SYNCHRONIZED SIMULTANEOUS APPROXIMATE LIFTING FOR THE MULTIPLE KNAPSACK POLYTOPE

by

THOMAS BRADEN MORRISON

B.S., Kansas State University, 2012

A THESIS

Submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Industrial and Manufacturing Systems Engineering

College of Engineering

KANSAS STATE UNIVERSITY

Manhattan, Kansas

2012

Approved by:

Major Professor

Dr. Todd Easton

ABSTRACT

Integer programs (IPs) are mathematical models that can provide an optimal solution to a variety of different problems. They have the ability to maximize profitability and decrease wasteful spending, but IPs are NP-complete resulting in many IPs that cannot be solved in reasonable periods of time. Cutting planes or valid inequalities have been used to decrease the time required to solve IPs.

These valid inequalities are commonly created using a procedure called lifting. Lifting is a technique that strengthens existing valid inequalities without cutting off feasible solutions. Lifting inequalities can result in facet defining inequalities, the theoretically strongest valid inequalities. Because of these properties, lifting procedures are used in software to reduce the time required to solve an IP.

This thesis introduces a new algorithm for synchronized simultaneous approximate lifting for multiple knapsack problems. Synchronized Simultaneous Approximate Lifting (SSAL) requires $O(|E_1|S_{LP_{|E_1|+|E_2|,m}} + |E_1|^2)$ effort, where $|E_1|$ and $|E_2|$ are the sizes of sets used in the algorithm and S_{LP} is the time to solve a linear program. It approximately uplifts two sets simultaneously to creates multiple inequalities of a particular form. These new valid inequalities generated by SSAL can be facet defining.

A small computational study shows that SSAL is quick to execute, requiring fractions of a second. Additionally, applying SSAL inequalities to large knapsack problem enabled commercial software to solve faster and also eliminate off the initial linear relaxation point.

Contents

List of Figures

List of Tables

Dedication

My work is dedicated to my family and all the other people who have helped me achieve especially the teachers and professors who have truly taught me.

Acknowledgments

There are many who, through their support and efforts, aided in the success of this thesis. Most notably is Dr. Todd Easton whose impact on my academic career extends far beyond this thesis. Without his help, encouragement, and persistence, this thesis would be non-existent and my graduate career might have only been a path I thought about taking in college. Also I would like to thank the other graduate students whose research this has been an extension of. Without their work, my work could not exist. I would also like to thank Dr. Shing Chang and Dr. Craig Spencer for their service on my board.

Chapter 1

Introduction

Integer programming is a method for solving problems which can maximize revenue, reduce costs, and optimize systems and businesses. Integer programs (IPs) are mathematical models that can provide an optimal solution to a variety of different problems and take the form maximize $c^T x$ subject to $Ax \leq b$ and $x \in \mathbb{Z}^n_+$, where $A \in \mathbb{R}^{mxn}$, $c \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$. This thesis presents a new algorithm, Synchronized Simultaneous Approximate Lifting (SSAL), to generate cutting planes. SSAL works on the multiple knapsack problems (MKP), which is a common class of IPs.

Examples of the multiple knapsack problem are faced in everyday life when faced by yes/no choices. A simple example is deciding who to invite to a party. There might be problems between two friends which prevent inviting both. This type of problem also appears in industrial application. For example, which boxes should be sent on a particular truck. Perhaps two products cannot be shipped together for fear of contamination.

Multiple knapsack problems can be use to solve numerous optimization applications. One such popular use is in finances and investments [6, 27, 30]. It also has application in road and highway construction selection, which results in more efficient placement of roads reducing traffic congestion with minimal cost [33]. It even has less conventional uses, which includes cryptography [7]. This wide range of uses for multiple knapsack problems makes it a powerful tool for optimization.

The most common method to solve IPs is the branch and bound algorithm which uses the optimal solution from linear relaxations. A linear relaxation (LR) is a linear program (LP) which has the IP formulation. Since it is an LP instead of an IP, it doesn't have the integer requirement. When the LR contains fractional values branch and bound creates two nodes, also referred to as children. One child adds the constraint that a fractional variable is less than or equal to the floor of its value from the LR. The other child adds the constraint that the variable must be greater than or equal to its ceiling. By adding these constraints, the non integer space between some integer points is no longer valid in either of these problems. Running this process iteratively enumerates all integer points. Eventually branch and bound finds the optimal solution, but it can require exponential time. Say for example that instead of trying to pack a single truck, you are in charge of packing every truck in a company which moves thousands of products. These problems can become large enough that they take days or even weeks to solve. For this reason, much research has been done in creating new inequalities, or cutting planes, that reduce the solution times of these IPs.

A cutting plane is a valid inequality that when added to the problem eliminates some fraction of the LR's feasible area. A valid inequality is satisfied by every feasible IP solution. Applying iterations of cutting planes can force the optimal LR solution to become an integer solution, thus the IP is solved. Facet defining cutting planes are the theoretically strongest valid inequalities because they can fully define the space of the problem allowing the LR to provide the optimal integer solution.

One method to obtain a facet defining inequality is through lifting. Lifting uses the restricted polyhedron which forces some variables in the problem to specific values. Lifting alters the coefficients on the variables of a valid inequality to make it stronger. This thesis focuses on the development of inequalities through synchronized simultaneous approximate lifting. The lifting technique in this thesis also has the ability to create facet defining inequalities.

1.1 Research Motivation

Bolton [3] developed an exact synchronized simultaneous uplifting algorithm. This algorithm was limited to a single constraint, which means that problems with more constraints cannot achieve as strong of cuts as would be possible if all constraints are considered together. The goal of this research is to develop a synchronized simultaneous approximate lifting algorithm, which creates stronger inequalities for the multiple knapsack instance. Thus presenting a new lifting method with the objective of generating cutting planes to reduce the time to solve IPs.

1.2 Research Contributions

This thesis presents a new synchronized simultaneous approximate lifting (SSAL) algorithm for the knapsack polytope which is capable of solving problems with multiple constraints. The input to SSAL is a multiple knapsack problem and two sets of mutually exclusive indices. A table of feasible points based on the indices selected from the initial valid inequality is found using LP. These points are used to calculate the approximate synchronized simultaneous uplifting coefficients.

The primary contributions of this thesis lie in the creation of synchronized simultaneous approximate lifted variables for the multiple knapsack polyhedron. Results from a small computational study show applying SSAL enabled CPLEX 10.0 [12], a commercial integer programming software, to solve large sample problems 6% faster. In addition, the initial linear relaxation solution decreased by between 2% in large problems and 6% in smaller problems.

1.3 Outline

Chapter 2 contains an overview of integer programming and polyhedral theory providing the background information necessary to understand the research presented in this thesis. Topics covered include: cutting planes and facet defining inequalities, the knapsack problem, and various forms of lifting including SSL. Formal definitions along with detailed examples aid in the understanding of this thesis.

Chapter 3 presents SSAL. First, notation is defined followed by an overview of the algorithm. Next, the pseudocode provides the details to execute SSAL. Proof of correctness and proof of advancement over previous algorithms and its ability to make facet defining inequalities are presented. Finally, an example demonstrates SSAL produces multiple facet defining inequalities.

The results from the computational study are found in Chapter 4. The class of problems generated is described along with data to support the effectiveness of SSAL. Data presented includes changes in the initial linear relaxation solution and the time required to solve to optimality.

Finally, Chapter 5 provides a conclusion of SSAL and its computational results. This chapter also contains ideas and extensions discovered during the development of SSAL that can be pursued as future research.

Chapter 2

Background Information

This chapter introduces the necessary operations research and mathmatical background necessary to understand this thesis. Concepts discussed include integer programming, the definition and use of cutting planes, and the concept of lifting and lifting techniques. Through the discussion in this chapter, a basic understanding of the concepts should lead to an appreciation of the work presented in this thesis.

2.1 Integer Programming

An integer program (IP) has a linear objective equation that can either be maximized or minimized to meet a specific goal. It is also subject to a finite set of linear constraints, and the variables are required to be integer. Therefore, IPs follow the form:

Maximize

$$z^{IP} = c^T x$$

subject to

$$Ax \le b$$

$$x \in \mathbb{Z}_+^n$$

where $c \in \Re^n$, $A \in \Re^{m \times n}$ and $b \in \Re^m$.

The feasible space for an IP is defined as $P = \{x \in \mathbb{Z}_+^n : Ax \le b\}$. The solution space, P contains a set of countable points. Denote N as the set of indices of an IP, $N = \{1, ..., n\}$.

The most popular method for solving IPs is the use of the branch and bound algorithm. This algorithm first solves the problem as though it were a linear program, which typically yields a solution with non-integer variables. This related problem is called the linear relaxation (LR) and represents the optimal solution to the problem if the variables need not be integer. The linear relaxation that corresponds to the given problem is referred to as IP^{LR} with the format: Maximize $z^{LR} = c^T x$ subject to $Ax \le b, x \in \Re_+^n$. Let P^{LR} be defined as the LR's feasible region, $P^{LR} = \{x \in \Re_+^n : Ax \le b\}$.

If the solution to IP^{LR} is fractional, branch and bound splits this LR into two subproblems, which would then continue to be split into subproblems that collectively still have every feasible integer point. If the solution to one of these subproblems is infeasible, it is instantly fathomed, which means that it no longer branches and is eliminated from the pool of problems. But if the solution is feasible, it continues to split. Though, not every point needs to be examined and its objective value found. Every time the problem is split, there is a possibility that the new child problem has an integer solution. In this case, the node is no longer split. The largest of these objective values is saved and used for comparison. If any non-integer solution is found to be lower than this value, it can be fathomed. Once every subproblem has been eliminated, the highest integer solution is known to be the optimal answer and the solution to the original IP.

There are many different classes of IP's which can be used in different situations. One class of IP is the knapsack problem (KP). This class of problems is called a knapsack problem because of the problem faced by a traveler preparing for a trip. They are faced with a collection of items that could be taken with them, but are limited by the amount they can carry in their "knapsack". There are n items they could take, each with their own benefit c_j, and weight a_j. The traveler also has a maximum amount they can carry b.

A formulation of KP has its variables $x_j = 1$ if the item is taken, and $x_j = 0$ if it is not. The formulation of a simple KP is

Maximize

$\sum_{j=1}^{n} c_j x_j$

subject to

$\sum_{j=1}^{n} a_j x_j \leq b$

$x_j \in \mathbb{B}$ for all $j = 1, 2, ..., n$

where $a_j \geq 0 \; \forall \; j = 1, ..., n$. Let P_{KP} represent the set of feasible solutions, $P^{KP} = \{x \in \mathbb{B}^n : \sum_{i=1}^{n} a_i x_i \leq b\}$.

A KP is such a simple formulation which only includes one constraint, a common extension is to have multiple knapsack constraints. Instead of only having the weight a hiker can carry, assume that there is only so much room inside one's knapsack or only have so much money to buy supplies. This introduces more constraints which would further limit the number of items that could be taken. This is referred to as a multiple knapsack problem, or MKP. Formally, an MKP is Maximize $\sum_{j=1}^{n} c_j x_j$ subject to $\sum_{j=1}^{n} a_{i,j} x_j \leq b_i$ for all $i = 1, ..., m$, $x_j \in \mathbb{B}$ for all $j = 1, 2, ..., n$ where $a_{i,j}$ and $b_i \geq 0$ for all $j = 1, ..., n$. Also, the set of feasible points, $\{x_j \in \mathbb{B}, Ax \leq b\}$ is denoted as P^{MKP}.

Example 1:

A hiker is considering taking 12 items on his trip. Each item has a benefit, weight, size and cost given in Table 2.1. Below is a formulation of this problem.

Item#	1	2	3	4	5	6	7	8	9	10	11	12	capacity
Benefit	20	20	18	16	15	14	13	12	12	12	11	10	
Weight	20	20	18	16	15	14	13	12	12	12	11	10	115
Size	2	12	29	17	13	4	17	18	20	16	8	11	110
Cost	5	16	16	5	7	8	13	9	15	10	17	19	95

Table 2.1: Benefit, weight, size, and cost of items that may be taken in the knapsack

Maximize

$$20x_1 + 7x_2 + 46x_3 + 79x_4 + 9x_5 + 84x_6 + 42x_7 + 34x_8 + 91x_9 + 107x_{10} + 117x_{11} + 3x_{12}$$

Subject to

$$20x_1+20x_2+18x_3+16x_4+15x_5+14x_6+13x_7+12x_8+12x_9+12x_{10}+11x_{11}+10x_{12} \leq 115$$

$$2x_1+12x_2+29x_3+17x_4+13x_5+4x_6+17x_7+18x_8+20x_9+16x_{10}+8x_{11}+11x_{12} \leq 105$$

$$5x_1+16x_2+16x_3+5x_4+7x_5+8x_6+13x_7+9x_8+15x_9+10x_{10}+17x_{11}+19x_{12} \leq 95$$

$$x_j \in \{0,1\}, j \in \{1,...,12\}.$$

The optimum solution to this problem is to select items $\{1,4,6,7,8,9,10,11\}$ leading to an objective benefit of 574 units. The individual would carry 110 and with a volume of 102 and a cost of 82 units.

2.2 Polyhedral Theory

Polyhedral theory is critical to integer programming research. Numerous researchers have applied concepts from polyhedral theory to decrease the time required to solve an integer program [2, 9, 13]. This section describes some of the basic polyhedral concepts.

Polyhedra are convex. A set is convex if and only if for any two points in the set, every point on the line between those two points is also in the set. The convex hull of a set S, S^{ch}, is the intersection of all convex sets that contain S. $S \subseteq \mathbb{R}^n$

Each linear inequality produces a half space by including all points on one side and eliminating the other from the solution space. Thus $\{x \in \Re^n : \sum_{j=1}^{n} \alpha_j x_j \leq \beta\}$ is a half space. The intersection of finitely many half spaces is a polyhedron.

Clearly, the feasible region of an IP^{LR} is a polyhedron and is always convex, because the feasible region is confined by linear constraints. Since IPs have the condition of always needing to be integer and the space between any two integer solutions is not integer, P is not convex, unless there is only zero or one feasible solutions. Of particular interest to integer programming research is the relationship between P^{ch} and P^{LR}. This particular research focuses on $P^{ch}_{MKP} = conv(P_{MKP})$.

While P^{ch} and P^{LR} are similar, there is a key distinction between them. While the P^{LR} consists of all the space inside the constraints, the convex hull is defined by the extreme integer points and the inequalities between them. If the constraints were ever tight enough to make P^{ch} and P^{LR} the same, then the initial solution to the IP^{LR} would produce an integer solution and therefore eliminate the need for branch and bound which could otherwise require exponentially many iterations.

For this reason, there is a large dedication of time in polyhedral theory spent to creating new inequalities for IP's. These new inequalities, or cutting planes, are valid inequalities that are used to restrict the area in P^{LR} without removing any points from P. The inequality $\sum_{j=1}^{n} \alpha_j x_j \leq \beta$ is a valid inequality of P^{ch} if and only if every $x \in P$ satisfies $\sum_{j=1}^{n} \alpha_j x_j \leq \beta$.

The dimension of a polyhedron is a significant characteristic critical to research in integer programming. The dimension of a polyhedron is the number of linearly independent vectors that can be used to define a space. A set of vectors, $v^1, ..., v^q \in \Re^n$, is independent if any combination of vectors cannot recreate any other vector. This concept is expressed mathematically as $\sum_{i=1}^{q} \lambda_i v_i = 0$ if and only if there exists a unique solution for $\lambda_i = 0$ for all $i = 1, ..., q$. Because P does not contain any feasible vectors in its solution space, affine independence is used instead.

Affine independence uses a set of points to determine the dimension of a space. A set of points $x_1, x_2, x_3, ..., x_r \in \Re_+^n$ is affinely independent if and only if, $\sum_{j=1}^{r} \lambda_j x_j = 0$ and $\sum_{j=1}^{r} \lambda_j = 0$ is uniquely solved by $\lambda_j = 0$ for all $j = 1, ..., r$. Since vectors can be made from two points, there has to be one additional point to act as the origin to determine the dimension of a set of points. For this reason, the dimension of a space equals the maximum number of affinely independent points minus one.

Every valid inequality induces a face on a polyhedron. The valid inequality, $\sum_{j=1}^{n} \alpha_j x_j \leq \beta$, defines a face $F \subseteq P^{ch}$ of the form $F = \{x \in P^{ch} : \sum_{j=1}^{n} \alpha_j x_j = \beta\}$. Any polyhedron can be defined as a set of faces. While any face can restrict the space of a polyhedron, some are redundant to others either because they are less restrictive or because they have a smaller dimension. The minimum set of faces that define the convex hull consists of only facets. Facets are the faces that are one dimension less than the polyhedron, and make the strongest inequalities. The following simple 2-dimensional example shows how certain inequalities are more effective than others.

Example 2

Consider the following integer program:

Maximize

$$5x_1 + 6x_2$$

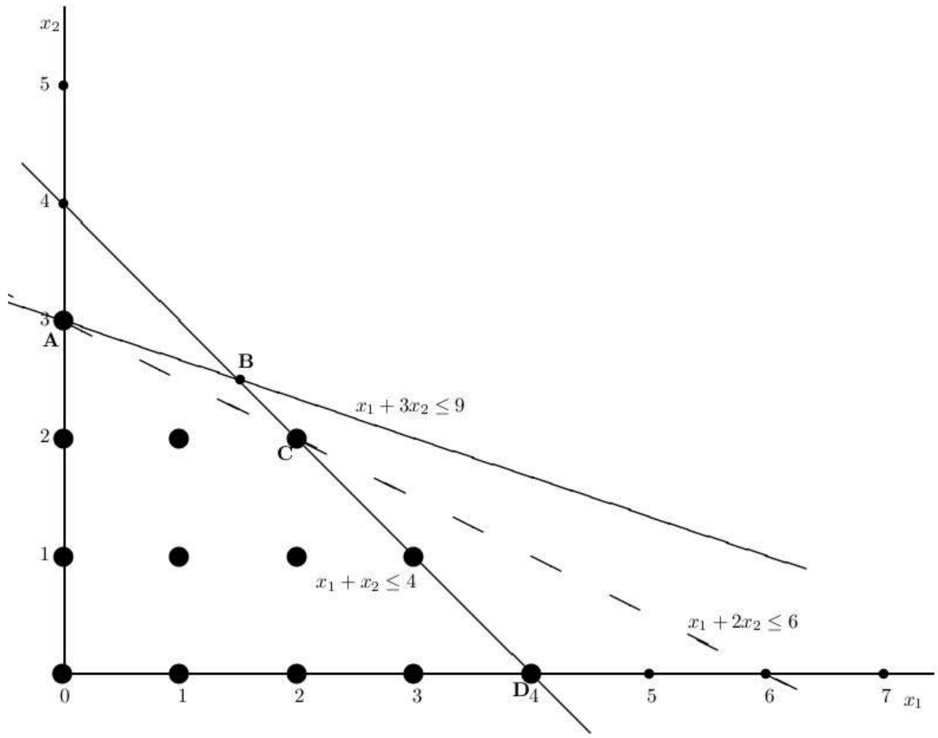

Figure 1.1: Cutting Plane Method in Example 2.1

Figure 2.1: 2-Dimensional IP example

Subject to

$x_1 + 3x_2 \leq 9$

$x_1 + x_2 \leq 4$

$x_1,\, x_2 \ \in \mathbb{Z}_+.$

Figure 2.1 provides a graphical view of this IP. The first constraint $x_1+x_2 \leq 4$ passes through the points $(0, 4)$, B, C, D, $(3, 1)$, and $(4, 0)$. The second constraint $x_1+3x_2 \leq 9$ passes through points A and B. Clearly P^{LR} is defined by these two constraints and the x_1 and x_2 axes. The solution to the LR is the point B which is $(1.5, 2.5)$ giving a z value of 22.5. The large circles represent P, the feasible integer points.

Clearly, P^{LR} is not P^{ch} because there is a non-integer vertex B. Both original constraints induce proper faces of P^{ch}, at least one point in P satisfies each inequality at equality. Since the dimension of P^{ch} is two, only 1 dimensional faces are facets. Using affine independence, we can examine both these constraints for their dimension. The first constraint, $x_1 + x_2 \leq 4$, passes through several feasible points, (4,0),(3,1) and (2,2), but since they are all in a line, only 2 of them are affinely independent. Thus, this constraint is facet defining. However, the second constraint, $x_1 + 3x_2 \leq 9$, is only met at equality by the integer point A. Thus, its dimension is 0 and it is not facet defining.

In this case, the solution to the LP is fractional or results in a non-integer solution. To fully define the convex hull, a new facet defining inequality can be added to cut off the rest of the linear relaxation space. This new constraint $x_1 + 2x_2 \leq 6$ is represented as the dotted line in Figure 2.1. With this final constraint added, the solution space, P^{LR} becomes the same as P^{ch} and the solution to IP^{LR} would be the optimal integer solution.

There are many different strategies for creating new constraints, and most of them are very specific to the type of problem that one is solving. Sometimes the constraints created are far from the solution space or are cutting off very little linear relaxation space. This gave rise to a process called lifting which is used to create stronger inequalities. Lifting is the focus of the next section and is the basis of this thesis.

2.3 Lifting

The purpose of lifting is to use weak valid inequalities and transform them into stronger valid inequalities. This is done by introducing new variables into the inequality or changing existing coefficients, which allows the inequality's dimension to increase and may enable it to become facet defining.

Lifting was originally developed by Gomory [16], and has expanded to many different classes of lifting. Lifting begins with a restricted polyhedron. Given set $E = \{e_1, e_2, ..., e_{|E|}\} \subseteq N$ and $K = (k_1, ..., k_{|N \setminus E|})$, then the restricted set of feasible points is $P_{E,K} = \{x \in \mathbf{Z}^n : Ax \leq b, x_{e_{|E|+1}} = k_1,\ e_2 = k_{|E|+2}, ..., x_{e_{|N|}} = k_{|N \setminus E|}\}$. Lifting requires a valid inequality $\sum_{i \in N} \alpha_i x_i \leq \beta$ over $P^{ch}_{E,K}$ and creates a valid inequality of the form $\sum_{i \in N \setminus E} \alpha'_i x_i + \sum_{i \in E} \alpha'_i x_i \leq \beta'$ for P^{ch}.

There are four classifications of different type of lifting techniques. These techniques are classified based upon the size of E, values of α' and β', values of K and also how many different inequalities are obtained. The four classes are: sequential or simultaneous lifting, exact or approximate lifting, up, down or middle lifting, and single or synchronized lifting.

So a specific lifting technique might be classified as approximate, synchronized, simultaneous, up lifting. To the best of my knowledge, this thesis is the first research done in this area.

2.3.1 Up, Down, and Middle Lifting

When lifting was originally developed, there was only up lifting, and it remains the most commonly studied variety of lifting. This version of lifting assumes the variables that are about to be lifted into the inequality have an initial coefficient, or K, equal to 0. Any up lifting techniques would then determine how high the coefficient can be increased to make the new inequality still valid for all possible solutions in the polyhedron.

Down lifting is different in that it starts with $K = u$ where u is a predetermined upper bound which causes the inequality being lifted to be overly restrictive. These coefficients are then systematically decreased until they are valid for all solutions which were feasible in the original formulation. This is much less commonly studied than up lifting. But even less studied is middle lifting. In middle lifting, the coefficients are neither at an upper bound or lower bound to start and can be both increased and/or decreased.

2.3.2 Exact vs. Approximate Lifting

Exact lifting seeks to increase the α' coefficients and/or decrease β as far as possible while still remaining valid. Any further manipulation of these coefficients would result in an invalid inequality because exact lifting techniques put their values exactly to the limit. This results in the most restrictive inequalities possible, but require intense calculation. This can result in a process which is actually harder computationally to solve than then original problem as shown in Gutierrez's work [20]. While exact lifting was the original form of lifting, highly complex problems have become commonplace in IP formulation, and most exact lifting techniques are too complex to be implemented which gave rise to approximate lifting.

Approximate lifting is a branch of lifting that does not result in an optimally restrictive inequality. But through giving up this exactness, also allows huge reductions in runtime to determine α' and β. This allows researchers to create lifting techniques that can quickly create coefficients that maintain a valid inequality, but it still has room to be improved more. Even if they are not exact, they still help to eliminate linear relaxation space which helps increase the solution time of IP's.

2.3.3 Single vs. Synchronized Lifting

Further types of lifting are single and synchronized lifting. Nearly all current lifting techniques are single lifting which generate exactly one inequality when applied. Synchronized lifting is a technique, originally used in Jennifer Bolton's thesis and refers to an algorithm that is capable of creating numerous inequalities as a single instance.

2.3.4 Sequential vs. Simultaneous Lifting

Another classification of lifting involves the number of variables that are added to the equation at a time. There are two categories and they describe the size of E. In sequential lifting, $|E| = 1$, meaning that only a single variable is being lifted. Simultaneous lifting lifts multiple variables at a time, meaning $|E| \geq 2$. Unlike the other categories of lifting, where one type is studied substantially more than the other, both sequential and simultaneous lifting are popularly studied.

Sequential lifting seeks to modify the coefficients of variables individually and in succession. The order in which the variables are lifted has an effect on the inequalities that are produced. The single sequential up lifting algorithm assumes that $\Sigma_{j=2}^{n}\alpha_j x_j \leq \beta$ is valid for $P_{\{1\}}^{ch}$, and seeks to create a valid inequality $\alpha_1 x_1 + \Sigma_{j=2}^{n}\alpha_j x_j \leq \beta$ for P^{ch}. Several individuals have performed research on sequential up lifting [4, 5, 31]

Simultaneously up lifting the variables of E results in inequalities of the form $\alpha \sum_{i\in E} w_i x_i + \sum_{i\in N\setminus E} \alpha_i x_i \leq \beta$, where $w_i \in R$ is a weight, as described by Gutierrez [20]. The goal is to seek the maximum α value for which this inequality is valid. Gutierrez also provided theory to show that the exact lifting coefficient can be obtained by solving a single integer program.

2.3.5 Prior Lifting Research

With an understanding of the distinct classes of lifting, prior research can now be classified into these categories of lifting. Thus, there are 24 different types of lifting. This section describes much of the prior work relating to lifting and categorizes them into these categories.

By far the most popular is single sequential up lifting. Some exact algorithms can be found in [5, 20, 21, 24, 25, 31]. In the node packing polyhedron, [10, 31] provide some results.

Sequence independent lifting [1, 17, 29] is considered a single approximate sequential uplifting method with Balas [4] also providing research on approximate sequential uplifting, but his method generates numerous inequalities and thus is considered as a synchronized method.

Hooker and Easton [14] developed a linear time algorithm to simultaneously lift variables into cover inequalities for P_{KP}^{ch}. Gutierrez's method can also perform single exact simultaneous up lifting. Since then, [18, 19] expanded on this theory by creating pseudopolynomial or polynomial time algorithm that allows multiple simultaneously lifted sets to be sequentially lifted into a valid inequality for P_{KP}^{ch}. [26, 11] provide some exact simultaneous lifting results on the node packing polyhedron.

Zemel [32] developed the first exact method to simultaneously lift multiple variables in 1978, but this method required the use of exponentially many integer programs and can be applied only to cover inequalities from the binary Knapsack Problems. Zemel's method generates many inequalities by finding the extreme point of the polar, but is too computationally intensive to be efficiently implemented. In actuality Zemel's method generates numerous inequalities that are all simultaneously lifted. Thus, his method should have been classified as a synchronized simultaneous lifting algorithm.

Although sychronized lifting was originally created by Zemel but was later given the name of synchronized lifting. Bolton generated a polynomial time exact synchronized simultaneous uplifting technique [3]. Later both Beyer and Harris extended upon these results [8, 22]. This thesis is also an extension of their research. Bolton's synchronized simultaneous lifting (SSL) which is critical to this thesis is briefly described next.

2.3.6 Exact Synchronized Simultaneous Up Lifting

The original SSL created by Jennifer Bolton requires a knapsack problem with a single constraint and two mutually exclusive lifting sets E_1 and E_2. SSL outputs valid inequalities of the form $\alpha_{E_1} \sum_{i \in E_1} x_i + \alpha_{E_2} \sum_{i \in E_2} x_i \leq \beta$. Her algorithm requires $O(n^2)$ effort.

To begin SSL, the feasible combinations of the two sets are listed as ordered pairs. This is achieved by finding the maximum number of variables from E_1 that can be included together and have a feasible solution. This number is then decremented by 1, and as many variables in E_2 are introduced. This continues until there are no more variables included from E_1.

Next, beginning at the first extreme point on the axis, the slope of the lines from the first point to all other points is found. The line that does not eliminate any points is the most extreme. The slope of the line is computed through finding α values for each set. The value of α_{E_1} is found by taking $(q - q^*)$, where p^* and q^* are the quantities from the first and second set, respectively, feasible at the first point, and p and q are the quantities from the first and second set, respectively, feasible at the second point. The coefficient α_{E_2} is found by taking $(p^* - p)$. Finally, β is equal to $(p^* q - q^* p)$.

The ratio of $\alpha_{E_2}/\alpha_{E_1}$ gives the slope of the line between the two points. By selecting the lowest ratio (or highest, depending on which axis is used as the starting point), the next extreme point can be found. Should a tie occur in the ratio of the α values, the point farthest down the list is selected. This corresponds to two or more points on the same line with the steepest slope. This extreme point is now considered (p^*, q^*), and the slope of the lines to all subsequent points is found. This process is repeated until the final extreme point candidate located on the other axis is selected as an extreme point.

To demonstrate SSL, consider the first constraint from the MKP Example 1 with an altered b to better illustrate the algorithm:

$$20x_1+20x_2+18x_3+16x_4+15x_5+14x_6+13x_7+12x_8+12x_9+12x_{10}+11x_{11}+10x_{12} \leq 90$$

For this example, arbitrarily set $E_1 = \{1, 2, 3, 4, 5\}$ and $E_2 = \{6, 7, 8, 9, 10, 11, 12\}$. As is seen above, the a coefficients have been sorted for each of the sets. All the feasible combinations of sets are found by starting the count for E_1 at its maximum possible of 5 which would allow no variables from E_2. Since $(5, 0)$ is feasible, it attempts $(5, 1)$, which is infeasible. The algorithm then attempts $(4, 1)$ which is feasible. The point $(4, 2)$ is also feasible and then $(4, 3)$ is infeasible. This procedure continues until it generates the following set of points in Table 2.2.

These are the potential candidates to be extreme points. These points can be reduced to the obvious set of candidate extreme points as shown in Table 2.3. The algorithm begins with the first point and determine the slopes to every other point. The results are in the table below.

Next, the lowest slope is selected which is $\frac{1}{2}$ produced when $E_1 = 4$ and $E_2 = 2$. This can more clearly be seen in Figure 2.2 with the bold line having the minimum slope.

$count$	$feas\ E_1[count]$	$feas\ E_2[count]$	Feasible
0	5	0	y
0	5	1	n
1	4	1	y
1	4	2	y
1	4	3	n
2	3	3	y
1	3	4	n
1	2	4	y
1	2	5	y
1	2	6	n
4	1	6	y
1	1	7	n
6	0	7	y

Table 2.2: Feasible point data

$count$	$feas\ E_1[count]$	$feas\ E_2[count]$
0	5	0
1	4	2
2	3	3
3	2	5
4	1	6
6	0	7

Table 2.3: List of candidate extreme points

It is clearer to see in the graph that only the $\frac{1}{2}$ slope line would result in an inequality that doesn't cut off any feasible points. This line is represented by the inequality $2\sum_{i\in E_1} x_i + \sum_{i\in E_2} x_i \leq 10$. This second point is used as a new start point, and the list is once again reviewed for the lowest slope. This is repeated until the last point is chosen. When completed, the algorithm results in two more inequalities, $3\sum_{i\in E_1} x_i + 2\sum_{i\in E_2} x_i \leq 16$ and $\sum_{i\in E_1} x_i + \sum_{i\in E_2} x_i \leq 7$ shown below in the graph.

Bolton also provided conditions for when these inequalities are facet defining. In this

E_1	E_2	α_{E_1}	α_{E_2}	$\frac{\alpha_{E_2}}{\alpha_{E_1}}$
4	**2**	**2**	**1**	$\mathbf{\frac{1}{2}}$
3	3	3	2	$\frac{2}{3}$
2	5	5	3	$\frac{3}{5}$
1	6	6	4	$\frac{2}{3}$
0	7	7	5	$\frac{5}{7}$

Table 2.4: Values for the first SSL inequality

particular case, none of these inequalities meet her condition and are not facet defining inequalities.

This algorithm provides good inequalities, but it relies on the original problem only having a single constraint. It doesn't provide as strong of results when moving into problems with multiple constraints. For this reason, the next chapter of this thesis is about a new algorithm which can create good valid inequalities for multiple knapsack problems.

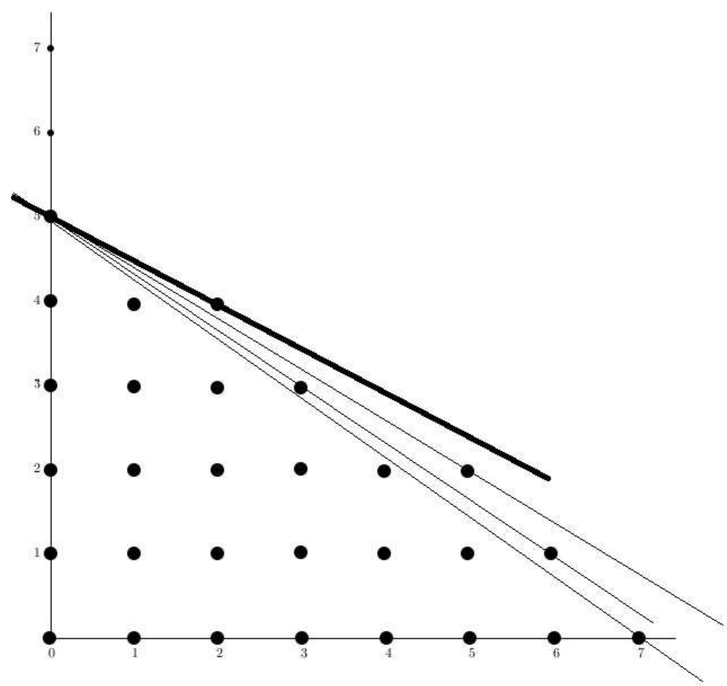

Figure 2.2: Example 2 SSL first constraint

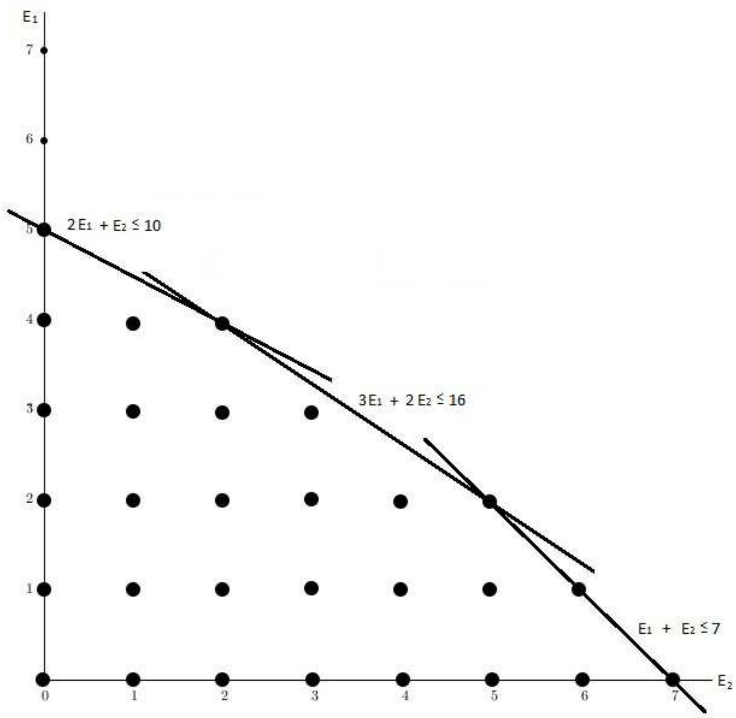

Figure 2.3: Example 2 SSL complete $E_1 - E_2$ graph

Chapter 3

SSAL Algorithm

This chapter formally introduces the Synchronized Simultaneous Approximate Lifting (SSAL) algorithm for MKP's. SSAL uplifts two sets into an arbitrary inequality simultaneously and generates multiple inequalities of the same form. Discussed in this chapter is an overview of SSAL introducing the notation. This is followed by the pseudocode for SSAL and the theoretical argument of correctness along with the ability to make facet defining inequalities. Finally, an example illustrates SSAL and shows that it creates new inequalities without the need for solving IP's which could possibly take longer than the original problem to solve.

One major weakness of Bolton's is that her algorithm is restricted to a single knapsack. The mechanics of extending Bolton's algorithm to multiple knapsack soley hangs on the identification of feasible points. The remainder of the algorithm is the same.

At the outset of this research, a dynamic program was built to identify the set of feasible points. This algorithm required far to much effort to generate these feasible points, and any computational benefit from the cuts was wasted in the generation of the set of feasible points. It was clearly necessary to find a faster or approximate method to find a set of "candidate extreme points".

Synchronized Simultaneous Approximate Up Lifting (SSAL) combines SSL with an approximate set of point that may or may not be feasible. Thus, the inequalities are not as strong as they may be because the algorithm may state that a point is feasible, when in actuality it is not.

Linear programs are solved to generate these approximate extreme points. Briefly, an LP is set up with the original IP constraints and an additional constraint $\sum_{i \in E_1} x_i = e_1$ where e_1 ranges from 0 to $|E_1|$. The objective function z_{e_1} = maximizes $\sum_{i \in E_2} x_i$. The candidate feasible point becomes $(e_1, \lfloor z_{e_1} \rfloor)$. These points are then fed into Bolton's algorithm and the inequalities are generated.

Formally, this process breaks into two subroutines. The first identifies the feasible points and is called *Find Points* . The inputs to SSAL are an MKP instance with n variables and m constraints and two mutually exclusive nonempty sets $E_1, \ E_2 \subset N$. The psuedocode is as follows:

Find Points Subroutine

Initialization:

Set $e_1 := |E_1|$.

Set $numpoints := 0$.

Main Step:

while $e_1 \geq 0$

Solve the following linear program:

$z_{e_1} = \text{Max} \sum_{i \in E_2} x_i$

subject to $Ax \leq b$

$\sum_{i \in E_1} x_i = e_1$

$0 \leq x \leq 1$.

if the LP is feasible, then

$feaspoints_{e_1}[numpoints] := e_1$.

$feaspoints_{e_2}[numpoints] := \lfloor z_{e_1} \rfloor$.

$numpoints := numpoints + 1$.

Set $e_1 := e_1 - 1$.

Output:

Report $feaspoints_{e_1}, feaspoints_{e_2}$ and $numpoints$.

Once the number of feasible points and the matrices with their values have been found, they can be passed into the second subroutine. This next subroutine identifies the inequalities created from these candidate extreme points.

Generate Valid Inequalities Subroutine

Initialization:

Set $loc := 0$.

Set $numconst := 0$.

Set $sumin := 0$.

Main Step:

Horizontal inequality

Set $validineq_{\alpha_1}[numconst] := 1$.

Set $validineq_{\alpha_2}[numconst] := 0$.

Set $validineq_{\beta}[numconst] := feaspoints_{e_1}[0]$.

Set $numconst := numconst + 1$.

Angled inequality

while $loc < numpoints$

Set $e_1^{start} := feaspoints_{e_1}[loc]$.

Set $e_2^{start} := feaspoints_{e_2}[loc]$.

Set $\alpha := M$ where M is arbitrarily high.

Set $k := loc + 1$.

while $k \leq numpoints$

Set $e_1^{end} := feaspoints_{e_1}[k]$.

Set $e_2^{end} := feaspoints_{e_2}[k]$.

Set $\alpha_{new} := (e_1^{start} - e_1^{end})/(e_2^{end} - e_2^{start})$.

if $\alpha_{new} \leq \alpha$, then

Set $\alpha := \alpha_{new}$.

Set $loc := k$.

Set $slope_1 := (e_1^{start} - e_1^{end})$.

Set $slope_2 := (e_2^{end} - e_2^{start})$.

Set $validineq_{\alpha_1}[numconst] := slope_2$.

Set $validineq_{\alpha_2}[numconst] := slope_1$.

Set $validineq_{\beta}[numconst] := e_1^{start} * slope_2 + e_2^{start} * slope_1$.

Set $numconst := numconst + 1$.

Vertical inequality

Set $validineq_{\alpha_1}[numconst] := 0$.

Set $validineq_{\alpha_2}[numconst] := 1$.

Set $validineq_{\beta}[numconst] := feaspoints_{e_2}[numpoints]$.

Set $numconst := numconst + 1$.

Output

Report $validineq_{\alpha_1}$, $validineq_{\alpha_2}$, $validineq_{\beta}$ and $numconst$ as the valid inequalities of the PMK instance.

3.1 SSAL Theoretical Results

Now that the procedure of SSAL has been examined, we must confirm that the inequalities created by SSAL are indeed valid. The following proof shows that each inequality returned by SSAL is valid.

Theorem 3.1.1 *The inequality $\sum_{i \in E_1} \alpha_{E_1} x_i + \sum_{i \in E_2} \alpha_{E_2} x_i \leq \beta$ returned from SSAL for an MKP instance is valid for P_{MK}^{ch}.*

Proof: Given an MKP instance, assume that SSAL returns an inequality of the form $\sum_{i \in E_1} \alpha^{E_1} x_i + \sum_{i \in E_2} \alpha^{E_2} x_i \leq \beta$. Let x' be any point in P_{MK} and let $\sum_{i \in E_1} x'_i = p$ and $\sum_{i \in E_2} x'_i = q$.

One step in SSAL solved the LP $max \sum_{i \in E_2} x_i$ subject to $Ax \leq b, \sum_{i \in E_1} x_i = p$, $x \in \mathbf{R}^n_+$. The solution to this LP has a value of at least q. Therefore, SSAL stores the point (p, r) for some $r \geq q$ and $r \in \mathbf{Z}$ within $feaspoints$.

The algorithm assures that none of the $feaspoints$ violate any of the valid inequalities. This is shown as follows. If loc has a value less than the point (p, r), then eventually (p, r) is tested in the loop and thus an $\alpha_{new} := (e_1^{start} - p)/(r - e_2^{start})$. If $\alpha_{new} > \alpha$, then $\alpha^{E_1} p + \alpha^{E_2} r \leq \beta$ and the result inequality is valid. Now if $\alpha_{new} \leq \alpha$ and no other updates occur to α, then $\alpha^{E_1} p + \alpha^{E_2} r = \beta$.

Finally if $\alpha_{new} \leq \alpha$ and another updates occur to α, then the slope of the line has been adjusted. In such a situation, the inequality generated during this iteration has $\alpha^{E_1}p + \alpha^{E_2}r < \beta$.

It is evident that $\frac{validineq_{\alpha_1}[j]}{validineq_{\alpha_2}[j]} < \frac{validineq_{\alpha_1}[k]}{validineq_{\alpha_2}[k]}$ for all $k \geq j$. Let loc be such that $feaspoints_{e_1}[loc] \leq p - 1$. Due to these slopes having this property and the fact that the polyhedron is convex, it must be that $\alpha^{E_1}p + \alpha^{E_2}r < \beta$.

Since $\alpha^{E_1}p + \alpha^{E_2}q \leq \alpha^{E_1}p + \alpha^{E_2}r$ and the point (p, r) satisfies each inequality, x' satisfies each generated inequality. Thus, $\sum_{i \in E_1} \alpha^{E_1} x_i + \sum_{i \in E_2} \alpha^{E_2} x_i \leq \beta$ is a valid inequality of P_{MK}^{ch}.

□

Now that the inequalities provided by SSAL are shown to be valid, it is natural to examine how much effort the algorithm takes to create them. The following result shows that SSAL is a polynomial time algorithm.

Theorem 3.1.2 *The SSAL algorithm requires* $O(|E_1|S_{LP_{|E_1|+|E_2|,m}} + |E_1|^2)$ *where* $S_{LP_{n,m}}$ *is the time required to solve an linear program with* m *constraints and* n *variables.*

Proof: The $findpointssubroutine$ has an initialization that requires $O(1)$. The Main Step solves $|E_1|$ linear programs and stores the desired numbers in each iteration. Observe that the linear programs are identical except for the right hand side of one constraint. Thus, swihching between linear programs requires $O(1)$ effort. Since these are MKP instances, only the variables in the E_1 and E_2 sets need to be considered. Consequently, the main step requires $O(|E_1|S_{LP_{|E_1|+|E_2|,m}})$ effort. Thus this subroutine runs in $O(|E_1|S_{LP_{|E_1|+|E_2|,m}})$ effort.

The *Generate Valid Inequalities Subroutine* has an initialization phase that trivially requires $O(1)$ effort. Both the horizontal and vertical inequalities also require $O(1)$ effort. The angled inequalities, has two loops that are both bounded by the size of E_1. Each loop requires $O(1)$ effort. Thus, this routine requires solving $|E_1|+1$ linear programs. Thus, the SSAL algorithm requires $O(|E_1|S_{LP_{|E_1|+|E_2|,m}} + |E_1|^2)$ effort.

□

Since linear programs can be solved in polynomial time [23], SSAL is a polynomial time algorithm. Now that the psuedocode, validity, and runtime have all been presented, an example problem can be presented. This small example would only take a fraction of a seconds to solve, but it demonstrates the algorithm and provides some areas of discussion.

3.2 SSAL Example

Recall the MKP from Example 1 regarding the hiker preparing for his trip. The constraints of this model are

$$20x_1+20x_2+18x_3+16x_4+15x_5+14x_6+13x_7+12x_8+12x_9+12x_{10}+11x_{11}+10x_{12} \leq 115$$

$$2x_1+12x_2+29x_3+17x_4+13x_5+4x_6+17x_7+18x_8+20x_9+16x_{10}+8x_{11}+11x_{12} \leq 105$$

$$5x_1+16x_2+16x_3+5x_4+7x_5+8x_6+13x_7+9x_8+15x_9+10x_{10}+17x_{11}+19x_{12} \leq 95.$$

Furthermore, let $E_1 = \{1, 2, 3, 4, 5\}$ and $E_2 = \{6, 7, 8, 9, 10, 11, 12\}$.

This subroutine starts by initializing e_1 to 5 and *numpoints* to 0. Next, the *Find Points* Subroutine solves several LPs of the form: $z_{e_1} = \text{Max} \sum_{i \in E_2} x_i$, subject to $Ax \leq b$, $\sum_{i \in E_1} x_i = e_1$, $0 \leq x \leq 1$.

In this first iteration the LP is

Maximize

$$z_5 = x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} + x_{12}$$

Subject to

$$20x_1+20x_2+18x_3+16x_4+15x_5+14x_6+13x_7+12x_8+12x_9+12x_{10}+11x_{11}+10x_{12} \leq 115$$

$$2x_1+12x_2+29x_3+17x_4+13x_5+4x_6+17x_7+18x_8+20x_9+16x_{10}+8x_{11}+11x_{12} \leq 105$$

$$5x_1+16x_2+16x_3+5x_4+7x_5+8x_6+13x_7+9x_8+15x_9+10x_{10}+17x_{11}+19x_{12} \leq 95$$

$$x_1 + x_2 + x_3 + x_4 + x_5 = 5$$

$$0 \leq x \leq 1.$$

The optimal solution is $z_5 = 2.416$ with $x_5^* = (1, 1, 1, 1, 1, 0, 0, 0, 0.4166, 0, 1, 1)$. This solution creates a new feasible point on the $e_1 - e_2$ graph. This point is $e_1 = 5$ and $e_2 = 2$ which is the objective function rounded down. These values are saved as $feaspoints_{e_1}[0]$:=5 and $feaspoints_{e_2}[0] := 2$. *numpoints* increments to 1, e_1 is reduced by 1 to 4, and this step is repeated for every value of e_1 until it reaches 0. This results in points shown in Table 3.1.

Next the algorithm moves to the *Generate Valid Inequalities Subroutine*. This sub-

$numpoints$	$feaspoints_{e_1}[numpoints]$	z^{e_1}	$feaspoints_{e_2}[numpoints]$
0	5	2.416	2
1	4	3.995	3
2	3	5.482	5
3	2	6.593	6
4	1	6.947	6
5	0	7	7

Table 3.1: Reported values from *Find Points Subroutine*

routine is initialized by setting $loc, numconst$, and $sumin$ to 0. First it creates an inequality from the first point that has the maximum number of e_1. This is represented generically by setting $validineq_{\alpha_1}[numconst] := 1$, $validineq_{\alpha_2}[numconst] := 0$ and $validineq_{\beta}[numconst] := feaspoints_{e_1}[0]$. In this example, $feaspoints_{e_1}[0] = 5$ and the valid inequality is $x_1 + x_2 + x_3 + x_4 + x_5 \leq 5$. The variable $numconst$ would then be incremented by 1 so the next inequality is saved in the next cell of each array. There is only one horizontal inequality.

Next sloped inequalities can be calculated. This begins by first saving the information of the starting point. Since the algorithm just began, this is the first point or when $numpoints = 0$, (5,2). Using the data reported from the *Find Points Subroutine*, $e_1^{start} :=$ 5 and $e_2^{start} := 2$ and α is set arbitrarily high. Any value higher than $|E_2|$ is large enough for even the most extreme conditions because the possible slopes are equal to the change in e_2 divided by e_1, and since e_1 changes by a minimum of 1, the greatest value of α_{new} is the change in e_2 which is the size of $|E_2|$.

Then for every point after the starting point, the slope of a line that passes through both the starting point and this possible ending point is calculated. This is done by first obtaining these new values from the *Find Points Subroutine*. In this example, the first possible ending point is when numpoints=1; This results in the following assignments: $e_1^{end} := 4$ and $e_2^{end} := 3$.

These new values allow α_{new} to be calculated using the equation $\alpha_{new} = (e_1^{start} - e_1^{end})/(e_2^{end} - e_2^{start})$ or in this case, $\alpha_{new} = \frac{5-4}{3-2} = 1$. This value of α_{new} is compared to the current α, which is still arbitrarily high. Since it is less than this value, α is assigned a new value of 1. This same procedure is continued for every possible ending point until the lowest value of α is found. Table 3.2 of the possible ending point for this first iteration:

numpoint	e_1^{end}	e_2^{end}	α_{new}	$slope_1$	$slope_2$
1	4	3	1	1	1
2	**3**	**5**	$\mathbf{\frac{2}{3}}$	**2**	**3**
3	2	6	$\frac{3}{4}$	3	4
4	1	6	1	4	4
5	0	7	1	5	5

Table 3.2: Possible ending points for start point: numpoints=0

The lowest α in Table 3.2 is $\frac{2}{3}$ and occurs between (5,2) and (3,5). The slopes from these two points can assign the coefficients and calculate the β for new valid inequalities. The inequality for the changes in e_1 and e_2 can be made using the following variables and values: $slope_1 := (e_1^{start} - e_1^{end}) = 5 - 3 = 2$, $slope_2 := (e_2^{end} - e_2^{start}) = 5 - 2 = 3$, and $\beta := e_1^{start} * slope_2 + e_2^{start} * slope_1$. These would be saved as $validineq_{\alpha_1}[0] := 3, validineq_{\alpha_2}[0] := 2, validineq_{\beta}[0] := 5 * 3 + 2 * 2 = 19$. Which would represent the valid inequality $3\sum_{i \in E_1} x_i + 2\sum_{i \in E_2} x_i \leq 19$. The variable *numconst* is incremented again after each inequality's values are added to the array.

This process is then repeated with this end point as the new start point with $e_1^{start} :=$ 3 and $e_2^{start} := 5$. Each point following this point is reassessed for a new lowest slope. The possible ending points for this second iteration are in Table 3.3. The lowest α is once again used to determine the coefficients and β for new valid inequalities. These represent the following inequality: $\sum_{i\in E_1} x_i + \sum_{i\in E_2} x_i \leq 8$.

$numpoint$	e_1^{end}	e_2^{end}	α_{new}	$slope_1$	$slope_2$
3	**2**	**6**	**1**	**1**	**1**
4	1	6	2	2	1
5	0	7	$\frac{3}{2}$	3	2

Table 3.3: Possible ending points for start point: numpoints=2

$numconst$	$validineq_{\alpha_1}[numconst]$	$validineq_{\alpha_2}[numconst]$	$validineq_{\beta}[numconst]$
0	1	0	5
1	3	2	19
2	1	1	8
3	1	2	14
4	0	1	7

Table 3.4: Arrays for inequality values

When the final point is used as the ending point, the algorithm makes the final angled inequality. In this example, there is one more inequality, $\sum_{i\in E_1} x_i + 2\sum_{i\in E_2} x_i \leq 14$. Next, SSAL creates the final cut, the vertical inequality. This final inequality uses only the final point, and takes the form $\sum_{i\in E_2} x_i \leq 7$. This is the final inequality and with the addition of its values to the arrays, the algorithm is complete. All the values for the valid inequalities which are stored in their respective arrays, $validineq_{\alpha_1}[numconst], validineq_{\alpha_2}[numconst]$, and $validineq_{\beta}[numconst]$, are used to create new inequalities. These arrays can be seen in Table 3.4 For this example, there are 5 cuts created and they are:

$\sum_{i \in E_1} x_i \leq 5$

$3 \sum_{i \in E_1} x_i + 2 \sum_{i \in E_2} x_i \leq 19$

$\sum_{i \in E_1} x_i + \sum_{i \in E_2} x_i \leq 8$

$\sum_{i \in E_1} x_i + 2 \sum_{i \in E_2} x_i \leq 14$

$\sum_{i \in E_2} x_i \leq 7$

These valid inequalities from SSAL can be examined to determine their usefulness in solving the IP. As stated before, the strongest inequalities are facet defining inequalities. To determine whether the inequalities are facet defining, first we must examine the dimension of the polytope. P^{MKP} is fully dimensional and so its dimension is 12. This implies that a facet defining inequalities must be 11-dimensional or has 12 affinely independent points on its face. In this example problem, two of the generated inequalities are facet defining: $3 \sum_{i \in E_1} x_i + 2 \sum_{i \in E_2} x_i \leq 19$ and $\sum_{i \in E_1} x_i + \sum_{i \in E_2} x_i \leq 8$.

x_1	1	1	1	1	1	1	1	1	0	1	1	1
x_2	1	1	1	1	1	1	1	0	1	0	1	1
x_3	1	1	1	1	1	1	1	1	0	0	0	0
x_4	1	1	1	1	1	1	1	0	1	1	0	1
x_5	1	1	1	0	1	1	1	1	1	1	1	0
x_6	1	0	0	0	0	0	0	1	1	1	1	1
x_7	0	1	0	0	0	0	0	0	0	0	0	0
x_8	0	0	1	0	0	0	0	1	1	1	1	1
x_9	0	0	0	1	0	0	0	0	0	0	0	0
x_{10}	0	0	0	0	1	0	1	1	1	1	1	1
x_{11}	1	1	1	1	0	1	1	1	1	1	1	1
x_{12}	0	0	0	0	1	1	0	1	1	1	1	1

Figure 3.1: Affinely independent points for $3 \sum_{i \in E_1} x_i + 2 \sum_{i \in E_2} x_i \leq 19$

Figures 3.1 and 3.2 provide 12 feasible affinely independent points. This means that these two SSAL cuts induce 11-dimensional faces and are therefore facet defining to P_{ch}^{MKP}.

x_1	1	1	1	1	1	1	1	1	0	1	1	1
x_2	0	0	0	0	0	0	0	0	1	0	1	1
x_3	0	0	0	0	0	0	0	1	0	0	0	0
x_4	0	0	0	0	0	0	0	0	1	1	0	1
x_5	1	1	1	1	1	1	1	1	1	1	1	0
x_6	0	1	1	1	1	1	1	1	1	1	1	1
x_7	1	0	1	1	1	1	1	0	0	0	0	0
x_8	1	1	0	1	1	1	1	1	1	1	1	1
x_9	1	1	1	0	1	1	1	0	0	0	0	0
x_{10}	1	1	1	1	0	1	1	1	1	1	1	1
x_{11}	1	1	1	1	1	0	1	1	1	1	1	1
x_{12}	1	1	1	1	1	1	0	1	1	1	1	1

Figure 3.2: Affinely independent points for $\sum_{i\in E_1} x_i + \sum_{i\in E_2} x_i \leq 8$

This ability to make facet defining inequalities more consistently than previous algorithms is a good indicator of the usefulness of the algorithm. This is a small example problem so the appearance of facet defining inequalities shows the ability to make facet defining inequalities even when not working on large problems.

3.3 Advancements of SSAL

One natural question is whether or not SSAL is better than SSL examined on the individual constraints. This example also shows that SSAL has stronger inequalities than SSL applied on each individual constraint and thus it has the potential to be useful in practice.

Consider the example problem and let the initial sets E_1 and E_2 be the same as in the SSAL example. The SSL algorithm begins the same as SSAL in that it sets e_1 to its maximum and works its way down. SSL also calculates a set of feasible points which are the optimal solution to

Maximize

$z_{e_1}^j = \sum_{i \in E_2} x_i$

subject to

$Ax \leq b^j$

$\sum_{j \in E_1} x_j = e_1$

$x \in \{0, 1\}$

Due to the knapsack structure, these problems are simple enough that Bolton created a linear time algorithm to determine these feasible points. Thus, SSL should be applied to each constraint. Clearly, there are two methods to perform this. One adds valid inequalities based upon each constraint. The second and clearly stronger version finds the feasible points for each constraint and then takes the minimum. Valid inequalities can be created from these feasible points. SSAL is still better than this stronger application as the following discussion shows.

The first constraint has (5,2), (4,4), (3,5), (2,7), (1,7), (0,7) as potential extreme points. Continuing for the second and third constraints yields Table 3.5. Clearly, the candidate point would be the minimum for a particular e_1 value. Thus, the set of potential extreme points are (5,2), (4,4), (3,5), (2,6), (1,6), (0,7).

e_1	$z^0_{e_1}$	$z^1_{e_1}$	$z^2_{e_1}$	z_{e_1}
5	**2**	3	4	**2**
4	**4**	5	5	**4**
3	**5**	6	6	**5**
2	7	**6**	**6**	**6**
1	7	7	**6**	**6**
0	**7**	**7**	**7**	**7**

Table 3.5: SSL IP solutions

These values can then be used as points to make valid inequalities. As before e_1 values can be saved in $feaspoints_{e_1}[numpoints]$ and z_{e_1} can be saved in $feaspoints_{e_2}[numpoints]$ Since both SSL and SSAL make constraints using the same methods, we only need to examine the points to determine the difference in strength of the output inequalities. The table of both algorithms points can be found in Table 3.6.

$numpoints$	$feaspoints_{e_1}$	$SSALfeaspoints_{e_2}$	$SSLfeaspoints_{e_2}$
0	5	2	2
1	**4**	**3**	**4**
2	3	5	5
3	2	6	6
4	1	6	6
5	0	7	7

Table 3.6: Points reported from SSAL and SSL

This example problem only has one point change, but this is most likely caused by the original formulation having 3 similarly formed inequalities. With more constraints that vary more in nature, SSL loses its ability to create strong inequalities due to its inability to use multiple constraints at a time. The point that changes in this example is when $feaspoints_{e_1}[numpoints] = 4$. While SSAL provides the $feaspoints_{e_2}[numpoints]$ value

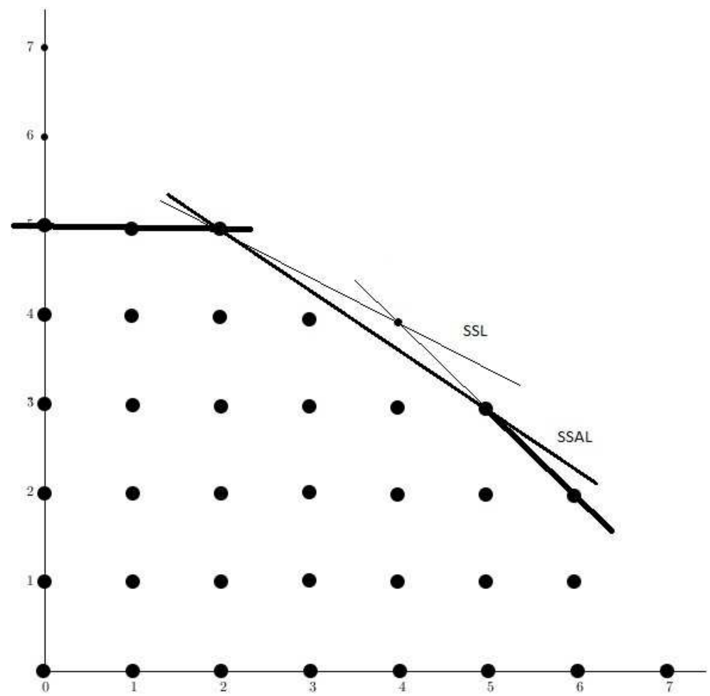

Figure 3.3: SSL and SSAL constraint differences

of 3, SSL provided a less restrictive value of 4. This point is extreme and therefore would be used when creating new constraints during the *Generate Valid Inequalities*Subroutine. The effect on this inequality can be seen more clearly in Figure 3.3.

The thinest line shows the inequalities generated only by SSL and the regular line shows the inequality generated by just SSAL, while the thickest lines show the inequalities that are present in both SSL and SSAL. Each point that is eliminated by using SSAL instead of SSL, has the possibility of affecting two inequalities. This example only affects one because the new point (4,4) sits on a pre-existing inequality $\sum_{i \in E_1} x_i + \sum_{i \in E_2} x_i \leq 8$, so this inequality remains the same. The other inequality is affected. The SSL inequality $2 \sum_{i \in E_1} x_i + \sum_{i \in E_2} x_i \leq 12$, is still valid as all its output equalities are but is not as restrictive on the inequality produced by SSAL. This inequality is clearly less restrictive than $3\sum_{i\in E_1} x_i + 2\sum_{i\in E_2} x_i \leq 19$ produced by SSAL, but what does that actually mean in terms of dimension. There are only 7 affinely independent points that meet $2\sum_{i\in E_1} x_i + \sum_{i\in E_2} x_i \leq 12$ at equality. These points are shown as a matrix in Figure 3.4.

x_1	1	1	1	1	1	1	1
x_2	1	1	1	1	1	1	1
x_3	1	1	1	1	1	1	1
x_4	1	1	1	1	1	1	1
x_5	1	1	1	1	1	1	1
x_6	1	0	0	0	0	0	0
x_7	0	1	0	0	0	0	0
x_8	0	0	1	0	0	0	0
x_9	0	0	0	1	0	0	0
x_{10}	0	0	0	0	1	0	1
x_{11}	1	1	1	1	0	1	1
x_{12}	0	0	0	0	1	1	0

Figure 3.4: Affinely independent points for $2\sum_{i\in E_1} x_i + \sum_{i\in E_2} x_i \leq 12$

Since there are only 7 affinely points, that means that the dimension of the inequality is 6. This SSL inequality is 6 dimensions smaller than P_{ch}^{MKP}, which means that it is not facet defining. Notice that this inequality only has feasible points at the (5,2) point. This is because there are no feasible points at the (4,4) point. This is what allows SSAL to cut that point off and create a stronger inequality. This proves that the inequalities created by SSAL can be stronger than those created by SSL.

Chapter 4

SSAL Computational Results

Now that we have seen the theoretical advances of SSAL over previous algorithms, an obvious question is how well it performs in practice? In this chapter, computational studies of SSAL are reported. First, the creation of the sample problems is described. Next, the implementation issues of SSAL for this particular study are discussed. Finally, the results from the computational study are presented and analyzed.

4.1 Computational Instances

Various multiple knapsack problems comprise this computational study. Recall that a multiple knapsack takes the form Maximize $c^T x$ subject to $Ax \leq b$, $x \in \mathbb{B}^n$ where A is required to be nonnegative. One of the largest issues with knapsack instances, is that many such problems are either prohibitively challenging to solve or surprisingly easy [15].

For instance, MIPLIB has 270 multiple knapsack instances, of these 150 solve in less than a minute. The other 120 require many hours or are unable to solve in a reasonable amount of time.

In an effort to find problems in this middle ground, the spirit of the instances from MIPLIB was followed. There are 3 categories of variations in the setup of the problem groups: number of variables, number of constraints, and acceptance probability. Number of variables and constraints are represented in the MKP formulation by n and m respectively and acceptance probability, p, is discussed in the instance formation.

The instances are created by first assigning each $a_{i,j}$ to a random integer between 1,000 and 10,000. Since most IPs are sparse, there is a acceptance probability. A uniform random number between 0 and 1 is generated and if is larger than the acceptance probability, then $a_{i,j} = 0$, else $a_{i,j}$ remains unchanged. The right hand side of each constraint, b_i, is calculated by summing the $a_{i,j}$ in that row and multiplying by the slackness coefficient ρ, $b_i = \sum_{j=1}^{n} \rho a_{i,j}$. The benefits c_j are equal to $c_j = u_j + \sum_{i=1}^{m} a_{i,j}$ where u_j is a uniform number between 0 and 1000.

Through preliminary research, it was discovered that problems with many constraints tend to have better computational results with SSAL as expected. For this reason, the parameters were set so there will be approximately 10 times the number of constraints as there are variables. Since the solution time of the MKP can grow exponentially, relatively small values of n were chosen so solutions could be found in reasonable amounts of time. For this reason, the values used for n are 20, 30, and 40 while the values for m are 200, 300, and 400. To get a full view of the scarcity of the problems, the values of p chosen are .25, .50 and .75. Every combination of the 3 parameters were tested, resulting in 27 test groups.

Since these are randomly generated instances and to avoid lucky or unlucky instances, 10 instances of each group of problems are created and solved. This helps to provide a more comprehensive view of the benefits of SSAL.

4.2 Implementation of SSAL

This section is about the different methods SSAL used on the sample problems, and descriptions about the most effective methods attempted. These are not changes in the algorithm, merely how sets are chosen, which inequalities to include and how many different pairs of sets selected.

The inputs to SSAL are a multiple knapsack problem and two variable sets: E_1 and E_2. How these two sets are formed is very important. Several set creation methods were attempted and involved sorting the variables for inclusion, but in the end, setting a condition for acceptance and arbitrarily assigning to groups was the most effective. This primary criterion for set selection is based upon a variable's reduced cost, π_j. The π_j represents the cost of moving a previously non-basic variable or excluding an included variable. If π_j is greater than or equal to a cutoff point, π', then x_j is included into one of the sets. This π' is a value determined by the programmer and has no set value. If π' is equal to 0, then the algorithm includes all the variables with positive x values in the linear relaxation. For this reason a value less than 0 is suggested. The best value for π' is largely dependent on the problems in which it is used, but with these sample problems, a relatively small value, $\pi' = -50$, was effective and is the value used for these examples. Define $E = \{j \in N : \pi_j \geq \pi'\}$.

Once E is determined, there are numerous methods to divide this set into two groups. Again several methods were attempted and alternating between set E_1 and E_2 provided stronger computational results. In order to provide numerous sets to generate constraints, the following rules were implemented. During the first pass, the variables alternate between the sets. For the second pair, two are placed into set E_1 followed by two into set E_2, which repeats until E is partitioned. Next three went to E_1 and three to E_2. This process continues until there were 6 in one set before changing to the other set. Minimal computational improvements occurred by adding more sets in this fashion.

The following subroutine, *Create Sets*, provides psuedocode for this process. Although this is not an actual subroutine from SSAL, it is very crucial to the successful implementation of SSAL and the psuedo code for this is as follows:

Create Sets

Set $k := 0$.

for each i=1..n

if $(\pi_i) \geq (\pi')$, then

Set $E_0[k] := i$.

Set $k := k + 1$.

Set $E_0[k]$:=-1.

Set $cnt := 0$.

Set $cntt := 0$.

Set $i := 0$.

Set $control := 1$.

while $E_0[i] \neq -1$

if $control$ equals 1, then

Set $E_1[cnt] := E_0[i]$.

Set $cnt := cnt + 1$.

else

Set $E_2[cntt] := E_0[i]$.

Set $cntt := cntt + 1$.

if $(cnt + cntt)\text{mod}(set + 1) = 0$, then

Set $control$:= ($control$+1) mod 2.

Set $i := i + 1$.

Set $E_1[cnt] := -1$.

Set $E_2[cntt] := -1$.

The final change in implementation is the acceptance or rejection of inequalities. When SSAL is run multiple times, it creates many inequalities, but these inequalities are not equally strong. For this reason, not every inequality is accepted and used in the solving of the MKP. Since the linear relaxation is available, and represents the optimal non-integer solution, it provides a good benchmark for analyzing these inequalities. The values for each variable from the linear relaxation solution are substituted into the inequality to get its current value. This is compared to the inequality's rhs. When directly compared, it would show whether or not the inequality cut off the linear relaxation point. Since the goal is to cut off as much non-integer space as possible and not just the linear relaxation, the rhs of each inequality is multiplied by a scalar, s, which allows inequalities that are close to eliminating the linear relaxation to be included. This allows for only the top inequalities to be included which increases the effectiveness of SSAL. For these problems, a value of 1.25 is used for s.

These three implementational changes have allowed SSAL to be effective. In the next section, the results of applying the recently discussed implementation to the sample problems shows the real usefulness of SSAL.

4.3 Computational Results

The instances were solved with and without the use of SSAL inequalities. The SSAL inequalities were added as preprocessing cutting planes two results were saved from each solution: the solution times, t^{CPLEX} and t^{SSAL}, and the original linear relaxation values,

LR^{CPLEX} and LR^{SSAL}. The values returned from the 10 instances from each group were averaged. Next the percent changes from CPLEX to SSAL,t^{δ} and LR^{δ} were calculated, so a better picture of SSAL's results can be observed in Table 4.1.

m	n	p	t^{CPLEX}	t^{SSAL}	t^{δ}	LR^{CPLEX}	LR^{SSAL}	LR^{δ}
200	20	.25	.0172	.0161	93%	2735	2510	92%
		.50	.264	.263	99%	5516	5042	91%
		.75	1.40	1.46	104%	8251	7549	91%
	30	.25	.219	.238	108%	4153	3937	95%
		.50	23.4	23.7	101%	8288	7851	95%
		.75	276	267	96%	12296	11643	95%
	40	.25	7.00	7.36	105%	5522	5320	96%
		.50	2650	2690	102%	10999	10583	96%
		.75	9763	9981	102%	16430	16247	99%
300	20	.25	.0156	.0138	88%	4152	3737	90%
		.50	.423	.394	93%	8288	7554	91%
		.75	1.40	1.46	104%	12296	11197	91%
	30	.25	.151	.151	100%	6221	5880	94%
		.50	30.2	31.7	104%	12386	11685	94%
		.75	288	300	104%	18484	17456	94%
	40	.25	6.20	7.03	113%	8198	7879	96%
		.50	3890	3830	98%	16425	15744	96%
		.75	14700	15300	104%	24738	24234	98%
400	20	.25	.0132	.0129	97%	5547	4819	87%
		.50	.474	.433	91%	11012	10028	91%
		.75	3.01	2.99	99%	16433	14932	91%
	30	.25	.149	.146	97%	8299	7829	94%
		.50	32.7	30	92%	16480	15524	94%
		.75	335	344	102%	24665	23239	94%
	40	.25	6.53	7.14	109%	10939	10497	96%
		.50	5910	6120	103%	22003	21074	96%
		.75	7276	7190	98%	33036	32691	99%

Table 4.1: Computational results of SSAL

Upon inspection of the table, it is clear that the LR solution is reduced in nearly all cases. This is because SSAL consistently is able to cut off the LR point, because it always includes the variables used in the solution. It averages a 94% reduction across all these example problems. The change in solution time is less obvious, and can commonly result in an increase in solution times, due to the addition of inequalities that slow the branch and bound iteration.

However, through statistical analysis, it can be seen with an 80% confidence that the correlation between the number of variables and number of constraints affects the solution times of the problem. It shows that as the number of constraints compared to the number of variables increases, the solution times were reduced. The most significant showing of this, is when the number of constraints was at least 10 times the number of variables. Table 4.2 shows only the results when this condition is met with the solutions where constraints are strictly greater than 10 times the number of variables in bold.

m	n	p	t^{CPLEX}	t^{SSAL}	t^{δ}	LR^{CPLEX}	LR^{SSAL}	LR^{δ}
200	20	.25	.0172	.0161	93%	2735	2510	92%
		.50	.264	.263	99%	5516	5042	91%
		.75	1.40	1.46	104%	8251	7549	91%
300	20	.25	.0156	.0138	**88**%	4152	3737	90%
		.50	.423	.394	**93**%	8288	7554	91%
		.75	1.40	1.46	**104**%	12296	11197	91%
	30	.25	.151	.151	100%	6221	5880	94%
		.50	30.2	31.7	104%	12386	11685	94%
		.75	288	300	104%	18484	17456	94%
400	20	.25	.0132	.0129	**97**%	5547	4819	87%
		.50	.474	.433	**91**%	11012	10028	91%
		.75	3.01	2.99	**99**%	16433	14932	91%
	30	.25	.149	.146	**97**%	8299	7829	94%
		.50	32.7	30	**92**%	16480	15524	94%
		.75	335	344	**102**%	24665	23239	94%
	40	.25	6.53	7.14	109%	10939	10497	96%
		.50	5910	6120	103%	22003	21074	96%
		.75	7276	7190	98%	33036	32691	99%

Table 4.2: Computational results of SSAL with 10:1 or greater constraint to variable ratio

Table 4.2 begins to provide a much stronger showing of SSAL's beneficial impact on the solution times in problems with many constraints compared to variables. This was not the only information obtained from the original computational study. Another statistical test on the original data showed that the acceptance probability was not linear but instead showed a parabolic nature with the minimum in the middle. This means that the problems that had a .5 acceptance probability tended to have lower solutions times with SSAL. Table 4.3 shows only these results, again with the solutions greater than 10:1 constraint to variable ratio shown in bold.

m	n	p	t^{CPLEX}	t^{SSAL}	t^{δ}	LR^{CPLEX}	LR^{SSAL}	LR^{δ}
200	20	.50	.264	.263	99%	5516	5042	91%
	30	.50	23.4	23.7	101%	8288	7851	95%
	40	.50	2650	2690	102%	10999	10583	96%
300	20	.50	.423	.394	**93**%	8288	7554	91%
	30	.50	30.2	31.7	104%	12386	11685	94%
	40	.50	3890	3830	98%	16425	15744	96%
400	20	.50	.474	.433	**91**%	11012	10028	91%
	30	.50	32.7	30	**92**%	16480	15524	94%
	40	.50	5910	6120	103%	22003	21074	96%

Table 4.3: Computational results of SSAL with .50 acceptance probability

This area when there are 10 times more constraints than variables and the constraint coefficients are 50% sparse is where SSAL has shown the most improvement in problem solution times. To test this hypothesis, larger MKP instances were created. This new group has 70 variables and 1000 constraints. Again, ten instances were created in this group and solved as before. Table 4.4 reports every problem from these larger instances.

Table 4.4 shows a clear improvement on solution solve times with all but one instance improving. This average 6% drop in solution time is a clear sign of the beneficial nature

trial	t^{CPLEX}	t^{SSAL}	t^{δ}	LR^{CPLEX}	LR^{SSAL}	LR^{δ}
1	15769	14072	89%	83553047	79798884	95%
2	14546	13543	93%	82405224	80421564	97%
3	13989	13544	97%	81987908	81976908	100%
4	14387	13206	92%	81529016	79122952	97%
5	13925	11329	81%	82101225	82101225	100%
6	15392	14295	93%	81920394	79918596	97%
7	13732	12952	94%	82495839	81847258	99%
8	14285	14001	98%	81593020	81593020	100%
9	13593	13920	102%	82495392	81149302	98%
10	14120	13819	98%	81395830	78284738	96%
Average	14374	13468	94%	82147690	81121445	98%

Table 4.4: Computational results of SSAL for large scale problems

of SSAL in large scale problems. The effect on the change in linear relaxation however is much less profound. This is due to relativity between the size of the problem and how much linear relaxation space there is to be cut off. Because the amount of space that can be cut off does not grow as fast as the problem, only a 2% reduction is observed. Although the inequalities appear not to be cutting off as much linear relaxation space, the new cuts dramatically reduce the solve time, which is the most significant result. This 6% reduction is the equivalent of 14 minutes while solving these problems that averaged 4 hour solve times. If SSAL continues to improve this class of problems in even larger instances, the reduction of solve time could quickly reach hours.

The processing time of SSAL has always been minute compared to the solution time of the multiple knapsack problem, commonly taking fractions of a second. These larger problems were the first to show actual times for processing with times still no longer than 3 seconds. Compared to the 14 minutes which were cut from the problem solution time, this 3 seconds is irrelevant.

These computational studies clearly show the improvement possible with SSAL. This algorithm commonly cuts off initial linear relaxation solution, and has a large improvement in solution times in problems where the number of constraints is larger than 10 times the number of variables and the constraints are 50% filled.

Chapter 5

Conclusions

The goal of this thesis was to expand on the SSL algorithm to expand its use to multiple knapsack problems and to do this without an exponentially long processing time. To achieve this, an approximate version of SSL was developed. SSAL lifts two sets into a valid inequality. Another objective is to generate cutting planes that perform better than traditional CPLEX and create stronger inequalities than SSL. The SSAL algorithm presented in this thesis achieves these goals.

SSAL requires $O(|E_1|S_{LP_{|E_1|+|E_2|,m}} + |E_1|^2)$ effort where S_{LP} is the time to solve an LP which is polynomial. The inequalities generated are valid and have the possibility of being facet defining. This is illustrated by an example which demonstrates all critical aspects of the algorithm. After executing SSAL, five inequalities are generated. Two of these inequalities are shown to be facet defining.

The computational study presented in Chapter 4 shows that adding SSAL inequalities to CPLEX enabled the software to solve some classes of problems 6% faster than with CPLEX alone. SSAL inequalities are created very quickly, requiring no more than a few seconds to generate.

5.1 Future Research

SSAL has moved the research in synchronized simultaneous lifting forward but there are still many new ideas that can still be explored. In this section, some possible continuations of this research will be presented.

At present, synchronized simultaneous approximate lifting is only able to use two sets. Harris [22] developed an algorithm to exact synchronized simultaneous up lift three sets. There is opportunity for research which allows approximate lifting of three or more sets. This would allow the creation of more inequalities with greater variety.

This thesis focuses only on an approximate computation which causes it to create weaker inequalities than are theoretically possible to save on computational time. Some of the extreme points might not however be extreme. An interesting expansion on SSAL could find the extreme points that are most likely to not have any feasible points, and run a quick exact check on these points. This would allow the algorithm to strengthen some of its inequalities without damaging computational times significantly.

The computational studies performed in this thesis are on sample problem in which all variables are created uniformly. This causes variables which are not included in the

LR to be as likely to be included in the final answer as the variables in the LR. This means that using the variables based on reduced cost is only barely more likely to get variables that are in the final solution as randomly selecting variables from the whole set. It is likely that SSAL would perform better under a more realistic situation where every variable is not created equal.

Bibliography

[1] Atamtürk, A. (2004). "Sequence independent lifting for mixed-integer programming," *Operations Research*, 52 (3), 487-491.

[2] Atamtürk, A. (2003). "On the facets of the mixed-integer knapsack polyhedron," *Mathematical Programming*, 98 (1-3), 145-175.

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

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

[5] Balas, E. and E. Zemel (1984). "Lifting and complementing yields all the facets of positive zero-one programming polytopes," in *Mathematical Programming, Proceedings of the International Conference on Mathematical Programming*, R.W. Cottle et al., eds., 13-24.

[6] Bennett, J.P. (1993). "Knapsack allocation of multiple resources in benefit-cost analysis by way of the analytic hierarchy process". Dept. of Political Sci., Syracuse Univ., NY, USA; Saaty, T.L. *Mathematical and Computer Modelling*, v 17, n 4-5, p 55-72.

[7] Beth, T.; Frisch, M.; Simmons, G.J. (1992). "Public-Key Cryptography: State of the Art and Future Directions". E.I.S.S. Workshop. Final Report eds. Source: , xi+97.

[8] Beyer, Carrie. (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.

[9] Cadoux, Florent. (2010). "Computing deep facet-defining disjunctive cuts for mixed-integer programming". *Mathematical Programming*. 2010-04-01.

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

[11] Conley, Clark. (2009). "Cliqued holes and other graphic structures for the node packing polytope" *MS Thesis*, Department of Industrial and Manufacturing Systems Engineering, Kansas State University.

[12] The CPLEX Solver on ILOG's Home Page, http://www.ilog.com/.

[13] De Farias Jr, I., E. Johnson, and G. Nemhouser (2002). "Facets of the complimentarity knapsack polytope," *Mathematics of Operations Research*, 27 (1), 210-227.

[14] Easton, T. and K. Hooker, "Simultaneously lifting sets of binary variables into cover inequalities for knapsack polytopes," *Discrete Optimization, Special Issue: In Memory of George B. Dantzig*, 5 (2) May 2008, 254-261.

[15] Easton, T., K. Hooker and E. K. Lee, (2003) "Facets of the Independent Set Polytope," *Mathematical Programming, Series B* 98(1-3), 177-199.

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

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

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

[19] Sharma, Kamana, (2007) "Simultaneously lifting sets of variables in binary knapsack problems," *MS Thesis*, Department of Industrial and Manufacturing Systems Engineering, Kansas State University.

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

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

[22] 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.

[23] Karmarkar,N. (1984). A new polynomial-time algorithm for linear programming. In Proceedings of the sixteenth annual ACM symposium on Theory of computing (STOC '84). ACM, New York, NY, USA, 302-311.

[24] Nemhauser, G. L. and P. H. Vance, (1994). "Lifted cover facets of the 0-1 knapsack polytope with GUB constraints," *Operations Research Letters*, 16 (5), 255-263.

[25] Park, K. (1997). "Lifting cover inequalities for the precedence-constrained knapsack problem," *Discrete Applied Mathematics*, 72 (3), 219-241.

[26] Pavelka, Jeff. (2011). "Sequential and simultaneous lifting in the node packing polyhedron". *MS Thesis*, Department of Industrial and Manufacturing Systems Engineering, Kansas State University.

[27] Pisinger, D. (2001). "Budgeting with bounded multiple-choice constraints". Dept. of Comput. Sci., Copenhagen Univ., Denmark. *European Journal of Operational Research*, v 129, n 3, p 471-80.

[28] Renegar, James. (1988). "A polynomial-time algorithm, based on Newton's method, for linear programming". *Mathematical Programming*. 1988-01-01.

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

[30] Subbu, R. (2007). ”Multi-criteria set partitioning for portfolio management: a visual interactive method”. Gen. Electr. Global Res., Niskayuna, NY, USA; Russo, G.; Chalermkraivuth, K.; Celaya, J. 2007 First IEEE Symposium on Computational Intelligence in Multicriteria Decision Making, p 6.

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

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

[33] Zongzhi Li. (2010). ”A heuristic approach for selecting highway investment alternatives” Dept. of Civil, Archit. Environ. Eng., Illinois Inst. of Technol., Chicago, IL, United States; Madanu, S.; Bei Zhou; Yuanqing Wang; Abbas, M. *Computer-Aided Civil and Infrastructure Engineering*, v 25, n 6, p 427-39.