}$.

An important concept in solving IP's is a linear relaxation (LR). Given an IP, its linear relaxation is the IP with the integer requirement removed, meaning that the problem is

a linear program of the form maximize $c^T x$ subject to $Ax \leq b, x \geq 0$. Define the linear relaxation space as $P^{LR} = \{x \in \mathbb{R} : Ax \leq b, x \geq 0\}$. The optimal solution to a linear relaxation can be found in polynomial time [29].

Branch and bound is the most widely used and standard algorithm to solve integer programming optimization problems [32]. This technique builds an ancestral branching tree where the nodes have properties determined by the nodes' lineage. This technique generates an optimal solution, but it may take an exponential amount of time.

The branch and bound algorithm starts with the linear relaxation, called the root node. If no noninteger variables exist, then the linear relaxation solution is optimal. If not, the node is split on a noninteger variable ($x_i = f$) of the LR solution. The resulting branched child nodes each have one constraint on them $x_i \geq \lceil f \rceil$ and $x_i \leq \lfloor f \rfloor$, the integer below and above the noninteger x_i. While there exists at least one unfathomed leaf node, the branch and bound algorithm solves the LR of that node with its corresponding linear relaxation solution, x^{*LR} and linear relaxation objective function value, z^{*LR}. This repeats until all nodes have been fathomed. A node is fathomed if one of three criteria are met: the linear relaxation becomes infeasible, the node's linear relaxation solution is integer or the objective function value is worse than the best integer solution found so far.

To reduce the run time of branch and bound, cutting planes are frequently applied. The area of research that studies cutting planes is called polyhedral theory. The goal of a cutting plane is to shrink the linear relaxation space. The next section gives an overview of polyhedral theory.

2.2 Polyhedral Theory

The geometry of any IP problem is important to the solution of the IP. Polyhedral theory is fundamental to understanding this type of research. A half space is $\{x \subseteq \mathbb{R}^n : \sum_{i=1}^{n} \alpha_i x_i \leq \beta\}$ and a polyhedron is defined as the intersection of finitely many half spaces.

A set $S \subseteq \mathbb{R}^n$ is convex if and only if $\lambda x^1 + (1-\lambda)x^2 \in S$ for every $\lambda \in [0,1]$ and all x^1 and $x^2 \in S$. The convex hull of a set S, $conv(S)$, is the intersection of all convex sets that contain S. Clearly, a polyhedron is convex. An important result is that P^{LR} and $conv(P)$ are both polyhedrons. If a polyhedron is bounded, then it is called a polytope.

An inequality $\sum_{i=1}^{n} \alpha_i x_i \leq \beta$ is valid for $conv(P)$ if every $x \in P$ satisfies this inequality. A valid inequality is a cutting plane if there exists an $x' \in P^{LR}$ such that $\sum_{i=1}^{n} \alpha_i x'_i > \beta$. Only cutting planes can reduce the time required to solve an IP.

Every valid inequality induces a face, F, of $conv(P)$ where $F = \{x \in conv(P) : \sum_{i=1}^{n} \alpha_i x_i = \beta\}$. Every face is a polyhedron. Theoretically, the usefulness of a valid inequality is measured by the dimension of this face.

The dimension of a polyhedron equals the maximum number of linearly independent vectors. However, an IP only has feasible points and so no nonzero vector creates a feasible direction. Due to this fact, affine independence is used to determine the dimension of $conv(P)$.

Let V be a finite set of points in $\mathbb{R}^n$, $V = \{v^i \in \mathbb{R}^n : i = 1, ..., w\}$. The points in V are affinely independent if and only if the unique solution to $\sum_{i=1}^{w} \lambda_i v_i = 0$ and $\sum_{i=1}^{w} \lambda_i = 0$ is $\lambda_i = 0$ for all $i = 1, ..., w$. Furthermore, the dimension of the convex hull of the set of V

points is equal to the maximum number of affinely independent points minus one. In order to define points in $\mathbb{R}^n$, let ξ_j be the origin in n dimensions translated one positive unit in the j^{th} dimension, i.e. $\xi_2 = (0, 1, 0, ..., 0)$.

The larger the dimension of the face, up to one less than the dimension of $conv(P)$, the theoretically stronger the inequality is. The strongest such cutting planes are called facet defining inequalities. An inequality is facet defining if its face has dimension one less than the dimension of $conv(P)$. A facet defining inequality is an inequality that defines a facet of the $conv(P)$. If one could find all of the facet defining inequalities for a given problem, then including all these facets would define $conv(P)$. Thus, the extreme points would be integer and the linear relaxation solution would be an integer solution.

2.2.1 2-Dimensional Integer Programming Example

The following small example of a two dimensional IP is shown to explain these concepts.

$$\begin{aligned} \text{Maximize} \quad & x_1 + x_2 \\ \text{Subject to:} \quad & 5x_1 + 4x_2 \leq 20 \\ & x_1 + 2x_2 \leq 8 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z} \end{aligned}$$

Figure 2.1 shows a graphical representation of this problem. The set of feasible integer points, P, are identified by large circles. The extreme points of the linear relaxation space are shown at A, B, C and the origin. The dashed line denotes the $conv(P)$ and is formed by connecting the extreme integer points in the feasible region. Note the x axis from the origin

to (4,0) and the y axis from the origin to (0,4) are also assumed to be dashed in this picture.

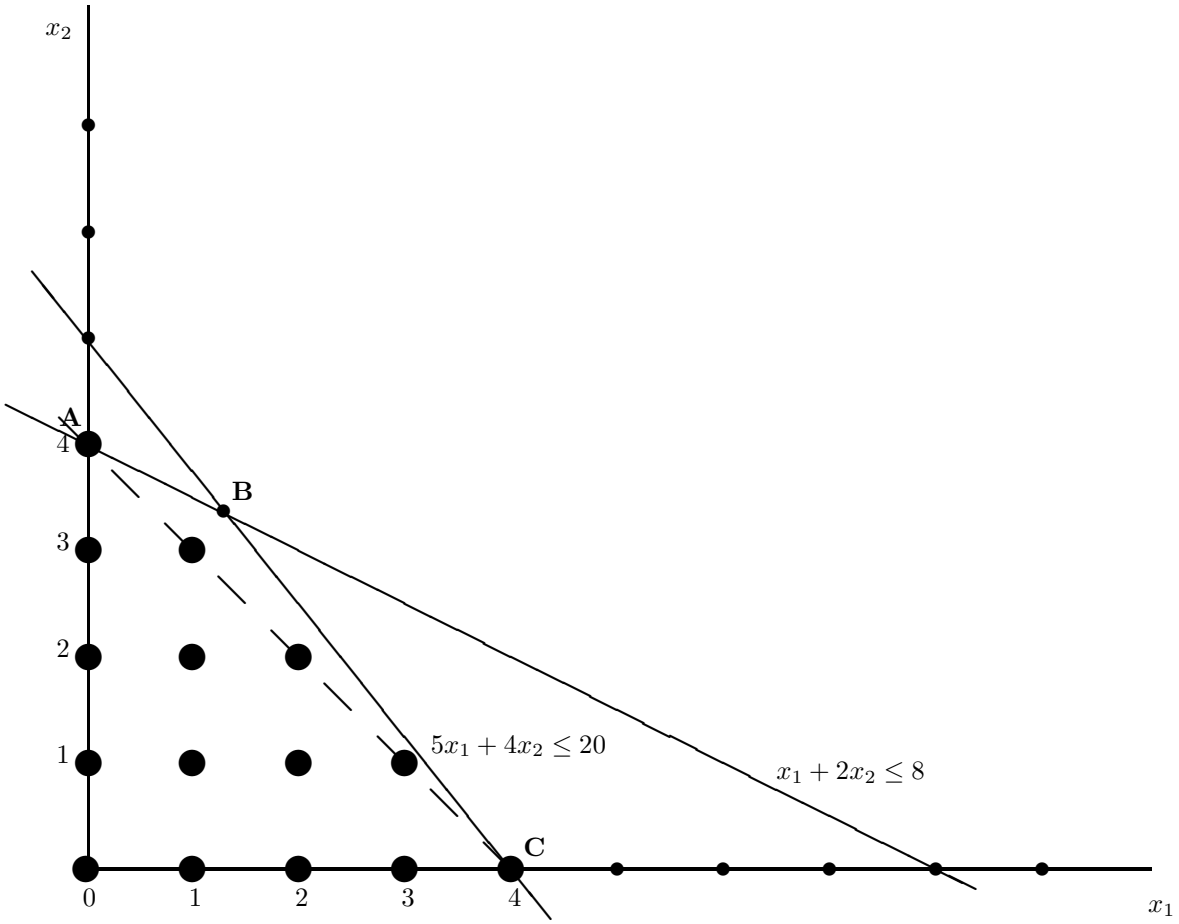

Figure 2.1: 2-Dimensional IP Example

The first step in solving this problem with cutting planes is to find the linear relaxation solution. Once that is found, a cutting plane is added to the linear relaxation and the linear relaxation solution is recalculated to be (x^{*LR}, z^{*LR}). This process is repeated until the solution is an integer solution in which case z^{*LR} is the optimal integer objective function value z^{*IP}.

Because the objective function is simply $x_1 + x_2$, the best solution to this problem occurs at (1.3,3.375), which is point A, and yields $z^{*LR} = 4.675$. Adding the valid inequality, $x_1 + x_2 \leq 4$, to the formulation eliminates the point (1.3,3.375). Thus, this valid inequality is a cutting plane. Resolving the linear relaxation results in an optimal solution of $z^{*LR} = 4$ at the point $(4, 0)$. Since this is an integer point, the optimal IP solution is found.

To prove $x_1 + x_2 \leq 4$ is a facet defining inequality, one must first determine the dimension of $conv(P)$. The dimension of $conv(P)$ is 2 because it is bounded from the top by the fact that there are 2 variables and bounded from below by the fact that there are 3 affinely independent points in P : (0,0),(0,1) and (1,0). Combining these two statements means that the dimension of $conv(P)$ is 2.

The constraint $x_1 + x_2 \leq 4$ is valid as it does not eliminate any points in P. Furthermore, the face $F = \{x \in conv(P) : x_1 + x_2 = 4\}$ is not $conv(P)$ since $(0, 0)$ is in P and not in F. Thus, the dimension of the face, $dim(F)$, is less than or equal to 1. The points $(0, 4)$ and $(2, 2)$ are in P and also in F. These points are affinely independent. Thus, $dim(F) \geq 1$. Thus $x_1 + x_2 \leq 4$ is a facet defining inequality.

Knapsack problems are used extensively in this thesis. Whence, the next section formally introduces knapsack problems and demonstrates them through the use of an example.

2.3 Knapsack Problems

Knapsack problems (KP) are a special class of IPs. The knapsack problem is so named by the concept problem where a hiker needs to select which items to bring on an overnight

camping trip and has limited strength. Each possible item has a corresponding non-negative weight and benefit. The purpose is to maximize the benefit the hiker receives from the items in his knapsack subject to constraints on how much weight the hiker can carry.

To model the knapsack as an IP, let $x_j = 1$ if item j is taken and 0 else. Then the KP is Maximize $\sum_{j=1}^{n} c_j x_j$ subject to $\sum_{j=1}^{n} a_j x_j \leq b$, and $x_j \in \{0,1\}$ for all $j \in N$ where c and $a \in \mathbb{R}_+^n, b \in \mathbb{R}_+$. Denote the feasible solutions of a KP as $P^{KP} = \{x \in \{0,1\}^n : \sum_{j=1}^{n} a_j x_j \leq b\}$.

The multiple knapsack problem (MK) is a type of integer program with a finite number of knapsack constraints. It is defined as Maximize $c^T x$ subject to $Ax \leq b$ and $x \in \{0,1\}^n$ where $c \in \mathbb{R}_+^n, A \in \mathbb{R}_+^{m*n}$. Define the feasible solutions of an MK to be $P^{MK} = \{x \in \{0,1\}^n : Ax \leq b\}$.

Both KP and MK problems have many real life applications that many researchers have worked on in a variety of fields and industries. These applications include project and portfolio selection [7], production planning and inventory problems [11], profit maximization applications [13, 36] and machine scheduling [30].

Without loss of generality, assume that any KP has the property that $a_1 \geq a_2 \geq ... \geq a_n$. Furthermore, in the remainder of the document, assume that every subset of N is sorted in ascending order. Finally, assume $a_1 \leq b$. If not, then $x_1 = 0$ for all feasible solutions and x_1 can be removed from the problem. Given these assumptions, $dim(conv(P^{KP})) = n$ because the origin and ξ_i for all $i \in N$ are in P^{KP}. The $conv(P^{MK})$ is full dimensional assuming that each $a_{ij} \in A$ satisfies $a_{ij} \leq b_i$, for all $i = 1, ..., m$ and $j = 1, ..., n$.

To assist with understanding the concepts of knapsack problems an example is shown. This example contains the classic hiker with a knapsack. This example shows the setup and

solution to the KP problem.

A hiker considers taking 15 items on a fishing trip to Yellowstone National Park. He can take any of the items at his own discretion, subject to the constraints. Each item has a corresponding weight and benefit as shown in Table 2.1. Furthermore, the hiker can only carry 40 units.

Object	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Weight	21	17	16	15	12	11	10	8	7	6	5	4	3	3	1
Benefit	30	30	16	15	1	18	14	11	10	7	9	12	7	2	4

Table 2.1: Benefits and Weights of Items

The hiker can choose to take an item in which case $x_j = 1$, or not take the item in which case $x_j = 0$. The problem can be formulated as an integer programming model as shown.

$$\begin{aligned}
\text{Maximize:} \quad & 30x_1 + 30x_2 + 16x_3 + 15x_4 + 1x_5 + 18x_6 + 14x_7 + 11x_8 + 10x_9 \\
& +7x_{10} + 9x_{11} + 12x_{12} + 7x_{13} + 2x_{14} + 4x_{15} \\
\text{Subject to} \quad & 21x_1 + 17x_2 + 16x_3 + 15x_4 + 12x_5 + 11x_6 + 10x_7 + 8x_8 + 7x_9 + 6x_{10} \\
& +5x_{11} + 4x_{12} + 3x_{13} + 3x_{14} + 1x_{15} \leq 40 \\
& x_i \in \{0, 1\} \quad \forall i = \{1, ..., 15\}
\end{aligned}$$

Solving this shows that the maximum benefit the hiker can obtain is 76 by carrying items 2, 6, 11, 12 and 13. The hiker is carrying a total weight to carry of 40 units.

2.4 Cover Inequalities

One widely used technique for solving KP and MK problems is by adding a cover cut to the constraints of the problem. Given a knapsack constraint, a set $C \subseteq N$ is a cover for a KP if $\sum_{j \in C} a_j > b$. The corresponding cover inequality takes the form $\sum_{j \in C} x_j \leq |C| - 1$. This inequality is valid because it is created from the original KP constraints and carrying every element in the cover is too heavy. Thus, the hiker can carry at most one item less than the number in the cover.

A cover C is a minimal cover if $C \setminus \{j\}$ is not a cover for all $j \in C$. If $C \subseteq N$ is a cover, define an extended cover as $E(C) = C \cup \{j \in N : a_j \geq a_i, \forall i \in C\}$. An extended cover has a valid inequality of the form $\sum_{j \in E(C)} x_j \leq |C| - 1$.

Recall the example of the hiker in Yellowstone. A cover of the weight constraint, for example, would be $\{1, 2, 3\}$ because $21 + 17 + 16 > 40$ and the resulting cover constraint would be $x_1 + x_2 + x_3 \leq 2$. This is also a minimal cover because if any element is taken out of the cover, it ceases to be a cover. An example of a non-minimal cover is $\{1, 2, 3, 4\}$ because element 4 can be taken out of the cover and it remains a cover. Minimal covers are the most desirable covers. In fact, a nonminimal cover inequality is dominated by a minimal cover inequality.

In order to understand that the concepts in this thesis are new, it is important to understand different types of lifting. Lifting is similar to multiple variable merging. Thus, to understand the difference between multiple variable merging and lifting, background information on lifting is covered next.

2.5 Lifting

Lifting is used to strengthen inequalities. Lifting can increase the dimension of a cutting plane. This makes a stronger inequality by cutting off a larger linear relaxation space. Lifting was developed by Gomory [17].

In formal terms, lifting requires a set $E \subseteq N$, an ordered set K containing $|E|$ integers and a valid inequality, $\sum_{i \in E} \alpha_i x_i + \sum_{i \in N/E} \alpha_i x_i \leq \beta$ of the restricted space on E and K. Formally, define the restricted space of P on E and K to be $P^{E,K} = conv\{x \in P : x_i = k_i$ for all $i \in E\}$ where $k_i \in \mathbb{Z}$ and $K = (k_1, k_2, ..., k_{|E|})$. Lifting ends with a valid inequality of $conv(P)$ that takes the form $\sum_{i \in E} \alpha'_i x_i + \sum_{i \in N/E} \alpha_i x_i \leq \beta'$.

The types of lifting are dependent upon the choice of E, K, α' and β'. There are 24 different types of lifting and include up, down, middle, exact, approximate, sequential, simultaneous, single and synchronized.

Up lifting, the most common lifting technique, assumes each k is at the lower bound, typically $K = (0, 0, ..., 0)$. Down lifting assumes that all of the k_i's are forced to be the upper bound of x_i. Middle lifting is when the k_i's are set to values in between.

Exact lifting generates the strongest inequality possible given a starting valid inequality. Thus, any increase to α' or decrease in β' would make the inequality invalid. Typically exact lifting requires solving an optimization problem. Approximate lifting has weaker α' or β' values.

Sequential lifting requires $|E| = 1$. Thus, each variable is lifted by individually solving an optimization problem. Simultaneous lifting requires $|E| \geq 2$. In this case, numerous

coefficients can be changed by solving a single optimization problem.

An alternate class of lifting was proposed by Bolton in 2009 [5]. Her work identified single and synchronized lifting. A lifting algorithm is single if it creates exactly one inequality. A lifting method is synchronized if the technique produces multiple valid inequalities.

These lifting classes are frequently combined and the most popular is exact sequential single up lifting. The following example demonstrates how to perform this type of lifting on a cover inequality and helps to clarify these methods.

2.5.1 Lifting Example

Consider the knapsack constraint

$$\begin{aligned} & 23x_1 + 22x_2 + 17x_3 + 15x_4 + 14x_5 + 14x_6 + 13x_7 + 12x_8 + 10x_9 + 9x_{10} + \\ & 8x_{11} + 7x_{12} + 7x_{13} + 5x_{14} + 4x_{15} && \leq 86 \\ & x_i \in \{0,1\} \ \ \forall \ \ i = 1, ..., 8 \end{aligned}$$

Observe that $C = \{4, 5, 6, 7, 8, 9, 10, 11\}$ is a cover. Thus the valid cover inequality is $x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} \leq 7$. In order to sequentially uplift x_3, solve the following IP:

$$\begin{aligned} z^* = Maximize \quad & x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} \\ \text{subject to} \quad & 23x_1 + 22x_2 + 17x_3 + 15x_4 + 14x_5 + 14x_6 + 13x_7 + 12x_8 + \\ & 10x_9 + 9x_{10} + 8x_{11} + 7x_{12} + 7x_{13} + 5x_{14} + 4x_{15} && \leq 86 \\ & x_3 && = 1 \\ & x_i \in \{0,1\} \ \ \forall \ \ i = 1, ..., 8 \end{aligned}$$

The solution to the above IP is $z^* = 6$. This means that $\alpha_3 = \beta - z^* = 7 - 6 = 1$. The resulting valid inequality is $x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} \leq 7$.

In order to lift x_2, solve $z* =$ Maximize $x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11}$ subject to $23x_1 + 22x_2 + 17x_3 + 15x_4 + 14x_5 + 14x_6 + 13x_7 + 12x_8 + 10x_9 + 9x_{10} + 8x_{11} + 7x_{12} + 7x_{13} + 5x_{14} + 4x_{15} \leq 86$ and $x_2 = 1$. The solution is z^*=5 and $\beta - z^* = 2$. The resulting valid inequality is $2x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} \leq 7$.

After this is run for x_1 the result is a stronger inequality with $\alpha_1 = 1$. The resulting inequality is $x_1 + 2x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} \leq 7$. This inequality is facet defining.

There has been a significant amount of research into sequential exact up lifting by Cho et al. [8], Gutierrez [22], Wolsey [40] and Hammer et. al. [23]. Approximate lifting techniques can be found in Balas [2] Wolsey [42] and Gu, et al.[21]. Exact simultaneous up lifting results are located in Easton and Hooker [15] and Kubik [31]. In 2009 Bolton developed exact synchronized simultaneous uplifting (SSL) for binary knapsack problems. This thesis' focus is not on lifting and these references merely scratch the surface of lifting results.

2.6 Inequality Merging

This thesis' research supplies the theoretical foundations for developing a variation of merging inequalities. Hickman and Easton [24] first introduced the concept of merging inequalities and their work provides related background information. Merging cover inequalities requires a knapsack constraint and two covers that overlap by exactly one index. One cover is called

the host and the other is called the donor. Due to the related nature of their work, an example is provided to demonstrate their method.

Consider the following knapsack constraint.

$$\begin{aligned} &19x_1 + 18x_2 + 17x_3 + 15x_4 + 14x_5 + 14x_6 + 13x_7 + 12x_8 + 10x_9 + 9x_{10} + \\ &8x_{11} + 8x_{12} + 7x_{13} + 7x_{14} + 5x_{15} + 4x_{16} && \leq 86 \\ &x_i \in \{0,1\} \ \ \forall \ \ i = 1, ..., 8 \end{aligned}$$

Let the host cover be $C^1 = \{1, 2, 3, 4, 5, 6\}$ and the donor cover be $C^2 = \{6, 7, 8, 9, 10, 11, 12, 13, 14\}$. Observe that C^1 and C^2 are covers with a single overlapping index, 6. The two cover inequalities are

$$\begin{aligned} x_1 + x_2 + x_3 + x_4 + x_5 + x_6 &\leq 5 \\ x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} + x_{12} + x_{13} + x_{14} &\leq 8. \end{aligned}$$

To merge these two constraints, x_6 is removed from the first cover inequality and is replaced by the left hand side of the donor cover divided its right hand side. Thus, the merged cover inequality is

$$\begin{aligned} &x_1 + x_2 + x_3 + x_4 + x_5 + \tfrac{1}{8}x_6 + \tfrac{1}{8}x_7 + \tfrac{1}{8}x_8 + \tfrac{1}{8}x_9 + \tfrac{1}{8}x_{10} + \tfrac{1}{8}x_{11} + \tfrac{1}{8}x_{12} + \\ &\tfrac{1}{8}x_{13} + \tfrac{1}{8}x_{14} && \leq 5 \end{aligned}$$

This work proved that a merged cover inequality is valid if the set of the host cover minus the overlapping element union the largest index in the donor cover is a cover. It is trivially verified that the set $C^1 \setminus \{6\} \cup \{14\}$ is a cover. Thus, the merged inequality is valid.

Besides introducing this class of inequalities, their work also provided conditions for facet defining and proved that merging creates a new class of inequalities. Furthermore, a

computational study showed that including these cuts can reduce the effort required to solve some integer programs.

While Hickman and Easton's results are strong, there still remains several unresolved issues. Can one merge covers that overlap by multiple variables? Does the coefficient by which the inequality is scaled have to be one divided by the right hand side of the inequality? The following chapter provides answers to these important questions and expands on Hickman and Easton's work by merging inequalities on multiple variables.

Chapter 3

Merging Valid Inequalities on Multiple Variables

This chapter introduces multiple variable merging of valid inequalities (MVMI) emphasizing on the knapsack polytope and cover inequalities. A linear time algorithm, the multiple variable merging cover algorithm (MVMCA), is developed to find these inequalities. Conditions under which MVMCA can produce a facet defining inequality are presented. The chapter concludes with an example that shows an MVMCI facet defining inequality and steps through MVMCA's process.

3.1 Multiple Variable Merging of Valid Inequalities

Hickman and Easton [24] introduced merging valid inequalities on a single variable. This section strengthens that research by merging valid inequalities on multiple variables using

a host inequality and donor inequality. Furthermore the type of merging introduced here dominates Hickman and Easton's results.

To begin the discussion of multiple variable merging of valid inequalities, consider an IP. Let there exist a host set of indices, $H \subseteq N$, and a donor set of indices, $D \subseteq N$. Assume that $\sum_{i \in H} \alpha_i^h x_i \leq \beta^h$ and $\sum_{i \in D} \alpha_i^d x_i \leq \beta^d$ are valid inequalities of $conv(P)$. A multiple variable merged inequality (MVMI) takes the form $\sum_{i \in H \setminus D} \alpha_i^h x_i + \alpha' \sum_{i \in D} \alpha_i^d x_i \leq \beta^h$. Several questions instantly are created from this procedure. How does one determine an α' where the MVMI is valid? Can an MVMI be facet defining? Does there exist useful MVMIs?

A natural direction to begin exploring answers to these questions occurs if P is restricted to P^{KP} and the sets H and D are covers. Multiple variable merging of covers requires a knapsack constraint and two covers. The two covers from now on will be referred to as a host and donor cover. These are so named because the host cover receives the variables from the donor cover. The donor cover provides the overlapping and non-overlapping variables. The resulting equations are merged with a merging coefficient, α, that is multiplied by the coefficients of the donor cover.

Formally, given a knapsack constraint and two covers $C^h \subseteq N$ and $C^d \subseteq N$, a multiple variable merging cover inequality, $MVMCI_{C^h,C^d,\alpha}$, takes the form $\sum_{i \in C^h \setminus C^d} x_i + \alpha \sum_{i \in C^d} x_i \leq |C^h| - 1$.

Clearly, if $\alpha = 0$, then $MVMCI_{C^h,C^d,0}$ is valid. Furthermore, this inequality is dominated by the C^h cover inequality. Additionally, if $\alpha > |C^h| - 1$, then $MVMCI_{C^h,C^d,\alpha}$ is invalid. One should seek the maximum α that retains validity.

The multiple variable merging cover algorithm determines an α that creates a valid $MVMCI_{C^h,C^d,\alpha}$. The input to MVMCA is a knapsack constraint and two covers $C^h \subseteq N$ and $C^d \subseteq N$ where $C^h = \{i_1^h, i_2^h, ..., i_{|C^h|}^h\}$ and $C^d = \{i_1^d, i_2^d, ..., i_{|C^d|}^d\}$. Define the overlapping variables $M = C^h \cap C^d$ where $M = \{i_1^m, i_2^m, ..., i_{|M|}^m\}$ and $|m| \geq 1$.

The basic idea of MVMCA is to determine feasible points and to adjust alpha accordingly. The feasible points are calculated by summing the smallest *counth* coefficients associated with elements of C^h and the smallest *countd* coefficients associated with elements of C^d. If this sum is less than b, then a feasible point exists and an α' value is calculated. If this α' value is less than α, then α is replaced. At the end of the process the α is multiplied by the donor variables to provide a valid MVMCI. Formally,

Multiple Variable Merging Cover Algorithm (MVMCA)

$\alpha \leftarrow \infty$

$countd \leftarrow 1$

$counth \leftarrow |C^h \setminus C^d|$

$sum \leftarrow \sum_{i \in C^h \setminus C^d} a_i + a_{i^d_{|C^d|}}$

If $sum \leq b$ *And* $|C^h \setminus C^d| = |C^h| - 1$, *Then* $\alpha \leftarrow 0$.

While $countd \leq |C^d|$ *Begin*

If $sum \leq b$, *Then*

If $\frac{|C^h|-1-counth}{countd} < \alpha$, *Then* $\alpha \leftarrow \frac{|C^h|-1-counth}{countd}$

$sum \leftarrow sum + a_{i^d_{|C^d|}-countd}$

$countd \leftarrow countd + 1$

End If Else

$sum \leftarrow sum - a_{i^h_{|C^h \setminus C^d| - counth + 1}}$

If $counth = 0$, *Then* $countd \leftarrow countd + 1$

Else $counth \leftarrow counth - 1$

End Else

End While

Report α

Determining the computational effort required by MVMCA is accomplished through amortized analysis. The initialization requires $O(|C^h| + |C^d|)$. In each iteration of the main loop, either *countd* is increased by one or *counth* is decreased by one. Consequently, the loop is repeated $O(|C^h| + |C^d|)$. Each iteration of the loop trivially requires $O(1)$. Thus, the main loop has $O(|C^h| + |C^d|)$ effort. Reporting α is $O(1)$. Therefore, the effort required to run MVMCA is $O(|C^h| + |C^d|)$, which is bounded by $O(n)$ and MVMCA is a linear time algorithm.

Any cutting plane algorithm on a knapsack constraint requires reading in the instance, which is $\Omega(n)$. Thus, multiple variable merging on covers is a problem that is $\Theta(n)$. Consequently, MVMCA theoretically requires the least effort to generate valid MVMCIs.

The following theorem proves that MVMCA returns an α that creates a valid MVMCI. Prior to providing this result and to simplify the notation, define $C^h \setminus C^d = \{i^{hd}_1, ..., i^{hd}_{|C^h \setminus C^d|}\}$.

Theorem 3.1 *Given a knapsack constraint* $\sum_{i \in N} a_i x_i \leq b$ *and two covers* C^h *and* $C^d \subseteq N$ *such that* $|C^h \cap C^d| \geq 1$. *Then* $\sum_{i \in C^h \setminus C^d} x_i + \alpha' \sum_{i \in C^d} x_i \leq |C^h| - 1$ *is valid for conv*(P^{KP}) *for any* $\alpha' \leq \alpha$ *where* α *is returned from MVMCA.*

Proof: Let $\sum_{i \in N} a_i x_i \leq b$ be a knapsack constraint with two covers $C^h \subseteq N$ and $C^h \subseteq N$. For contradiction, assume there exists an $\alpha' \leq \alpha$ such that $\sum_{i \in C^h \setminus C^d} x_i + \alpha' \sum_{i \in C^d} x_i \leq |C^h| - 1$ is not valid for $conv(P^{KP})$. Therefore, there exists an $x' \in P^{KP}$ such that $\sum_{i \in C^h \setminus C^d} x'_i + \alpha' \sum_{i \in C^d} x'_i > |C^h| - 1$.

Define $q = \sum_{i \in C^h \setminus C^d} x'_i$ and $p = \sum_{i \in C^d} x'_i$. Then $\alpha' p > |C^h| - 1 - q$ and $\alpha' > \frac{|C^h|-1-q}{p}$. During the iteration when $counth = q$ of MVMCA, $sum = \sum_{k=|C^h \setminus C^d|-q+1}^{|C^h \setminus C^d|} a_{i_k^{hd}} + \sum_{k=|C^d|-p+1}^{|C^d|} a_{i_k^d} \leq \sum_{i \in N} a_i x'_i \leq b$ with the last inequality due to the feasibility. Thus, MVMCA reports an $\alpha \leq \frac{|C^h|-1-q}{p}$, which contradicts $\alpha' > \alpha$.

□

The MVMCA algorithm provides the strongest α coefficient possible. Observe that the α returned from MVMCI is a supporting cutting plane, because there exists a feasible point that meets this inequality at equality. The point on MVMCI's face is $\sum_{j=|C^h \setminus C^d|-counth'+1}^{|C^h \setminus C^d|} \xi_{i_j^{hd}} + \sum_{j=|C^d|-countd'+1}^{|C^d|} \xi_{i_j}$ where $counth'$ and $countd'$ are the values of $counth$ and $countd$ that generated the smallest α from MVMCA. Thus, any increase in α would create an invalid inequality.

Besides finding the strongest α, MVMCA also aids in identifying whether an MVMCI can be facet defining. The following result provides these conditions. This theorem has a substantial amount of notation and an example can be seen later in this chapter.

Theorem 3.2 *Given a knapsack constraint* $\sum_{i\in N} a_i x_i \leq b$ *and two covers* C^h *and* $C^d \subseteq N$ *with* $|C^h \cap C^d| \geq 1$. *The inequality* $\sum_{i\in C^h\setminus C^d} x_i + \alpha \sum_{i\in C^d} x_i \leq |C^h| - 1$ *is facet defining for* $conv(P^{KP})$ *if* α *is returned from MVMCA, MVMCA had a tie for the minimum* α *value occuring from* $counth_1, countd_1$ *and* $counth_2, countd_2$ *and the following conditions are met.*

i) The set $S_1 = \{i^{hd}_1, i^{hd}_{|C^h\setminus C^d|-counth_1+2}, ..., i^{hd}_{|C^h\setminus C^d|}\} \cup \{i^{d}_{|C^d|-countd_1+1}, ..., i^{d}_{|C^d|}\}$ *is not a cover.*

ii) The set $S_2 = \{i^{hd}_{|C^h\setminus C^d|-counth_1}, ..., i^{hd}_{|C^h\setminus C^d|-1}\} \cup \{i^{d}_{|C^d|-countd_1-1}, ..., i^{d}_{|C^d|}\}$ *is not a cover.*

iii) The set $S_3 = \{i^{hd}_{|C^h\setminus C^d|-counth_2+1}, ..., i^{hd}_{|C^h\setminus C^d|}\} \cup \{i^{d}_1, i^{d}_{|C^d|-countd_2+2}, ..., i^{d}_{|C^d|}\}$ *is not a cover.*

iv) The set $S_4 = \{i^{hd}_{|C^h\setminus C^d|-counth_2+1}, ..., i^{hd}_{|C^h\setminus C^d|}\} \cup \{i^{d}_{|C^d|-countd_2}, ..., i^{d}_{|C^d|-1}\}$ *is not a cover.*

v) Either $\{l \in N : l = argmax\{a_i : i \in N \setminus (C^h \cup C^d)\}\} \cup \{i^{hd}_{|C^h\setminus C^d|-counth+1}, ..., i^{hd}_{|C^h\setminus C^d|}\} \cup \{i^{d}_{|C^d|-countd_1+1}, ..., i^{d}_{|C^d|}\}$ *or* $\{k\} \cup \{i^{hd}_{|C^h\setminus C^d|-counth_2+1}, ..., i^{hd}_{|C^h\setminus C^d|}\} \cup \{i^{d}_{|C^d|-countd_2+1}, ..., i^{d}_{|C^d|}\}$ *are not covers.*

Proof: Assume $\sum_{i\in N} a_i x_i \leq b$ is a knapsack constraint with two covers C^h and $C^d \subseteq N$ and α is returned from MVMCA. Furthermore assume MVMCA had a tie for the minimum α value occuring from $counth_1, countd_1$ and $counth_2, countd_2$ and conditions $i)$, $ii)$, $iii)$, $iv)$ and $v)$ are met.

The MVMCA returns an α value where the MVMCI is valid by Theorem 3.1. The point $x' \in P^{KP}$ where $x'_i = 0$ for all $i \in N$ is feasible and it does not meet the MVMCI at equality, thus the face generated by the MVMCI is not $conv(P^{KP})$. Hence this face's dimension is bounded by $n-1$. Consequently, it is sufficient to show that there are n affinely independent points in P^{KP} that meet the MVMCI at equality.

Consider the following $|C^h \cup C^d|$ points:

$\xi_{i_k^{hd}} + \sum_{j=|C^h \setminus C^d|-counth_1+2}^{|C^h \setminus C^d|} \xi_{i_j^{hd}} + \sum_{j=|C^d|-countd_1+1}^{|C^d|} \xi_{i_j}$ for $k = 1, ..., |C^h \setminus C^d| - counth_1 - 1$. The point when $k = 1$ is feasible since S_1 is not a cover. The remaining points are feasible due to the sorted order of the sets and the fact that a is sorted in descending order.

$\sum_{j=|C^h \setminus C^d|-counth_1}^{|C^h \setminus C^d|} \xi_{i_j^{hd}} - \xi_{i_k^{hd}} + \sum_{j=|C^d|-countd_1+1}^{|C^d|} \xi_{i_j}$ for $k = |C^h \setminus C^d| - counth_1, ..., |C^h \setminus C^d|$. The point when $k = |C^h \setminus C^d|$ is feasible due to S_2 not being a cover. The remaining points are feasible due to the sorted order of the set and the fact that a is sorted in descending order.

$\sum_{j=|C^h \setminus C^d|-counth_2+1}^{|C^h \setminus C^d|} \xi_{i_j^{hd}} + \xi_{i_k^d} + \sum_{j=|C^d|-countd_2+2}^{|C^d|} \xi_{i_j^d}$ for $k = 1, ..., |C^d| - countd_2 - 1$. The point when $k = 1$ is feasible since S_3 is not a cover. The remaining points are feasible due to the sorted order of the set and the fact that a is sorted in descending order.

$\sum_{j=|C^h \setminus C^d|-counth_2+1}^{|C^h \setminus C^d|} \xi_{i_j^{hd}} + \sum_{j=|C^d|-countd_2}^{|C^d|} \xi_{i_j^d} - \xi_{i_k^d}$ for $k = |C^d| - countd_2, ..., |C^d|$. The point when $k = |C^d|$ is feasible due to S_4 not being a cover. The remaining points are feasible due to the sorted order of the set and the fact that a is sorted in descending order.

The remaining $n - |C^h \cup C^d|$ points divide into two cases depending upon which set is not a cover from assumption v).

Assume $\{k\} \cup \{i^{hd}_{|C^h \setminus C^d|-counth_1+1}, ..., i^{hd}_{|C^h \setminus C^d|}\} \cup \{i^d_{|C^d|-countd_1+1}, ..., i^d_{|C^d|}\}$ is not a cover where $k = argmax\{a_i : i \in N \setminus (C^h \cup C^d)\}$. Consider the points $\sum_{j=|C^h \setminus C^d|-counth_1+1}^{|C^h \setminus C^d|} \xi_{i_j^{hd}} + \sum_{j=|C^d|-countd_1+1}^{|C^d|} \xi_{i_j^d} + \xi_k$ for all $k \in N \setminus (C^h \cup C^d)$. The point when $k = l = argmax\{a_i : i \in N \setminus (C^h \cup C^d)\}$ is feasible, due to the assumption of no cover for this set. All other k are feasible since this is the maximum such a coefficient not in $C^h \cup C^d$.

Alternately, assume $\{k\} \cup \{i^{hd}_{|C^h \setminus C^d| - counth_2 + 1}, ..., i^{hd}_{|C^h \setminus C^d|}\} \cup \{i^{d}_{|C^d| - countd_2 + 1}, ..., i^{d}_{|C^d|}\}$ is not a cover where $k = argmax\{a_i : i \in N \setminus (C^h \cup C^d)\}$. Consider the points $\sum_{j=|C^h \setminus C^d| - counth_2 + 1}^{|C^h \setminus C^d|} \xi_{i^{hd}_j} + \sum_{j=|C^d| - countd_2 + 1}^{|C^d|} \xi_{i^{d}_j} + \xi_k$ for all $k \in N \setminus (C^h \cup C^d)$. The point when $k = l = argmax\{a_i : i \in N \setminus (C^h \cup C^d)\}$ is feasible, due to the assumption of no cover for this set. All other k are feasible since this is the maximum such a coefficient not in $C^h \cup C^d$.

These n points have just been shown to be feasible. Furthermore, each of these n points has the property that $\sum_{i \in C^h \setminus C^d} x_i + \alpha \sum_{i \in C^d} x_i = |C^h| - 1$, since each point either has $counth_1$ and $countd_1$ variables set to one corresponding to elements in C^h and C^d or $counth_2$ and $countd_2$ variables set to one corresponding to elements in C^h and C^d. Thus these points are in MVMCI's face.

To show that these points are affinely independent, consider the first $|C^h \setminus C^d|$ points (the first two sets of points). The first $k = 1, ..., |C^h \setminus C^d| - counth_1 - 1$ points are affinely independent because each $x_{i^{hd}_k}$ only has a single 1 for $k = 1, ..., |C^h \setminus C^d| - counth_1 - 1$. The next $counth_1 + 1$ points are a cyclic permutation of $counth_1$ consecutive ones over $counth_1 + 1$ columns. Since $counth_1$ and $counth_1 + 1$ are relatively prime, these points are affinely independent.

The next $|C^d|$ points (the third and fourth sets of points) are affinely independent. Observe that the first $k = 1, ..., |C^d| - countd_2 - 1$ points are affinely independent because each $x_{i^{hd}_k}$ only has a single 1 for $k = 1, ..., |C^d| - countd_2 - 1$. The next $countd_2 + 1$ points are a cyclic permutation of $countd_2$ consecutive ones over $countd_2 + 1$ columns. Since $countd_2$ and $countd_2 + 1$ are relatively prime, these points are affinely independent.

Because the points $(counth_1, countd_1) \neq (counth_2, countd_2)$, these two sets of points

are affinely independent from each other. The final $N \setminus (C^h \cup C^d)$ points are independent from the other points because they are the only points to have a 1 in the x_k's row where $k \in N \setminus (C^h \cup C^d)$. Thus, these are n feasible affinely independent points that meet the MVMCI at equality and its face is at least dimension $n-1$. Thus the MVMCI is a facet defining inequality.

□

The next section steps through the process of calculating an MVMCI for an example instance, and the implementation of MVMCA. It then demonstrates that the identified MVMCI is facet defining. The chapter concludes with changing C^h and C^d and rerunning MVMCA.

3.2 MVMCI Example

Consider the feasible region defined by

$$19x_1 + 18x_2 + 17x_3 + 15x_4 + 14x_5 + 14x_6 + 13x_6 + 12x_7 + 10x_9 + 9x_{10} + 8x_{11} \quad \leq 86$$

$$x_i \in \{0,1\} \quad \forall \quad i = 1, ..., 11.$$

The input for MVMCA is a knapsack constraint and two covers. Observe that $C^h = \{1,2,3,4,5,6\}$ and $C^d = \{4,5,6,7,8,9,10,11\}$ are covers. Thus, the overlapping variables are $C^h \cap C^d = \{4,5,6\}$ and this example merges on 3 variables. The form of this merged inequality is

$$MVMCI_{C^h,C^d,\alpha} \quad x_1 + x_2 + x_3 + \alpha(x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11}) \quad \leq 5.$$

MVMCA begins by setting α arbitrarily high, *countd* is 1 and $counth = |C^h \setminus C^d| = 3$. Furthermore, $C^h \setminus C^d = \{1, 2, 3\}$, $a_{i^d_{|C^d|}} = a_{11} = 8$, and $sum = a_1 + a_2 + a_3 + a_{11} = 62$. Since $|C^h \setminus C^d| \leq 5$, α is not set to 0.

The main loops checks whether or not *sum* is less than $b = 86$. Since $sum \leq 86$ and the condition $\frac{|C^h|-1-counth}{countd} = \frac{6-1-3}{1} = 2 < \alpha$, α is updated to $\frac{6-1-3}{1} = 2$. The reasoning behind this step is to determine whether or not P^{KP} has a feasible point with *counth* variables corresponding to elements in $C^h \setminus C^d$ set to one and *countd* variables corresponding to elements in C^d set to one. In this case, this requires determining whether there exists a point in P^{KP} with 3 elements in $C^h \setminus C^d$ and one element in C^d set to one. Since $(1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1) \in P^{KP}$, any $\alpha > 2$ would create an invalid inequality, due to this point. Consequently, MVMCA must return a value of $\alpha \leq 2$.

Next *sum* is updated to $sum + a_{i^d_{|C^d|}-countd} = sum + a_{10} = 71$ and $countd = 2$. Once the counts have been updated, the elements corresponding to these counts have a sum of 71 which is less than 86 so α is calculated to be $\frac{5-2}{3} = 1$. This is less than the previous α of 2, so α is updated to 1.

The loop repeats itself to find the next α adding in the next C^d variable. This eventually finds an alpha for every valid combination of $C^h \cup C^d$ variables by taking advantage of the sorted orders. In the next iteration, the indices $\{1, 2, 3\}$ and $\{9, 10, 11\}$ are tested and $sum = 83$ with $\alpha' = \frac{5-3}{3} = \frac{2}{3}$. Because $\alpha' < \alpha = 1$, α is set to $\frac{2}{3}$.

The next iteration has $sum = 95$, which is greater than $b = 86$. Since *sum* is greater than b from the knapsack constraint, α is not updated and the smallest index in C^h is taken out of the set, leaving $counth = 2, countd = 4$, equivalent to $(0, 1, 1, 0, 0, 0, 1, 1, 1, 1)$. Thus,

$sum = 95 - 19 = 76$. This new point is now feasible, so α is calculated again to be $\frac{(5-2)}{4} = \frac{3}{4}$.

This process is repeated until $counth = 0$ and $countd = |C^d|$, in this case 8. This is by definition infeasible because $\{4, ..., 11\}$ is a cover and thus including all the donor variables is infeasible.

A summary of this process is shown below in Table 3.1. This table shows every iteration of MVMCA. The first column in Table 3.1 contains the set of host indices that are included in the calculation of that iteration's *sum*, meaning that the corresponding coefficients in the original inequality for those variables are added to obtain *sum*. The second column is *counth*, which is a reference to the actual values that are entered into the α' equation. Column 3 has the donor indices used to calculate *sum*. The *countd* column has the number of donor variables' coefficients and is used to calculate α'. The sum of the coefficients of the host and donor indices is presented in the fifth column. The sixth column tells whether the indicated point is feasible in the original inequality. The last column is the resulting α' from calculating $\frac{(|C^h|-1)-counth}{countd}$. A value of N/A in the last column means that the point indicated by that row is not feasible, and thus no α' needs to be calculated to accommodate it.

Notice the two bolded rows. These rows have the lowest α' values and generate the α returned by MVMCA to create $MVMCI_{C^h, C^d, \frac{2}{3}}$. The fact that there is a tie at $\alpha' = \frac{2}{3}$ becomes important to prove facet defining later in this chapter. Thus MVMCA returns $\alpha = \frac{2}{3}$ and creates $MVMCI_{C^h, C^d, \frac{2}{3}}$,

$$MVMCI_{C^h, C^d, \frac{2}{3}} \quad x_1 + x_2 + x_3 + \tfrac{2}{3}(x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11}) \quad \leq 5,$$

which is valid by Theorem 3.1.

Host Indices	$Counth$	Donor Indices	$Countd$	sum	Feasible?	α'
$\{1,2,3\}$	3	$\{11\}$	1	64	Yes	2
$\{1,2,3\}$	3	$\{10,11\}$	2	73	Yes	1
$\mathbf{\{1,2,3\}}$	**3**	$\mathbf{\{9,10,11\}}$	**3**	**83**	**Yes**	$\mathbf{\frac{2}{3}}$
$\{1,2,3\}$	3	$\{8,9,10,11\}$	4	95	No	N/A
$\{2,3\}$	2	$\{8,9,10,11\}$	4	75	Yes	$\frac{3}{4}$
$\{2,3\}$	2	$\{7,8,9,10,11\}$	5	88	No	N/A
$\{3\}$	1	$\{7,8,9,10,11\}$	5	67	Yes	$\frac{4}{5}$
$\mathbf{\{3\}}$	**1**	$\mathbf{\{6,7,8,9,10,11\}}$	**6**	**81**	**Yes**	$\mathbf{\frac{2}{3}}$
$\{3\}$	1	$\{5,6,7,8,9,10,11\}$	7	95	No	N/A
$\emptyset$	0	$\{5,6,7,8,9,10,11\}$	7	81	Yes	$\frac{5}{7}$
$\emptyset$	0	$\{4,5,6,7,8,9,10,11\}$	8	95	No	N/A

Table 3.1: Calculating MVMCA α' Results

An obvious goal is to show that $MVMCI_{C^h,C^d,\frac{2}{3}}$ eliminates a linear relaxation solution. The point $(\frac{1}{20},\frac{3}{10},\frac{3}{10},\frac{1}{10},\frac{3}{4},\frac{3}{4},1,1,1,1,1)$ is in P^{LR}, because this point requires the individual to carry a weight of 85.95, which is less than $b = 86$. When this point is substituted into the $MVMCI_{C^h,C^d,\frac{2}{3}}$, the left hand side sums to 5.049. However, the right hand side is $|C^h| - 1 = 5$. Thus, $MVMCI_{C^h,C^d,\frac{2}{3}}$ is a cutting plane and eliminates space from P_{LR}.

Now that it has been proven that $x_1+x_2+x_3+\frac{2}{3}(x_4+x_5+x_6+x_7+x_8+x_9+x_{10}+x_{11}) \leq 5$ is a cutting plane, proving that it is facet defining becomes important, because facet defining

$C^h \setminus C^d$	1	0	0	1	1	1	1	1	1	1	1
	0	1	0	1	1	1	1	1	1	1	1
	0	0	1	1	1	1	1	1	1	1	1
$C^h \cap C^d$	0	0	0	1	0	0	0	0	0	0	0
	0	0	0	0	1	0	0	0	0	0	0
	1	1	1	0	0	1	0	0	0	0	0
$C^d \setminus C^h$	1	1	1	0	0	0	1	0	0	0	0
	1	1	1	0	0	0	0	0	1	1	1
	1	1	1	0	0	0	0	1	0	1	1
	1	1	1	1	1	1	1	1	1	0	1
	1	1	1	1	1	1	1	1	1	1	0

Table 3.2: Affinely Independent Points

inequalities are the theoretically strongest inequalities. To prove that $MVMCI_{C^h,C^d,\frac{2}{3}}$ is facet defining one must show that its face has dimension 10.

The dimension is bounded by the number of affinely independent points in the face. There are 11 such affinely independent points, because the conditions of Theorem 3.2 are satisfied. These 11 points are given in Table 3.2. The following discussion proves these points are in the face and are affinely independent. It also provides additional insight into this theorem. Note that the lines in Table 3.2 are there to assist with understanding of the different indices in the sets $C^h \setminus C^d$, $C^h \cap C^d$, and $C^d \setminus C^h$.

A critical assumption in Theorem 3.2 is that there was a tie in the smallest α value. In this case, observe that MVMCA has two iterations where $\alpha' = \frac{2}{3}$. One case has $counth = 1$ and $countd = 6$, and the second case has $counth = 3$ and $countd = 3$. For the assumptions of Theorem 3.2, let $(counth_1, countd_1) = (1, 6)$ and $(counth_2, countd_2) = (3, 3)$.

The set $S_i = \{i_1^{hd}, i_{|C^h \setminus C^d| - counth_1 + 2}^{hd}, ..., i_{|C^h \setminus C^d|}^{hd}\} \cup \{i_{|C^d| - countd_1 + 1}^{d}, ..., i_{|C^d|}^{d}\} = \{1, 6, 7, 8, 9, 10, 11\}$ is not a cover. Thus, condition i) is true and the point in the first column in Table 3.2 is feasible. Since $a_1 \geq a_2 \geq a_3$, the points in the second and third column are also feasible. Each of these points is in the face because $1 + \frac{2}{3} * 6 = 5$, which is a direct result of choosing $counth_1$ variables associated with indices in $C^h \setminus C^d$ and $countd_1$ variables associated with indices in C^d.

The set $S_{iii} = \{i_{|C^h \setminus C^d| - counth_2 + 1}^{hd}, ..., i_{|C^h \setminus C^d|}^{hd}\} \cup \{i_1^{d}, i_{|C^d| - countd_2 + 2}^{d}, ..., i_{|C^d|}^{d}\} = \{1, 2, 3, 4, 10, 11\}$ is not a cover. Thus condition iii) is true and the point in the fourth column in Table 3.2 is feasible. Since $a_4 \geq a_5 \geq a_6 \geq a_7$, the fifth, sixth and seventh columns are feasible points also. These points all have corresponding $counth = 3$ and $countd = 3$ and thus they are in $MVMCI_{C^h, C^d, \frac{2}{3}}$'s face.

The set $S_{iv} = \{i_{|C^h \setminus C^d| - counth_2 + 1}^{hd}, ..., i_{|C^h \setminus C^d|}^{hd}\} \cup \{i_{|C^d| - countd_2}^{d}, ..., i_{|C^d| - 1}^{d}\} = \{1, 2, 3, 8, 9, 10\}$ is not a cover. Thus condition iv) is true and the point in the eleventh column of Table 3.2 is feasible. Because $a_8 \geq a_9 \geq a_{10} \geq a_{11}$, the eighth, ninth and tenth columns in Table 3.2 are feasible points. Again these points all have corresponding $counth = 3$ and $countd = 3$ and are in the face of $MVMCI_{C^h, C^d, \frac{2}{3}}$. Thus, these are 11 points that are feasible and on the desired face.

The reader should observe that $counth_1 = 1$. Thus, condition ii) is vacuously true.

Furthermore, $C^h \cup C^d = N$ and condition v) is also vacuously true.

To prove that these 11 points are affinely independent, observe that the first 3 points only have a single 1 in each of the first 3 rows. Thus, this set of points is affinely independent.

Now consider the fourth through eleventh points. The fourth to seventh points only have a single 1 in the fourth to seventh row. The eighth through eleventh columns are all constant except for the cyclical permutation of 3 ones over the last four rows. Since 3 and 4 are relatively prime, these points are affinely independent. Consequently, the fourth through eleventh points are affinely independent.

Since (1,6) is affinely independent from (3,3); these two sets of points are also affinely independent. Thus, these points are affinely independent, the face of $MVMCI_{C^h,C^d,\frac{2}{3}}$ has a dimension of 10, and this face is a facet of $conv(P^{KP})$.

Now that MVMCA has been shown to produce a facet defining inequality, it is necessary to show that this type of inequality could not have been obtained using any other method currently in use and also needs to be shown that it is not simply a slight variation on an existing strategy of finding cutting planes for the knapsack problem.

As mentioned in the introduction, Hickman and Easton's [24] work on merging inequalities was the catalyst to this research. Their method can only merge on a single variable and could not create $MVMCI_{C^h,C^d,\frac{2}{3}}$. Furthermore, MVMCA would generate stronger inequalities even if $|C^h \cap C^d| = 1$. That is, MVMCA would generate an α coefficient that is at least as large as the $\frac{1}{|C^d|-1}$ scaling coefficient generated by Hickman and Easton's method.

Starting with any integer inequality and performing any type of sequential lifting only

generates integer coefficients. Thus, starting with the C^h cover inequality and lifting sequentially could not create $MVMCI_{C^h,C^d,\frac{2}{3}}$. If $MVMCI_{C^h,C^d,\frac{2}{3}}$ is multiplied by $\frac{3}{2}$, it becomes $\frac{3}{2}x_1+\frac{3}{2}x_2+\frac{3}{2}x_3+x_4+x_5+x_6+x_7+x_8+x_9+x_{10}+x_{11} \leq \frac{15}{2}$. Clearly, this is not a sequentially lifted version of the C^d donor inequality. Consequently, sequential lifting cannot generate this $MVMCI_{C^h,C^d,\frac{2}{3}}$.

Simultaneous lifting could theoretically generate $MVMCI_{C^h,C^d,\frac{2}{3}}$ if given the correct initial valid inequality and the proper lifting weights. This inequality could be generated by starting with either of the valid inequalities $x_1+x_2+x_3 \leq 5$ or $x_4+x_5+x_6+x_7+x_8+x_9+x_{10}+x_{11} \leq \frac{15}{2}$. Neither of these inequalities are tight and both inequalities can be trivially strengthened to $x_1+x_2+x_3 \leq 3$ and $x_4+x_5+x_6+x_7+x_8+x_9+x_{10}+x_{11} \leq 7$. Thus, without consulting an oracle one could not generate $MVMCI_{C^h,C^d,\frac{2}{3}}$ with simultaneous uplifting.

Finally, numerous other methods exist to find valid inequalities. Such methods as Chvatal-Gomory cuts [9], mixed integer rounding cuts [12], superadditive cuts [18] and modular cuts [17] may be able to generate $MVMCI_{C^h,C^d,\frac{2}{3}}$, but not without substantial effort and possibly the consultation of an oracle. Thus, MVMCIs are a new class of cutting planes for the knapsack polytope.

After the implementation of MVMCA on an example, it is prudent to show that MVMCA could also be run by reversing the donor and host covers. Thus let $C'^h = \{6,7,8,9,10,11,12\}$ and $C'^d = \{1,2,3,4,5,6\}$. Table 3.3 provides a summary of the loops of MVMCA.

MVMCA returns α equal to 1, thus the valid inequality is $MVMCI_{C'^d,C'^h,1} = \sum_{i=1}^{11} x_i \leq 7$. This inequality is merely an extended cover inequality. In this particular case, little is gained from MVMCA. However, if one reduced the right hand side of the knapsack constraint

Host Indices	$Counth$	sum	Donor Indices	$Countd$	Feasible?	α
$\{7,8,9,10,11\}$	5	$\{6\}$	1	66	Yes	2
$\{7,8,9,10,11\}$	5	$\{5,6\}$	2	80	Yes	1
$\{7,8,9,10,11\}$	5	$\{4,5,6\}$	3	95	No	N/A
$\{8,9,10,11\}$	4	$\{4,5,6\}$	3	82	Yes	1
$\{8,9,10,11\}$	4	$\{3,4,5,6\}$	4	99	No	N/A
$\{9,10,11\}$	3	$\{3,4,5,6\}$	4	87	No	N/A
$\{10,11\}$	2	$\{3,4,5,6\}$	4	77	Yes	$\frac{5}{4}$
$\{10,11\}$	2	$\{2,3,4,5,6\}$	5	95	No	N/A
$\{11\}$	1	$\{2,3,4,5,6\}$	5	86	Yes	$\frac{6}{5}$
$\{11\}$	1	$\{1,2,3,4,5,6\}$	6	105	No	N/A
$\emptyset$	0	$\{1,2,3,4,5,6\}$	6	97	No	N/A

Table 3.3: Reversed MVMCA α Results

to 79, then the α value reported from MVMCA would have been $\frac{5}{4}$ and it would have resulted in an inequality that is stronger than an extended cover inequality.

To demonstrate the impact of MVMCA in more complicated examples, Chapter 4 presents a computational study. This study performs multiple runs with several examples of MVMCA being implemented on varying sizes of MK instances. Once the best strategy is chosen larger problems are solved to show the usefulness of MVMCIs.

Chapter 4

Computational Study

This section describes a computational study that is performed to demonstrate the usefulness of MVMCA in reducing the time required to obtain the optimal integer solution. The results show that MVMCI can improve this time an average of 6.3% with some situations yielding up to a 40% improvement.

This computational study is performed using a PC with an Intel i7 2.6 GHz processor and 8 GB of RAM. The study solves random knapsack instances using CPLEX 12.5 [10], a commercial optimization software package. The instances are solved with and without an MVMCI and the times are reported.

The random instances were generated according to standard practices [34] for benchmark instances. Various combinations of rows and columns are created and the instances follow the same format. The $a_{i,j}$ coefficients are randomly generated integers between 1 and 1,000. The tightness ratio is set to .35 and thus $b_i = .35 \sum_{j \in N} a_{i,j} \ \forall \ i \in \{1, ..., m\}$ where m is the number of rows. The objective coefficients are $c_j = \sum_{i=1}^{m} a_{i,j} + u$ where u is a uniformly

distributed random integer between 0 and 200.

To begin some small instances are thoroughly studied to provide recommended settings for more complex instances. To avoid random anomalies 15 instances of each size are created and the average of these 15 are reported. There are four sizes of instances run in this study, 5 rows with 50 and 75 variables, and 10 rows with 50 and 75 variables.

In a first attempt, the two covers for MVMCA were sorted purely by the size of the coefficients. This strategy did not return results with an improvement to CPLEX. Gu [19] suggests that sorting the indices based on the variables' reduced costs leads to better computational results.

The covers for MVMCA are generated by weighting the reduced costs by $\frac{3}{2}$ and adding that to the size of the coefficients of the first row. With the variables ranked, the program adds indices in this order until a host cover is obtained. A size of overlapping variables is selected. These overlapping variables are the largest indices in C^h. Next the program adds indices in the weighted order to the overlapping indices until a donor cover is obtained. The program sorts the indices of each cover in descending order by their coefficients and runs MVMCA.

The entire process of creating a valid MVMCI is extremely fast and required less than .01 seconds for every instance. Every instance generated an MVMCI off of the first row. Thus, MVMCIs are abundant. The MVMCIs are added as part of a preprocessing step and not as local cuts in the branching tree.

A primary concern is to determine whether or not to overlap the covers and if so, by how many variables. To accomplish this, the computational study on the smaller instances was

tested with different numbers of overlapping variables 1, 3, 5 and 10. Table 4.1 provides the data for these instances. The first 3 columns identify the size and settings of the instance. The next two report the average run times for the 15 replications of CPLEX with and without MVMCA added. The last column contains the percent decrease in run time from CPLEX to MVMCA implemented with CPLEX. Note that a negative number in this column means that CPLEX alone ran faster than the MVMCA problem formulation.

As can be seen, the improvement in these runs range from 40% to -7.15%. On average implementing MVMCA resulted in an improvement of 5.5%. Furthermore, the smaller the problem is, the worse MVMCA performs. This is most likely due to the fact that adding MVMCA, while linear in computational effort, does increase the number of constraints for each iteration of branch and bound. Thus, the cutting plane improvement has to be somewhat sizeable to be able to improve the run time. The room for improvement is so small because the run times themselves are so short that any improvement will be minimal at best. As the problems become larger, the results improve.

Table 4.2 summarizes the results from Table 4.1 based upon the number of overlapping variables. Table 4.2 shows that 3 overlapping variables provide the best results. A setting of 3 overlapping variables provides at least a 1% improvement over the other runs. On average using 3 overlapping variables improved the solution time by 8.7%. Thus, the larger instances are implemented with 3 overlapping variables.

To test the impact of MVMCIs on more difficult problems, instances containing 100 variables with 10 and 15 constraints were solved. Additionally, instances with 125 variables

Constraints	Variables	Overlapping Variables	CPLEX Average (Seconds)	MVMCA Average (Seconds)	Improvement(%)
5	50	1	.64	.68	-7.2%
5	50	3	.64	.65	-2.02%
5	50	5	.64	.65	-2.02%
5	50	10	.64	.67	-4.8%
5	75	1	4.55	4.10	9.8%
5	75	3	4.55	4.50	.9%
5	75	5	4.55	4.33	4.7%
5	75	10	4.55	4.36	4.2%
10	50	1	7.70	6.49	15.7%
10	50	3	7.70	6.20	19.5%
10	50	5	7.70	6.25	18.8%
10	50	10	7.70	6.62	17.0%
10	75	1	69.53	65.58	5.6%
10	75	3	69.56	63.86	8.2%
10	75	5	69.56	65.35	6.0%
10	75	10	69.56	71.23	2.4%
Average			**20.61**	**19.46**	**5.5%**

Table 4.1: MVMCA Smaller Problem Runs

Overlapping Variables	CPLEX Average (Seconds)	MVMCA Average (Seconds)	Improvement(%)
1	20.61	19.21	6.74%
3	20.61	18.80	8.74%
5	20.61	19.14	7.07%
10	20.61	20.72	-.53%

Table 4.2: Overlapping Variables Averages

and 10 constraints are solved. The instances with 125 variables and 15 constraints required over a day to solve each instance. Since each class has 15 instances, a study on 125 variables with 15 constraints was not performed. In this study, the covers are selected in the same fashion as in the smaller instances and there are 3 overlapping variables.

The results of the larger problems are shown in Table 4.3. On average MVCMA reduced CPLEX's run time by 6.3%. While 6.3% may not seem like a large improvement, when dealing with average run times of over 6,000 seconds, the total improvement is over an hour of less run time.

Knapsack problems by nature are inconsistent and it should be noted that while performing this computational study some instances were run with 10 constraints and 101 variables. These runs actually showed the best results with an average of 21% improvement and the data are shown in Table 4.4. It is believed practitioners and researchers may receive more than a 6.3% improvement by implementing MVMCA.

Constraints	Variables	Overlapping Variables	CPLEX Average (Seconds)	MVMCA Average (Seconds)	Improvement(%)
10	100	3	1003.95	943.67	6.0%
15	100	3	6570.85	6165.75	6.1%
10	125	3	4309.15	4022.71	6.6%
Average			**3961.32**	**3710.71**	**6.3%**

Table 4.3: MVMCA Larger Problem Runs

The computational results show that MVMCA has the potential to reduce the time required to solve an MK instance. To implement MVMCA, it is recommended to generate covers based upon a combination of reduced costs and size of the knapsack coefficients. Additionally, one should have about 3 overlapping variables between the two covers. The next chapter reviews the main points of this thesis and discusses some areas of future research.

Instance	CPLEX(Seconds)	MVMCA(Seconds)	Improvement(%)
1	621.726	506.403	18.55%
2	76.517	76.501	.02%
3	392.884	348.161	11.38%
4	404.193	344.926	14.66%
5	1404.423	1274.823	9.23%
6	2826.997	2333.602	17.45%
7	1266.215	1153.874	8.87%
8	301.071	285.774	5.08%
9	338.001	310.051	8.27%
10	254.672	235.751	7.43%
11	28.093	29.628	-5.46%
12	1948.624	1140.575	41.47%
13	1780.683	1059.187	40.52%
14	104.533	94.329	9.76%
15	717.365	562.164	21.63%
Average	**831.066**	**650.383**	**21.74%**

Table 4.4: MVMCA vs. CPLEX on 10 Constraints, 101 Variables, 3 Overlap

Chapter 5

Conclusion

The purpose of this thesis was to explore a new way to merge cover inequalities. Building on the work of Hickman and Easton [24], there was opportunity to improve their work by merging on more than one variable as opposed to only one variable. Furthermore, in numerous instances Hickman's method would create inequalities that were weak and these inequalities could be strengthened. Thus, the motivation of this thesis was to remedy these two weaknesses.

This thesis provides the base for a new class of cutting planes obtained by merging two cover constraints on multiple variables with a merging coefficient α. This class of cutting planes cannot be obtained through any existing method without the consultation of an oracle. One of the biggest advantages of MVMCA is that the cutting planes it produces are obtained using a linear time algorithm, the provably best runtime for such instances. Additionally, these inequalities can be facet defining.

To show the value of MVMCA, a computational study implemented MVMCA with

CPLEX solver software. CPLEX with and without MVMCA was tested on various cases. When implementing MVMCA, one should rank the two covers on reduced costs and coefficient size. The covers should overlap by about three indices when run with similiar sized instances presented in this thesis. MVMCA can reduce the amount of time required for CPLEX to process and solve multiple knapsack problems. Some cases reduced the solution time by 40%. On the larger problems, MVMCI decreased the average run time by 6.3%, which is about an hour per group of instances. This shows that MVMCA can provide positive results over CPLEX alone and in the right circumstances could aid researchers and practitioners in solving more complex integer programs.

5.1 Future Research

The results show that MVMCA can be used to generate facet defining inequalities for the knapsack polytope and opens the doors to significant additional research. This research can largely be divided into computational and theoretical.

Many computational research questions related to MVMCI exist. In this research only one MVMCI is included and a computational study to determine the optimal number of MVMCIs may further reduce the runtime. The KP instances selected were random and do MVMCIs perform better on real-world instances? Since any binary constraint can be represented as a knapsack constraint through substitution, what is the effect of MVMCA on general IPs?

MVMCIs represent a new technique to create cutting planes so much of the future research

involves extending this methodology to create new classes of cutting planes. Theoretically, this knowledge provides some exciting directions for future research.

Can the host cover be merged with a donor cover from a different knapsack constraint? One would expect this type of merging to provide better computational results as the inequality takes into account information from more than a single constraint.

The results presented in this thesis are for MK. Can these concepts be extended to arbitrary IPs? Since covers are merely edges in a conflict hypergraph, one could examine hypergraphic structures to assist in answering this question.

Another possible application is to merge any two valid inequalities, not restricted to cover inequalities. Positive results can lead to more widely applicable inequalities and better computational results. Several possible cutting planes come to mind and merging disjunctive cuts, Gomory cuts or other common inequalities.

Bibliography

[1] Bakirli, B.,C. Gencer, and E. Aydogan, (2014). A combined approach for fuzzy multi-objective multiple knapsack problems for defence project selection. *The Journal of the Operational Research Society*, Vol. 65, No. 7, 1001-1016.

[2] Balas, E.,(1975). Facets of the knapsack polytope,*Mathematical Programming*, Vol. 8, No. 1, 146-164.

[3] Bektas, T., and S. Elmastas,(2007). Solving school bus routing problems through integer programming. *The Journal of the Operational Research Society*, Vol. 58, No. 12, 1599-1604.

[4] Bichler, M., J. Kalagnanam, K. Katirciglu and A. King,(2002). Applications of flexible pricing in business-to-business electronic commerce. *IBM Systems Journal*, Vol. 41, No. 2, 287-302.

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

[6] Brooks, J. (2012). The Court of Appeals of Virginia uses integer programming and cloud computing to schedule sessions.*Interfaces*, Vol. 42, No. 6, 544-553.

[7] Chang, P. and J. Lee,(2012). A fuzzy DEA and knapsack formulation integrated model for project selection.*Computers and Operations Research*, Vol. 39, No. 1, 112-125.

[8] Cho, C., D. Padberg, and M. Rao,(1983). On the uncapacitated plant location problem. II. Facets and lifting theorems.*Mathematics of Operations Research*, Vol. 8, No. 4, 590-612.

[9] Chvátal, V.,(1973). Edmonds polytopes and a hierarchy of combinatorial problems.*Discrete Mathematics*, Vol. 4, No. 4, 305-337.

[10] IBM ILOG CPLEX Optimizer, Version 12.5.1, see http://www-01.ibm.com/software/info/ilog/. (2013).

[11] Dawande, M., J. Kalagnanam, P. Keskinocak, R. Ravi, and F. Salman,(2000). Approximation algorithms for the multiple knapsack problem with assignment restrictions. *Journal of Combinatorial Optimization*, Vol. 4, No. 2, 171-186.

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

[13] Dizdar, D., A. Gershkov, and B. Moldovanu,(2011). Revenue maximization in the dynamic knapsack problem.*Theoretical Economics*, Vol. 6, No. 2, 157-184.

[14] Easton, K., G. Nemhauser, and M. Trick,(2003). Solving the traveling tournament problem: a combined integer and constraint programming approach.*PATAT'2002, Springer*

Lecture Notes in Computer Science, E. Burke and P. Causmaecker (eds), Vol. 2740, 63-77.

[15] Easton, T. and K. Hooker,(2008). Simultaneously lifting sets of binary variables into cover inequalities for knapsack polytopes.*Discrete Optimization*, *Special Issue: In Memory of George B. Dantzig*, Vol. 5, No. 2, 254-261.

[16] Foulds, L., H. Hamacher and J. Wilson,(1998). Integer Programming approaches to facilities layout models with forbidden areas.*Annals of Operations Research*, Vol. 81, 405-417.

[17] Gomory, R.,(1958). Outline of an algorithm for integer solutions to linear programs.*Bulletin of the American Mathematical Society*, Vol. 64, No. 5, 275-278.

[18] Gomory, R. and E. Johnson,(1972). Some continuous functions related to corner polyhedra 1.*Mathematical Programming*, Vol. 3, No. 1, 23-85.

[19] Gu, Z., G. Nemhauser, and M. Savelsbergh,(1998). Lifted cover inequalities for 0-1 integer programs: computation.*INFORMS Journal on Computing*, Vol. 10, No. 4, 427-437.

[20] Gu, Z., G. Nemhauser, and M. Savelsbergh,(1999). Lifted cover inequalities for 0-1 integer programs: complexity.*INFORMS Journal on Computing*, Vol. 11, No. 1, 117-123.

[21] Gu, Z., G. Nemhauser, and M. Savelsbergh,(2000). Sequence independent lifting in mixed integer programming.*Journal of Combinatorial Optimization*, Vol. 4, No. 1, 109-129.

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

[23] Hammer, P., E. Johnson, and U. Peled, Facets of regular 0-1 polytopes.*Mathematical Programming*, Vol. 8, No. 1, (1975), 179-206.

[24] Hickman, R. and T. Easton, "Merging valid inequalities over the multiple knapsack polyhedron," (In Press). *International Journal of Operations Research.*

[25] Hochbaum, D.,(1995. A non-linear knapsack problem.*Operations Research Letters*, Vol. 17, No. 3, 103-110.

[26] Hooker, J.,(2005). A search-infer-and-relax framework for integrating solution methods.*Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, *Conference Proceedings*, *CPAIOR 2005*, *Spinger Lecture Notes in Computer Science*, R. Barták and M. Milano (eds), Vol. 3524, 243-257.

[27] Hooker, K. and T. Easton,(2007). Using hyperstars to create facial defining inequalities of general binary integer programs. *The International Journal of Operations Research*, Vol. 4, No. 3, 138-145.

[28] Jacobson, S., J. Kobza, and A. Easterling,(2001). A detection theoretic approach to modeling aviation security problem using the knapsack problem.*IIE Transactions*, Vol. 33 No. 9, 747-759.

[29] Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming.*Combinatorica*, Vol. 4, No.4, 373-395.

[30] Kellerer, H. and V. Strusevich,(2010). Fully polynomial approximation schemes for a symmetric quadratic knapsack problem and its scheduling applications.*Algorithmica*, Vol. 57, No. 4, 769-795.

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

[32] Land, A., and A. Doig,(1960). An automatic method of solving discrete programming problems.*Econometrica*, Vol. 28, No. 3, 497-520.

[33] Li, Fulhai, X. Zhou, J. Ma, S. Wong,(2010). Multiple nuclei tracking using integer programming for quantitative cancer cell cycle analysis.*IEEE Transactions on Medical Imaging*, Vol. 29, No. 1, 96-105.

[34] OR Library webpage, see http://people.brunel.ac.uk/ mastjjb/jeb/info.html (2013).

[35] Rushmeier, R., and S. Kontogiorgis. Advances in the optimization of airline fleet assignment.*Transportation Science* Vol. 31, No. 2, (1997), 159.

[36] Szeto, K. and M. Lo, (2004). An application of adaptive genetic algorithm in financial knapsack problem.*Innovations in Applied Artificial Intelligence, Lecture Notes in Computer Science*, Springer: Berlin/Heidelberg, 1220-1228.

[37] Taylor III, B., A. Keown, and A. Greenwood.(1983), An integer goal programming model for determining military aircraft expenditures.*The Journal of the Operational Research Society*, Vol. 34, No. 5, 379-90.

[38] Tavana, M., K. Khalili-damghani, and A. Abtahi, (2013). A fuzzy multidimensional multiple-choice knapsack model for project portfolio selection using an evolutionary algorithm.*Annals of Operations Research*, Vol. 206, No. 1, 449-483.

[39] Vladislav, M., J. Lazic, T. Davidovic and N. Mladenovic, (2013). Routing of barge container ships by mixed-integer programming heuristics.*Applied Soft Computing*, Vol. 13, No. 8, 3515-3528.

[40] Wolsey, L., (1975). Faces for a linear inequality in 0-1 variables.*Mathematical Programming*, Vol. 8, No. 1, 165-178.

[41] Wolsey, L.,(1976). Facets and strong valid inequalities for integer programs.*Operations Research*, Vol. 24, No. 2, 367-372.

[42] Wolsey, L.,(1977). Valid inequalities and superadditivity of 0/1 integer programs.*Mathematics of Operations Research*, Vol. 2, No. 1, 66-77.