Valid Inequalities and Computational Results for SOS1-, SOS2-, and Cardinality-Constrained Linear Programs

by

Ernee Kozyreff Filho, M.S.

A Dissertation

In

Industrial Engineering

Submitted to the Graduate Faculty
of Texas Tech University in
Partial Fulfillment of
the Requirements for
the Degree of

Doctor of Philosophy

Approved

Dr. Ismael R. de Farias
Chair of Committee

Dr. Jennifer Farris

Dr. Arne Ledet

Dr. Philip W. Smith

Dr. Shiren Wang

Dr. Mark Sheridan
Dean of the Graduate School

May, 2014

ACKNOWLEDGEMENTS

First, I thank Dr. Ismael Regis de Farias Jr. for the every kind of support given to me during the last 5 years. Without him, I would not have completed this program successfully. Second, I thank my family for everything they do and for keeping it together during all these years I was away from home. Third, I thank my professors who have made me push my boundaries of imagination and reasoning. Finally, I thank all my friends who accompanied me during this exceptional time of my life.

TABLE OF CONTENTS

ABSTRACT

In this dissertation, we consider three types of constraints that are frequently used in mixed integer programming (MIP), namely: SOS1, SOS2, and cardinality. We study and develop cutting planes for those types of constraints, and use them within branch-and-cut to solve difficult MIP instances.

Our computational results indicate that the cutting planes used are very effective in reducing the solution times of our instances. We also study the effect of other common features present in commercial solvers, such as preprocessing, "standard" cutting planes, and heuristics.

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Optimization is an area of study that lies in the intersection of pure mathematics and real-life industry applications. Practitioners need to make processes faster, reduce costs, improve quality of products, and, ultimately, increase profits. Some of the tools that can be used to reach those goals rely on mathematical models that, in many cases, are developed by academic researchers.

Besides the ability to develop and apply models to industrial problems, nowadays there exists a need for efficient and robust software, which, in the case of optimization, are commonly called *solvers*. Commercial solvers are built to be as general as possible, i.e., they should be capable of solving diverse types of problems, and, for this reason, must incorporate a number of routines that account for the most common types of constraints encountered in practice.

Take, for example, binary variables, i.e., whose only possible values are 0 or 1. This type of variable often represents the answers "YES" or "NO" to a decision, e.g. "build" or "not build", "schedule" or "not schedule", "buy" or "not buy". It is easy to see their importance in modeling problems, and, given their high occurrence, commercial solvers have incorporated several facilities that take into account binary variables and speed up the time taken to find the best possible solution.

Likewise, there are certain types of constraints that solvers can identify and take advantage of their structure. In this dissertation, we will study three kinds of constraints that, we believe, are currently not well handled by state-of-the-art solvers. The constraints we will study are:

- Multiple Choice
- Piecewise Linear
- Cardinality

The first two types of constraints have been considerably explored in the literature. Here, our contribution is mainly in *testing* the computational effectiveness of different strategies to solve problems with those constraints. We implement those strategies and try to speed up the solution time of the solver (using different settings).

The last type of constraint has not been as much explored as the others in the literature, and here we present a theoretical study of it followed by computational tests.

1.1 Dissertation Outline

In Chapter 2, we review fundamental concepts and establish some notation that will be used throughout the text. In chapters 3, 4, and 5, we study linear programming problems with the aforementioned types of constraints, i.e., *multiple choice*, *piecewise linear*, and *cardinality*, respectively. In chapter 6, we give final conclusions and suggest topics for further research.

CHAPTER 2
REVIEW OF FUNDAMENTAL CONCEPTS

In this chapter we state important concepts that will be used throughout the text. All results are given without proof, and the reader is referred to [8] and [20] for more details.

2.1 Linear and Affine Algebra

Let m and n be positive integers, and $v^1, \ldots, v^m$ be vectors (or points) in $\mathbb{R}^n$. We say that the point v is a *linear combination* of $v^1, \ldots, v^m$ if there exist $\lambda_1, \ldots, \lambda_m \in \mathbb{R}$ such that

$$v = \lambda_1 v^1 + \ldots + \lambda_m v^m.$$

If $\lambda_1 + \ldots + \lambda_m = 1$ as well, we say that v is an *affine combination* of $v^1, \ldots, v^m$, and if, additionally, $\lambda_1, \ldots, \lambda_m \geq 0$, we say that v is a *convex combination* of $v^1, \ldots, v^m$.

A set of points is said to be *linearly independent* if none of the points in the set can be written as a linear combination of the others. Alternatively, we say that $v^1, \ldots, v^m$ are linearly independent if $\lambda_1 = \ldots = \lambda_m = 0$ is the unique solution to $\lambda_1 v^1 + \ldots + \lambda_m v^m = 0$. The maximum number of linearly independents points in $\mathbb{R}^n$ is n. Similarly, a set of points is said to be *affinely independent* if none of the points in the set can be written as an affine combination of the others. Alternatively, we say that $v^1, \ldots, v^m$ are affinely independent if $\lambda_1 = \ldots = \lambda_m = 0$ is the unique solution to both $\lambda_1 v^1 + \ldots + \lambda_m v^m = 0$ and $\lambda_1 + \ldots + \lambda_m = 0$. The maximum number of

affinely independents points in $\mathbb{R}^n$ is $n+1$.

2.2 Polyhedral Theory

A *polyhedron* is the solution set to a finite system of linear inequalities. The *dimension* of a polyhedron P, denoted $dim(P)$, is equal to k if the maximum number of affinely independent points in P is $k+1$. If $P \in \mathbb{R}^n$ and $dim(P) = n$, we say that P is *full-dimensional*.

This definition of dimension of a polyhedron agrees with our intuition. For instance, in $\mathbb{R}^3$, the entire space has dimension 3, and so does any object with volume, such as a cube. A plane, or a planar figure, has dimension 2. Lines and line segments have dimension 1, and a single point has dimension zero.

Example 2.2.1. The system of inequalities in $\mathbb{R}^2$

$$\begin{cases} -2x_1 + x_2 \leq 4 & \text{(i)} \\ 2x_1 + 5x_2 \leq 50 & \text{(ii)} \\ x_2 \leq 9 & \text{(iii)} \\ 5x_1 + x_2 \leq 48 & \text{(iv)} \\ 5x_1 - x_2 \leq 40 & \text{(v)} \\ x_1 \leq 10 & \text{(vi)} \\ x_1 \geq 0 & \text{(vii)} \\ x_2 \geq 0 & \text{(viii)} \end{cases} \tag{2.1}$$

defines a polyhedron, call it Q, which is displayed in Figure 2.1.

The points $(0,1)$, $(1,0)$ and $(1,1)$, for instance, are affinely independent and belong to Q. Because Q lies in $\mathbb{R}^2$, there cannot be more than three affinely independent

points in it. Hence, we conclude that $dim(Q) = 3 - 1 = 2$, and thus Q is full-dimensional.

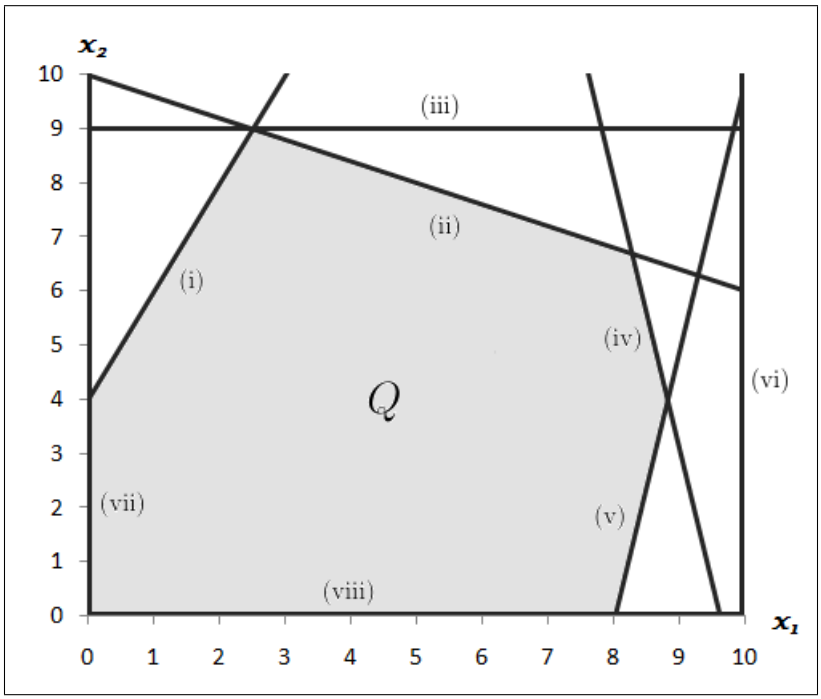


Figure 2.1. The polyhedron Q Defined by Inequalities (2.1).

In general, we can denote a polyhedron $P \in \mathbb{R}^n$ as a set $\{x \in \mathbb{R}^n : Ax \leq \beta\}$. Note that we can always reverse the sign of inequalities by multiplying all coefficients by -1 and thus get appropriate matrix A and vector β. Note also that every equation can be described as a pair inequalities, i.e.,

$$\sum_{i=1}^{n} a_i x_i = b \quad \Leftrightarrow \quad \begin{cases} \sum_{i=1}^{n} a_i x_i \leq b \\ -\sum_{i=1}^{n} a_i x_i \leq -b \end{cases}$$

where $a_1, \ldots, a_n, b \in \mathbb{R}$. Thus, a polyhedron can have equations in its description as

well.

We say that the system of inequalities $Ax \leq \beta$ is *feasible* if there is at least one point x that satisfies it. Otherwise, it is *infeasible*. If the system $Ax \leq \beta$ is infeasible, then $P = \emptyset$, and we say that $dim(P) = -1$. If there exists a $\omega \in \mathbb{R}$ such that $P \subset [-\omega, \omega]^n$, we say that P is *bounded*, and we call it a *polytope*.

An inequality $\sum_{i=1}^{n} a_i x_i \leq b$ is *valid* for P if it is satisfied by all points of P. The set $F = \{x \in P : \sum_{i=1}^{n} a_i x_i = b\}$ is called a *face* of P, and the inequality in question is said to *define a face* of P. If $F = \emptyset$, it is an *empty face* of P. Otherwise it *supports* P. Every face of a polyhedron is also a polyhedron.

Let F be a face of P. If $dim(F) = dim(P) - 1$, we say that F is a *facet* of P. If $dim(F) = 0$, we say that F is an *extreme point*, or a *vertex*, of P. Whenever P is not bounded, it can have *extreme rays*. However, in this dissertation we will not be concerned with those since all polyhedra studied here will be polytopes.

The inequalities that define facets of a full-dimensional polyhedron are the only ones that are necessary for its description. All other inequalities are irrelevant, or redundant. Hence, identifying facet-defining inequalities of a polyhedron is often desirable. It may also be important to identify its extreme points.

Example 2.2.1 (continued) Inequality (iii) defines a face of dimension 0 of Q, so it is redundant. Inequality (vi) defines an empty face of Q, so it is also redundant. All the remaining inequalities define facets of Q and thus are required for its description. It is easy to see in the figure that removing (iii) and (vi) does not change Q, whereas removing any of the other inequalities will change it.

Finally, the extreme points of Q are $(0, 0), (8, 0), (\frac{44}{5}, 4), (\frac{190}{23}, \frac{154}{23}), (\frac{5}{2}, 9)$ and $(0, 4)$.

Theoretically, it is possible to determine all extreme points of a polyhedron given its facets, and vice-versa. In practice, this may be infeasible as the number of extreme points of a polyhedron grows exponentially with its number of facets. For small enough systems $Ax \leq \beta$, though, one can use software such as PORTA [7] to get the facet description of the polyhedron and also all its extreme points (and extreme rays).

2.3 Linear Programming

In Linear Programming (LP), we are interested in maximizing a linear function over a polyhedron in $\mathbb{R}^n$. The usual notation for this problem is

$$\begin{aligned} \text{maximize} \quad & z = \sum_{i=1}^{n} c_i x_i \\ \text{subject to} \quad & Ax \leq \beta, \end{aligned}$$

where z is the *objective function* and $c_i \in \mathbb{R}$, $i = 1, \ldots, n$, are the *objective function coefficients*. Usually, non-negativity of the variables x_i, $i = 1, \ldots, n$, is required, and we tend to write this explicitly, "pulling it out" of the *constraint matrix* A. We then write the problem as

$$\begin{aligned} \text{maximize} \quad & z = \sum_{i=1}^{n} c_i x_i \\ \text{subject to} \quad & Ax \leq \beta \\ & x_i \geq 0, \qquad i = 1, \ldots, n. \end{aligned}$$

We may also be interested in *minimizing* an objective function, but, without loss of generality (WLOG), we may consider the maximization problem only, since min-

imizing z is equivalent to maximizing $-z$. We will use an asterisk do denote the optimal solutions of problems, i.e., we will write x^* for an optimal solution and z^* for the optimal value of the objective function.

Solving LP problems has become a very fast task in the last decades. State-of-the-art solvers such as CPLEX and GUROBI, which use the *Simplex* algorithm (or some variation of it), will usually take less than a few seconds in a modest computer to solve problems with thousands of variables and hundreds of rows in the constraint matrix. The Simplex algorithm is also used as a sub-routine to solve harder problems, which are discussed next.

2.4 Integer Programming

In applications, it is common to require that some variables of the model be integers (or integral). When this is true for all the variables in the formulation, we are dealing with an *integer programming (IP)* problem. If all variables are binary, we call it a *binary integer programming (BIP)* problem. Finally, if the problem has also *continuous* (or real) variables, then we call it a *mixed integer programming (MIP)* problem.

MIP problems are, in general, NP-hard, but in practice the above mentioned solvers can handle very difficult instances of it and, thus, are used by major companies. The approach used to solve this kind of problems is *branch-and-bound*, which we describe next.

2.4.1 Branch-and-Bound

Suppose we want to maximize an objective function over a domain S, i.e., we want to determine $z^* = \max\{\sum_{i=1}^n c_i x_i : x \in S\}$. If we can find sets S_1 and S_2 such that $S = S_1 \cup S_2$, and, additionally, we can find $z_1^* = \max\{\sum_{i=1}^n c_i x_i : x \in S_1\}$ and $z_2^* = \max\{\sum_{i=1}^n c_i x_i : x \in S_2\}$, then our objective is fulfilled: we just take $z^* = \max\{z_1^*, z_2^*\}$. This approach is commonly known as *divide-and-conquer*, and is one of the central ideas behind branch-and-bound—it is the *branch* part of it.

The *bound* part is based on the following idea. Consider again the problem of determining $z^* = \max\{\sum_{i=1}^n c_i x_i : x \in S\}$. Suppose that we are able to easily find $\bar{z}^* = \max\{\sum_{i=1}^n c_i x_i : x \in S_0\}$, where $S \subseteq S_0$. Although this does not solve our initial problem, at least we have an upper bound for z^*, namely $\bar{z}^*$.

But how can we find such a set S_0? One way of doing so is by dropping one or more constraints from S. For example, if $x_1 \in \{0, 1\}$ in S, we may allow $x_1 \in [0, 1]$ in S_0 and thus get model with one less binary variable (remember that linear problems are easier to solve). In this case, we say that S_0 is a *relaxation* of S. If S_0 is defined only by linear constraints, we call it a *linear programming relaxation* (LPR) of S.

The following example illustrates how branch-and-bound works.

Example 2.2.1 (continued) Suppose we want to solve the following problem:

$$\begin{aligned} \text{maximize} \quad & z = -\tfrac{4}{3}x_1 + x_2 \\ \text{subject to} \quad & x \in S \end{aligned} \tag{2.2}$$

where $S = Q \cap \mathbb{Z}^2$ is the feasible set of the problem, shown in Figure 2.2.

In order to apply the Simplex method, we have to ignore the integrality constraint,

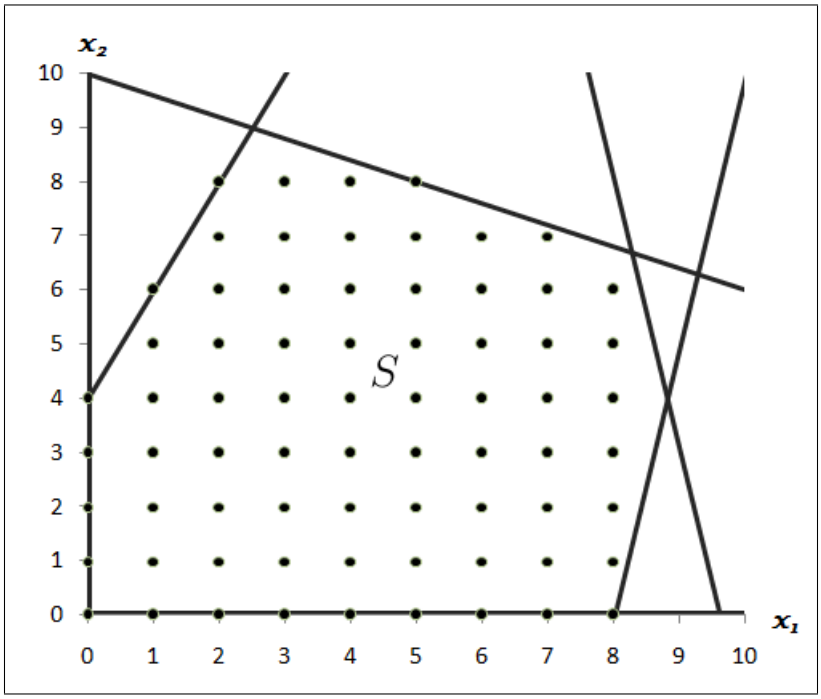


Figure 2.2. The Feasible Set S.

i.e., we consider a LPR of S, which, in this case, is Q. We have that

$$\bar{z}^* = \max\{-\tfrac{4}{3}x_1 + x_2 : x \in Q\} = \tfrac{17}{3}, \text{ with } x_1 = \tfrac{5}{2} \text{ and } x_2 = 9.$$

Since the solution is not integral and $2 < x_1 < 3$, we divide S into $S_1 = \{x \in S : x_1 \leq 2\}$ and $S_2 = \{x \in S : x_1 \geq 3\}$ (note that $S = S_1 \cup S_2$), and solve the following two problems:

$$\begin{array}{lll} \text{maximize} & z_1 = -\frac{4}{3}x_1 + x_2 & \\ \text{subject to} & x \in S_1 & \end{array} \quad \text{and} \quad \begin{array}{lll} \text{maximize} & z_2 = -\frac{4}{3}x_1 + x_2 & \\ \text{subject to} & x \in S_2 & \end{array}$$

Again, we want to apply the Simplex method, so instead of considering S_1 and S_2,

we solve the problems over $Q_1 = \{x \in Q : x_1 \leq 2\}$ and $Q_2 = \{x \in Q : x_1 \geq 3\}$. We get:

$$\bar{z_1}^* = \max\{-\tfrac{4}{3}x_1 + x_2 : x \in Q_1\} = \tfrac{16}{3}, \text{ with } x_1 = 2 \text{ and } x_2 = 8$$

and

$$\bar{z_2}^* = \max\{-\tfrac{4}{3}x_1 + x_2 : x \in Q_2\} = \tfrac{24}{5}, \text{ with } x_1 = 3 \text{ and } x_2 = \tfrac{44}{5}.$$

To keep track of the different optimization problems we are solving, let us build a tree, where each node will represent a problem. For this example, our tree is shown in Figure 2.3.

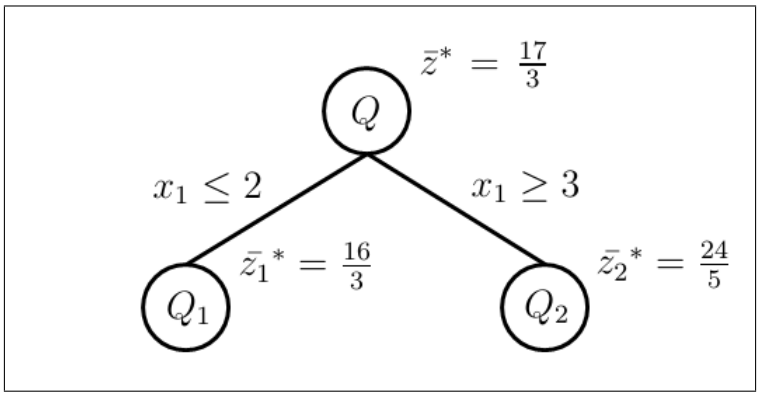


Figure 2.3. A Branch-and-Bound Tree.

The solution found at the root node gives us an upper bound on z, i.e., we can say that $z \leq \bar{z}^* = \frac{17}{3}$.

Solving the problem on the left-most node, we calculated $\bar{z_1}^* = \frac{16}{3}$, which is an upper bound for z_1. And because the solution at this particular node was integral

($x_1 = 2$ and $x_2 = 8$), we have in fact found an optimal solution not only over Q_1, but also over S_1, and, therefore, no further branching is necessary on this node. Moreover, we have now a candidate for an optimal solution to our original problem.

On the right-most node, we found the upper bound $\bar{z_2}^* = \frac{24}{5}$ for z_2. Since our solution was not integral ($x_1 = 3$ and $x_2 = \frac{44}{5}$), we may wish to branch on x_2 and create two more nodes. However, in every child node, the upper bound $\bar{z_2}^*$ will still hold, so even if we find solutions in S, their objective function values will be at most $\bar{z_2}^*$. Because $\bar{z_2}^* \leq \bar{z_1}^*$, we need not continue on this side of the tree. We have, therefore, found the optimal value of z, namely $z^* = \frac{16}{3}$.

The branch-and-bound algorithm terminates whenever we do not need to branch the tree anymore. The previous example shows two kinds of tree *pruning*. We pruned the left-most node of the tree by *integrality*. The right-most node was pruned by *bound*. The third possible type of pruning is by *infeasibility*, and that occurs when the feasible set of the node is empty.

2.4.2 Branch-and-Cut

Given a set S, its *convex hull*, denoted $conv(S)$, is the set of all points that are convex combinations of points of S. If S can be described by a system of linear inequalities together with integrality restrictions on some variables, then $conv(S)$ is a polyhedron, and the following property holds:

$$\max\left\{\sum_{i=1}^{n} c_i x_i : x \in S\right\} = \max\left\{\sum_{i=1}^{n} c_i x_i : x \in conv(S)\right\}.$$

This means that having the an inequality description of the convex hull of the

feasible set allows us to simply apply the Simplex algorithm and solve mixed integer problems very quickly. The problem of finding such a description, however, is not easy. In many cases, what we can do is approximate it via the addition of inequalities to the description of S.

Example 2.2.1 (continued) By looking at Figure 2.2, it is clear that the inequality $x_2 \leq 8$ is valid for S. If we had this inequality in the description of S to start with, i.e., if we were given that

$$S = \left\{ x \in Q \cap \mathbb{Z}^2 : x_2 \leq 8 \right\},$$

then solving problem (2.2) would have been an easier task. The reason is that solving the LP problem

$$\begin{aligned} \text{maximize} \quad & \bar{z} = -\tfrac{4}{3}x_1 + x_2 \\ \text{subject to} \quad & x \in Q', \end{aligned}$$

where $Q' = \{x \in Q : x_2 \leq 8\}$, via the Simplex method, would have given us $\bar{z}^* = \frac{16}{3}$, with $x_1 = 2$ and $x_2 = 8$, which is the same optimal solution we had calculated previously.

Figure 2.4 shows the set Q' and Figure 2.5 shows $conv(S)$. Notice how Q' is a better approximation of $conv(S)$ than is Q.

The inequality that we added to the formulation in the previous example "cut off" part of the LPR of our original feasible set and helped us solve the problem. Such inequality is often called a *cutting plane*, or simply a *cut*. The idea of *branch-and-cut*

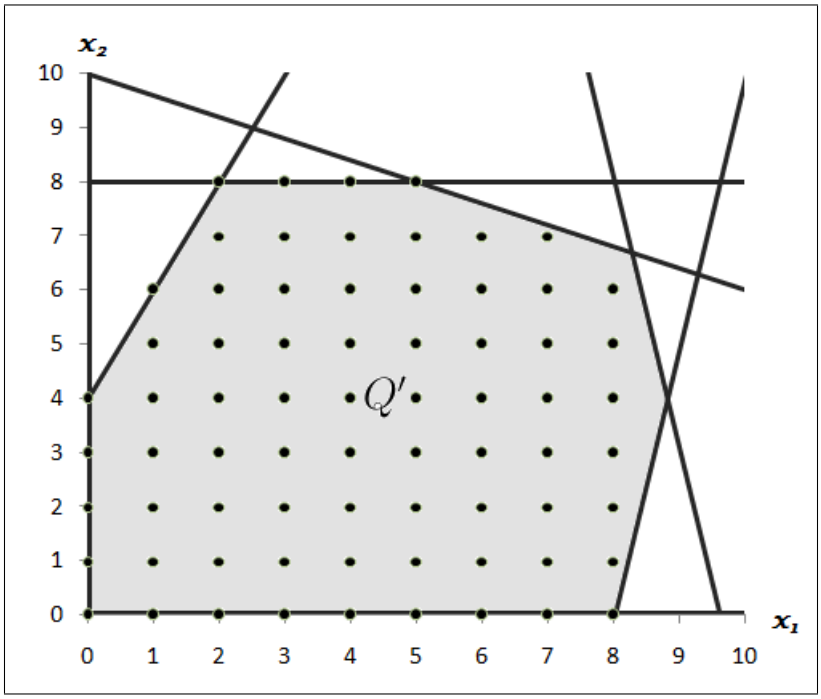


Figure 2.4. The Set Q'.

is to combine branch-and-bound with cutting planes.

Several "families" of cutting planes have been developed in the literature throughout the years (e.g. in [1, 2, 15, 21, 25, 27]). They take into account specific structures in the problem formulation. Here, we will study cutting planes developed for three kinds of constraints. Chapters 3, 4, and 5 treat each of them individually.

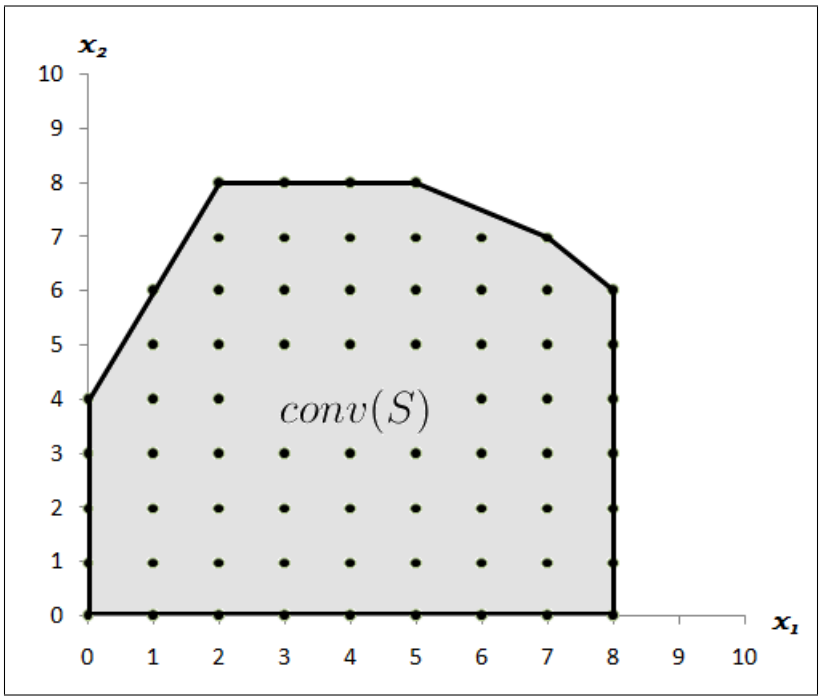


Figure 2.5. The Convex Hull of S.

CHAPTER 3
MULTIPLE CHOICE OPTIMIZATION PROBLEMS

A set of variables is said to be a *special ordered set of type 1 (SOS1)* if, in the problem formulation, at most one of the variables in the set can be non-zero. This type of constraint is also referred to as *multiple choice*, *complementarity*, or, in case all the variables in the set are binary, as a *generalized upper bound* (often abbreviated as GUB) constraint.

Multiple choice constraints occur, in practice, in a variety of situations. Here are some examples:

- In scheduling crews for flights at an airline company, we have to choose, for each flight, exactly one pilot and one co-pilot from a set of pilots and a set of co-pilots available.
- At a glass manufacturing line, at most one type of windshield can be produced at a time.
- While planning the design of a shopping mall, we may wish to have at most one big department store in each section of the mall.

Top commercial solvers have implemented in their algorithm a routine to account for SOS1 sets, i.e., the user can simply declare a set of variables to be an SOS1 and the solver will handle it automatically.

Despite its relatively simple statement, multiple choice constraints can be very fruitful in terms of mathematical results. This fact, allied to its importance, makes SOS1 constraints a very frequent topic studied in the literature.

The "usual" way of enforcing the multiple choice constraint, which has been used since the late 1950's, when problems with integer variables started to appear frequently in the literature (e.g. [11, 19]), is by adding binary variables and additional constraints to the formulation. Recently, Vielma and Nemhauser [24] proposed another formulation to model SOS1 constraints that uses a "logarithmic" number of binary variables and additional constraints, while the "usual" model requires a "linear" number of binary variables and additional constraints. Later we will elaborate on what "linear" and "logarithmic" mean in this context.

The term *special ordered set* was introduced in 1970, by Beale and Tomlin [3]. In their paper, they propose another approach to deal with SOS1 constraints, which dispenses with the use of binary variables. Their method focuses on *branching*, i.e., they enforce the SOS1 constraint via a specialized branching scheme.

Other works have focused on exploring the polyhedral structure of the multiple choice problem's solution set and deriving cutting planes to be used in branch-and-cut (see, for example, [12, 16, 17, 26, 28]). In this chapter, we will study and test computationally two families of cutting planes derived by de Farias el. al [12] using the above mentioned different methods to account for the SOS1 constraints.

3.1 Problem Statement

Let m be a positive integer and $M = \{1, \ldots, m\}$. For each $i \in M$, let n_i be a positive integer and $N_i = \{1, \ldots, n_i\}$. For simplicity, we denote as ij the ordered pair (i, j) and any set with one element by the element itself. We are interested in the *multiple choice optimization problem* (MCOP):

$$\text{maximize} \qquad \sum_{i \in M} \sum_{j \in N_i} c_{ij} x_{ij}$$

$$\text{subject to} \qquad Ax \leq b \tag{3.1}$$

$$\{x_{i1}, \ldots, x_{in_i}\} \text{ is SOS1 }, \quad i \in M \tag{3.2}$$

$$x_{ij} \in [0, u_{ij}], \qquad j \in N_i, i \in M. \tag{3.3}$$

We denote $d = \sum_{i \in M} n_i$, i.e., d is the number of variables. Also, we let $I = \cup_{i \in M} (i \times N_i)$, i.e., I is the set of indices of x. For $T \subseteq I$, we let $M_T = \{i \in M : ij \in T \text{ for some } j \in N_i\}$. We also assume, WLOG, that $u_{ij} = 1 \ \forall ij \in I$, which can be achieved by scaling the variables. Finally, if $r_j \in \mathbb{R}$, then $R = \emptyset \Rightarrow \sum_{j \in R} r_j = 0$.

3.2 Formulations to Model SOS1 Constraints

To model constraint (3.2), the "usual" approach is to introduce a binary variable y_{ij} for each continuous variable x_{ij}, plus upper bounds on the summations of the y_{ij}'s for every $i \in M$. More specifically, (3.2) is equivalent to

$$\begin{cases} x_{ij} \leq y_{ij}, & ij \in I \\ \sum_{j \in N_i} y_{ij} \leq 1, & i \in M \\ y_{ij} \in \{0, 1\}, & ij \in I. \end{cases} \tag{3.4}$$

Here, we have introduced d binary variables and $d + m$ extra constraints to the formulation of MCOP. Thus, the number of extra variables and extra constraints is "linear" in the number of variables.

The following alternative way to model (3.2), introduced by Vielma and Nemhauser [24], is "logarithmic" in the number of variables, using $\sum_{i\in M}\lceil \log_2 n_i\rceil$ binary variables and $2\sum_{i\in M}\lceil \log_2 n_i\rceil$ extra constraints.

For each $i \in M$ and each $k \in \{1,\dots,\lceil \log_2 n_i\rceil\}$, let

$$L_{ik} = \bigcup_{\ell=0}^{\lceil \log_2 n_i\rceil - k} \left\{ j \in N_i : \left| j - \left(\tfrac{1}{2} + 2^{k+1}\ell\right)\right| < 2^{k-1} \right\}.$$

Then the constraints

$$\begin{cases} \displaystyle\sum_{j\in L_{ik}} x_{ij} \le y_{ik}, & i \in M, k \in \{1,\dots,\lceil \log_2 n_i\rceil\} \\ \displaystyle\sum_{j\in N_i - L_{ik}} x_{ij} \le 1 - y_{ik}, & i \in M, k \in \{1,\dots,\lceil \log_2 n_i\rceil\} \\ y_{ik} \in \{0,1\}, & i \in M, k \in \{1,\dots,\lceil \log_2 n_i\rceil\} \end{cases} \tag{3.5}$$

are equivalent to (3.2). We will refer to (3.4) as the *MIP formulation* for SOS1, and to (3.5) as the *LOG formulation* for SOS1.

A third approach is to simply declare the sets of variables to be SOS1 and let the solver take care of it. Theoretically, the solver should have a specialized branching scheme to deal with the constraints (such as the one introduced by Beale and Tomlin [3]), but because we do not know how the solvers in fact work internally, we will not elaborate on it. We will, nevertheless, test this method computationally, and will call it the *SOS approach*.

We now illustrate both the MIP and the LOG formulations.

Example 3.2.1. Suppose we want to model the constraint

$$\{x_1, \ldots, x_8\} \text{ is } SOS1. \tag{3.6}$$

The MIP formulation for (3.6) is

$$\begin{cases} x_1 \leq y_1 \\ x_2 \leq y_2 \\ x_3 \leq y_3 \\ x_4 \leq y_4 \\ x_5 \leq y_5 \\ x_6 \leq y_6 \\ x_7 \leq y_7 \\ x_8 \leq y_8 \\ y_1 + y_2 + y_3 + y_4 + y_5 + y_6 + y_7 + y_8 \leq 1 \\ y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8 \in \{0, 1\}, \end{cases} \tag{3.7}$$

and the LOG formulation for (3.6) is

$$\begin{cases} x_1 + x_4 + x_5 + x_8 \leq y_1 \\ x_2 + x_3 + x_6 + x_7 \leq 1 - y_1 \\ x_1 + x_2 + x_7 + x_8 \leq y_2 \\ x_3 + x_4 + x_5 + x_6 \leq 1 - y_2 \\ x_1 + x_2 + x_3 + x_4 \leq y_3 \\ x_5 + x_6 + x_7 + x_8 \leq 1 - y_3 \\ y_1, y_2, y_3 \in \{0, 1\}. \end{cases} \tag{3.8}$$

In [24], a computational experiment indicated the superiority of LOG over other formulations for piecewise linear instances (to be discussed in the next chapter). However, we are not aware of a computational study of the LOG formulation applied to SOS1-constrained problems. One of the goals of this work is to fill this gap.

3.3 Cutting Planes for SOS1-Constrained Problems

As commented in the previous chapter, cutting planes, or simply *cuts*, can be useful to solve mixed integer programming problems. For this reason, commercial solvers have certain families of cuts implemented in their branch-and-cut routine (e.g. Gomory, cover, flow cover, and mixed integer rounding (MIR) cuts. See [20] for details).

Two families of cuts for SOS1-constrained problems were given by de Farias et al. [12], but not tested computationally. Here, we generalize those results and test them in branch-and-cut to solve difficult instances of MCOP. The strategy we adopt is based on that of Crowder et al. [10]. Specifically, we use as cuts inequalities valid for the convex hull of the set defined by one of the inequalities of (3.1), together with constraints (3.2) and (3.3). Let

$$\sum_{i \in M} \sum_{j \in N_i} a_{ij} x_{ij} \leq b \tag{3.9}$$

be one of the rows of the constraint matrix A. Our polyhedron of interest is $P = conv(S)$, where

$$S = \{x \in \mathbb{R}^d : x \text{ satisfies } (3.3), (3.9), \text{ and } (3.2)\}.$$

We note that, if an inequality is valid for P, then it is also valid for the feasible set of MCOP, since the latter is a subset of the former. The set S is often called the *knapsack relaxation* of the feasible set of MCOP. We assume that:

Assumption 3.3.1. $a_{i1} \geq a_{i2} \geq \cdots \geq a_{in_i}, \forall i \in M$.

Assumption 3.3.2. $n_i \geq 2$ *for some* $i \in M$.

Assumption 3.3.3. $\sum_{i \in M} a_{i1} > b$.

Assumption 3.3.4. $b > 0$ *and* $a_{ij} \geq 0 \ \forall ij \in I$.

Assumption 3.3.1 is WLOG, since it can be achieved by a simple relabeling of the variables (remember that we are considering only one row of A). If Assumptions 3.3.2 and 3.3.3 do not hold, the problem is trivial, so they are also WLOG. Although there is loss of generality in Assumption 3.3.4, the problem is still interesting and difficult under this assumption, as the computational experiments will show.

We now start the presentation of valid inequalities for P. Some of them, which are often called *trivial inequalities*, can be easily inferred from the problem. For instance, for $i \in M$, the inequality

$$\sum_{j \in N_i} x_{ij} \leq 1 \tag{3.10}$$

is obviously valid for P, which can be derived using constraints (3.2), (3.3), and our assumption that $u_{ij} = 1 \ \forall ij \in I$.

In order to use branch-and-cut, we need to work with an LPR of our feasible set S, i.e., a set defined only by continuous variables and linear constraints, and which

contains S. Here, the SOS1 constraint is the one which prevents our problem from being an LP, so our LPR must not contain it. Instead, we replace the SOS1 constraint with (3.10). We let

$$LPR(S) = \{x \in \mathbb{R}^d : x \text{ satisfies } (3.3), (3.9), \text{ and } (3.10)\}.$$

Just like (3.10), one can find other inequalities that are valid for S. However, if such inequalities are also valid for $LPR(S)$, they cannot be used as cuts. Hence, what we need are inequalities that are valid for S, but not for $LPR(S)$. We note that an inequality is valid for S if and only if it is valid for P. Thus, we may use S and P interchangeably when considering valid inequalities.

Another type of inequality that, in this case, could not be used as cuts, is a *cover inequality*. However, one can *strengthen* cover inequalities through a processes called *lifting* (see [20] for details). We now define cover inequality and give an example of lifting.

Definition 3.3.1. Let $C = \{i_1j_1, \ldots, i_kj_k\} \subseteq I$, with $i_1, \ldots, i_k$ all distinct. The set C is called a *cover* if $\sum_{ij \in C} a_{ij} > b$. Given a cover C, the inequality

$$\sum_{ij \in C} a_{ij} x_{ij} \leq b \tag{3.11}$$

is called a *cover inequality*.

It is easy to see that (3.11) is valid for P, since $a_{ij} \geq 0 \; \forall ij \in I$, $x_{ij} \in [0, 1] \; \forall ij \in I$, and $C \subset I$ (i.e., (3.11) is (3.9) without some terms on its left-hand side).

Example 3.3.1. Let $m = 5$, $n_1 = n_2 = n_3 = n_5 = 2$, $n_4 = 3$, and (3.9) be given by

$$(6x_{11} + x_{12}) + (2x_{21} + x_{22}) + (4x_{31} + 3x_{32}) + (8x_{41} + 6x_{42} + x_{43}) + (9x_{51} + 4x_{52}) \leq 13. \tag{3.12}$$

Then, $C = \{11, 21, 32, 42\}$ is a cover and

$$6x_{11} + 2x_{21} + 3x_{32} + 6x_{42} \leq 13 \tag{3.13}$$

is a cover inequality. Inequality (3.13) and constraints (3.2) imply that

$$(6x_{11} + 2x_{12}) + 2x_{21} + 3x_{32} + 6x_{42} \leq 13 \tag{3.14}$$

is valid for P, and the reason is the following: because $\{x_{11}, x_{12}\}$ is SOS1, either x_{11} or x_{12} can be positive, but not both. If $x_{11} > 0$, then $x_{12} = 0$, and (3.27) becomes (3.13), which we know is valid for P. If $x_{12} > 0$, then $x_{11} = 0$, and thus the maximum value that the sum $2x_{21} + 3x_{32} + 6x_{42}$ can achieve is 11. Therefore, there is "room" for another 2 on the left-hand side of (3.13), and hence we conclude that the *lifting coefficient* of x_{12} is 2. We say that (3.13) was *lifted with respect to* x_{12}.

Inequality (3.27) cuts off, for example, $x_{11} = \frac{1}{5}$, $x_{12} = \frac{4}{5}$, $x_{21} = x_{32} = x_{42} = 1$, and $x_{ij} = 0$ otherwise, which is a vertex of $LPR(S)$.

We now reproduce two families of lifted cover inequalities derived by de Farias et al. [12].

Theorem 3.3.1. (de Farias et al. [12]) *Let C be a cover, and suppose that $j = 1$ $\forall ij \in C$. Additionally, assume that $\exists i' \in M_C$ and $j' \in N_{i'}$ such that*

$$\sum_{i \in M_C - i'} a_{i1} + a_{i'j'} < b. \tag{3.15}$$

Then,

$$\sum_{i \in M_C} \sum_{j=1}^{n_i} \max \left\{ a_{ij}, b - \sum_{k \in M_C - i} a_{k1} \right\} x_{ij} \leq b \tag{3.16}$$

is valid and facet-defining for P.

Example 3.3.1 (continued) The set $C = \{41, 51\}$ is a cover that satisfies the assumptions of Theorem 3.3.1 with $i' = 4$, $j' = 3$ (or $i' = 5$, $j' = 2$). Thus

$$(8x_{41} + 6x_{42} + 4x_{43}) + (9x_{51} + 5x_{52}) \leq 13 \tag{3.17}$$

is valid and facet-defining for P. Inequality (3.19) cuts off, for example, $x_{41} = 1$, $x_{51} = \frac{1}{5}$, $x_{52} = \frac{4}{5}$, and $x_{ij} = 0$ otherwise, which is a vertex of $LPR(S)$.

Theorem 3.3.2. (de Farias et al. [12]) *Let C be a cover and $i'j' \in C$ with $j' < n_{i'}$. Suppose that $j = n_i \ \forall ij \in C - i'j'$. Finally, suppose that $\sum_{i \in M_C} a_{in_i} < b$. Then,*

$$\sum_{j \in N_{i'}} \max \left\{ a_{i'j}, b - \sum_{k \in M_C - i'} a_{kn_k} \right\} x_{i'j} +$$

$$\sum_{i \in M_C - i'} \sum_{j=1}^{n_i} a_{in_i} \max \left\{ 1, \frac{a_{ij}}{b - \sum_{k \in M_C - i} a_{kn_k}} \right\} x_{ij} \leq b \tag{3.18}$$

is valid and facet-defining for P.

Example 3.3.1 (continued) The set $C = \{22, 32, 43, 51\}$ is a cover that satisfies the assumptions of Theorem 3.3.2 with $i'j' = 51$. Thus,

$$(x_{21} + x_{22}) + (3x_{31} + 3x_{32}) + \left(\frac{8}{5}x_{41} + \frac{6}{5}x_{42} + x_{43}\right) + (9x_{51} + 8x_{52}) \leq 13 \quad (3.19)$$

is valid and facet-defining for P. Inequality (3.19) cuts off, for example, $x_{22} = x_{32} = x_{43} = 1$, $x_{51} = \frac{4}{5}$, $x_{52} = \frac{1}{5}$, and $x_{ij} = 0$ otherwise, which is a vertex of $LPR(S)$.

As mentioned earlier, these two families of cuts had not been tested computationally, and in the following sections we present computational results that indicate that they can be very useful in solving MCOP. The cuts used in our computational tests, however, are generalizations of (3.16) and (3.18), which we present in theorems 3.3.3 and 3.3.4. We start with two propositions.

Proposition 3.3.1. *Let $T \subset I$. Suppose that*

$$\sum_{ij \in T} \alpha_{ij} x_{ij} \leq \gamma \quad (3.20)$$

is valid for $\bar{P} = P \cap \left\{x \in \mathbb{R}^d : x_{ij} = 0 \ \forall ij \in I - T\right\}$. Then, (3.20) is valid for P.

Proof. Let $x^* \in S$, and $\bar{x} \in \mathbb{R}^d$ be given by

$$\bar{x}_{ij} = \begin{cases} x^*_{ij}, & \text{if } ij \in T \\ 0, & \text{otherwise.} \end{cases}$$

Clearly,

$$\sum_{ij\in T} \alpha_{ij} x^*_{ij} = \sum_{ij\in T} \alpha_{ij} \bar{x}_{ij}. \tag{3.21}$$

Because of Assumption 3.3.4, $\bar{x} \in \bar{P}$, and so it satisfies (3.20). It then follows from (3.21) that x^* satisfies (3.20). Therefore, (3.20) is valid for P. □

Proposition 3.3.2. *Let $\bar{M} \subseteq M$, and, for all $i \in \bar{M}$, let $r_i, t_i \in N_i$ with $r_i < t_i$. Suppose that*

$$\sum_{i\in\bar{M}} \sum_{j=r_i}^{t_i} \alpha_{ij} x_{ij} \leq b \tag{3.22}$$

is valid for P. Let $R = \left\{ i \in \bar{M} : \alpha_{ir_i} = a_{ir_i} \right\}$ and $T = \left\{ i \in \bar{M} : \alpha_{it_i} = a_{it_i} \right\}$. Then,

$$\sum_{i\in R} \sum_{j=1}^{r_i-1} a_{ir_i} x_{ij} + \sum_{i\in\bar{M}} \sum_{j=r_i}^{t_i} \alpha_{ij} x_{ij} + \sum_{i\in T} \sum_{j=t_i+1}^{n_i} a_{ij} x_{ij} \leq b \tag{3.23}$$

is valid for P.

Proof. Let $x^* \in S$, and $\bar{x} \in \mathbb{R}^d$ be given by

$$\bar{x}_{ij} = \begin{cases} x^*_{ij}, & \text{if } i \in \bar{M}, j \in \{r_i, \ldots, t_i\}, \text{ and } x^*_{ij} > 0 \\ x^*_{il}, & \text{if } i \in R, j = r_i, \text{ and } x^*_{il} > 0 \text{ for some } l \in \{1, \ldots, r_i - 1\} \\ \frac{a_{il}}{a_{it_i}} x^*_{il}, & \text{if } i \in T, j = t_i, \text{ and } x^*_{il} > 0 \text{ for some } l \in \{t_i + 1, \ldots, n_i\} \\ 0, & \text{otherwise.} \end{cases}$$

Clearly,

$$\sum_{i\in R}\sum_{j=1}^{r_i-1} a_{ir_i}x^*_{ij} + \sum_{i\in \bar{M}}\sum_{j=r_i}^{t_i} \alpha_{ij}x^*_{ij} + \sum_{i\in T}\sum_{j=t_i+1}^{n_i} a_{ij}x^*_{ij} = \sum_{i\in \bar{M}}\sum_{j=r_i}^{t_i} \alpha_{ij}\bar{x}_{ij}. \tag{3.24}$$

It is also easy to see that $\bar{x} \in S$. So, $\bar{x}$ satisfies (3.22). It then follows from (3.24) that x^* satisfies (3.23). Therefore, (3.23) is valid for P. □

We now give the first family of inequalities we will use in our computational tests.

Theorem 3.3.3. *Let C be a cover. Denote the elements of C as ir_i ($\forall i \in M_C$). Assume that $\sum_{i\in M_C - i'} a_{ir_i} + a_{i'j'} < b$ for some $i' \in M_C$ and $j' \in \{j \in N_{i'} : r_{i'} < j' \leq n_{i'}\}$. Then,*

$$\sum_{i\in M_C}\sum_{j=1}^{r_i-1} a_{ir_i}x_{ij} + \sum_{i\in M_C}\sum_{j=r_i}^{n_i} \max\left\{a_{ij}, b - \sum_{k\in M_C - i} a_{kr_k}\right\} x_{ij} \leq b \tag{3.25}$$

is valid for P. Inequality (3.25) is facet-defining when $r_i = 1\ \forall i \in M_C$.

Proof. By applying Theorem 3.3.1 we see that

$$\sum_{i\in M_C}\sum_{j=r_i}^{n_i} \max\left\{a_{ij}, b - \sum_{k\in M_C - i} a_{kr_k}\right\} x_{ij} \leq b \tag{3.26}$$

is valid for $P \cap \{x \in \mathbb{R}^d : x_{ij} = 0 \text{ when } i \in M_C \text{ and } j < r_i\}$. By applying Proposition 3.3.1, it follows that (3.26) is valid for P. Finally, we apply Proposition 3.3.2 with $\bar{M} = M_C$ and $t_i = n_i\ \forall i \in \bar{M}$ to obtain (3.25) (note that $R = \bar{M}$ and that T is irrelevant, since $t_i = n_i\ \forall i \in \bar{M}$, and consequently the third term of (3.23) is not present). That (3.25) is facet-defining when $r_i = 1\ \forall i \in M_C$ follows directly from Theorem 3.3.1. □

Theorem 3.3.3 generalizes Theorem 3.3.1 by allowing covers C with $j > 1$ for $ij \in C$.

Example 3.3.1 (continued) The set $C = \{42, 51\}$ is a cover that satisfies the assumptions of Theorem 3.3.3 with $i'j' = 43$ (or $i'j' = 52$). Thus,

$$(6x_{41} + 6x_{42} + 4x_{43}) + (9x_{51} + 7x_{52}) \leq 13 \tag{3.27}$$

is valid for P.

Inequality (3.27) is facet-defining for P and cuts off, for example, $x_{22} = x_{42} = 1$, $x_{51} = \frac{2}{5}$, $x_{52} = \frac{3}{5}$, and $x_{ij} = 0$ otherwise, which is a vertex of $LPR(S)$.

Finally, we give the second family of inequalities we will use in our computational tests.

Theorem 3.3.4. *Let C be a cover and $i'j' \in C$ with $j' < n_{i'}$. Denote the elements of C other than $i'j'$ as it_i ($\forall i \in M_C - i'$). Suppose that $\sum_{i \in M_C - i'} a_{it_i} + a_{i'n_{i'}} < b$. Then,*

$$\sum_{j \in N_{i'}} \max \left\{ a_{i'j}, b - \sum_{k \in M_C - i'} a_{kt_k} \right\} x_{i'j} +$$

$$\sum_{i \in M_C - i'} \left(\sum_{j=1}^{t_i} a_{it_i} \max \left\{ 1, \frac{a_{ij}}{b - \sum_{k \in M_C - \{i, i'\}} a_{kt_k} - a_{i'n_{i'}}} \right\} x_{ij} + \sum_{j=t_i+1}^{n_i} a_{ij} x_{ij} \right) \leq b \tag{3.28}$$

is valid for P. Inequality (3.28) is facet-defining when $t_i = n_i$ $\forall i \in M_C - i'$.

Proof. By applying Theorem 3.3.2 we see that

$$\sum_{j \in N_{i'}} \max \left\{ a_{i'j}, b - \sum_{k \in M_C - i'} a_{kt_k} \right\} x_{i'j} +$$

$$\sum_{i \in M_C - i'} \left(\sum_{j=1}^{t_i} a_{it_i} \max \left\{ 1, \frac{a_{ij}}{b - \sum_{k \in M_C - \{i,i'\}} a_{kt_k} - a_{i'n_{i'}}} \right\} x_{ij} \right) \leq b \tag{3.29}$$

is valid for $P \cap \left\{ x \in \mathbb{R}^d : x_{ij} = 0 \text{ when } i \in M_C - i \text{ and } j > t_i \right\}$. By applying Proposition 3.3.1, it follows that (3.29) is valid for P. Finally, we apply Proposition 3.3.2 with $\bar{M} = M_C - i'$ and $r_i = 1 \ \forall i \in \bar{M}$ to obtain (3.28) (note that $T = \bar{M}$ and that R is irrelevant, since $r_i = 1 \ \forall i \in \bar{M}$, and consequently the first term of (3.23) is not present). That (3.28) is facet-defining when $t_i = n_i \ \forall i \in M_C - i'$ follows directly from Theorem 3.3.2. □

Theorem 3.3.4 generalizes Theorem 3.3.2 by allowing covers C with $j < n_i$ for $ij \in C - i'j'$.

Example 3.3.1 (continued) The set $C = \{21, 31, 43, 51\}$ is a cover that satisfies the assumptions of Theorem 3.3.4 with $i'j' = 51$. Thus,

$$(2x_{21} + x_{22}) + (4x_{31} + 3x_{32}) + \left(\frac{8}{3} x_{41} + 2x_{42} + x_{43} \right) + (9x_{51} + 6x_{52}) \leq 13 \tag{3.30}$$

is valid for P.

Inequality (3.30) is facet-defining for P and cuts off, for example, $x_{31} = x_{43} = 1$, $x_{51} = \frac{4}{5}$, $x_{52} = \frac{1}{5}$, and $x_{ij} = 0$ otherwise, which is a vertex of $LPR(S)$.

For the remainder of this dissertation, we will refer to inequalities (3.25) and (3.28) as *SOS1 cuts*.

3.4 Separation of SOS1 Cuts

We now describe our procedure to generate inequalities (3.25) and (3.28). The term for this is *separation* of inequalities. Separation of valid inequalities is an area of study on its own, given its importance and difficulty (e.g., separation of 0-1 cover inequalities is NP-Complete [20]). Here, we use simple heuristics to derive SOS1 cuts, and we do it at every node of the branch-and-cut tree.

Let x^* be the optimal solution to the LPR of the current branch-and-cut node. Suppose that x^* does not satisfy (3.2) for some SOS1 (it it does, then x^* is an optimal solution for our instance already). For each knapsack constraint within $Ax \leq \beta$, we attempt to find an inequality (3.25) and an inequality (3.28) to cut off x^*. To simplify the presentation, we assume that the entries of the row of A being considered satisfy Assumption 3.3.1 (in practice, they need to be sorted beforehand). We also omit the index of the row.

Let $M_1 = \{i \in M : a_{ij}x_{ij} > 0 \text{ for some } j \in N_i\}$ and $M_2 = \{i \in M : x_{ij} > 0 \text{ for some } j \in N_i\}$. Suppose that $M_1 \neq \emptyset$ and $M_2 \neq \emptyset$. For each $i \in M_1$ let

$$r_i = \min\{j \in N_i : a_{ij}x_{ij} > 0\},$$

and for each $i \in M_2$ let

$$t_i = \max\{j \in N_i : x_{ij} > 0\}.$$

Inequality (3.25) We form the set $C = \{ir_i : i \in M_1\}$. If $\sum_{ij \in C} a_{ij} > b$, C is a cover, and we continue; otherwise we quit. We then search, in increasing order of $i \in M_C$ and $j \in \{r_i, \ldots, n_i\}$, a pair $i'j'$ satisfying $\sum_{i \in M_C - i'} a_{ir_i} + a_{i'j'} < b$. If none is found, we quit. Otherwise, for the first pair found we form an inequality (3.25). If x^* violates the resulting inequality (3.25) by at least 0.01 (the difference between the left-hand-side and the right-hand-side of (3.25)), we add the inequality to CPLEX's cut pool. Otherwise, we quit. We separate at most 1,000 inequalities (3.25).

Inequality (3.28) We first test whether $\sum_{i \in M_2} a_{it_i} > b$. If no, we quit. If yes, let $s \in M$ be such that

$$\sum_{\substack{i \in M_2 \\ i < s}} a_{it_i} < b \quad \text{and} \quad \sum_{\substack{i \in M_2 \\ i \leq s}} a_{it_i} \geq b.$$

We search, in increasing order of $i \in \{k \in M_2 : k < s\}$ and $j \in \{1, \ldots, t_i\}$, a pair $i'j'$ satisfying

$$\sum_{\substack{i \in M_2 - i' \\ i < s}} a_{it_i} + a_{i'j'} > b.$$

If none is found, we quit. Otherwise, for the first pair found we form the set $C = \{it_i : i \in M_2 - i',\ i < s\} \cup i'j'$, which is a cover. We then apply Theorem 3.3.4 to derive an inequality (3.28). As before, if x^* violates the resulting inequality (3.25) by at least 0.01, we add the inequality to the cut pool. Otherwise, we quit. We separate at most 1,000 inequalities (3.28).

3.5 Computational Experience

In this section, we describe the instances we used in our computational tests, as well as the computer platform and how the tests were conducted.

3.5.1 Instances

Let ℓ denote the number of rows of A, and let $n_i = n, \forall i \in M$, where n is a positive integer (in other words, the number of elements of every SOS1 of an instance is the same). Finally, we denote $N = \{1, \ldots, n\}$.

We generated 5 instances for each combination of ℓ, m, and n, with $\ell \in \{120, 160\}$, $m \in \{60, 80, 100\}$, and $n \in \{5, 10, 15, 20\}$. To enforce the SOS1 constraint, we used either the LOG formulation, the MIP formulation, or the SOS approach (which, henceforth, we may call simply MIP, LOG, and SOS). Thus, the total number of MCOP instances created is 120 with each formulation, for a total of 360.

The instances were generated as follows. Let $v_{ij} \sim \text{Unif}\{1, \ldots, 4\}$, $i \in M$, $j \in N$. The objective function coefficients were generated as:

$$c_{ij} = \begin{cases} v_{in}, & \text{if } j = n \\ c_{i(j+1)} + v_{ij}, & \text{otherwise.} \end{cases}$$

We now describe how the entries of each row of A were generated. Let $D_i \sim \text{Unif}[0, 1]$, $i \in M$, and $d_{ij} \sim \text{Unif}[0, 1]$, $i \in M$, $j \in N$. Let $w_{in} \sim \text{Unif}\{1, \ldots, 3\}$, $i \in M$, and $w_{ij} \sim \text{Unif}\{1, \ldots, 5\}$, $i \in M$, $j \in N - n$. Finally, let

$$z_{ij} = \begin{cases} w_{ij}, & \text{if } j = n \\ w_{ij} + \left\lceil \dfrac{c_{ij} a_{i(j+1)}}{c_{i(j+1)}} \right\rceil, & \text{otherwise.} \end{cases}$$

We set

$$a_{ij} = \begin{cases} 0, & \text{if } D_i \geq \Delta \text{ or } d_{ij} \geq \delta \\ z_{ij}, & \text{otherwise,} \end{cases}$$

and

$$b = \max \left\{ 1 + \max\{a_{ij} : i \in M, j \in N\}, \left\lfloor 1.3 \cdot \sum_{i \in M} a_{in} \right\rfloor \right\}.$$

The instances generated can be interpreted as a production planning problem, in which an SOS1 represents a machine (or a production line) where no more than one product can be manufactured at the same time, and we wish to allocate, in a given time window, products to machines, with the objective of maximizing total profit.

Specifically, m is the number of machines in a manufacturing plant, and n_i is the number of different products that can be produced in machine i, for all $i \in M$. For every $ij \in I$, x_{ij} is the amount of product j manufactured in line i.

Every row of the constraint matrix A represents a restriction on a type of resource, with b as the total amount of the resource available, and a_{ij}, $ij \in I$, the amount of the resource used per unit (or batch) of product x_{ij} manufactured. (We note that the constraint matrix has ℓ rows with different b's and a_{ij}'s in each one of them. For simplicity, and be consistent with the rest of the text, we omit the extra index here.) Finally, c_{ij} is the unit profit associated with x_{ij}, $ij \in I$.

3.5.2 Tests Conducted

We performed our computational tests in the Texas Tech High Performance Computing Center's Intel Xeon E5450 3.0GHz CPU with 16GB RAM nodes. We used the callable libraries of CPLEX 12.2.0.0, which we ran on a single thread. We allowed a maximum computational time of 3,600 seconds of CPU for solving each instance.

We evaluated CPLEX's performance when solving MCOP to proven optimality. We compared the performance that results when:

- the LOG formulation, MIP formulation, or SOS approach is used
- CPLEX's internal features are turned ON or OFF
- SOS1 cuts (i.e., inequalities (3.25) and (3.28)) are used or not

Nowadays, CPLEX has a large number of parameters that can be tuned in order to better handle different kinds of problems. The features that we believe can have the biggest impact on the solution time are: *Presolve*, *Heuristics*, and *MIP cuts*. Thus, in our tests, we turn these parameters ON and OFF.

In short, Presolve runs a set of procedures before the branch-and-cut process starts with the objective of reducing the number of variables and strengthening constraints, among others (see [20] for details). Heuristics is run throughout the solution process and aims at finding feasible solutions in order to prune nodes by bound. MIP cuts are "standard" cutting planes, such as Gomory, cover, and clique inequalities, which are internally derived and added to the current node being solved in the branch-and-cut tree. Finally, the addition of SOS1 cuts is performed via callable libraries and, in our case, was implemented using C. Like the MIP cuts, the SOS1 cuts are also added to the current node being solved.

So, given an instance, we have $2^4 = 16$ different *strategies* for solving it, namely with Presolve ON or OFF, Heuristics ON or OFF, MIP cuts ON or OFF, and SOS1 cuts ON or OFF. To simplify our presentation, when considering a strategy, we will use, in this order, the letter `p` to denote Presolve ON, `h` to denote Heuristics ON, `m` to denote MIP cuts ON, and `s` to denote SOS1 cuts ON. To denote a feature being OFF, we will use the letter `n`. For example, strategy `phms` has everything ON, while strategy `pnhn` has Presolve and Heuristics ON, but MIP cuts and SOS1 cuts OFF. Table 3.1 describes all possible strategies used.

Table 3.1. Features and Strategies

Strategy	Presolve	Heuristics	MIP cuts	SOS1 cuts
`phms`	ON	ON	ON	ON
`phmn`	ON	ON	ON	OFF
`phns`	ON	ON	OFF	ON
`phnn`	ON	ON	OFF	OFF
`pnms`	ON	OFF	ON	ON
`pnmn`	ON	OFF	ON	OFF
`pnns`	ON	OFF	OFF	ON
`pnnn`	ON	OFF	OFF	OFF
`nhms`	OFF	ON	ON	ON
`nhmn`	OFF	ON	ON	OFF
`nhns`	OFF	ON	OFF	ON
`nhnn`	OFF	ON	OFF	OFF
`nnms`	OFF	OFF	ON	ON
`nnmn`	OFF	OFF	ON	OFF
`nnns`	OFF	OFF	OFF	ON
`nnnn`	OFF	OFF	OFF	OFF

Given an MCOP instance, we will say that it was *solved* when it was solved to proven optimality within our prescribed time limit, and we consider a strategy to

be *more efficient* than another when the computational time required to solve the instance was smaller for that strategy.

Finally, we note that CPLEX has two search strategies, namely *Traditional Branch-and-Cut* and *Dynamic Search*. Of the two, only Traditional Branch-and-Cut allows the user to add their own cutting planes, so we only tested this option. Once this strategy is chosen, CPLEX's default setting have Presolve, Heuristics, and MIP cuts are all turned ON, and thus is represented by the strategy `phmn`.

3.6 Analysis of Results

We now report and analyze the results of our computational tests. For different formulations and strategies, we consider four outputs: number of instances solved, solution time, number of nodes used in the branch-and-cut tree, and number of cuts applied.

3.6.1 Number of Instances Solved

We start our analysis with the *number of instances solved* using each strategy, separated by type of formulation (MIP, LOG, or SOS). The results are shown in Table 3.2.

The number of tests run with each formulation was $120 \cdot 16 = 1920$. Thus, we have that 77% of the instances were solved with the MIP formulation, 74% were solved with the LOG formulation, and only 36% were solved with the SOS approach. In particular, Table 3.2 indicates that having Presolve ON with the SOS approach is extremely inefficient. If we consider only the tests in which Presolve was OFF, then the fraction of instances solved with SOS was 70%.

Table 3.2. Number of Instances Solved by Formulation and Strategy

Strategy	MIP	LOG	SOS
phms	116	115	9
phmn	70	66	0
phns	116	115	4
phnn	57	61	0
pnms	116	115	9
pnmn	71	66	0
pnns	116	115	4
pnnn	56	61	0
nhms	116	115	113
nhmn	85	67	55
nhns	116	115	113
nhnn	62	59	55
nnms	116	115	113
nnmn	85	67	54
nnns	116	114	113
nnnn	62	59	52
Total	1476	1425	694

For the MIP and the LOG formulation, having Presolve ON did not really affect the number of instances solved when SOS1 cuts were also ON, and it affected negatively the number of instances solved when SOS1 cuts were OFF. This effect, however, was minor in comparison with the use of MIP cuts and SOS1 cuts. The use of Heuristics did not have a significant impact on the ability to solve our instances with any of the formulations considered.

The results indicate that the use of MIP cuts had a positive impact on the number of instances solved. For MIP, the eight strategies that use MIP cuts solved our instances in 775, or 81%, of the tests considered (against 701, or 73%, without MIP cuts), and, for LOG, this number was 726, or 76% (against 699, or 73%, without

MIP cuts). For the SOS approach, the four strategies that had MIP cuts ON and Presolve OFF solved our instances in 335, or 70%, of the tests considered (against 333, or 69%, without MIP cuts).

The use of SOS1 cuts also had a positive (and more substantial) impact on the number of instances solved for all formulations. For MIP, the eight strategies that use SOS1 cuts solved our instances in 928, or 97%, of the tests considered (against 548, or 57%, without SOS1 cuts), and, for LOG, this number was 919, or 96% (against 506, or 53%, without SOS1 cuts). For the SOS approach, the four strategies that had SOS1 cuts ON and Presolve OFF solved our instances in 452, or 94%, of the tests considered (against 215, or 45%, without SOS1 cuts).

Overall, the best formulation in terms of number of instances solved was MIP, followed by LOG, followed by SOS. In particular, the combination of the MIP formulation with the use of SOS1 cuts was the one which solved the biggest number of instances ($116/120 = 97\%$), regardless of how other parameters were set.

Finally, in a comparison of the strategy `phms` with CPLEX's default settings (i.e., `phmn`), the use of SOS1 cuts increased the number of instances solved from 70 to 116 for MIP, from 66 to 115 for LOG, and from 0 to 9 for SOS.

3.6.2 Solution Time

We now turn our attention to the *solution time* of the instances. The summarized results are shown in Table 3.3. We note that, in this table, the times shown are averages that take into account only the instances solved (and shown in Table 3.2).

Overall, MIP and LOG had close performances, and SOS was worse. And because SOS was worse for every strategy, we will concentrate our analysis of solution time

Table 3.3. Average Solution Times (s) and Strategies

Strategy	MIP	LOG	SOS
phms	85	83	446
phmn	274	275	—
phns	78	81	938
phnn	451	346	—
pnms	85	77	288
pnmn	324	250	—
pnns	83	73	1047
pnnn	431	347	—
nhms	68	95	104
nhmn	189	232	573
nhns	89	97	108
nhnn	253	316	573
nnms	70	103	115
nnmn	204	230	485
nnns	85	77	110
nnnn	231	332	431
Total	156	158	253

on the other two formulations.

To do this, it is important that we use the *same instances* when comparing their solution times. For example, suppose we want to compare the strategies `nnmn` vs. `nnnn`, with the MIP formulation. Although the average times shown in Table 3.3 are somewhat close (204 vs. 231), the number for `nnmn` is an average for 85 instances, while the number for `nnnn` is an average for 62 instances. Thus the two averages are not comparable, and so we need to identify the instances that were solved by both strategies before inferring which one was faster when using MIP.

In the next pages we will consider the use of each of the three CPLEX features, namely Presolve, Heuristics, and MIP cuts, plus the use of SOS1 cuts, and will see

how they affected the solution time of our instances. Because the instances considered in this analysis may vary according to each formulation, the comparison of the times with and without a certain feature will be analyzed *within* a formulation (MIP or LOG). Only in Section 3.6.2.5 will we make a comparison across formulations, i.e., LOG vs. SOS.

3.6.2.1 Presolve

Table 3.4 compares the average solution time of instances when they are solved with and without *Presolve*, having everything else constant. The column "# Inst." shows the number of instances considered in the comparison, and values in **bold** indicate the better strategy.

Table 3.4. Strategies and Average Solution Times (s) with and without Presolve

Strategies	MIP				LOG			
	# Inst.	Avg. Time			# Inst.	Avg. Time		
`phms` vs. `nhms`	116	85	vs.	**68**	115	**83**	vs.	95
`phmn` vs. `nhmn`	70	274	vs.	**43**	65	230	vs.	**186**
`phns` vs. `nhns`	116	**78**	vs.	89	115	**81**	vs.	97
`phnn` vs. `nhnn`	57	451	vs.	**129**	59	**250**	vs.	316
`pnms` vs. `nnms`	116	85	vs.	**70**	115	**77**	vs.	103
`pnmn` vs. `nnmn`	71	324	vs.	**45**	66	250	vs.	**203**
`pnns` vs. `nnns`	116	**83**	vs.	85	114	**60**	vs.	77
`pnnn` vs. `nnnn`	56	431	vs.	**127**	59	**252**	vs.	332

The use of Presolve was helpful with the LOG formulation, but degraded the solution times with the MIP formulation. (Recall that Presolve had a very negative effect with SOS, too.) We do not know the reason for this. Nevertheless, for all formulations, the impact of Presolve was substantial in most cases. We note that,

for MIP, the impact of Presolve was greater when SOS1 cuts were not used, and it was minor when SOS1 cuts were used.

3.6.2.2 Heuristics

Table 3.5 compares the average solution time of instances when they are solved with and without *Heuristics*, having everything else constant.

Table 3.5. Strategies and Average Solution Times (s) with and without Heuristics

Strategies	MIP				LOG			
	# Inst.	Avg. Time			# Inst.	Avg. Time		
`phms` vs. `pnms`	116	85	vs.	85	115	83	vs.	**77**
`phmn` vs. `pnmn`	70	**274**	vs.	285	65	230	vs.	**227**
`phns` vs. `pnns`	116	**78**	vs.	83	115	81	vs.	**73**
`phnn` vs. `pnnn`	56	**411**	vs.	431	61	**346**	vs.	347
`nhms` vs. `nnms`	116	**68**	vs.	70	115	**95**	vs.	103
`nhmn` vs. `nnmn`	85	**189**	vs.	204	67	232	vs.	**230**
`nhns` vs. `nnns`	116	89	vs.	**85**	114	77	vs.	77
`nhnn` vs. `nnnn`	62	253	vs.	**231**	59	**316**	vs.	332

Because the disparity between every pair of values is small, we conclude that the use of Heuristics did not have a significant effect on the solution time of our instances.

3.6.2.3 MIP Cuts

Table 3.6 compares the average solution time of instances when they are solved with and without *MIP cuts*, having everything else constant.

Here, we notice a major influence of the parameter MIP cuts on the average solution time for both MIP and LOG, in particular when SOS1 cuts are OFF. In fact, in the few cases when the use of MIP cuts did not improve the solution time, SOS1

Table 3.6. Strategies and Average Solution Times (s) with and without MIP Cuts

Strategies	MIP			LOG		
	# Inst.	Avg. Time		# Inst.	Avg. Time	
`phms` vs. `phns`	116	85 vs.	**78**	115	83 vs.	**81**
`phmn` vs. `phnn`	57	**56** vs.	451	65	**101** vs.	303
`pnms` vs. `pnns`	116	85 vs.	**83**	115	77 vs.	**73**
`pnmn` vs. `pnnn`	56	**56** vs.	431	60	**105** vs.	305
`nhms` vs. `nhns`	116	**68** vs.	89	115	**95** vs.	97
`nhmn` vs. `nhnn`	62	**27** vs.	253	59	**78** vs.	316
`nnms` vs. `nnns`	116	**70** vs.	85	114	90 vs.	**77**
`nnmn` vs. `nnnn`	62	**24** vs.	231	59	**73** vs.	332

cuts were ON, and the difference between the two values shown was not as big as when we had SOS1 cuts OFF. This indicates that, first: the use of cutting planes was very important in the solution or our instances; and second: SOS1 cuts play a similar role to MIP cuts, and, perhaps, can even be used in place of MIP cuts, giving similar or better results. We will look further into this conjecture next, when we compare instances solved with MIP cuts, and with and without SOS1 cuts.

3.6.2.4 SOS1 Cuts

Table 3.7 compares the average solution time of instances when they are solved with and without *SOS1 cuts*, having everything else constant.

Here, we see a massive effect due to the use of SOS1 cuts, in every single comparison shown in the table. These results show that, for our instances, the use of SOS1 cuts was indispensable for solving them quickly. As suspected, SOS1 cuts overwhelmed MIP cuts, which suggests that, perhaps, CPLEX does not currently have internal cutting planes that can handle well instances with continuous variables and SOS1

Table 3.7. Strategies and Average Solution Times (s) with and without SOS1 Cuts

Strategies	MIP		LOG	
	# Inst.	Avg. Time	# Inst.	Avg. Time
`phms` vs. `phmn`	70	**3** vs. 274	66	**2** vs. 275
`phns` vs. `phnn`	57	**1** vs. 451	61	**2** vs. 346
`pnms` vs. `pnmn`	71	**1** vs. 324	66	**2** vs. 250
`pnns` vs. `pnnn`	56	**1** vs. 431	61	**1** vs. 347
`nhms` vs. `nhmn`	85	**5** vs. 189	67	**2** vs. 232
`nhns` vs. `nhnn`	62	**2** vs. 253	59	**1** vs. 316
`nnms` vs. `nnmn`	85	**5** vs. 204	67	**2** vs. 230
`nnns` vs. `nnnn`	62	**2** vs. 231	59	**1** vs. 332

constraints.

3.6.2.5 MIP vs. LOG

We now compare MIP vs. LOG. In Table 3.8, we show the average solution times of the 56 instances that were solved with all strategies, using both MIP and LOG.

The tables shows, as expected, that the strategies that use SOS1 cuts performed much better than the ones without those cuts. And because the numbers are too small, we will focus, for now, on the strategies that have SOS1 cuts OFF. For those strategies, MIP had a significantly better performance with 6 out of 8 strategies, and it was worse with `phnn` and `pnnn`. If one were to use CPLEX to solve our instances *without* SOS1 cuts, the best approach would be to use the MIP formulation and a strategy which has Presolve OFF and MIP cuts ON, with Heuristics ON or OFF being irrelevant.

Finally, we compare MIP vs. LOG *with* the use of SOS1 cuts. To do so, we consider all 114 instances that were solved using all strategies that have SOS1 cuts ON, using

Table 3.8. Strategies and Average Solution Times (s) for Instances Solved by All Strategies with Both MIP and LOG

Strategy	MIP	LOG
phms	1	1
phmn	**53**	72
phns	1	1
phnn	411	**217**
pnms	1	1
pnmn	**56**	71
pnns	1	1
pnnn	431	**209**
nhms	2	2
nhmn	**13**	68
nhns	2	**1**
nhnn	**126**	265
nnms	1	1
nnmn	**13**	63
nnns	1	1
nnnn	**127**	269

both MIP and LOG. The results are shown in Table 3.9.

Table 3.9. Strategies and Average Solution Times (s) for Instances Solved by All Strategies Using SOS1 Cuts with Both MIP and LOG

Strategy	MIP	LOG
phms	**64**	72
phns	**62**	69
pnms	**65**	68
pnns	65	**60**
nhms	**52**	84
nhns	**68**	77
nnms	**52**	90
nnns	**64**	77

MIP had a better performance in 7 out of 8 strategies considered, and it was worse only with `pnns`. We conclude that if one were to use CPLEX to solve our instances *with* SOS1 cuts, the best strategies would be to have Presolve OFF and MIP cuts ON, with Heuristics ON or OFF being irrelevant.

3.6.3 Number of Nodes

We now analyze the effect of SOS1 cuts and the other CPLEX features considered in the previous sections on the number of nodes in the branch-and-cut tree necessary to solve our instances. Because this is a secondary measure of efficiency, we consider only the 56 instances used to build Table 3.8, which are the ones that were solved with all strategies, using both MIP and LOG.

It is easy to see that there is a correlation between the values shown in tables 3.8 and 3.10. This is logical, since a tree with more nodes will likely take longer to solve. This behavior has been observed in other works that report computational results, such as [4] and [10].

Notice that the number of nodes with strategies that use SOS1 cuts is a few orders of magnitude smaller than strategies without SOS1 cuts. Also, the table shows that, when SOS1 cuts are on, the difference between the number of nodes for MIP vs. LOG is very small.

3.6.4 Types of Cuts Used

We now report the types and average number of cutting planes used by CPLEX in our tests. For SOS1 cuts, recall that we set a limit of 1,000 cuts (3.25) and 1,000 cuts (3.28) to be given to CPLEX. For the internal CPLEX cuts, no limit was previously set. Again, we consider the 56 instances that were solved by all strategies and with

Table 3.10. Strategies and Average Number of Nodes for Instances Solved by All Strategies with Both MIP and LOG

Strategy	MIP	LOG
phms	292	299
phmn	108,577	137,492
phns	339	322
phnn	1,431,665	742,710
pnms	327	307
pnmn	125,047	150,035
pnns	387	345
pnnn	1,634,455	775,351
nhms	207	301
nhmn	19,977	101,120
nhns	291	346
nhnn	348,506	733,733
nnms	224	317
nnmn	21,144	102,746
nnns	299	367
nnnn	383,080	802,315

both MIP and LOG. The results for MIP are shown in Table 3.11, and for LOG in Table 3.12.

In those tables, the column "Used" shows the number of SOS1 cuts that were actually used by CPLEX. This number may not be equal to the sum of the columns (3.25) and (3.28) because, for example, our separation algorithm may generate identical cuts, in which case at most one cut is counted as used. Another reason for this is the fact that CPLEX has a cut filter that may reject a cut if, for instance, it considers the cut too dense, or less effective than other cuts [9]. We do not have information on exactly how many cuts (3.25) and (3.28) were individually used, but only the combined total.

Table 3.11. Strategies and Average Number of MIP Cuts Using the MIP Formulation for Instances Solved by All Strategies with Both MIP and LOG

Strategy	Flow Cover	Gomory	Implied Bound	(3.25)	(3.28)	Used
phms	25	8	0	95	194	233
phmn	42	21	6	—	—	—
phns	—	—	—	100	208	263
pnms	25	8	0	100	208	253
pnmn	42	21	6	—	—	—
pnns	—	—	—	103	216	275
nhms	26	14	0	91	164	233
nhmn	37	21	1	—	—	—
nhns	—	—	—	103	190	249
nnms	26	14	0	96	176	217
nnmn	37	22	1	—	—	—
nnns	—	—	—	107	200	262

Table 3.12. Strategies and Average Number of MIP Cuts Using the LOG Formulation for Instances Solved by All Strategies with Both MIP and LOG

Strategy	Flow Cover	Gomory	Implied Bound	(3.25)	(3.28)	Used
phms	18	9	0	106	205	252
phmn	37	25	3	—	—	—
phns	—	—	—	105	206	266
pnms	18	9	0	108	209	258
pnmn	37	25	3	—	—	—
pnns	—	—	—	107	212	274
nhms	24	9	0	102	198	244
nhmn	41	20	1	—	—	—
nhns	—	—	—	105	208	269
nnms	23	9	0	103	200	246
nnmn	42	20	1	—	—	—
nnns	—	—	—	108	214	277

We see that the corresponding values of tables 3.11 and 3.12 are close, giving evidence that the number of cuts generated did not vary much depending on the formulation used (MIP or LOG). Thus, unless explicitly noted, we make comments that apply to both tables.

It is clear that the number of SOS1 cuts used is much greater than those of other types, when MIP cuts is ON. Moreover, the ratio of used cuts over generated cuts is around 84%. These facts suggest that the SOS1 cuts are strong and useful, at least according to CPLEX's filter.

In a comparison between strategies with SOS1 cuts ON vs. SOS1 cuts OFF (e.g. `phms` vs. `phmn`), we notice that the number of MIP cuts is smaller when SOS1 cuts are ON. This suggests that some SOS1 cuts "replace" some of the MIP cuts that would be used if SOS1 cuts were not derived. On the other hand, in a comparison between strategies with MIP cuts ON vs. MIP cuts OFF (e.g. `phms` vs. `phns`), we do not see a significant variation on the number of SOS1 cuts, suggesting that MIP cuts do not "replace" SOS1 cuts.

Regarding SOS1 cuts individually, the number of cuts (3.25) is consistently about half as much the number of cuts (3.28). We believe the reason for this is our separation procedures, which tries harder to separate cuts of the second type. We do not know which one of the two SOS1 cuts is stronger or more useful.

Presolve has a small effect in the number of cuts generated. For MIP, turning it ON decreased the number of Flow Cover, Implied Bound, and (3.28) cuts, increased the number of Gomory cuts, and did not affect the number of (3.25) cuts. For LOG, turning it ON decreased the number of Implied Bound cuts, increased the number of Flow Cover and Gomory cuts, and did not affect the number of SOS1 cuts.

Finally, turning Heuristics ON caused a small increase in the number of (3.28) cuts for MIP, and a smaller, but noticeable, increase in the number of (3.28) cuts for LOG. It did not affect significantly the number of other cuts, for either MIP or LOG.

3.7 Summary of Conclusions and Further Research

In this chapter, we considered MCOP (Multiple Choice Optimization Problem), which is a linear problem with complementarity constraints (also referred to as SOS1 constraints). We looked into three ways of enforcing such constraints to model MCOP, namely the MIP and LOG formulations, and the SOS approach.

We then generalized inequalities developed by de Farias el. al [12] (which we called SOS1 cuts) and used them within branch-and-cut to solve instances of MCOP using CPLEX. We studied the use of three CPLEX parameters (Presolve, Heuristics, and MIP cuts), plus the use of the SOS1 cuts.

We considered as the main measure of efficiency the solution time of our instances, and our results indicate that SOS1 cuts were extremely useful in solving our instances, with MIP, LOG, and SOS. Among those three ways of enforcing the SOS constraint, the best was MIP, followed closely by LOG, followed by SOS (which gave much poorer results). Among the CPLEX features considered, MIP cuts was the most useful. Presolve had a small negative impact when combined with MIP, a small positive impact when combined with LOG, and a huge negative effect when combined with SOS. Heuristics did not have a significant impact for any of the formulations.

The number of nodes of the branch-and-cut trees necessary to solve our instances was correlated with the solution time, as expected. We also reported the types and

number of cutting planes used (both SOS1 and other derived internally by CPLEX).

Some related topics of further research are: the development of other families of inequalities to be used as cutting planes for MCOP; and a polyhedral study of MCOP without Assumption 3.3.4.

CHAPTER 4
PIECEWISE LINEAR OPTIMIZATION PROBLEMS

A set of variables is said to be a *special ordered set of type 2 (SOS2)* if, in the problem formulation, at most two of the variables in the set can be non-zero, and, in case two variables are non-zero, their indices must be adjacent.

One main application of this type of constraint is to model *piecewise linear functions*, i.e., functions that are linear between predefined *breakpoints*. These types of functions can occur naturally, or they can be used to approximate non-linear functions to any desired degree of accuracy.

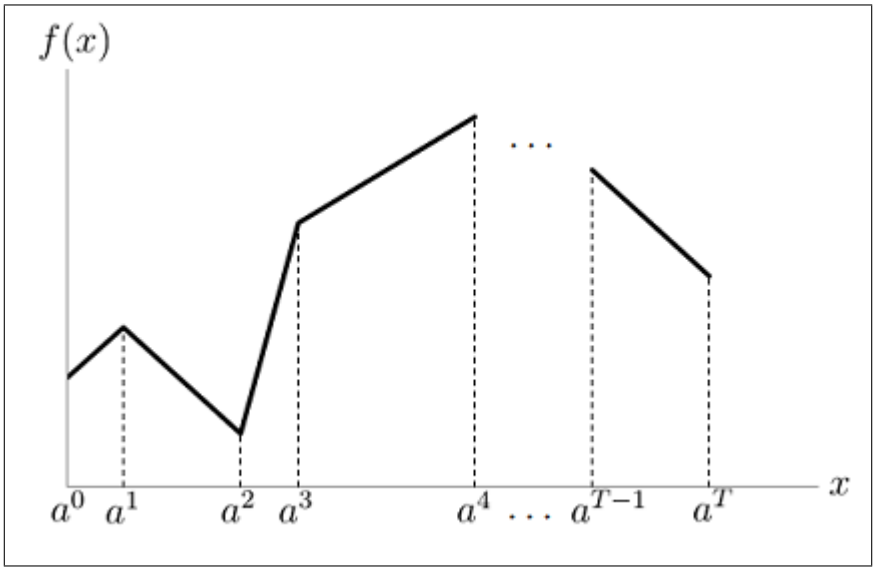


Figure 4.1. A Piecewise Linear Function.

Figure 4.1 shows a piecewise linear function $f(x) : [a^0, a^T] \to \mathbb{R}$ and its breakpoints $a^0, a^1, \ldots, a^T$. (Here, the superscript does *not* indicate the power of a number.) The values $f(a^i)$, $i = 0, \ldots, T$, are known, and thus $f(x)$ is completely defined.

A question of interest is how to model a piecewise linear function using linear constraints and (possibly) binary variables. The earliest formulation for this kind of function was introduced in 1957 by Markovitz [19], and is called the *incremental cost* model. In this dissertation, however, we will formulate our problems using the concepts of SOS2 and convex combinations of variables. Specifically, we let

$$x = \sum_{i=0}^{T} a^i \lambda^i \quad \text{and} \quad f(x) = \sum_{i=0}^{T} f(a^i)\lambda^i,$$

where

$$\begin{cases} \lambda^i \geq 0, \quad i = 0, \ldots, T \\ \displaystyle\sum_{i=0}^{T} \lambda^i = 1 \\ \{\lambda^0, \ldots, \lambda^T\} \text{ is SOS2.} \end{cases}$$

Here, we associate each breakpoint a^i with a variable λ^i and write x as a convex combination of two adjacent breakpoints (or a breakpoint itself). Once we know x, the value of $f(x)$ is well defined.

The "usual" formulation to model SOS2 constraints was introduced in 1960 by Dantzig [11], where he used a "linear" number of binary variables and linear constraints. Later, Beale and Tomlin [3] proposed specialized branching schemes to enforce SOS2 constraints (they also introduced the terminology "SOS2") that dispensed the use of binary variables and additional constraints. Finally, Vielma and Nemhauser [24] proposed a "logarithmic" formulation for SOS2 constraints.

In this chapter, we test these three approaches computationally.

4.1 Problem Statement

Let n be a positive integer and $N = \{1, \ldots, n\}$. We are interested in the *piecewise linear optimization problem (PLOP)*:

$$\begin{aligned} \text{minimize} \quad & \sum_{j \in N} g_j(x_j) \\ \text{subject to} \quad & Ax \leq \beta \\ & x_j \in [0, \omega_j], \quad j \in N, \end{aligned} \tag{4.1}$$

where g_j is a continuous piecewise linear function $\forall j \in N$. For each variable x_j, $j \in N$, let $a_j^0 = 0, \ldots, a_j^T = \omega_j$ be the breakpoints of function g_j (i.e., the points where the slope of g_j changes plus the endpoints 0 and ω_j), with $a_j^0 < \cdots < a_j^T$. For notational simplicity, we assume WLOG that all functions g_j have the same number $T + 1$ of breakpoints (in this case we may have to add breakpoints at places where the slope does not change). We write:

$$x_j = \sum_{k=0}^{T} a_j^k \lambda_j^k, \quad j \in N, \tag{4.2}$$

$$\sum_{k=0}^{T} \lambda_j^k = 1, \quad j \in N, \tag{4.3}$$

$$\lambda_j^k \geq 0, \quad j \in N, k = 0, \ldots, T, \tag{4.4}$$

$$\left\{\lambda_j^0, \ldots, \lambda_j^T\right\} \text{ is SOS2}, \quad j \in N. \tag{4.5}$$

4.2 Formulations to Model SOS2 Constraints

As mentioned above, the "usual" model to account for the SOS2 constraint (4.5) was introduced by Dantzig [11]. His model uses nT binary variables and $n(T+1)$ linear constraints, as follows:

$$\begin{cases} \lambda_j^0 \leq y_j^0, & j \in N \\ \lambda_j^k \leq y_j^{k-1} + y_j^k, & j \in N, k = 1, \ldots, T-1 \\ \lambda_j^T \leq y_j^{T-1}, & j \in N \\ \sum_{k=0}^{T-1} y_j^k = 1, & j \in N \\ y_j^k \in \{0,1\}, & j \in N, k = 0, \ldots, T-1. \end{cases} \tag{4.6}$$

Another way to enforce the SOS2 constraint is by using Vielma and Nemhauser's [24] "logarithmic" formulation: for each $j \in N$ and each $t \in \{1, \ldots, \lceil \log_2 T \rceil\}$, let

$$\begin{cases} L_{1_j}^t = \bigcup_{\ell=0}^{\lceil \log_2 T \rceil - t} \{k \in K : |k - 2^{t+1}\ell| < 2^{t-1}\} \\ L_{2_j}^t = \bigcup_{\ell=0}^{\lceil \log_2 T \rceil - t} \{k \in K : |k - (2^{t+1}\ell + 2^t)| < 2^{t-1}\}. \end{cases}$$

Then the constraints

$$\begin{cases} \sum_{k \in L_{1_j}^t} \lambda_j^k \leq y_j^t, & j \in N, t \in \{1, \ldots, \lceil \log_2 T \rceil\} \\ \sum_{k \in L_{2_j}^t} \lambda_j^k \leq 1 - y_j^t, & j \in N, t \in \{1, \ldots, \lceil \log_2 T \rceil\} \\ y_j^t \in \{0,1\}, & j \in N, t \in \{1, \ldots, \lceil \log_2 T \rceil\} \end{cases} \tag{4.7}$$

are equivalent to (4.5). We will call (4.6) the *MIP formulation* for SOS2 and (4.7)

the *LOG formulation* for SOS2.

An alternative to adding binary variables and linear constraints to our formulation is to use specialized branching schemes. This idea was pioneered by Beale and Tomlin [3], and commercial softwares can make use of it when the user declares a set to be SOS2. However, we do not know how these solvers have this implemented, and, for this reason, we will not elaborate further on branching schemes. We will, nevertheless, test this alternative computationally, and refer to it as the *SOS approach*.

We now illustrate both the MIP and the LOG formulation.

Example 4.2.1. Suppose we want to model the constraint

$$\{\lambda^0, \ldots, \lambda^8\} \text{ is } SOS2. \tag{4.8}$$

The MIP formulation for (4.8) is

$$
\begin{cases}
\lambda^0 & & & & & & & & & & & & & \leq & y^0 \\
& \lambda^1 & & & & & & & & & & & & \leq & y^0 + y^1 \\
& & \lambda^2 & & & & & & & & & & & \leq & y^1 + y^2 \\
& & & & \lambda^3 & & & & & & & & & \leq & y^2 + y^3 \\
& & & & & & \lambda^4 & & & & & & & \leq & y^3 + y^4 \\
& & & & & & & & \lambda^5 & & & & & \leq & y^4 + y^5 \\
& & & & & & & & & & \lambda^6 & & & \leq & y^5 + y^6 \\
& & & & & & & & & & & & \lambda^7 & \leq & y^6 + y^7 \\
& & & & & & & & & & & & & \lambda^8 \leq & y^7 \\
y^0 & + & y^1 & + & y^2 & + & y^3 & + & y^4 & + & y^5 & + & y^6 & + & y^7 & = & 1 \\
y^0, y^1, y^2, y^3, y^4, y^5, y^6, y^7 \in \{0,1\}, & & & & & & & & & & & & & & & &
\end{cases}
$$

and the LOG formulation for (4.8) is

$$
\begin{cases}
\lambda^0 & & & & & & \lambda^4 & & & & & & \lambda^8 & \leq & y^1 \\
& & \lambda^2 & & & & & & & + & \lambda^6 & & & \leq & 1 - y^1 \\
\lambda^0 & + & \lambda^1 & & & & & & & & & + & \lambda^7 + \lambda^8 & \leq & y^2 \\
& & & & \lambda^3 & + & \lambda^4 & + & \lambda^5 & & & & & \leq & 1 - y^2 \\
\lambda^0 & + & \lambda^1 & + & \lambda^2 & + & \lambda^3 & & & & & & & \leq & y^3 \\
& & & & & & & & \lambda^5 & + & \lambda^6 & + & \lambda^7 + \lambda^8 & \leq & 1 - y^3 \\
y^1, y^2, y^3 \in \{0,1\}. & & & & & & & & & & & & & & &
\end{cases}
$$

In Section 4.5, we will compare the computational performance of both the MIP and LOG formulations, and also the SOS approach.

4.3 Cutting Planes for SOS2-Constrained Problems

Recently, Zhao and de Farias [29] derived several families of inequalities to be used within branch-and-cut to solve SOS2-constrained problems. In this work, we will use "simplified" versions of their inequalities, since our instances will only have non-negative entries in the constraint matrix A.

Let $K = \{1, \ldots, T\}$. As in the case of SOS1, the inequalities are derived using one of the rows of A, represented by

$$\sum_{j \in N} \sum_{k \in K} a_j^k \lambda_j^k \leq b \tag{4.9}$$

(Here, we have omitted the terms $a_j^0 \lambda_j^0$ because $a_j^0 = 0, \forall j \in N$.) Our polyhedron of interest is $P = conv(S)$, where

$$S = \{\lambda \in \mathbb{R}^{n(T+1)} : \lambda \text{ satisfies } (4.3), (4.4), (4.9), \text{ and } (4.5)\}.$$

We define the linear programming relaxation of S by dropping the SOS2 constraint:

$$LPR(S) = \{\lambda \in \mathbb{R}^{n(T+1)} : \lambda \text{ satisfies } (4.3), (4.4), \text{ and } (4.9)\}.$$

The four families of inequalities we will use in our computational experiments are divided in two groups: *lifted convexity inequalities* and *lifted cover inequalities*. Each group has two families of inequalities. Below we give the first family of lifted convexity inequalities.

Theorem 4.3.1. (Zhao and de Farias [29]) *Let $j \in N$ and $s \in K$ be such that*

$a_j^s < b$. *Let* $I = \left\{i \in N - \{j\} : a_j^s + a_i^T > b\right\}$, *and* $k_i = \min\left\{k \in K : a_j^s + a_i^k > b\right\}$ $\forall i \in I$. *Suppose that* $I \neq \emptyset$. *Then,*

$$\frac{1}{a_j^s}\sum_{k=1}^{s-1} a_j^k \lambda_j^k + \sum_{k=s}^{T} \lambda_j^k + \sum_{i\in I}\sum_{k=\max\{1,k_i-1\}}^{T} \alpha_i^k \lambda_i^k \leq 1 \tag{4.10}$$

is valid for P, *where*

$$\begin{gathered}\left(\alpha_i^{k_i-1}, \alpha_i^{k_i}\right) \in \left\{(0,0), \left(\frac{a_j^s + a_i^{k_i-1} - b}{a_j^s}, \frac{a_j^s + a_i^{k_i} - b}{a_j^s}\right)\right\}, \\ \forall i \in I \text{ with } k_i > 1 \text{ and } a_j^s + a_i^{k_i-1} < b,\end{gathered} \tag{4.11}$$

$$\left(\alpha_i^{k_i-1}, \alpha_i^{k_i}\right) = \left(0, \frac{a_j^s + a_i^{k_i} - b}{a_j^s}\right), \forall i \in I \text{ with } k_i > 1 \text{ and } a_j^s + a_i^{k_i-1} = b,$$

$$\alpha_i^{k_i} = 0, \forall i \in I \text{ with } k_i = 1,$$

and

$$\alpha_i^k = \frac{a_j^s + a_i^k - b}{a_j^s}, \forall i \in I \text{ with } k > k_i.$$

Example 4.3.1. Let $n = 3$, $T = 3$, and (4.9) be given by

$$\left(2\lambda_1^1 + 6\lambda_1^2 + 8\lambda_1^3\right) + \left(3\lambda_2^1 + 7\lambda_2^2 + 10\lambda_2^3\right) + \left(5\lambda_3^1 + 7\lambda_3^2 + 9\lambda_3^3\right) \leq 10.$$

Let $j = 1$ and $s = 2$. Then $I = \{2, 3\}$, $k_2 = 2, k_3 = 1$, and, by choosing $\left(\alpha_i^{k_i-1}, \alpha_i^{k_i}\right) = (0, 0)$ in (4.11), inequality (4.10) is

$$\left(\frac{1}{3}\lambda_1^1 + \lambda_1^2 + \lambda_1^3\right) + \left(\lambda_2^3\right) + \left(\frac{1}{2}\lambda_3^2 + \frac{5}{6}\lambda_3^3\right) \leq 1. \tag{4.12}$$

By choosing the other option in (4.11), inequality (4.10) is

$$\left(\frac{1}{3}\lambda_1^1 + \lambda_1^2 + \lambda_1^3\right) + \left(-\frac{1}{6}\lambda_2^1 + \frac{1}{2}\lambda_2^2 + \lambda_2^3\right) + \left(\frac{1}{2}\lambda_3^2 + \frac{5}{6}\lambda_3^3\right) \leq 1. \tag{4.13}$$

Inequalities (4.12) and (4.13) both define facets of P.

We now give the second family of lifted convexity inequalities.

Theorem 4.3.2. (Zhao and de Farias [29]) *Let $j \in N$, $s \in K - \{1\}$, and $I = \{i \in N - \{j\} : a_j^{s-1} + a_i^T \geq b\}$. Suppose that $I \neq \emptyset$. Let $k_i = \min\{k \in K : a_j^{s-1} + a_i^k \geq b\}$, $i \in I$, and $L = \left\{i \in I : a_j^s + a_i^{k_i - 1} \geq b \text{ and } k_i > 1\right\}$. Suppose that $L \neq \emptyset$, and let $a_L = \min\{a_i^{k_i-1} : i \in L\}$. Then,*

$$\sum_{k=1}^{s-1} \gamma_j^k \lambda_j^k + \sum_{k=s}^{T} \lambda_j^k + \sum_{i \in L} \sum_{k=k_i}^{T} \alpha_i^k \lambda_i^k \leq 1 \tag{4.14}$$

is valid for P, where

$$\alpha_i^k = \frac{a_i^k - a_L}{b - a_L} \ \forall i \in L \text{ with } k \geq k_i.$$

Example 4.3.1 (continued) *Let $j = 1$ and $s = 2$. Then $I = \{2, 3\}$, $k_2 = k_3 = 3$, $L = \{2, 3\}$, $a_L = 7$, and (4.14) is*

$$\left(\frac{2}{3}\lambda_1^1 + \lambda_1^2 + \lambda_1^3\right) + \left(\lambda_2^3\right) + \left(\frac{2}{3}\lambda_3^3\right) \leq 1.$$

This inequality defines a facet of P.

We now give the two families of lifted cover inequalities derived in Zhao and de Farias [29] that we will test computationally. Given $j \in N$, we denote

$$u_j^k = a_j^k - a_j^{k-1} \ \forall k \in K - \{1\} \text{ and } u_j^1 = a_j^1.$$

Definition 4.3.1. Let $C \subseteq N$ and $l_j \in \{2, \ldots, T\} \ \forall j \in C$ be such that

$$\sum_{j \in C} a_j^{l_j} = b + \rho \tag{4.15}$$

with $\rho > 0$. The set C is a *cover*.

For the remainder of this chapter, C will denote a cover. We now give the first family of lifted cover inequalities.

Theorem 4.3.3. (Zhao and de Farias [29]) *Let C_1 and C_2 be two disjoint subsets of C such that $C = C_1 \cup C_2$, and $l_j > 2 \ \forall j \in C_1$. Then,*

$$\sum_{j \in C_1} (\alpha_j \lambda_j^{l_j - 2} + \beta_j \lambda_j^{l_j - 1} + \sum_{k=l_j}^{T} \lambda_j^k) + \sum_{j \in C_2} (\gamma_j \lambda_j^{l_j - 1} + \sum_{k=l_j}^{T} \lambda_j^k) \leq |C| - 1 \tag{4.16}$$

is valid for P, where

$$\alpha_j = min \left\{ 0, \frac{\rho - u_j^{l_j} - u_j^{l_j - 1}}{\rho} \right\}, \ \beta_j = \frac{\rho - u_j^{l_j}}{\rho}, \ \text{and } \gamma_j = min \left\{ 0, \beta_j \right\}.$$

Example 4.3.2. Let $n = 4$, $T = 3$, and (4.9) be

$$\left(2\lambda_1^1 + 6\lambda_1^2 + 8\lambda_1^3\right) + \left(3\lambda_2^1 + 7\lambda_2^2 + 10\lambda_2^3\right) + \left(4\lambda_3^1 + 8\lambda_3^2 + 10\lambda_3^3\right) + \left(5\lambda_4^1 + 7\lambda_4^2 + 9\lambda_4^3\right) \leq 10.$$

We take $C_1 = \{1\}$ with $l_1 = 3$ and $C_2 = \{3\}$ with $l_3 = 2$. Then $\rho = 6$, and (4.16) is

$$\left(\frac{1}{3}\lambda_1^2 + \lambda_1^3\right) + \left(\lambda_3^2 + \lambda_3^3\right) \leq 1.$$

This inequality cuts off the point $\lambda_1^3 = \frac{1}{5}$, $\lambda_3^2 = 1$, and $\lambda_j^k = 0$ otherwise, which is a vertex of $LPR(S)$.

Next, we give the second family of lifted cover inequalities.

Theorem 4.3.4. (Zhao and de Farias [29]) *Let* $a_C = max\left\{a_i^{l_i} : \forall i \in C\right\}$, $\bar{C} = \{j \in N - C : a_j^T \geq a_C$ *and* $a_j^{T-1} \geq a_C - \rho\}$, *and* $t_j = min\left\{k : a_j^k \geq a_C$ *and* $a_j^{k-1} \geq a_C - \rho\right\}$ $\forall j \in \bar{C}$. *Suppose that* $\bar{C} \neq \emptyset$. *The lifted cover inequality*

$$\sum_{j \in C}(\gamma_j \lambda_j^{l_j - 1} + \sum_{k=l_j}^{T} \lambda_j^k) + \sum_{j \in \bar{C}} \sum_{k=t_j}^{T} \lambda_j^k \leq |C| - 1 \tag{4.17}$$

is valid for P, *where the* γ_j*'s are as in Theorem 4.3.3.*

Example 4.3.2 (continued) *We take* $C = \{1, 2\}$ *and* $l_1 = l_2 = 2$. *Then,* $\bar{C} \in \{3, 4\}$, $t_3 = t_4 = 2$, *and (4.3.4) is*

$$\left(-\frac{1}{3}\lambda_1^1 + \lambda_1^2 + \lambda_1^3\right) + \left(-\frac{1}{3}\lambda_2^1 + \lambda_2^2 + \lambda_2^3\right) + \left(\lambda_3^2 + \lambda_3^3\right) + \left(\lambda_4^2 + \lambda_4^3\right) \leq 1. \tag{4.18}$$

This inequality defines a facet of P.

4.4 Separation of SOS2 Cuts

We now explain how we separated inequalities (4.10), (4.14), (4.16), and (4.17), which we call collectively *SOS2 cuts*, to be used within branch-and-cut to solve instances of PLOP. For the remainder of this section we let $\bar{\lambda}$ be the optimal LP relaxation solution just obtained.

Let $j \in N$ and suppose that $\bar{\lambda}_j^k > 0$ for some $k \in K$. Let

$$\epsilon_j = \min\left\{k \in K : \bar{\lambda}_j^k > 0\right\}$$

and

$$\phi_j = \max\left\{k \in K : \bar{\lambda}_j^k > 0\right\}.$$

We repeat the procedure below for each row of the constraint matrix.

Inequality (4.10) This inequality is determined by choosing j and s, and, when (4.11) holds, one of the two alternative values for $\left(\alpha_i^{k_i-1}, \alpha_i^{k_i}\right)$. The choice between the two alternatives is made as follows: for every $i \in I$, if $\alpha_i^{k_i-1}\bar{\lambda}_i^{k_i-1} + \alpha_i^{k_i}\bar{\lambda}_i^{k_i} \leq 0$, we take $\left(\alpha_i^{k_i-1}, \alpha_i^{k_i}\right) = (0,0)$, otherwise we choose the other option.

For every $j \in N$, we test whether $\bar{\lambda}$ violates (4.10) with $s = 1$ and, in case $\epsilon_j > 1$, we also test whether $\bar{\lambda}$ violates (4.10) with $s = \epsilon_j$. If it does, we add the inequality to CPLEX's cut pool.

Inequality (4.14) Again, the inequality is determined by choosing j and s. For every $j \in N$ and $s = 2, \ldots, T-1$, we test whether $\bar{\lambda}$ violates (4.14) and, if it

does, we add the inequality to CPLEX's cut pool.

Inequality (4.16) For every $j \in N$, we construct covers as follows: $C = \{j\} \cup \{i \in N : i \neq j \text{ and } \epsilon_i \geq 2\}$, with $2 \leq l_j \leq \epsilon_j$, $l_i = \epsilon_i$, and provided $\sum_{j \in C} a_j^{l_j} > b$. We let

$$C_1 = \left\{ j \in C : \epsilon_j \geq 3 \text{ and } a_j^{\epsilon_j - 1} > b - \sum_{i \in C - \{j\}} a_i^{\epsilon_i} \right\},$$

and $C_2 = C - C_1$. We then test whether $\bar{\lambda}$ violates (4.16) and, if it does, we add the inequality to the CPLEX's pool.

Inequality (4.17) We construct covers C as in the previous case, and $\bar{C}$ as in Theorem 4.3.4. We then test whether $\bar{\lambda}$ violates (4.17) and, if it does, we add the inequality to the CPLEX's pool.

In all of the above procedures, a *minimum violation* had to be met by $\bar{\lambda}$ in order for an inequality to be given to CPLEX. This value was chosen based on preliminary tests, and was set to 0.01. We performed separation at every node of tree in all instances we tested. We separate at most 1,000 inequalities of each type.

4.5 Computational Experience

4.5.1 Instances

We tested branch-and-cut with the inequalities of Section 4.3 on challenging instances of the *transportation problem*:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i\in I}\sum_{j\in J} g_{ij}(x_{ij}) \\
\text{subject to} \quad & \sum_{j\in J} x_{ij} = \sigma_i, \qquad i \in I \\
& \sum_{i\in I} x_{ij} = \delta_j, \qquad j \in J \\
& x_{ij} \geq 0, \qquad i \in I, j \in J,
\end{aligned}$$

where I is the set of supply nodes, J the set of demand nodes, σ_i the supply of node i, and δ_j the demand of node j. We assume WLOG that $\sum_{i\in I}\sigma_i = \sum_{j\in J}\delta_j$.

In terms of the λ variables, the transportation model becomes

$$\begin{aligned}
\text{minimize} \quad & \sum_{i\in I}\sum_{j\in J}\sum_{k\in K} c_{ij}^k \lambda_{ij}^k \\
\text{subject to} \quad & \sum_{j\in J}\sum_{k\in K} a_{ij}^k \lambda_{ij}^k = \sigma_i, \qquad i \in I \\
& \sum_{i\in I}\sum_{k\in K} a_{ij}^k \lambda_{ij}^k = \delta_j, \qquad j \in J \\
& \sum_{k=0}^{T} \lambda_{ij}^k = 1, \qquad i \in I, j \in J \\
& \lambda_{ij} \geq 0, \qquad i \in I, j \in J, k \in K \\
& \left\{\lambda_{ij}^0, \ldots, \lambda_{ij}^T\right\} \text{ is } SOS2, \quad i \in I, j \in J.
\end{aligned}$$

To generate our test instances, we followed the procedures described in [13], with $|I| \in \{20, 40\}$, $|J| \in \{40, 60, 80\}$, and $T \in \{4, 6, 8, 10\}$. For each combination of $|I|$, $|J|$, and T, we created 5 instances using formulations LOG, MIP, and SOS.

Therefore, the number of instances created with each formulation is 120, for a total of 360 instances.

4.5.2 Tests Conducted

We performed our computational tests in the Texas Tech High Performance Computing Center's Intel Xeon E5450 3.0GHz CPU with 16GB RAM nodes. We used the callable libraries of CPLEX 12.2.0.0, which we ran on a single thread. We allowed a maximum computational time of 3,600 seconds of CPU for solving each instance.

As with MCOP (Chapter 3), we evaluated CPLEX's performance when solving PLOP to proven optimality. We compared the performance that results when:

- the LOG formulation, MIP formulation, or SOS approach is used
- CPLEX's internal features are turned ON or OFF
- SOS2 cuts (i.e., inequalities (4.10), (4.14), (4.16) and (4.17)) are used or not

The CPLEX features that we consider are: *Presolve*, *Heuristics*, and *MIP cuts*. In our tests, we turn these parameters ON and OFF. To simplify our presentation, when considering a strategy we will use the letter `p` to denote Presolve ON, `h` to denote Heuristics ON, `m` to denote MIP cuts ON, and `s` to denote SOS2 cuts ON, always in this order. To denote a feature being OFF, we will use the letter `n`. Table 4.1 describes all possible strategies used. (This table is identical to Table 3.1. We reproduce it here for completeness.)

Given a PLOP instance, we will say that it was *solved* when it was solved to proven optimality within our prescribed time limit, and we consider a strategy to be *more*

Table 4.1. Features and Strategies

Strategy	Presolve	Heuristics	MIP cuts	SOS2 cuts
`phms`	ON	ON	ON	ON
`phmn`	ON	ON	ON	OFF
`phns`	ON	ON	OFF	ON
`phnn`	ON	ON	OFF	OFF
`pnms`	ON	OFF	ON	ON
`pnmn`	ON	OFF	ON	OFF
`pnns`	ON	OFF	OFF	ON
`pnnn`	ON	OFF	OFF	OFF
`nhms`	OFF	ON	ON	ON
`nhmn`	OFF	ON	ON	OFF
`nhns`	OFF	ON	OFF	ON
`nhnn`	OFF	ON	OFF	OFF
`nnms`	OFF	OFF	ON	ON
`nnmn`	OFF	OFF	ON	OFF
`nnns`	OFF	OFF	OFF	ON
`nnnn`	OFF	OFF	OFF	OFF

efficient than another when the computational time required to solve the instance was smaller for that strategy.

Finally, we note that CPLEX has two search strategies, namely *Traditional Branch-and-Cut* and *Dynamic Search*. Of the two, only Traditional Branch-and-Cut allows the user to add their own cutting planes, so we only tested this option. Once this strategy is chosen, CPLEX's default setting have Presolve, Heuristics, and MIP cuts are all turned ON, and thus is represented by the strategy `phmn`.

4.6 Analysis of Results

We now report and analyze the results of our computational tests. For different formulations and strategies, we consider four outputs: number of instances solved,

solution time, number of nodes used in the branch-and-cut tree, and number of cuts applied.

4.6.1 Number of Instances Solved

We start our analysis with the *number of instances solved* using each strategy, separated by type of formulation (MIP, LOG, or SOS). The results are shown in Table 4.2.

Table 4.2. Number of Instances Solved by Formulation and Strategy

Strategy	MIP	LOG	SOS
`phms`	80	111	104
`phmn`	37	67	49
`phns`	81	106	103
`phnn`	21	44	55
`pnms`	76	113	102
`pnmn`	37	67	55
`pnns`	79	107	103
`pnnn`	22	46	55
`nhms`	102	117	106
`nhmn`	54	83	53
`nhns`	93	106	104
`nhnn`	27	38	53
`nnms`	102	116	105
`nnmn`	61	83	54
`nnns`	93	106	105
`nnnn`	29	34	57
Total	994	1344	1263

The number of tests run with each formulation was $120 \cdot 16 = 1920$. Thus, we have that 52% of the instances were solved with the MIP formulation, 70% were solved with the LOG formulation, and 66% were solved with the SOS approach.

Presolve had a negative effect on the number of instances solved for MIP: the number of instances solved with Presolve ON was 433, or 46% of the instances considered (against 561, or 58%, with Presolve OFF). For LOG and SOS, however, Presolve had virtually no effect on the number of instances solved.

The use of Heuristics did not have a significant impact on the ability to solve our instances with any of the formulations.

The results indicate that turning MIP cuts ON had a positive impact on the number of instances solved for MIP and LOG, and no impact for SOS. For MIP, the eight strategies that use MIP cuts solved our instances in 549, or 57%, of the tests considered (against 445, or 46%, without MIP cuts), and, for LOG, this number was 757, or 79% (against 587, or 61%, without MIP cuts).

The use of SOS2 cuts also had a positive (and more substantial) impact on the number of instances solved for all formulations. For MIP, the eight strategies that use SOS2 cuts solved our instances in 706, or 74%, of the tests considered (against 288, or 30%, without SOS2 cuts); for LOG, this number was 882, or 92% (against 462, or 48%, without SOS2 cuts); for the SOS approach, this number was 832, or 87% (against 431, or 45%, without SOS2 cuts).

Overall, the best formulation in terms of number of instances solved was LOG, followed by SOS, followed by MIP. In particular, the combination of the LOG formulation with the strategy `nhms` was the one which solved the biggest number of instances: 117, or 97.5%.

Finally, in a comparison of the strategy `nhms` vs. `phmn` (which represents CPLEX's default settings), the number of instances solved was 102 vs. 37 for MIP, 117 vs. 67 for LOG, and 106 vs. 49 for SOS.

4.6.2 Solution Time

We now turn our attention to the *solution time* of the instances. The summarized results are shown in Table 4.3. We note that, in this table, the times shown are averages that take into account only the instances solved (and shown in Table 4.2).

Table 4.3. Average Solution Times (s) and Strategies

Strategy	MIP	LOG	SOS
phms	613	190	402
phmn	808	720	641
phns	785	268	367
phnn	1305	940	639
pnms	610	265	347
pnmn	873	618	630
pnns	747	296	1369
pnnn	1227	958	666
nhms	271	275	455
nhmn	712	753	609
nhns	270	287	398
nhnn	945	968	621
nnms	263	288	406
nnmn	877	856	724
nnns	262	320	413
nnnn	896	801	675
Total	585	457	482

Overall, LOG and SOS had close performances, and MIP was worse. Since MIP solved considerably less instances and had a significantly higher solution time average than the other two formulations, we conclude that MIP's performance was the least effective of all. Thus, we will focus our analysis on LOG and SOS only, first individually and then altogether. And to properly compare strategies, it is important that we use the *same instances* when considering their solution times.

In the next pages we will consider the use of each of the three CPLEX features, namely Presolve, Heuristics, and MIP cuts, plus the use of SOS1 cuts, and will see how they affected the solution time of our instances. Because the instances considered in this analysis may vary according to each formulation, the comparison of the times with and without a certain feature will be analyzed *within* a formulation (LOG or SOS). Only in Section 4.6.2.5 will we make a comparison across formulations, i.e., LOG vs. SOS.

4.6.2.1 Presolve

Table 4.4 compares the average solution time of instances when they are solved with and without *Presolve*, having everything else constant. The column "# Inst." shows the number of instances considered in the comparison, and values in **bold** indicate the better strategy. We note that this table does not enable us to compare the two formulations (LOG vs. SOS), since the instances considered for each may be different.

Table 4.4. Strategies and Average Solution Times (s) with and without Presolve

Strategies	LOG			SOS		
	# Inst.	Avg. Time		# Inst.	Avg. Time	
`phms` vs. `nhms`	104	190 vs.	**176**	104	**402** vs.	406
`phmn` vs. `nhmn`	67	720 vs.	**508**	48	609 vs.	**605**
`phns` vs. `nhns`	106	**268** vs.	287	102	**346** vs.	364
`phnn` vs. `nhnn`	38	**669** vs.	968	50	665 vs.	**597**
`pnms` vs. `nnms`	113	265 vs.	**234**	102	347 vs.	**343**
`pnmn` vs. `nnmn`	67	618 vs.	**595**	48	655 vs.	**651**
`pnns` vs. `nnns`	106	**276** vs.	320	103	369 vs.	**368**
`pnnn` vs. `nnnn`	34	**446** vs.	801	50	**599** vs.	605

For the LOG formulation, Presolve was harmful when used along with MIP cuts, and helpful when MIP cuts were OFF. With SOS2 cuts ON, Presolve had a smaller impact on the solution times. For the SOS approach, the differences between the values of the columns with and without Presolve is relativity small, so we conclude that this feature did not have a significant impact on the solution times.

4.6.2.2 Heuristics

Table 4.5 compares the average solution time of instances when they are solved with and without *Heuristics*, having everything else constant.

Table 4.5. Strategies and Average Solution Times (s) with and without Heuristics

Strategies	LOG			SOS		
	# Inst.	Avg. Time		# Inst.	Avg. Time	
`phms` vs. `pnms`	111	**190** vs.	211	102	350 vs.	**347**
`phmn` vs. `pnmn`	66	676 vs.	**595**	49	641 vs.	**577**
`phns` vs. `pnns`	106	**268** vs.	270	102	**338** vs.	357
`phnn` vs. `pnnn`	43	884 vs.	**827**	51	690 vs.	**587**
`nhms` vs. `nnms`	116	**262** vs.	288	105	427 vs.	**406**
`nhmn` vs. `nnmn`	80	**676** vs.	767	48	605 vs.	**552**
`nhns` vs. `nnns`	105	**269** vs.	294	104	398 vs.	**392**
`nhnn` vs. `nnnn`	34	**764** vs.	801	52	633 vs.	**606**

For LOG, having Heuristics ON was slightly better than having it OFF, while for SOS the opposite was true. As with Presolve, having SOS2 cuts ON caused the impact of Heuristics to be minor, for both formulations.

4.6.2.3 MIP Cuts

Table 4.6 compares the average solution time of instances when they are solved with and without *MIP cuts*, having everything else constant.

Table 4.6. Strategies and Average Solution Times (s) with and without MIP Cuts

Strategies	LOG		SOS	
	# Inst.	Avg. Time	# Inst.	Avg. Time
`phms` vs. `phns`	106	**124** vs. 268	103	385 vs. **367**
`phmn` vs. `phnn`	44	**285** vs. 940	49	**641** vs. 646
`pnms` vs. `pnns`	107	**144** vs. 296	102	347 vs. **341**
`pnmn` vs. `pnnn`	45	**307** vs. 899	52	666 vs. **635**
`nhms` vs. `nhns`	106	**138** vs. 287	104	399 vs. **398**
`nhmn` vs. `nhnn`	38	**298** vs. 968	48	548 vs. **502**
`nnms` vs. `nnns`	106	**156** vs. 320	105	**406** vs. **413**
`nnmn` vs. `nnnn`	34	**258** vs. 801	52	752 vs. **740**

We notice a major influence of the parameter MIP cuts on the average solution time for LOG, in every row of the table. The average solution time reductions of MIP cuts ON vs. OFF vary between 51% and 70%. The table indicates that CPLEX performed best when MIP cuts and SOS2 cuts were used. Differently from the case of SOS1, it seems that SOS2 cuts do not "replace" MIP cuts, but are effective when used simultaneously with them.

For SOS, setting MIP cuts ON or OFF had virtually no influence on the solution time. A further investigation of the results showed that, in fact, CPLEX did not add any internal cuts when the SOS formulation was used. This is an evidence that the solver does not derive internal cuts that specifically take into account the SOS2 constraint. The small difference in times showed in Table 4.6 probably is, then, the effect of random causes (e.g. computer node used).

4.6.2.4 SOS2 Cuts

Table 4.7 compares the average solution time of instances when they are solved with and without *SOS2 cuts*, having everything else constant.

Table 4.7. Strategies and Average Solution Times (s) with and without SOS2 Cuts

Strategies	LOG			SOS		
	# Inst.	Avg. Time		# Inst.	Avg. Time	
`phms` vs. `phmn`	67	**75**	vs. 720	48	**292**	vs. 655
`phns` vs. `phnn`	44	**115**	vs. 940	50	**306**	vs. 703
`pnms` vs. `pnmn`	67	**82**	vs. 618	49	**233**	vs. 707
`pnns` vs. `pnnn`	46	**188**	vs. 958	51	**291**	vs. 718
`nhms` vs. `nhmn`	83	**112**	vs. 753	48	**277**	vs. 672
`nhns` vs. `nhnn`	38	**126**	vs. 968	49	**319**	vs. 609
`nnms` vs. `nnmn`	83	**111**	vs. 856	51	**288**	vs. 713
`nnns` vs. `nnnn`	34	**129**	vs. 801	52	**357**	vs. 679

Here, we see a huge positive effect of SOS2 cuts, in every single comparison shown in the table. Although the reductions are not as big as in the case of SOS1, they are still extremely significant, varying from 80% to 90% for LOG and from 48% to 57% for SOS. These results show that, for our instances, the use of SOS2 cuts was indispensable for solving them more quickly.

4.6.2.5 LOG vs. SOS

We now compare LOG vs. SOS. In Table 4.8, we show the average solution times of the 33 instances that were solved with all strategies, using both LOG and SOS.

First, the table shows that the strategies that use SOS2 cuts performed better than the ones without those cuts, for both LOG and SOS. In a comparison between LOG and SOS, LOG was faster with most strategies, but significantly slower for

Table 4.8. Strategies and Average Solution Times (s) for Instances Solved by All Strategies with Both LOG and SOS

Strategy	LOG	SOS
phms	**56**	197
phmn	**182**	280
phns	**101**	186
phnn	489	**284**
pnms	**58**	200
pnmn	**166**	243
pnns	**106**	189
pnnn	434	**256**
nhms	**65**	190
nhmn	**255**	304
nhns	**124**	189
nhnn	751	**302**
nnms	**68**	190
nnmn	**258**	326
nnns	**133**	197
nnnn	797	**304**

the four strategies that had MIP cuts and SOS2 cuts both OFF. In particular, for strategy `nnnn`, in which the formulations themselves play a bigger role in differentiating between the instances (because the four parameters considered are OFF), SOS performed much better than LOG. This suggests that LOG needs cutting planes in order to be fast. Out of the 16 strategies, LOG performed best with `phms`, and SOS, with `phns`.

We conclude that if one were to use CPLEX to solve our instances, the best formulation to use would be LOG. *Without* SOS2 cuts, the best strategy to use would `pnmn`, while *with* SOS2 cuts, the best strategy would be `phms`.

4.6.3 Number of Nodes

We now analyze the effect of SOS2 cuts and the other CPLEX features considered in the previous sections on the number of nodes in the branch-and-cut tree necessary to solve our instances. Because this is a secondary measure of efficiency of a strategy, we consider only the 33 instances used to build Table 4.8, which are the ones that were solved with all strategies, using both LOG and SOS.

Table 4.9. Strategies and Average Number of Nodes for Instances Solved by All Strategies with Both LOG and SOS

Strategy	LOG	SOS
`phms`	625	2,308
`phmn`	21,913	163,718
`phns`	1,186	2,322
`phnn`	52,223	163,718
`pnms`	679	21,814
`pnmn`	21,814	162,559
`pnns`	33	2,338
`pnnn`	56,820	162,559
`nhms`	480	2,215
`nhmn`	6,676	189,018
`nhns`	1,154	2,247
`nhnn`	39,619	173,871
`nnms`	523	2,338
`nnmn`	7,643	210,055
`nnns`	1,267	44,272
`nnnn`	3,279	210,055

By comparing tables 4.8 and 4.9, we see that there is a correspondence between the values shown in both tables, in the sense that instances that are solved faster tend to require less nodes. This, however, seems to be true only within each formulation, but not across them. Take, for example, strategy `nnnn`: while the average solution

time is bigger for LOG by a factor of two, the number of nodes is larger for SOS by a factor of 150. In particular, the amount of nodes relative to the solution time is much bigger with SOS, which suggests that this formulation spends less time in each node in comparison with LOG. This could be explained by SOS's lack of binary variables, which makes its LPR's less time consuming than LOG's.

4.6.4 Types of Cuts Used

We now report the types and average number of cutting planes used by CPLEX in our tests. For SOS2 cuts, recall that we set a limit of 1,000 cuts for each of the inequalities (4.10), (4.14), (4.16), and (4.17), to be given to CPLEX. For the internal CPLEX cuts, no limit was previously set. Again, we consider only the 33 instances that were solved by all strategies and with both LOG and SOS. The results for LOG are shown in Table 4.10, and, for SOS, in Table 4.11.

In those tables, the column "Used" shows the number of SOS2 cuts that were actually used by CPLEX. This number may not be equal to the sum of the columns (4.10), (4.14), (4.16), and (4.17) because, for example, our separation algorithm may generate identical cuts, in which case at most one cut is counted as used. Another reason for this is the fact that CPLEX has a cut filter that may reject a cut if, for instance, it considers the cut too dense, or less effective than other cuts [9]. We do not have information on exactly how many SOS2 cuts of each type were individually used, but only the combined total. The column "Imp. Bnd" refers to implied bound cuts, and we note that cuts of the type MIR, Cover, and Zero-half were also used, but, because their averages were below 1 for every strategy, we omitted them from the tables.

Table 4.10. Strategies and Average Number of MIP Cuts Using the LOG Formulation for Instances BSolved by All Strategies with both LOG and SOS

Strat.	Flow Cover	Gomory	Imp. Bnd	(4.10)	(4.14)	(4.16)	(4.17)	Used
phms	18	12	0	143	54	5	34	195
phmn	23	16	1	—	—	—	—	—
phns	—	—	—	156	60	6	37	228
pnms	18	12	0	159	53	6	37	214
pnmn	23	16	0	—	—	—	—	—
pnns	—	—	—	168	64	6	40	247
nhms	27	11	0	128	53	5	30	178
nhmn	37	15	0	—	—	—	—	—
nhns	—	—	—	151	61	6	37	224
nnms	28	11	0	137	54	5	32	190
nnmn	37	15	0	—	—	—	—	—
nnns	—	—	—	161	62	5	39	237

Table 4.11. Strategies and Average Number of MIP Cuts Using the SOS Formulation for Instances Solved by All Strategies with Both LOG and SOS

Strat.	(4.10)	(4.14)	(4.16)	(4.17)	Used
phms	262	90	8	66	396
phns	262	90	8	66	396
pnms	254	85	8	67	384
pnns	250	84	8	64	376
nhms	245	86	7	66	374
nhns	246	86	7	66	374
nnms	245	86	7	65	373
nnns	246	87	7	65	375

For LOG (Table 4.10), we see that the number of SOS2 cuts generated (in particular (4.10)) is large in comparison with the internal MIP cuts generated by CPLEX. This alone does not say much, since those cuts could have been generated but not

in fact used. However, because the numbers on the column "Used" are also large in comparison with CPLEX's internal cuts, we conclude that SOS2 cuts played an important role in the solution of those instances. The fraction of SOS2 cuts used relative to the total cuts generated was around 86%. This is another evidence that SOS2 cuts were strong ones for our instances. Regarding the differences in the amount of individual SOS2 cuts, we believe that it is related to our separation heuristics and, perhaps, the intrinsic difficulty of deriving some of the cuts. Finally, the table shows that, perhaps, some SOS2 cuts and some MIP cuts played a similar role. This is suggested by the fact that when MIP cuts are ON, the number of SOS2 cuts that were used decreases (compare, for instance, `phns` vs. `phms`), while when SOS2 cuts are ON, the number of MIP cuts used decreases (compare, for instance, `phmn` vs. `phms`).

For SOS (Table 4.11), no MIP cuts were generated by CPLEX with any of the strategies, and the number of SOS2 cuts generated and used was fairly uniform across all strategies that had them ON. Also, the relative amount of different SOS2 cuts generated was uniform across strategies. The fraction of SOS2 cuts used relative to the total cuts generated was around 93%. Again, this is evidence that SOS2 cuts were strong ones for our instances.

In a comparison between SOS2 cuts used for LOG vs. SOS, the number for SOS is significantly larger. We believe that this is due to the fact that the number of nodes for SOS instances was much bigger (see Table 4.9), which allowed our separation heuristics to derive more cuts for this formulation.

Presolve had a small effect in the number of cuts generated. Turning it ON increased the number of MIP cuts (specially Flow Cover), but decreased the number

of SOS2 cuts generated (specially (4.10)) and used.

Turning Heuristics ON increased the number of cuts (4.10) generated and the number of SOS2 cuts used, but had virtually no effect on the other types of cuts.

4.7 Summary of Conclusions and Further Research

In this chapter, we considered the problem of minimizing a piecewise linear objective function subject to linear constraints, which we called PLOP (Piecewise Linear Optimization Problem). We looked into three ways of modeling PLOP, namely the MIP and LOG formulations, and the SOS approach.

We used recently developed cutting planes for this type of problem, which we called SOS2 cuts, and tested them with instances of the transportation problem with piecewise linear cost functions. We used CPLEX as our solver, and considered, besides the use SOS2 cuts, three of its parameters, namely Presolve, Heuristics, and MIP cuts.

Our main measure of efficiency was the solution time of our instances, and our results indicate that SOS2 cuts were extremely useful in solving our instances, with either MIP, LOG, or SOS. Among those three ways of enforcing the SOS constraint, the best was LOG, followed closely by SOS, followed by MIP. Among the CPLEX features considered, Presolve's effect varied depending on which formulation was used and how the other parameters were set. Heuristics effect was also dependent on other factors, but had a minor impact on the solution times of the instances. MIP cuts had a significant and positive impact for LOG, but were not generated when the SOS approach was used. Finally, the use of SOS2 cuts had the highest impact on the solution times of our tests, reducing them tremendously.

The number of nodes of the branch-and-cut trees necessary to solve our instances was correlated with the solution time, as expected. We also reported the types and number of cutting planes used (both SOS2 and other derived internally by CPLEX).

Some related topics of further research are: the development of other families of inequalities to be used as cutting planes for PLOP; and the use of SOS2 cuts for semi-continuous variables.

CHAPTER 5
CARDINALITY-CONSTRAINED OPTIMIZATION PROBLEMS

In some applications, one may wish to limit the number of active (or non-zero) variables in a set. In chapter 3, we discussed special ordered sets of type 1, in which at most one variable in the set is active. A generalization of this idea is to allow at most K variables in the set to be non-zero, where K is a positive integer. We will refer to this type of restriction as a *cardinality constraint*.

A few examples where cardinality constraints occur are:

- Portfolio selection, in which the investor has a variety of assets to choose, but shall not pick too many.
- Location of petroleum wells in a map, where we may want to drill at most a certain number of holes due to high drilling costs.

The scientific literature on cardinality constraints is relatively scarce (specially if compared with SOS1 and SOS2). Most studies, e.g. [5, 22, 23], focus more on applications rather than theoretical studies of the solution set of the problem and, when they do, the variables are restricted to being binary (see [6] and [18] for a comprehensive list of works). In fact, we are aware of only two articles that deal with continuous variables and cardinality constraints at the same time, namely [4] and [14].

Bienstock [4] studied a branching scheme for cardinality constrained problems, as well as cutting planes that take into account this type of constraint. He also introduced the idea of *critical set inequalities*, which we generalize here. The paper

by de Farias and Nemhauser [14] is entirely focused on branch-and-cut for cardinality constrained linear problems. In their paper, the inequalities of the constraint matrix have non-negative coefficients, less-than-or-equal signs, and positive right-hand-sides (in other words, the linear constraints are of the type $Ax \leq \beta$ with $A \geq 0$ and $\beta > 0$). In this chapter, we study the case where $Ax \geq \beta$ with $A \geq 0$ and $\beta > 0$.

5.1 Problem Statement

Let m, n and K be positive integers, $M = \{1, \ldots, m\}$, and $N = \{1, \ldots, n\}$. The *cardinality-constrained optimization problem* (CCOP) is

$$\begin{aligned}
\text{maximize} \quad & \sum_{j \in N} c_j x_j & & \\
\text{subject to} \quad & \sum_{j \in N} a_{ij} x_j \geq b_i & i \in M & \quad (5.1)\\
& x_j \geq 0 & j \in N & \quad (5.2)\\
& x_j \leq 1 & j \in N & \quad (5.3)\\
& \text{at most } K \text{ variables } x_j \text{ can be nonzero.} & & \quad (5.4)
\end{aligned}$$

Given a set of indices $A \subseteq N$, we denote the minimum and maximum elements of A as $\min(A)$ and $\max(A)$, respectively. We let $\overline{A} = N \setminus A$ and $\hat{A} = A \cup \{j \in N : j > \max(A)\}$. For a given nonnegative integer t, we let A_t be the set consisting of the t smallest elements of A (if $t = 0$, $A_t = \emptyset$, and if $t > |A|$, $A_t = A$). Finally, we define $\sum_{i \in \emptyset} a_i = 0$.

5.2 Formulation to Model the Cardinality Constraint

Unlike SOS1 and SOS2 constraints, we do not know of a "logarithmic" formulation for cardinality constraints. Therefore, we will use a formulation that requires a "linear" number of binary variables and additional constraints. Specifically, we model (5.4) as follows:

$$\begin{cases} x_j \leq y_j, \quad j \in N \\ \sum_{j \in N} y_j \leq K \\ y_j \in \{0,1\}, \quad j \in N. \end{cases} \tag{5.5}$$

We now illustrate our MIP formulation for (5.4).

Example 5.2.1. Let $n = 8$ and $K = 5$. Then (5.5) is

$$\begin{cases} x_1 \leq y_1 \\ x_2 \leq y_2 \\ x_3 \leq y_3 \\ x_4 \leq y_4 \\ x_5 \leq y_5 \\ x_6 \leq y_6 \\ x_7 \leq y_7 \\ x_8 \leq y_8 \\ y_1 + y_2 + y_3 + y_4 + y_5 + y_6 + y_7 + y_8 \leq 5 \\ y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8 \in \{0,1\}. \end{cases} \tag{5.6}$$

5.3 Polyhedral Study of CCOP

In this section we study the polyhedron defined by one of the inequalities (5.4). As with MCOP and PLOP, we will derive results using a single inequality of the constraint matrix and then use those results for the problem with multiple constraints. We start by defining our polyhedron of interest.

5.3.1 The Polyhedron P

Let $P = conv(S)$, with $S = \{x \in \mathbb{R}^n : (5.2), (5.3), (5.4), \text{ and } (5.7) \text{ hold}\}$, where

$$\sum_{j \in N} a_j x_j \geq b \tag{5.7}$$

is one of the inequalities (5.1).

We assume that:

Assumption 5.3.1. $a_1 \geq \cdots \geq a_n$.

Assumption 5.3.2. $K \leq n - 1$.

Assumption 5.3.3. $\sum_{j=1}^{K} a_j \geq b$.

Assumption 5.3.4. $b \geq 0$ *and* $a_j \geq 0, \ \forall j \in N$.

Assumption 5.3.1 is WLOG since we are considering only one of the inequalities (5.1). When Assumption 5.3.2 does not hold (i.e., if the number of variables is not greater than K), the cardinality constraint (5.4) is redundant, so it is also WLOG. Given Assumption 5.3.1, Assumption 5.3.3 guarantees feasibility. Assumption 5.3.4 carries loss of generality. However, it is satisfied in important applications, e.g. portfolio selection [4].

We now give necessary and sufficient conditions for P to be full-dimensional.

Proposition 5.3.1. *The polytope P is full-dimensional if and only if the following conditions hold:*

$$\sum_{j=1}^{K} a_j > b \tag{5.8}$$

and

$$\sum_{j=1}^{K-1} a_j + a_n \geq b. \tag{5.9}$$

Proof. Suppose that (5.8) and (5.9) hold, and let $y = \frac{b}{\sum_{j=1}^{K} a_j}$. Then the following $n+1$ points are linearly independent and belong to P:

For $i = 1, \ldots, K$: $x_i^i = y$, $x_j^i = 1$ for all $j \in \{1, \ldots, K\} - \{i\}$, and $x_j^i = 0$ otherwise; for $i = K+1, \ldots, n$: $x_j^i = 1$ for all $j \in \{1, \ldots, K-1\} \cup \{i\}$, and $x_j^i = 0$ otherwise; plus the point $x_j^{n+1} = 1$ for all $j \in \{1, \ldots, K\}$, and $x_j^{n+1} = 0$ otherwise.

Suppose now that P is full dimensional. Then there exists a point $x' \in P$ with $x'_n > 0$, i.e. we have $\sum_{j=1}^{K-1} a_j x'_j + a_n x'_n \geq b$, which implies (5.9). If (5.8) does not hold, then (5.9) implies that $\sum_{j=1}^{k} a_j = b$, and so the dimension of P is at most $n-1$, a contradiction. Therefore (5.8) must hold. □

Because the inequality description of a polyhedron is simpler when it is full-dimensional, we assume that:

Assumption 5.3.5. *Conditions (5.8) and (5.9) hold.*

5.3.2 Trivial Inequalities

In this section, we study *trivial inequalities* valid for P. We call them trivial because they are easily inferred by the definition of P. In this section we provide necessary and sufficient conditions for those inequalities to be facet-defining. The first facet-defining inequality of P is the knapsack inequality.

Proposition 5.3.2. *Inequality (5.7) defines a facet of P.*

Proof. Let $m_i = \min\left\{1, \frac{b}{a_i}\right\}$, $i \in N$. For $i \in \{1, \ldots, K\}$, let $y_i = \frac{b - a_i m_i}{\sum_{j=1}^{K} a_j - a_i}$, and, for $i \in \{K+1, \ldots, n\}$, let $y_i = \frac{b - a_i m_i}{\sum_{j=1}^{K-1} a_j}$. Then the following n points are linearly independent, belong to P, and satisfy (5.7) at equality:

For $i = 1, \ldots, K$: $x_i^i = m_i$, $x_j^i = y_i$ for all $j \in \{1, \ldots, K\} - \{i\}$, and $x_j^i = 0$ otherwise; for $i = K+1, \ldots, n$: $x_i^i = m_i$, $x_j^i = y_i$ for all $j \in \{1, \ldots, K-1\}$, and $x_j^i = 0$ otherwise. □

The next two propositions follow directly from the previous one. For an index $j \in N$, inequality $x_j \geq 0$ defines a facet of P if and only if the polyhedron $\{x \in P : x_j = 0\}$ has dimension $n-1$. Similarly $x_j \leq 1$ is facet-defining for P if and only if $\{x \in P : x_j = 1\}$ has dimension $n-1$.

Proposition 5.3.3. *Inequality (5.2) defines a facet of P $\forall j > K$. It is facet-defining for $j \leq K$ if and only if $\sum_{i=1}^{K+1} a_i - a_j > b$ and $\sum_{i=1}^{k} a_i - a_j + a_n \geq b$.*

If (5.2) is not facet-defining for some $j \leq K$, then, by the previous proposition, at least one of the two conditions must fail. If $\sum_{i=i}^{K+1} a_i - a_j \leq b$, then (5.7) forces every x_i with $i \leq j$ to be nonnegative. If $\sum_{i=1}^{K} a_i - a_j + a_n < b$, then $x_n > 0$ forces x_j to be positive, i.e., there exists a $\gamma_j > 0$ so that $x_j \geq \gamma_j x_n$ is valid for P.

Proposition 5.3.4. *Inequality (5.3) defines a facet of P $\forall j < K$. It is facet-defining for $K \leq j \leq n-1$ if and only if $\sum_{i=1}^{K-1} a_i + a_j > b$ and $\sum_{i=1}^{K-2} a_i + a_j + a_n \geq b$. Finally, it is facet-defining for $j = n$ if and only if $\sum_{i=1}^{K-1} a_i + a_n > b$ and $\sum_{i=1}^{K-2} a_i + a_{n-1} + a_n \geq b$.*

If (5.3) is not facet-defining for some $j \geq K$, then there exists a set $C \subset N$, $|C| \geq 2$, such that $j \in C$ and $\sum_{i \in C} x_i \leq 1$ is valid for P. We will return to this in the next section, when we study *critical-set inequalities*.

Proposition 5.3.5. *The inequality*

$$\sum_{j \in N} x_j \leq K \tag{5.10}$$

is valid for P. It defines a facet of P if and only if $\sum_{j=2}^{K+1} a_j \geq b$.

Proof. It it clear that (5.3) and (5.4) imply (5.10). Suppose that $\sum_{j=2}^{K+1} a_j \geq b$. Then the following n points are linearly independent, belong to P, and satisfy (5.10) at equality:

For $i = 1, \ldots, K$: $x_j^i = 1$ for all $j \in \{1, \ldots, K+1\} - \{i\}$ and $x_j^i = 0$ otherwise; for $i = K+1, \ldots, n$: $x_j^i = 1$ for all $j \in \{2, \ldots, K\} \cup \{i\}$ and $x_j^i = 0$ otherwise.

If $\sum_{j=2}^{K+1} a_j < b$, then at most $K-1$ variables with index in $\{2, \ldots, n\}$ can be positive, and we get the valid inequality $\sum_{j=2}^{n} x_j \leq K-1$, which is stronger than (5.10). □

Example 5.3.1. Let $S = \{x \in [0,1]^8 : 11x_1 + 10x_2 + 9x_3 + 8x_4 + 5x_5 + 2x_6 + 2x_7 + 1x_8 \geq 21,$ and at most 3 variables x_j can be positive$\}$. Then P is full dimensional, $x_2 \geq 0, \ldots, x_8 \geq 0$ define facets of P ($x_1 \geq 0$ does not), $x_1 \leq 1, \ldots, x_3 \leq 1$ define facets of P ($x_4 \leq 1, \ldots, x_8 \leq 1$ do not), and $\sum_{j=1}^{8} x_j \leq 3$ is facet-defining for P.

Example 5.3.2. Let $S = \{x \in [0,1]^8 : 11x_1+10x_2+8x_3+7x_4+6x_5+5x_6+2x_7+1x_8 \geq 23,$ and at most 4 variables x_j can be positive$\}$. Then P is full dimensional, both $x_j \geq 0$ and $x_j \leq 1$ define facets of P for all $j \in \{1,\ldots,8\}$, and $\sum_{j=1}^{8} x_j \leq 4$ is facet-defining for P.

5.4 Cutting Planes for CCOP

In this section, we study two families of inequalities valid for P. Our first family of inequalities generalizes Bienstock's *critical-set inequalities* [4]. Our second family of inequalities is obtained by lifting *surrogate cardinality constraints*, i.e., inequalities of the type

$$\sum_{j \in N'} x_j \leq K, \tag{5.11}$$

where $N' \subset N$.

5.4.1 Generalized Critical-Set Inequalities

We now generalize Bienstock's critical-set inequalities [4], and we give necessary and sufficient conditions for them to define facets of P.

Definition 5.4.1. (Bienstock [4]) A set $C \subset N$ is *critical* if $\forall J \subset C$ with $|J| = K$, $\sum_{j \in J} a_j < b$.

Definition 5.4.2. Let $d \in \{2,\ldots,K\}$. A set $C \subset N$, with $|C| \geq d$, is *d-critical* if

$$\sum_{j \in \overline{C}_{K-d}} a_j + \sum_{j \in C_d} a_j < b.$$

A d-critical set is *maximal* if $\forall j \in \overline{C}$, $C \cup \{j\}$ is not d-critical.

If C is d-critical, then no more than $d-1$ variables with index in C can be positive. Thus,

$$\sum_{j \in C} x_j \leq d - 1 \tag{5.12}$$

is valid for P. Note that if C is d-critical, then so is $\hat{C}$. We call (5.12) a *generalized critical-set inequality*. We now give necessary and sufficient conditions for generalized critical-set inequalities to define facets.

Proposition 5.4.1. *Let $C \subset N$ be a maximal d-critical set, with $d \in \{2, \ldots, K\}$, and suppose that $\max(\overline{C}_{K-d+1}) < \min(C)$. Then (5.12) is facet-defining for P if and only if the following conditions hold:*

(i) $\sum_{j \in \overline{C}_{K-d+1}} a_j + \sum_{i \in C_d} a_j - a_{\min(C)} \geq b$

(ii) $\sum_{j \in \overline{C}_{K-d+1}} a_j + \sum_{i \in C_{d-2}} a_j + a_{\max(C)} \geq b$

(iii) $\sum_{j \in \overline{C}_{K-d+1}} a_j + \sum_{i \in C_{d-1}} a_j > b$.

Proof. Suppose that (i)–(iii) hold. Since C is maximal, $\max(C) = n$, and (ii) implies that $\sum_{j \in \overline{C}_{K-d}} a_j + a_i + \sum_{j \in C_{d-1}} a_j \geq b$, for all $i \in \overline{C} \setminus \overline{C}_{K-d+1}$.

Let $y = \frac{b}{\sum_{j \in \overline{C}_{K-d+1} \cup C_{d-1}} a_j}$. The following n points belong to P, are linearly independent, and satisfy (5.12) at equality:

For $i \in C_d$: $x^i_j = 1$ $\forall j \in \overline{C}_{K-d+1}$, $x^i_j = 1$ $\forall j \in C_d \setminus \{i\}$, and $x^i_j = 0$ otherwise; for $i \in C \setminus C_d$: $x^i_j = 1$ $\forall j \in \overline{C}_{K-d+1}$, $x^i_j = 1$ $\forall j \in C_{d-2}$, $x^i_i = 1$, and $x^i_j = 0$ otherwise;

for $i \in \overline{C}_{K-d+1}$: $x^i_j = 1\ \forall j \in \overline{C}_{K-d+1} \setminus \{i\}$, $x^i_i = 1\ \forall j \in C_{d-1}$, $x^i_i = y$, and $x^i_j = 0$ otherwise; for $i \in \overline{C} \setminus \overline{C}_{K-d+1}$: $x^i_j = 1\ \forall j \in \overline{C}_{K-d}$, $x^i_j = 1\ \forall j \in C_{d-1}$, $x^i_i = 1$, and $x^i_j = 0$ otherwise.

Suppose now that (5.12) is facet-defining for P. If (i) does not hold, then $x_{\min(C)} = 1$ for every feasible point satisfying (5.12) at equality. If (ii) does not hold, then $x_n = 0$ for every feasible point satisfying (5.12) at equality. If (iii) does not hold, then every point satisfying (5.12) at equality also satisfies (5.7) at equality. □

Example 5.3.1 (continued) The sets $\{4, 8\}$ and $\{5, 6, 7, 8\}$ are 2-critical and satisfy all conditions of Proposition 5.4.1. Thus, $x_4 + x_8 \leq 1$ and $x_5 + x_6 + x_7 + x_8 \leq 1$ are facet-defining for P.

The set $\{2, 3, 8\}$ is 3-critical and satisfies all conditions of Proposition 5.4.1, so $x_2 + x_3 + x_8 \leq 2$ is facet-defining for P.

Example 5.3.2 (continued) The sets $\{6, 7, 8\}$ and $\{5, 7, 8\}$ are 3-critical and satisfy all conditions of Proposition 5.4.1. Thus, $x_6 + x_7 + x_8 \leq 2$ and $x_5 + x_7 + x_8 \leq 2$ are facet-defining for P.

The sets $\{3, 4, 5, 8\}$ and $\{4, 5, 6, 7, 8\}$ are 4-critical and satisfy all conditions of Proposition 5.4.1. Thus, $x_3 + x_4 + x_5 + x_8 \leq 3$ and $x_4 + x_5 + x_6 + x_7 + x_8 \leq 3$ are facet-defining for P. No subset of N is 2-critical.

5.4.2 Lifted Surrogate Cardinality Constraints

Consider the surrogate cardinality constraint (5.11). If $N' \subsetneq N$, then (5.11) is not facet-defining since it is implied by (5.2) and (5.10). By performing lifting on (5.11), however, we can derive non-trivial facets using the following result.

Proposition 5.4.2. *Let $N' \subset N$, and $\ell \in N'$. Suppose that*

$$\sum_{j \in N' \setminus \{\ell\}} \alpha_j x_j \leq \eta \tag{5.13}$$

is valid for P. Then

$$\sum_{j \in N' \setminus \{\ell\}} \alpha_j x_j + \alpha_\ell x_\ell \leq \eta \tag{5.14}$$

is valid for P as well, where

$$\alpha_\ell \leq \min \left\{ \eta - \textstyle\sum_{j \in N' \setminus \{\ell\}} \alpha_j x_j : x_\ell = 1 \text{ and } x \in S \right\}. \tag{5.15}$$

Proof. We use (5.13) as a seed inequality to lift x_ℓ. We have that

$$\alpha_\ell \leq \min \left\{ \frac{\eta - \sum_{j \in N' \setminus \{\ell\}} \alpha_j x_j}{x_\ell} : x \in S \right\}.$$

Because (5.13) is satisfied by all points of S, we may assume that $x_\ell = 1$ when computing α_ℓ, and (5.15) follows. □

The ability to assume $x_\ell = 1$ simplifies the calculation of α_ℓ because we solve a linear problem instead of a non-linear one. If we further assume that $\alpha_j \geq 0, \forall j \in N' \setminus \{\ell\}$, (and we will from this point on), the procedure can be used iteratively to lift other variables with index in N'.

Example 5.3.1 (continued) Let (5.13) be $x_2 + x_3 + x_4 + x_5 + x_6 + x_7 \leq 3$. We lift x_8 to get $\alpha_8 = 2$ and thus $x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + 2x_8 \leq 3$, which is facet-defining for P.

Example 5.3.2 (continued) Let (5.13) be $x_4 + x_5 + x_6 \leq 4$. We sequentially lift x_7 and x_8 to get $\alpha_7 = \alpha_8 = 2$ and thus $x_4 + x_5 + x_6 + 2x_7 + 2x_8 \leq 4$, which is facet-defining for P.

In our computational tests, we use (5.11) as the seed inequality (5.13), so $\alpha_j = 1, \forall j \in N' \setminus \{\ell\}$, and $\eta = K$. We will show that, by a judicious choice of N', we can apply Proposition 5.4.2 to successfully derive cutting planes for CCOP. One advantage of this approach is that building (5.13) is straightforward and fast once we have N'. The main task, then, is finding an efficient way to compute α_ℓ.

The problem of computing the minimum of the expression in (5.15) is, in fact, a CCOP. Equivalently, we may solve

$$\beta_\ell = \max\left\{\sum_{j\in N'\setminus\{\ell\}} \alpha_j x_j : x_\ell = 1 \text{ and } (5.2), (5.3), (5.4), \text{ and } (5.7) \text{ hold}\right\} \quad (5.16)$$

and choose $\alpha_\ell \leq \eta - \beta_\ell$.

Computing β_ℓ, however, proved to be computationally inefficient, so we developed an approximate lifting procedure, which we now explain. It is easy to see that the assumption that $\alpha_j \geq 0, \forall j \in N' \setminus \{\ell\}$, implies that there exists an optimal solution to (5.16) with $\sum_{j\in N} x_j = K$. Hence, we have that

$$\beta_\ell = \max\left\{\sum_{j\in N'\setminus\{\ell\}} \alpha_j x_j : x_\ell = 1, \sum_{j\in N} x_j = K, \text{ and } (5.2), (5.3), (5.4), (5.7) \text{ hold}\right\}.$$

By relaxing the cardinality constraint, we arrive at a bounded continuous knapsack problem, given by

$$\gamma_\ell = \max\left\{\sum_{j\in N'\setminus\{\ell\}} \alpha_j x_j : x_\ell = 1, \sum_{j\in N} x_j = K, \text{ and (5.2), (5.3), and (5.7) hold}\right\}. \tag{5.17}$$

Since γ_ℓ is an upper bound for β_ℓ, by choosing $\alpha_\ell = \eta - \gamma_\ell$, we will have derived a valid lifting coefficient for (5.14). In case all α_j's are integral, β_ℓ is also integral, and so $\lfloor\gamma_\ell\rfloor$ is a (possibly) tighter upper bound for β_ℓ. We may then choose $\alpha_\ell = \eta - \lfloor\gamma_\ell\rfloor$, and thus get a stronger lifting coefficient for x_ℓ.

In the next section, we illustrate this procedure with an example.

5.5 Separation of Cardinality Cuts

We use relatively simple procedures to separate the two families of inequalities, namely Critical-set and Lifted Surrogate, using Propositions 5.4.1 and 5.4.2. We will collectively call those families *Cardinality cuts*.

Let x^* be the optimal solution to the linear programming relaxation of the current branch-and-cut node, $N^+ = \{j \in N : x_j^* > 0\}$, and suppose that $|N^+| > K$. For each knapsack inequality $i \in M$, we try to find inequalities (5.18) and (5.21) to cut off x^*. Because the procedure is the same for every row i, to simplify our presentation, we assume that i is fixed and drop this index. We also assume that the coefficients of each knapsack satisfy Assumption 5.3.1.

Separation of (5.18) For $d = 2, \ldots, K$, in this order, we repeat the following: Try to find $p \in N^+$ such that

$$\sum_{j \in N_{K-d}} a_j + \sum_{\substack{j=p \\ j \in N^+}}^{p+d-1} a_j \geq b \quad \text{and} \quad \sum_{j \in N_{K-d}} a_j + \sum_{\substack{j=p+1 \\ j \in N^+}}^{p+d} a_j < b.$$

If no such p is found, we move to the next value of d. If p is found, we set $C = \{j \in N^+ : j > p\}$, which d-critical. We then test whether the inequality

$$\sum_{j \in \hat{C}} x_j \leq d - 1 \tag{5.18}$$

cuts off x^*. If it does, we add (5.18) to CPLEX's cut pool and start the separation of (5.21). Otherwise we move to the next value of d. In case we are not successful in deriving (5.18) for any d, we start separation of (5.21). We separate at most 1,000 inequalities (5.18).

Example 5.3.1 (continued) Let $x^* = (\frac{7}{9}, 1, 0, 0, 0, \frac{2}{9}, 1, 0)$, and $d = 2$. Then $p = 2$, $C = \{6, 7\}$, and (5.18) is $x_6 + x_7 + x_8 \leq 1$, which cuts off x^* by $\frac{2}{9}$.

Separation of (5.21) Let $N' = \{j \in N : x_j^* > 0.8, j = 1, \ldots, K-1, \text{ or } x_j^* > 0, j = K, \ldots, n\}$. We try to sequentially lift all variables with index in $\{j \in N' : x_j^* > 0.8\}$ using the following iterative procedure:

1. Set $\alpha_j = 1, \forall j \in N'$.

2. Let $N'' = \{j \in N' : x_j^* > 0.8 \text{ and } \alpha_j = 1\}$. If $N'' = \emptyset$, go to step 4. Otherwise, let $\ell = \max(N'')$, and let

$$\sum_{j \in N' \setminus \{\ell\}} \alpha_j x_j \leq K \tag{5.19}$$

be the seed inequality for the lifting procedure.

3. Compute

$$\begin{aligned} \alpha_\ell &= K - \Big\lfloor \max \big\{ \textstyle\sum_{j \in N' \setminus \{\ell\}} \alpha_j x_j : \sum_{j \in N \setminus \{\ell\}} a_j x_j \geq b - a_\ell, \\ &\qquad 0 \leq x_j \leq 1, \forall j \in N \setminus \{\ell\}, \textstyle\sum_{j \in N \setminus \{\ell\}} x_j = K - 1 \big\} \Big\rfloor \end{aligned} \tag{5.20}$$

If $\alpha_\ell \leq 1$, go to step 4. Otherwise, update the value of α_ℓ and go to step 2.

4. Test whether the inequality

$$\sum_{j \in N'} \alpha_j x_j \leq K \tag{5.21}$$

cuts off x^*. If it does, add (5.21) to CPLEX's cut pool. Otherwise quit. We separate at most 10 inequalities (5.21).

Example 5.3.2 (continued) Let $x^* = (\frac{7}{10}, 0, 0, 1, 1, 0, 1, \frac{3}{10}) \Rightarrow N' = \{4, 5, 7, 8\}$.

Iteration 1 $N'' = \{4, 5, 7\}$, $\ell = 7$, (5.19) is $x_4 + x_5 + x_8 \leq 4$, and

$$\begin{aligned}\alpha_7 &= 4 - \Big\lfloor \max\big\{x_4 + x_5 + x_8 : 11x_1 + 10x_2 + 8x_3 + 7x_4 + 6x_5 + 5x_6 + x_8 \geq 21,\\ &\quad 0 \leq x_j \leq 1, \forall j \in N \setminus \{7\}, \textstyle\sum_{j \in N \setminus \{7\}} x_j = 3\big\}\Big\rfloor\\ &= 4 - \lfloor 2.3 \rfloor = 2.\end{aligned}$$

Iteration 2 $N'' = \{4, 5\}$, $\ell = 5$, (5.19) is $x_4 + 2x_7 + x_8 \leq 4$, and

$$\begin{aligned}\alpha_5 &= 4 - \Big\lfloor \max\big\{x_4 + 2x_7 + x_8 : 11x_1 + 10x_2 + 8x_3 + 7x_4 + 5x_6 + 2x_7 + x_8 \geq 18,\\ &\quad 0 \leq x_j \leq 1, \forall j \in N \setminus \{5\}, \textstyle\sum_{j \in N \setminus \{5\}} x_j = 3\big\}\Big\rfloor\\ &= 4 - \lfloor 3.3 \rfloor = 1.\end{aligned}$$

We obtain the inequality $x_4 + x_5 + 2x_7 + x_8 \leq 4$, which cuts off x^* by $\frac{3}{10}$.

In all of the above procedures, a *minimum violation* had to be met by x^* in order for an inequality to be given to CPLEX. This value was chosen based on preliminary tests, and was set to 0.10. We performed separation at every node of the tree in all instances we tested.

5.6 Computational Experience

In this section, we describe the instances we used in our computational tests, as well as the computer platform and how the tests were conducted.

5.6.1 Instances

The instances we tested were created with the intention of being difficult despite their small number of variables and constraints. We let $(m, n) = (10, 200), (10, 300),$

and $(20, 200)$, and, for each n, $K \in \{\lfloor \frac{n}{12} \rfloor + 1, \ldots, \lfloor \frac{n}{3} \rfloor\}$. (The reason for choosing this interval for K is that, if $K \leq \lfloor \frac{n}{12} \rfloor$, then the instance, as we generated it, is infeasible, and, for $K > \lfloor \frac{n}{3} \rfloor$, our instances become too easy.) For every combination of m, n and K, we used 5 different objective functions and set of constraints, for a total of 875 instances. We used the following procedure to generate the instances coefficients.

Let $v_j \sim \text{Unif}\{0, \ldots, 10\}$, $j \in N$, and $v_{ij} \sim \text{Unif}\{0, \ldots, 20\}$, $i \in M$ and $j \in N$. Then

$$c_j = -n + j + \left\lfloor \left(\frac{j}{10} \right)^2 \right\rfloor + r_j,$$

$$a_{ij} = 3n - 3j + r_{ij},$$

and

$$b_i = \sum_{j=1}^{\lfloor \frac{n}{12} \rfloor} s_{ij},$$

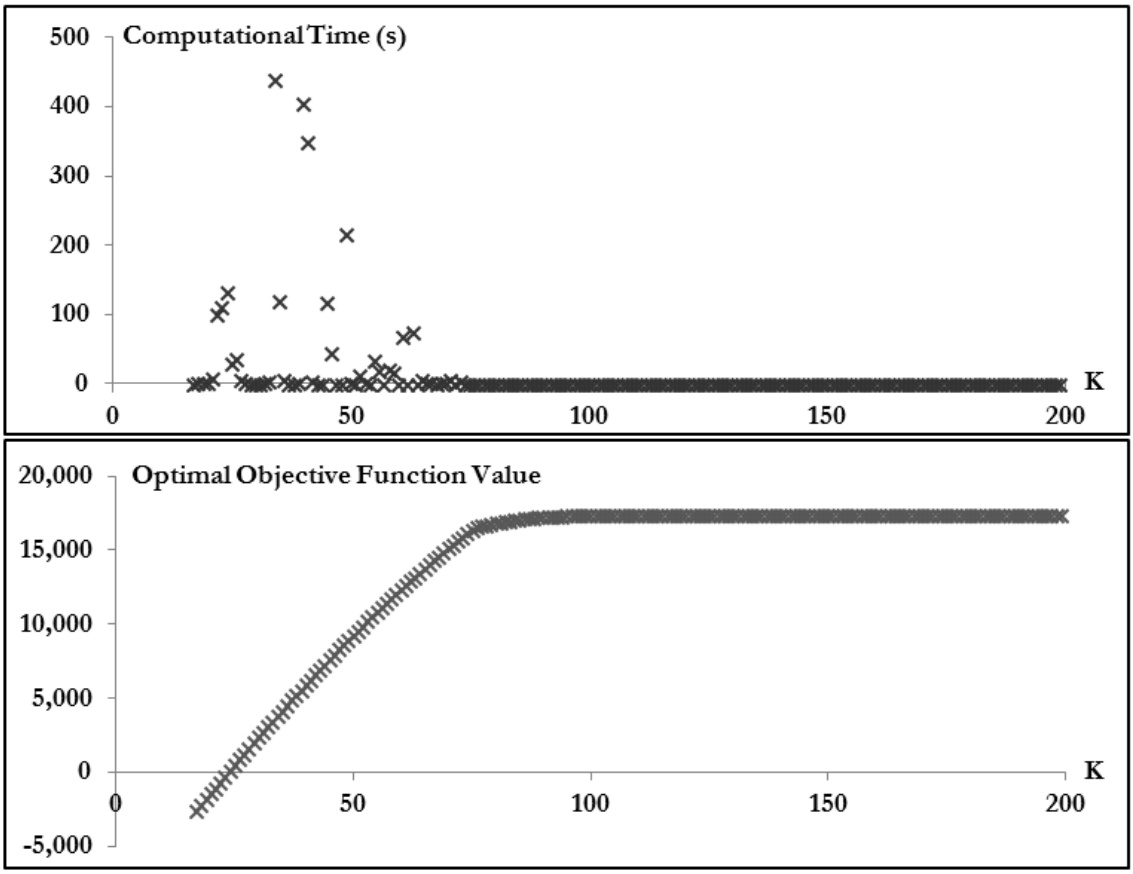
where $(s_{i1}, \ldots, s_{in})$ is the array $(a_{i1}, \ldots, a_{in})$ sorted in decreasing order, for all $i \in M$.

Notice how the expected values of c_j increase with j, whereas the expected values of a_{ij} decrease with j. This configuration forces variables with a large values of j (and, consequently, small values of a_{ij}) to be positive. On the other hand, because of the knapsack constraints, variables with a small value of j (large value of a_{ij}) also need to be positive in order to guarantee feasibility. Thus, finding an optimal solution

becomes challenging for many instances, as we will see further in this section.

The computational time of CCOP instances as a function of K can be rather unpredictable. Figure 5.1 shows the time, in seconds, needed to solve an instance with $m = 20$ and $n = 200$, along with its corresponding optimal objective function value, as a function of K. Here, we used CPLEX with default settings.

Figure 5.1. Computational Time and Optimal Objective Function Value of an Instance with $m = 20$ and $n = 200$, as a Function of K



The peak at $K = 24$ can be partially explained by the optimal value of the objective function being close to 0 (the smallest the absolute value of the objective function,

the harder it is to close the integrality gap). However, for other peaks, such as in $K = 34$ and $K = 49$, we do not have a plausible explanation, and this could be an area of further research. This "erratic" behavior was observed for all sizes of instances tested.

The monotonicity of the optimal objective function value as a function of K is clear: allowing more variables to be positive cannot decrease the value of the objective function. Starting around $K = 75$, there is a "stabilization" of the optimal objective function's value, and after $K = 99$, the value does not change at all. This is due to the fact that this instance's objective function has 101 negative and 99 positive coefficients, with 21 of the latter being under the value of 50.

5.6.2 Tests Conducted

We performed our computational tests in the Texas Tech High Performance Computing Center's Intel Xeon E5450 3.0GHz CPU with 16GB RAM nodes. We used the callable libraries of CPLEX 12.2.0.0, which we ran on a single thread. We allowed a maximum computational time of 18,000 seconds of CPU for solving each instance.

We evaluated CPLEX's performance when solving CCOP to proven optimality. We compared the performance that results when:

- CPLEX's internal features are turned ON or OFF
- Cardinality cuts (i.e., inequalities (5.18) and (5.21) are used or not

The CPLEX features that we consider are: *Presolve*, *Heuristics*, and *MIP cuts*. In our tests, we turn these parameters ON and OFF. To simplify our presentation, when considering a strategy we will use the letter `p` to denote Presolve ON, `h` to

denote Heuristics ON, `m` to denote MIP cuts ON, and `c` to denote Cardinality cuts ON, always in this order. To denote a feature being OFF, we will use the letter `n`. Table 5.1 describes all possible strategies used.

Table 5.1. Features and Strategies

Strategy	Presolve	Heuristics	MIP cuts	Cardinality cuts
`phmc`	ON	ON	ON	ON
`phmn`	ON	ON	ON	OFF
`phnc`	ON	ON	OFF	ON
`phnn`	ON	ON	OFF	OFF
`pnmc`	ON	OFF	ON	ON
`pnmn`	ON	OFF	ON	OFF
`pnnc`	ON	OFF	OFF	ON
`pnnn`	ON	OFF	OFF	OFF
`nhmc`	OFF	ON	ON	ON
`nhmn`	OFF	ON	ON	OFF
`nhnc`	OFF	ON	OFF	ON
`nhnn`	OFF	ON	OFF	OFF
`nnmc`	OFF	OFF	ON	ON
`nnmn`	OFF	OFF	ON	OFF
`nnnc`	OFF	OFF	OFF	ON
`nnnn`	OFF	OFF	OFF	OFF

Given a CCOP instance, we will say that it was *solved* when it was solved to proven optimality within our prescribed time limit, and we consider a strategy to be *more efficient* than another when the computational time required to solve the instance was smaller for that strategy.

Finally, we note that CPLEX has two search strategies, namely *Traditional Branch-and-Cut* and *Dynamic Search*. Of the two, only Traditional Branch-and-Cut allows the user to add their own cutting planes, so we only tested this option. Once this

strategy is chosen, CPLEX's default setting have Presolve, Heuristics, and MIP cuts are all turned ON, and thus is represented by the strategy `phmn`.

5.7 Analysis of Results

We now report and analyze the results of our computational tests. We consider four outputs: number of instances solved, solution time, number of nodes used in the branch-and-cut tree, and number of cuts applied.

5.7.1 Number of Instances Solved

We start our analysis with the *number of instances solved* using each strategy. The results are shown in Table 5.2.

Table 5.2. Number of Instances Solved and Strategies

Strategy	# Solved
`phmc`	862
`phmn`	847
`phnc`	867
`phnn`	860
`pnmc`	865
`pnmn`	847
`pnnc`	869
`pnnn`	862
`nhmc`	871
`nhmn`	867
`nhnc`	868
`nhnn`	863
`nnmc`	873
`nnmn`	869
`nnnc`	867
`nnnn`	860

The number of tests run with each strategy was 875, and the table shows that the great majority of the instances were solved. This is because it is not clear how one can create a difficult instance by varying the value of K, and so we ended up creating many instances that were easy. Nevertheless, there is a significant difference, for example, between strategies `phmn` and `nnmc`. The latter solved 26 instances more than the former, and was not able to solve only 2 instances of our set.

In terms of number of instances solved, Table 5.2 indicates that Presolve was not helpful (6,897 were solved with vs. 6,938 without it), Heuristics and MIP cuts did not play a significant role, and Cardinality cuts were very helpful (6,942 were solved with them vs. 6,875 without them).

5.7.2 Solution Time

We now turn our attention to the *solution time* of the instances. The summarized results are shown in Table 5.3. We note that, in this table, the times shown are averages that take into account only the instances solved (and shown in Table 5.2).

Here, the number associated with `nnmc` stands out, not only because it is the lowest, but also due to the fact that it is an average taken over a bigger number of instances.

In the next pages we will consider the use of each of the three CPLEX features, namely Presolve, Heuristics, and MIP cuts, plus the use of Cardinality cuts, and will see how they affected the solution time of our instances.

5.7.2.1 Presolve

Table 5.4 compares the average solution time of instances when they are solved with and without *Presolve*, having everything else constant. The column "# Inst."

Table 5.3. Average Solution Times (s) and Strategies

Strategy	Time(s)
phmc	346
phmn	423
phnc	204
phnn	455
pnmc	277
pnmn	370
pnnc	300
pnnn	416
nhmc	216
nhmn	220
nhnc	391
nhnn	445
nnmc	180
nnmn	205
nnnc	363
nnnn	400

shows the number of instances considered in the comparison, and values in **bold** indicate the better strategy.

Table 5.4. Strategies and Average Solution Times (s) with and without Presolve

Strategies	# Inst.	Avg. Time
phmc vs. nhmc	858	312 vs. **167**
phmn vs. nhmn	845	414 vs. **136**
phnc vs. nhnc	861	**192** vs. 318
phnn vs. nhnn	855	409 vs. **399**
pnmc vs. nnmc	864	269 vs. **143**
pnmn vs. nnmn	846	368 vs. **146**
pnnc vs. nnnc	861	**246** vs. 324
pnnn vs. nnnn	855	380 vs. **361**

The table shows that having Presolve ON was not helpful in 6 out of 8 comparisons. In fact, it was only helpful whenever MIP cuts were OFF and Cardinality cuts were ON. With MIP cuts ON, Presolve increased the solution time considerably. With MIP cuts and Cardinality Cuts both OFF, Presolve did not have a substantial effect.

5.7.2.2 Heuristics

Table 5.5 compares the average solution time of instances when they are solved with and without *Heuristics*, having everything else constant.

Table 5.5. Strategies and Average Solution Times (s) with and without Heuristics

Strategies	# Inst.	Avg. Time
`phmc` vs. `pnmc`	857	308 vs. **246**
`phmn` vs. `pnmn`	840	361 vs. **318**
`phnc` vs. `pnnc`	863	**189** vs. 250
`phnn` vs. `pnnn`	854	387 vs. **359**
`nhmc` vs. `nnmc`	870	197 vs. **162**
`nhmn` vs. `nnmn`	866	206 vs. **186**
`nhnc` vs. `nnnc`	864	347 vs. **344**
`nhnn` vs. `nnnn`	857	392 vs. **355**

Heuristics was not helpful in 7 out of 8 comparisons. Its effect, however, was not major most of the time, although it was noticeable.

5.7.2.3 MIP Cuts

Table 5.6 compares the average solution time of instances when they are solved with and without *MIP cuts*, having everything else constant.

Here, we notice a major influence of the MIP cuts on the average solution time when Presolve is OFF. In the last four rows of the table, the reductions are in the

Table 5.6. Strategies and Average Solution Times (s) with and without MIP Cuts

Strategies	# Inst.	Avg. Time
phmc vs. phnc	855	320 vs. **159**
phmn vs. phnn	839	408 vs. **327**
pnmc vs. pnnc	860	**270** vs. 274
pnmn vs. pnnn	841	336 vs. **318**
nhmc vs. nhnc	867	**200** vs. 386
nhmn vs. nhnn	857	**190** vs. 404
nnmc vs. nnnc	856	**151** vs. 355
nnmn vs. nnnn	856	**175** vs. 377

order of 50%, which is very significant. However, with Presolve ON, MIP cuts had either a negative impact, or almost no influence at all.

5.7.2.4 Cardinality Cuts

Table 5.7 compares the average solution time of instances when they are solved with and without *Cardinality cuts*, having everything else constant.

Table 5.7. Strategies and Average Solution Times (s) with and without Cardinality Cuts

Strategies	# Inst.	Avg. Time
phmc vs. phmn	841	**216** vs. 386
phnc vs. phnn	852	**178** vs. 412
pnmc vs. pnmn	843	**187** vs. 333
pnnc vs. pnnn	857	**254** vs. 405
nhmc vs. nhmn	865	**184** vs. 212
nhnc vs. nhnn	858	**349** vs. 411
nnmc vs. nnmn	868	**155** vs. 204
nnnc vs. nnnn	854	**317** vs. 380

The table shows a huge positive effect of Cardinality cuts, in every single com-

parison. Although the reductions are not as big as in the case of SOS1, they are still extremely significant, varying from 13% to 57%. These results indicate that, for our instances, the use of Cardinality cuts was indispensable for solving them more quickly.

5.7.3 Comparison Among All Strategies

Out of the 875 instances created, 813 were solved by all strategies. In Table 5.8 we show the average solution time for each strategy.

Table 5.8. Average Solution Times (s) for the Instances Solved by All Strategies

Strategy	Time(s)
`phmc`	139
`phmn`	263
`phnc`	72
`phnn`	223
`pnmc`	121
`pnmn`	222
`pnnc`	91
`pnnn`	189
`nhmc`	68
`nhmn`	99
`nhnc`	133
`nhnn`	197
`nnmc`	44
`nnmn`	81
`nnnc`	127
`nnnn`	191

The table indicates that the best strategy to solve our instances was `nnmc`, i.e., with Presolve and Heuristics both OFF, and MIP cuts and Cardinality Cuts both ON. This strategy was considerably better than all others. It reduced the solution

time of the second best strategy, `phnc`, by 39%, and of CPLEX's default settings, represented by `phmn`, by 83%.

5.7.4 Number of Nodes

We now consider the number of nodes in the branch-and-cut tree necessary to solve our instances. Because this is a secondary measure of efficiency of a strategy, we consider only the 813 instances used to build Table 5.8, which are the ones that were solved with all strategies.

Table 5.9. Average Number of Nodes for Instances Solved by All Strategies

Strategy	Nodes
phmc	167,244
phmn	238,462
phnc	438,416
phnn	1,633,949
pnmc	163,507
pnmn	242,721
pnnc	612,398
pnnn	1,463,607
nhmc	172,403
nhmn	378,218
nhnc	465,622
nhnn	1,017,060
nnmc	162,404
nnmn	433,530
nnnc	477,283
nnnn	1,072,543

For all strategies with MIP cuts and Cardinality cuts both OFF, the average number of nodes was over 1 million, while strategies that had MIP cuts and Cardinality

cuts both ON had an average number of node under 200,000. This is evidence that cutting planes can reduce drastically the number of nodes of a branch-and-cut tree.

5.7.5 Types of Cuts Used

We now report the types and average number of cutting planes used by CPLEX in our tests. For Cardinality cuts, recall that we set a limit of 1,000 cuts of type (5.18) and 10 cuts of type (5.21) to be given to CPLEX. For the internal CPLEX cuts, no limit was previously set. Again, we consider only the 813 instances that were solved by all strategies. The results are shown in Table 5.10.

In this table, the column "Used" shows the number of Cardinality cuts that were actually used by CPLEX. This number may not be equal to the sum of the columns (5.18), (5.21) because, for example, our separation algorithm may generate identical cuts, in which case at most one cut is counted as used. Another reason for this is the fact that CPLEX has a cut filter that may reject a cut if, for instance, it considers the cut too dense, or less effective than other cuts [9]. We do not have information on exactly how many Cardinality cuts of each type were individually used, but only the combined total. We note that cuts of the type MIR, Cover, Implied Bound, and Zero-half were also used, but, because their averages were below 1 for every strategy, we omitted them from the tables.

In Table 5.10, we see that having Presolve ON made CPLEX generate many more internal cuts, in particular Cover cuts. Whenever MIP cuts were ON, the number of cuts (5.18) decreased. This is because these cuts, which are the critical-set inequalities, have a structure that is similar to cover inequalities. Hence, it is not a surprise that they are, in a sense, interchangeable. With Presolve OFF, the amount

Table 5.10. Strategies and Average Number of MIP Cuts Using the LOG Formulation for Instances Solved by All Strategies with Both LOG and SOS

Strategy	Cover	Flow Cover	Gomory	(5.18)	(5.21)	Used
phms	170	1	1	470	9	92
phmn	264	2	1	—	—	—
phns	—	—	—	554	9	101
pnms	174	1	1	472	9	92
pnmn	267	1	1	—	—	—
pnns	—	—	—	548	9	99
nhms	21	0	1	470	9	87
nhmn	24	0	1	—	—	—
nhns	—	—	—	560	9	95
nnms	4	0	1	470	9	88
nnmn	5	1	1	—	—	—
nnns	—	—	—	554	9	94

of internal CPLEX cuts was very low, which, in this case, seemed to be beneficial, since the strategies without Presolve performed better, on average (see tables 5.4 and 5.8).

The fraction of used Cardinality cuts was smaller than what was observed for SOS1 and SOS2 cuts. Here, CPLEX used around 18% of the cuts derived. We believe that this is due to the great amount of identical cuts generated by our separation procedures.

5.8 Summary of Conclusions and Further Research

In this chapter, we studied the polyhedral description of CCOP (Cardinality Constrained Optimization Problem) and developed valid inequalities that can be used within branch-and-cut. We showed simple separation procedures that can be used to derive such inequalities, and that they can improve our ability to solve instances

of CCOP substantially.

As with SOS1 and SOS2, our main measure of efficiency was the solution time of our instances. Presolve has a negative effect most of the time, and was only helpful with MIP cuts were OFF and Cardinality cuts ON. Heuristics also had a negative impact, but smaller than Presolve. MIP cuts were very helpful in decreasing the solution time of our instances, and so were Cardinality cut, which had an even better effect than MIP cuts.

We also reported the average number of nodes and types of cutting planes used. Overall, the best strategy was having Presolve and Heuristics both OFF, and MIP and Cardinality cuts both ON.

We focused on inequalities that have only nonnegative coefficients, but there exist valid inequalities for CCOP with negative coefficients as well. For example, the inequality

$$-11x_1 - 10x_2 + x_4 + 11x_6 + 11x_7 + 12x_8 \leq 1$$

is facet-defining for the polyhedron of Example 5.3.1. Developing new inequalities valid for P, or even generalizing the ones presented here, can be an area of further research. Also, dropping Assumption 5.3.4 is a natural continuation of this work.

We also suggest as topics of research the applications that involve cardinality constraints described in [5]. The literature on such problems is very scarce.

CHAPTER 6
CONCLUSIONS AND FURTHER RESEARCH

In this dissertation, we considered three common types of constraints used in MIP models: SOS1, SOS2, and cardinality. We studied cutting planes specific for each type of constraint, and applied those cutting planes within branch-and-cut to solve difficult MIP instances.

For SOS1, we generalized valid inequalities developed by de Farias et. al. [12]; for SOS2, we used inequalities recently published by Zhao and de Farias [29]; and for cardinality, we generalized results given by Bienstock [4] and developed a lifting procedure to derive new inequalities.

Additionally, we studied the effect of features that are present in most commercial solvers, such as preprocessing, "standard" cutting planes, and heuristics, on the ability to solve our instances.

Our results indicate that the use of cutting planes specific for the constraints we studied can be very effective in enhancing our ability to solve MIP instances. Regarding the other features mentioned above, we obtained mixed results: in some cases they were useful, in others they were not.

As topics of further research, we suggest

- Deriving new cutting planes or generalizing the ones presented here
- Studying other types of recurring constraints in MIP

Finally, we encourage the use of the cutting planes studied here in real-life applications.

BIBLIOGRAPHY

[1] E. Balas. Facets of the Knapsack Polytope. *Mathematical Programming*, 8(1):146–164, 1975.

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

[3] E. M. L. Beale and J. A. Tomlin. Special Facilities in a General Mathematical Programming System for Non-Convex Problems Using Ordered Sets of Variables. *OR*, 69:447–454, 1970.

[4] D. Bienstock. Computational Study of a Family of Mixed-Integer Quadratic Programming Problems. *Mathematical Programming*, 74(2):121–140, 1996.

[5] S. Boyd. L1-Norm Methods for Convex Cardinality Problems. *Lecture Notes for EE364b, Stanford University. Available at http://www.stanford.edu/class/ee364b/lectures/l1_slides.pdf*, 2007.

[6] M. Bruglieri, M. Ehrgott, H. W. Hamacher, and F. Maffioli. An Annotated Bibliography of Combinatorial Optimization Problems with Fixed Cardinality Constraints. *Discrete Applied Mathematics*, 154(9):1344–1357, 2006.

[7] T. Christof, A. Löbel, and M. Stoer. PORTA – POlyhedron Representation Transformation Algorithm. *Software package, available for download at http://www.iwr.uni-heidelberg.de/groups/comopt/software/PORTA*, 1997.

[8] V. Chvatal. *Linear Programming*. Macmillan, 1983.

[9] CPLEX, IBM ILOG. IBM Software Group. *User-Manual CPLEX 12*, 2011.

[10] H. Crowder, E. L. Johnson, and M. Padberg. Solving Large-Scale Zero-One Linear Programming Problems. *Operations Research*, 31(5):803–834, 1983.

[11] G. B. Dantzig. On the Significance of Solving Linear Programming Problems with Some Integer Variables. *Econometrica, Journal of the Econometric Society*, pages 30–44, 1960.

[12] I. de Farias, E. L. Johnson, and G. L. Nemhauser. Facets of the Complementarity Knapsack Polytope. *Mathematics of Operations Research*, 27(1):210–226, 2002.

[13] I. R. de Farias Jr, E. Kozyreff, R. Gupta, and M. Zhao. Branch-and-Cut for Separable Piecewise Linear Optimization and Intersection with Semi-Continuous Constraints. *Mathematical Programming Computation*, 5(1):75–112, 2013.

[14] I. R. de Farias Jr and G. L. Nemhauser. A Polyhedral Study of the Cardinality Constrained Knapsack Problem. *Mathematical programming*, 96(3):439–467, 2003.

[15] R. Gomory. An Algorithm for the Mixed Integer Problem. Technical report, DTIC Document, 1960.

[16] T. Ibaraki. The Use of Cuts in Complementary Programming. *Operations Research*, 21(1):353–359, 1973.

[17] E. L. Johnson and M. W. Padberg. A Note of the Knapsack Problem with Special Ordered Sets. *Operations Research Letters*, 1(1):18–22, 1981.

[18] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[19] H. M. Markowitz and A. S. Manne. On the Solution of Discrete Programming Problems. *Econometrica: journal of the Econometric Society*, pages 84–110, 1957.

[20] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley New York, 1988.

[21] M. W. Padberg. Technical Note – A Note on Zero-One Programming. *Operations Research*, 23(4):833–837, 1975.

[22] A. F. Perold. Large-Scale Portfolio Optimization. *Management Science*, 30(10):1143–1160, 1984.

[23] S. Vasantharajan and A. Cullick. Well Site Selection Using Integer Programming Optimization. In *Third Annual Conference of the Intl. Assn. for Mathematical Geology, pages=421–426, year=1997*.

[24] J. P. Vielma and G. L. Nemhauser. Modeling Disjunctive Constraints with a Logarithmic Number of Binary Variables and Constraints. *Mathematical programming*, 128(1-2):49–72, 2011.

[25] L. A. Wolsey. Technical Notes – Facets and Strong Valid Inequalities for Integer Programs. *Operations Research*, 24(2):367–372, 1976.

[26] L. A. Wolsey. Valid Inequalities for 0–1 Knapsacks and MIPs with Generalised Upper Bound Constraints. *Discrete Applied Mathematics*, 29(2):251–261, 1990.

[27] E. Zemel. Lifting the Facets of Zero–One Polytopes. *Mathematical Programming*, 15(1):268–277, 1978.

[28] E. Zemel. The Linear Multiple Choice Knapsack Problem. *Operations Research*, 28(6):1412–1423, 1980.

[29] M. Zhao and I. R. de Farias Jr. The Piecewise Linear Optimization Polytope: New Inequalities and Intersection with Semi-Continuous Constraints. *Mathematical Programming*, pages 1–39, 2012.