5-2023

Exploiting Symmetry in Linear and Integer Linear Programming

Ethan Jedidiah Deakins
Industrial and Systems Engineeering, edeakins@vols.utk.edu

To the Graduate Council:

I am submitting herewith a dissertation written by Ethan Jedidiah Deakins entitled "Exploiting Symmetry in Linear and Integer Linear Programming." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Industrial Engineering.

James Ostrowski, Major Professor

We have read this dissertation and recommend its acceptance:

James Ostrowski, Hugh Medal, Mingzhou Jin, Jean-Paul Watson

Accepted for the Council:

Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:
I am submitting herewith a thesis written by Ethan Jedidiah Deakins entitled "Exploiting Symmetry in Linear and Integer Linear Programming." I have examined the final paper copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Industrial & Systems Engineering.

Jim Ostrowski, Major Professor

We have read this thesis
and recommend its acceptance:

Mingzhou Jin

Hugh Medal

James Ostrowski

Jean-Paul Watson

Accepted for the Council:

Dixie Thompson

Vice Provost and Dean of the Graduate School

To the Graduate Council:
I am submitting herewith a thesis written by Ethan Jedidiah Deakins entitled "Exploiting Symmetry in Linear and Integer Linear Programming." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Industrial & Systems Engineering.

Jim Ostrowski, Major Professor

We have read this thesis
and recommend its acceptance:

Mingzhou Jin

Hugh Medal

James Ostrowski

Jean-Paul Watson

Accepted for the Council:

Dixie Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Exploiting Symmetry in Linear and Integer Linear Programming

A Dissertation Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Ethan Jedidiah Deakins

May 2023

Dedicated to my late father, Timothy Deakins, and mother, Melissa Deakins, who showed me the value of hard work and perseverance.

Acknowledgements

I would like to express my deepest gratitude to my advisor, Professor James Ostrowski, for all his support and guidance. He played a pivotal role in convincing me to continue my education in graduate school. I would like to thank Dr. Bernard Knueven and Dr. Tony Rodgriguez for their friendship and for being role models early on in graduate school. They showed me the dedication required to finish this degree and how to enjoy the ride at the same time. I would also like to thank Professors Mingzhou Jin and Hugh Medal and Dr. Jean-Paul Watson for serving on my dissertation committee.

I would like to express my appreciation to the University of Tennessee, the Nuclear Engineering University Program, and the U.S. Department of Energy, for providing funding for my graduate studies and the work in this dissertation.

I would like to thank my lovely wife, Kelsie Joy Deakins. Her love and support during my graduate schooling knew no rivals. I truly would not have been able to complete this degree without her by my side through thick and thin.

Finally, I would like to thank all the wonderful friends I have made these past four years: Christopher Muir, Amelia McIlvenna, Jonathan Schrock, Lorna Treffert, Najmaddin Akhundov, Marzieh Bakhsi, Rebekah Herman, Moises and Rachel Ponce, Christopher Ginart, and Tim Gallacher. I have benefited greatly from their friendship and encouragement.

Abstract

This thesis explores two algorithmic approaches for exploiting symmetries in linear and integer linear programs. The first in Chapter 1 is *orbital crossover*, a novel method of crossover designed to exploit symmetry in linear programs. Symmetry has long been considered a curse in combinatorial optimization problems, but significant progress has been made. Up until recently, symmetry exploitation in linear programs was not worth the cost of symmetry detection. However, recent results involving a generalization of symmetries, equitable partitions, has made the cost of symmetry detection much more manageable.

The motivation for orbital crossover is that many highly symmetric integer linear programs exist, and thus, solving symmetric linear programs is of major interest in order to efficiently solve symmetric integer linear programs. The results of this work indicate that a specialized linear programming algorithm that exploits symmetry is likely to be useful in the toolbox of linear programming solvers.

The second algorithm is *orbital cut generation*. The main issue brought forward by symmetric integer linear programs is multiple symmetric solutions having an equivalent objective value. This massively increases the search space for algorithms such as branch and bound or branch and cut. Orbital cut generation aims to tackle the issues of multiple equivalent symmetric solutions using symmetrically equivalent cutting planes.

Chapter 2 shows how to effectively exploit symmetry in integer linear programs by generating symmetrically equivalent cutting planes that remove all symmetric solutions in one go. Further, the method is strengthened using symmetry to aggregate integer linear

programs and generate cutting planes in aggregate spaces before lifting them to the original problem.

Table of Contents

1 Orbital Crossover **1**

1.1 Nomenclature . . . 2

1.1.1 Symmetries . . . 2

1.1.2 Partitions . . . 2

1.1.3 Linear Programs . . . 3

1.2 Introduction . . . 3

1.3 Background: Symmetry and EPs . . . 5

1.3.1 Partitions and Restrictions . . . 5

1.3.2 Symmetries and Fractional Symmetries . . . 6

1.3.3 Orbits and Generalizations . . . 7

1.3.4 Stabilizers and Generalizations . . . 8

1.4 Orbital Crossover . . . 9

1.4.1 Creating $ALP(A, b, c, \mathcal{P})$ . . . 9

1.4.2 Lifting Solutions from $ALP(A, b, c, \mathcal{P})$ to $ELP(A, b, c, \mathcal{P})$ . . . 10

1.4.3 Arriving at an optimal BFS to $LP(A, b, c)$ . . . 12

1.4.4 Reducing the number of r variables . . . 13

1.5 Iterative Lifting . . . 14

1.6 Benchmarks . . . 17

1.6.1 Implementation . . . 17

1.6.2 Test Sets, Methods, and setup . . . 17

1.6.3 Results . . . 19

1.7 Discussion . . . 22

2 Orbital Cut Generation **23**

2.1 Nomenclature . . . 23

2.1.1 Symmetries . . . 23

2.1.2 Partitions . . . 23

2.1.3 Linear and Integer Linear Programs . . . 24

2.2 Introduction . . . 25

2.3 Symmetry in $ILP(A, b, c)$ . . . 27

2.3.1 Symmetries . . . 27

2.3.2 Orbits and Orbital Partitions . . . 28

2.3.3 Stabilizers . . . 29

2.4 Orbital Cut Generation . . . 29

2.4.1 Generating Symmetric Valid cuts for $ILP(A, b, c)$ . . . 31

2.5 Aggregation . . . 35

2.5.1 Reducing the Dimension of $ILP(A, b, c)$ . . . 35

2.5.2 Lifting a Valid cut . . . 36

2.5.3 Iterative Lifting . . . 40

2.6 Implementation . . . 43

2.6.1 Generating Symmetric Cutting Planes . . . 43

2.6.2 Exploiting Stabilizer Subgroups . . . 44

2.7 Benchmarks . . . 45

2.7.1 Test Sets, Methods, and setup . . . 45

2.7.2 Results . . . 46

2.8 Discussion . . . 49

Appendices **56**

A Orbital Crossover Implementation **57**

A.1 Nomenclature . . . 57

A.1.1 Input Data 57
A.1.2 Output Data 57
A.2 Implementation Details 59

Vita **69**

List of Tables

1.1 Benchmark Results for the **HS-COV-COD** Instances 20
1.2 Benchmark Results for the **MIPLIB 2017** Instances 21

2.1 Bounds for SA, SA$^+$, SCG, and OCG 47
2.2 IP Gap (%) for LPR, SCG, and OCG 48

List of Figures

2.1 A Cutting Plane on the Optimal Face of $LP(A, b, c)$ 30
2.2 Symmetric Cutting Planes on the Optimal Face of $LP(A, b, c)$ 30

Chapter 1

Orbital Crossover

This chapter and Appendix A are based on a manuscript prepared for publication by Ethan Deakins, Bernard Knueven, and Jim Ostrowski:

Deakins, E., Knueven, B., and Ostrowski, J. (2023). Orbital Crossover. *Submitted*

Authors Ostrowski and Knueven proposed the method. Authors Deakins and Ostrowski developed the proofs of Theorems 1.2 and 1.3. Author Deakins developed the source code, conducted all computational experiments, and drafted the manuscript. Author Ostrowski edited the manuscript. A preprint of the paper is available at https://optimization-online.org/2022/12/orbital-crossover/

In this chapter we present *orbital crossover*. We use a generalization of symmetries, equitable partitions, to aggregate and solve a linear program. Then we show how to use the equitable partition as a guide to disaggregate a solution to a vertex. We extend orbital crossover to an iterative procedure allowing us to conduct the crossover method in smaller dimensions. We demonstrate the effectiveness of orbital crossover against multiple linear programming algorithms using a variety of linear programming instances.

1.1 Nomenclature

1.1.1 Symmetries

$\mathcal{G}$	Symmetry group of an integer linear program.
G	Formulation group of an integer linear program.
$\mathcal{F}$	Set of Fractional symmetries of a linear program.
P_π	Permutation/doubly stochastic matrix acting on the columns of an integer linear program.
P_σ	Permutation/doubly stochastic matrix acting on the rows of an integer linear program.
$\mathcal{S}^n$	Set of permutation matrices of dimension $n \times n$.
$\mathcal{S}^m$	Set of permutation matrices of dimension $m \times m$.
$\mathcal{D}^n$	Set of doubly stochastic matrices of dimension $n \times n$.
$\mathcal{D}^m$	Set of doubly stochastic matrices of dimension $m \times m$.
$orb(v, \mathcal{G})$	Operator giving the orbit of a vector v with respect to a group $\mathcal{G}$.
$stab(v, \mathcal{G})$	Operator giving the stabilizer subgroup of a group $\mathcal{G}$ with respect to a vector v.
$iso(v, \mathcal{F})$	Operator that isolates a vector with respect to $\mathcal{F}$.

1.1.2 Partitions

$\mathcal{O}$	Orbital partition of the variables (and constraints) of an integer linear program.
$\mathcal{P}$	Equitable partition of the variables (and constraints) of a linear program.
$\mathcal{V}$	Partition of the variables of a linear or integer linear program.
$\mathcal{C}$	Partition of the constraints of a linear or integer linear program.
O_k	The k^{th} set of an orbital partition $\mathcal{O}$.
P_k	The k^{th} set of an equitable partition $\mathcal{P}$.
V_k	The k^{th} set of a partition of the variables of a linear or integer linear program.
C_k	The k^{th} set of a partition of the constraints of a linear or integer linear program.

$EQ(\mathcal{F})$	Operator resulting in an equitable partition given a set of fractional symmetries $\mathcal{F}$.
$\mathcal{R}(\mathcal{P})$	Set of all representatives in a partition $\mathcal{P}$.
$R(x_i, \mathcal{P})$	Representative of x_i in a partition $\mathcal{P}$.

1.1.3 Linear Programs

A	Matrix of constraint coefficients for a linear or integer linear program.
b	Vector of constraint right-hand sides for a linear or integer linear program.
c	Vector of objective coefficients for a linear or integer linear program.
$LP(A, b, c)$	Linear program with respect to A, b, and c.
$RLP(A, b, c, \mathcal{P})$	Restricted linear program with respect to A, b, c, and $\mathcal{P}$.
$ALP(A, b, c, \mathcal{P})$	Aggregate linear program with respect to A, b, c, and $\mathcal{P}$.
$ELP(A, b, c, \mathcal{P})$	Extended linear program with respect to A, b, c, and $\mathcal{P}$.

1.2 Introduction

We consider linear programs $LP(A, b, c)$ of the following form:

$$\max\ c^t x \tag{1.1a}$$

$$\text{s.t. } Ax \leq b \tag{1.1b}$$

$$x \geq 0, \tag{1.1c}$$

with $c^t \in \mathbb{R}^n$ representing the objective coefficients and x in $\mathbb{R}^n$ representing the decision variables. We have that b is in $\mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}$.

In general, we solve problems of the form in (1.1) using either simplex methods (primal or dual simplex algorithms) or interior point methods (IPM)s (logarithmic barrier, predictor-corrector, etc). An interior point method improves a feasible interior solution point of

a linear program (LP) by steps through the interior of the feasible region in contrast to simplex methods that move through steps around the boundary [9]. Both theoretically and practically, IPMs are attractive due to their polynomial run-times. However, they produce interior point solutions, whereas, simplex methods produce vertex solutions that are sparser and may be preferable in various applications.

Typically, a crossover routine accompanies an IPM to achieve a vertex solution to the LP problem [3]. Crossover techniques are computationally expensive. Gurobi reports that 29% of the solver time is spent in crossover when the solver uses an IPM [6]. However, IPMs are not the only means of producing interior point solutions in the hopes of solving LPs faster. Symmetric structures, and their generalizations, of LPs can be exploited to reduce problem size, but doing so may return interior point solutions.

In [17], the authors show how to reduce the dimension of an LP with respect to an equitable partition (EP). Further, they show that a feasible solution to the reduced LP can be lifted to a feasible solution to the original LP with the same objective value, and vice versa. The conversion from a reduced solution to a full-dimension solution and vice versa occurs by matrix multiplication by a suitable matrix. However, this conversion does not guarantee one arrives at a basic solution.

In this paper, we introduce a novel method of crossover, *orbital crossover*, that uses symmetry in the LP as a guide to achieve an optimal basic feasible solution (BFS). We organize the remainder of this paper as follows. First, a brief discussion of preliminaries on symmetry and EPs. Next, an explanation of the ideas behind the construction of orbital crossover. After this, an extension of the foundation of orbital crossover and how to apply it in an iterative procedure corresponding to iterative refinements of an EP. Next, an explanation of the methodology behind benchmark setups and results. The paper rounds out with a discussion on the performance of orbital crossover.

1.3 Background: Symmetry and EPs

1.3.1 Partitions and Restrictions

Let $\mathcal{P} = \{P_1,\ P_2,\ \ldots,\ P_k\}$ be a partition of the decision variables and let the *representative* of $P_k \in \mathcal{P}$ be the lowest index member of P_k. Let $\mathcal{R}(\mathcal{P})$ be the set of all representatives associated with a partition $\mathcal{P}$. Further, we let $R(x_i, \mathcal{P})$ be the function that identifies the representative of x_i given partition $\mathcal{P}$, that is, the representative $x_j \in \mathcal{R}(\mathcal{P})$ such that x_i and x_j are in the same set $P_k \in \mathcal{P}$. We overload the R notation to act on sets $P_k \in \mathcal{P}$ as well, where $R(P_k)$ returns the representative of the set P_k, that is $R(P_k) = P_k \cap \mathcal{R}(\mathcal{P})$.

We define the restricted linear program, $RLP(A, b, c, \mathcal{P})$ to be that as (1.1) with the additional constraints:

$$x_i = x_{R(x_i,\mathcal{P})},\ i = 1, 2, \ldots, n,\ i \neq R(x_i, \mathcal{P}). \tag{1.2}$$

Obviously, $RLP(A, b, c, \mathcal{P})$ can be written in such a way as to aggregate variables that belong to the same set of the partition. Such an aggregation will yield a formulation with only k variables. It is possible that such a restriction may make a subset of the constraints linearly dependent. We assume that those constraints are handled via preprocessing. We will refer to the aggregated version as $ALP(A, b, c, \mathcal{P})$.

We will make use of the *extended linear program*, $ELP(A, b, c, \mathcal{P})$, where the set of constraints (1.2) are replaced by

$$r_{i,R(x_i,\mathcal{P})} = x_i - x_{R(x_i,\mathcal{P})} \tag{1.3}$$

and all constraints are written in standard form. We sometimes refer to $r_{i,R(x_i,\mathcal{P})}$, $\big(x_i, x_{R(x_i,\mathcal{P})}\big)$, and (1.3) as *linking* variables, pairs, and constraints, respectively. Note that since the newly formed r variables are unbounded and appear in just one (new) equality each, a basis for $LP(A, b, c)$ can easily be mapped to a basis for $ELP(A, b, c, \mathcal{P})$ by adding the r variables to the basis. Similarly, a basis for $ELP(A, b, c, \mathcal{P})$ can be mapped to a basis

in $LP(A, b, c)$ if all the r variables are basic, simply by truncating the r variables and their associated constraints. We will show how to efficiently arrive at an optimal basic solution of $LP(A, b, c)$ given an optimal BFS to $ALP(A, b, c, \mathcal{P})$, by way of $ELP(A, b, c, \mathcal{P})$, when $\mathcal{P}$ is generated via symmetry information.

1.3.2 Symmetries and Fractional Symmetries

In this section we mention some basic ideas in group theory. An in-depth review can be found in [42, 19, 4]. The symmetry group of $LP(A, b, c)$ is defined to be permutations that map feasible solutions to the LP to feasible solutions with the same objective value [35]. Let $\mathcal{G}$ represent the symmetry group. Computing $\mathcal{G}$ is not practical, as it requires the knowledge of all feasible solutions. The *formulation group*, G, is used to approximate $\mathcal{G}$ in practice. Note that $G \subset \mathcal{G}$. The formulation group of (1.1) contains the set of permutations P_π such that there exists a $(P_\pi,\ P_\sigma)$ with:

$$cP_\pi = c, \tag{1.4a}$$

$$P_\sigma b = b, \tag{1.4b}$$

$$AP_\pi = P_\sigma A, \tag{1.4c}$$

$$P_\pi \in \mathcal{S}^n,\ P_\sigma \in \mathcal{S}^m, \tag{1.4d}$$

where $\mathcal{S}^n$ and $\mathcal{S}^m$ are the sets of all permutation matrices of appropriate size [29].

Note that P_π represents permutations of the columns (variables) while P_σ represents permutations of the rows (constraints). Thus, we can think of constraints being equivalent with respect to the group generated by all feasible P_σ permutations. Recall that we are considering linear programs whose constraints are written as $Ax \leq b$. The pair (P_π, P_σ) can be considered a symmetry of the problem written in standard form, where P_σ describes the action on the slack variables. There is a correspondence between constraints that are

equivalent with respect to a group and slack variables that are equivalent with respect to that same group.

Allowing (P_π, P_σ) to be doubly stochastic matrices that satisfy (2.2) gives the set of *fractional symmetries*, $\mathcal{F}$. Now, we have $P_\pi \in \mathcal{D}^n$ and $P_\sigma \in \mathcal{D}^m$, where $\mathcal{D}^n$ and $\mathcal{D}^m$ are the sets of all doubly stochastic matrices of appropriate size [17]. Note that the set of all fractional symmetries can be thought of as a polyhedron (though they no longer form a group).

1.3.3 Orbits and Generalizations

A problem's symmetry group can be used to define equivalence classes on both the sets of variables as well as the set of feasible solutions. Two solutions are equivalent with respect to the group $\mathcal{G}$ if there exists a permutation in $\mathcal{G}$ that maps one solution to the other. We call the set of all solutions equivalent with respect to the group $\mathcal{G}$ to a solution x the *orbit* of x with respect to $\mathcal{G}$ [28]. This is denoted as $orb(x,\ \mathcal{G})$. We overload the *orb* notation to act on variables as well as vectors. We have that $j \in orb(i, \mathcal{G})$ if and only if $e_j \in orb(e_i, \mathcal{G})$. We say that two variables in the same orbit are equivalent. The variable orbits can be used to define a natural partition of the variables as $j \in orb(i, \mathcal{G})$ implies that $i \in orb(j, \mathcal{G})$. We let $\mathcal{O} = \{O_1, \ldots,\ O_k\}$ represent the *orbital partition* of the variables.

While not immediately obvious, *equitable partitions* (EP)s are the fractional symmetry analog of orbits [17]. Formally, $(\mathcal{V}, \mathcal{C})$, partitions of the variables $(\mathcal{V} = (V_1,\ \ldots,\ V_k))$ and constraints $(\mathcal{C} = (C_1,\ \ldots,\ C_k))$, is an EP with respect to $\mathcal{F}$, if it satisfies the following constraints:

$$\sum_{p \in C} (A_{p,i} - A_{p,j}) = 0 \ \forall i, j \in V \in \mathcal{V},\ C \in \mathcal{C} \tag{1.5a}$$

$$\sum_{i \in V} (A_{p,i} - A_{q,i}) = 0 \ \forall p, q \in C \in \mathcal{C}, V \in \mathcal{V} \tag{1.5b}$$

$$\sum_{p,q \in C} (b_p - b_q) = 0 \ \forall C \in \mathcal{C} \tag{1.5c}$$

$$\sum_{i,j \in V} (c_i - c_j) = 0 \ \forall V \in \mathcal{V} \tag{1.5d}$$

We say that partition $\mathcal{P}^2$ is a *refinement* of partition $\mathcal{P}^1$ if for all $P_i^2 \in \mathcal{P}^2$ there exists a $P_j^1 \in \mathcal{P}^1$ with $P_i^2 \subseteq P_j^1$. It can be shown that any orbital partition is a refinement of an EP. Conversely, $\mathcal{P}^1$ is *coarser* than $\mathcal{P}^2$. The coarsest EP can be computed efficiently [2], and is often used as the first step in computing the formulation group of an instance.

1.3.4 Stabilizers and Generalizations

The *stabilizer* of a group $\mathcal{G}$ with respect to an element v, $stab(v, \mathcal{G})$ is defined to be:

$$stab(v, \mathcal{G}) \stackrel{\text{def}}{=} \{P_\pi \in \mathcal{G} \mid P_\pi(v) = v\}.$$

Note that this definition allows for v to be either a vector or a variable. Using the constraints (2.2), this can be thought of as adding the constraint $P_\pi v = v$, or, in the case where v represents a variable, say x_i, by fixing $P_\pi(i, i)$ to one.

Similar to the stabilizer, we wish to *isolate* elements of an EP by creating a refinement where that element is in a singleton set in the refined partition. Formally, we have $iso(v, \mathcal{F})$ defined as:

$$iso(v, \mathcal{F}) \stackrel{\text{def}}{=} \{P_\pi \in \mathcal{F} \mid P_\pi(v) = v\}.$$

Again, this can be thought of adding the constraint $P_\pi v = v$, or, in the case where v represents a variable, say x_i, by fixing $P_\pi(i, i)$ to one.

EPs are useful in LP as they provide a method of aggregating variables based on symmetry information. Let $EQ(\mathcal{F})$ be an equitable partition of the variables and constraints of an LP with respect to $\mathcal{F}$. When an LP is aggregated based on $\mathcal{P} = EQ(\mathcal{F})$, we have the following theorem.

Theorem 1.1 (Lemma 5.1 [17]). *Let $\mathcal{F}$ be the set of fractional symmetries acting on a linear program, $\mathcal{P} = (\mathcal{V}, \mathcal{C})$ an EP with respect to $\mathcal{F}$, z_R^* the optimal objective value to $RLP(A, b, c, \mathcal{P})$ and z^* the optimal objective value to $LP(A, b, c)$. Then, $z_R^* = z^*$.*

1.4 Orbital Crossover

For $\mathcal{P} = (\mathcal{V}, \mathcal{C})$, an EP of $LP(A, b, c)$, we will show how to use an optimal basic solution to $ALP(A, b, c, \mathcal{P})$ to find an optimal basic solution to $LP(A, b, c)$ via $ELP(A, b, c, \mathcal{P})$.

1.4.1 Creating $ALP(A, b, c, \mathcal{P})$

First, we need to define how the aggregate formulation is created. Let $\mathcal{R}(\mathcal{V})$ be the set of variable representatives and $\mathcal{R}(\mathcal{C})$ be the set of constraint representatives in $\mathcal{R}(\mathcal{P})$. Then $ALP(A, b, c, \mathcal{P})$ has $|\mathcal{R}(\mathcal{V})|$-many variables and $|\mathcal{R}(\mathcal{C})|$-many constraints. Note that each of the $|R(C)|$-many constraints is written in standard form making them linearly independent. For each representative constraint $c_p \in \mathcal{R}(\mathcal{C})$, we construct an aggregated constraint by simply combining the coefficients of variables in the same set of the EP $\mathcal{P}$. That is, the coefficient of each representative variable $x_j, j \in R(V)$ in $ALP(A, b, c, \mathcal{P})$ will be

$$\sum_{i|R(i,\mathcal{P})=i} A_{p,i},$$

and the right hand side of the constraint remains unchanged. The objective function is similarly aggregated.

Example 1.1. Aggregating an instance of $LP(A, b, c)$

Consider the following example LP:

$$\max\ x_1 + x_2 + x_3 \tag{1.6a}$$

$$\text{s.t. } x_1 + x_2 \leq 1 \qquad (c_1) \tag{1.6b}$$

$$x_2 + x_3 \leq 1 \qquad (c_2) \tag{1.6c}$$

$$x_1 + x_3 \leq 1 \qquad (c_3) \tag{1.6d}$$

$$x_1 + x_2 + x_3 \leq 1 \qquad (c_4) \tag{1.6e}$$

$$x_i \geq 0, \ \forall i. \tag{1.6f}$$

We add slack variables for each constraint, denoted as s_i, to transform the problem into standard form. Here, s_i is the slack for constraint c_i. In this LP, the coarsest EP is

$$\mathcal{P} = (\mathcal{V}, \mathcal{C}) = \{(x_1, x_2, x_3), (s_1, s_2, s_3), (s_4), (c_1, c_2, c_3), (c_4)\}.$$

Note that the partitions of the slack variables correspond to the partitions of the constraints. Given this correspondence, we will omit constraint partitions going forward. The representatives are x_1, s_1, s_4, c_1, and c_4. Thus, we can write $ALP(A, b, c, \mathcal{P})$ for this example in standard form as an LP on three variables with two constraints:

$$\max \ 3x_1 \tag{1.7a}$$

$$\text{s.t.} \ \ 2x_1 + s_1 = 1 \qquad (c_1) \tag{1.7b}$$

$$3x_1 + s_4 = 1 \qquad (c_4) \tag{1.7c}$$

$$x_1, s_1, s_4 \in \mathbb{R}^+. \tag{1.7d}$$

We note that the optimal solution to (1.7) is $(x_1, s_1, s_4) = \left(\frac{1}{3}, \frac{1}{3}, 0\right)$. This can be interpreted as saying that the average of all variables in $V_1 \in \mathcal{P}$ is $\frac{1}{3}$, the average of all variables in $V_2 \in \mathcal{P}$ is $\frac{1}{3}$, and the average of variables in $V_3 \in \mathcal{P}$ is 0. Also, in this optimal solution we have x_1 and s_1 as basic variables and s_4 as a nonbasic variable.

1.4.2 Lifting Solutions from $ALP(A, b, c, \mathcal{P})$ to $ELP(A, b, c, \mathcal{P})$

After arriving at an optimal BFS to the $ALP(A, b, c, \mathcal{P})$, we then lift the solution to an optimal BFS to the extended linear program $ELP(A, b, c, \mathcal{P})$. Recall that the $ELP(A, b, c, \mathcal{P})$ contains all of the original variables in addition to the linking r variables (1.3).

Let x_i^{a*} be the optimal value of the aggregate variable x_i^a. The original variables can be recovered by:

$$x_i = x^a_{R(x_i,\mathcal{P})} \ \forall i \in \{1,\ \ldots,\ n\}, \tag{1.8}$$

recalling that $R(x_i, \mathcal{P})$ identifies the representative of i in partition $\mathcal{P}$. Note that this implies that

$$r_{i,R(x_i,\mathcal{P})} = 0 \ \ \forall\ i \neq R(i, \mathcal{P}). \tag{1.9}$$

Similarly, let s_i^{a*} be optimal value of the slack variable s_i^a. Then

$$s_i = s^a_{R(s_i,\mathcal{P})} \ \forall i \in \{1,\ \ldots,\ m\}. \tag{1.10}$$

Let $(x, s, r)^*$ denote the lifted solution from $(x^a, s^a)^*$, an optimal BFS from the $ALP(A, b, c, \mathcal{P})$. We have the following result.

Theorem 1.2. *$(x, s, r)^*$ is an optimal BFS to $ELP(A, b, c, \mathcal{P})$ with nonnegativity constraints on the r variables.*

Proof. First, note that as a consequence of [17], if we truncate $(x, s, r)^*$ to $(x, s)^*$ then $(x, s)^*$ is both optimal and feasible for $LP(A, b, c)$. Now, recall that equation (1.8) implies equation (1.9), thus $(x, s, r)^*$ satisfies $ELP(A, b, c, \mathcal{P})$. Next, note that there are $n+m+(n-|\mathcal{R}(V)|)$-many variables in $ELP(A, b, c, \mathcal{P})$. The constraints of $ELP(A, b, c, \mathcal{P})$ contribute $m + (n - |\mathcal{R}(V)|)$-many linearly independent binding constraints. Thus the set of nonbasic variables should be of size n.

Let NB_{ALP} be the set of nonbasic variables in the optimal BFS to $ALP(A, b, c, \mathcal{P})$. We have that $|NB_{ALP}| = |R(\mathcal{V})|$. The disaggregation of every nonbasic variable in NB_{ALP} contributes one non-basic variable in $ELP(A, b, c, \mathcal{P})$. In addition, there are $n - |R(\mathcal{V})|$-many r variables that are in the non-basis, giving a total of $|R(\mathcal{V})| + n - |R(\mathcal{V})| = n$-many variables.

□

Example 1.2. Lifting an instance of $ALP(A, b, c, \mathcal{P})$ to $ELP(A, b, c, \mathcal{P})$.

The extended formulation for the LP in the previous example is as follows:

$$\max\ x_1 + x_2 + x_3 \tag{1.11a}$$

$$\text{s.t. } x_1 + x_2 + s_1 = 1 \qquad (c_1) \tag{1.11b}$$

$$x_2 + x_3 + s_2 = 1 \qquad (c_2) \tag{1.11c}$$

$$x_1 + x_3 + s_3 = 1 \qquad (c_3) \tag{1.11d}$$

$$x_1 + x_2 + x_3 + s_4 = 1 \qquad (c_4) \tag{1.11e}$$

$$r_{2,1} = x_2 - x_1 \tag{1.11f}$$

$$r_{3,1} = x_3 - x_1 \tag{1.11g}$$

$$x, s, r \geq 0. \tag{1.11h}$$

Performing the lifting via equations (1.8) and (1.10), we have $x_1 = x_2 = x_3 = \frac{1}{3}$, $s_1 = s_2 = s_3 = \frac{1}{3}$, $s_4 = 0$ $r_{2,1} = 0$, and $r_{3,1} = 0$. There are nine variables and six equality constraints. Letting s_4, $r_{2,1}$, and $r_{3,1}$ be the set of nonbasic variables will result in a non-degenerate BFS.

1.4.3 Arriving at an optimal BFS to $LP(A, b, c)$

The previous section shows how to construct an optimal BFS to $ELP(A, b, c, \mathcal{P})$ from the solution to $ALP(A, b, c, \mathcal{P})$. However, we wish to have an optimal BFS to $LP(A, b, c)$. We note that this can be accomplished easily if all the r variables are in the basis by truncating the r variables. Starting at the lifted BFS, where all the r variables are nonbasic, we iteratively pivot on each r variable. To ensure that the r variables never re-enter the set of nonbasic variables, we relax their lower bounds to be negative infinity as they enter the basis, making them free variables. We need to ensure, however, that the objective value does not change as we pivot r variables into the basis.

Theorem 1.3. *One can pivot on any r variable without changing the objective function.*

Proof. Let z^* be the optimal solution value to $ALP(A, b, c, \mathcal{P})$. From Theorem 1.1, we have that the optimal solution value to $LP(A, b, c)$ is z^* and that the lifted basic solution also has an objective of z^*. Consider pivoting on the nonbasic variable r. If either the reduced cost of r is zero or the pivot is degenerate, then pivoting does not change the objective. Consider now the case where the reduced cost is strictly positive and the pivot is non-degenerate. Note that the non-negativity constraint on r is arbitrary and could have been written as a non-positivity constraint, and the current BFS would still remain a BFS if such a change were made. Switching the sense of r would negate the reduced cost of r. Since the current BFS is optimal, pivoting on r in the "down direction" must be degenerate, so the solution value does not change.

□

1.4.4 Reducing the number of r variables

In the above method, $n - |\mathcal{R}(V)|$-many r variables are created, and each of these r variables adds one simplex pivot to the crossover algorithm. The quantity of r variables can be reduced by taking advantage of the following observation. If a constraint (including a variable bound) is binding in the solution to $ALP(A, b, c, \mathcal{P})$, then there exists a vertex in $LP(A, b, c)$ where all equivalent constraints are binding. The intuition behind this is that in $ALP(A, b, c, \mathcal{P})$, x and s represent the average value of the variables they represent, so if one is at its bound, then all are at their bound. Consider then an aggregate x variable that is nonbasic in $ALP(A, b, c, \mathcal{P})$. In the lifted tableau, variables that are not representatives are always basic by construction. This leads to a situation where the tableau contains constraints of the form:

$$x_i - x_{R(x_i,\mathcal{P})} - r_{i,R(x_i,\mathcal{P})} = 0,$$

where $r_{i,R(x_i,\mathcal{P})}$ is nonbasic and $x_{R(x_i,\mathcal{P})} = 0$. Clearly, we can perform a degenerate (downward) pivot on $r_{i,R(x_i,\mathcal{P})}$ to remove it from the nonbasis and add x_i to the nonbasis.

Thus, we could have originally omitted the $r_{i,R(x_i,\mathcal{P})}$ variable (and its associated constraint) from the formulation and simply included x_i in the nonbasis.

Note that slack variables do not have associated r variables, so we cannot easily swap basic s variables that we know to be zero into the nonbasis in place of r variables. It is likely that enforcing some constraints to be binding will force a set of r variables to take the value of zero, so it might be possible to construct the lifted BFS with fewer r variables in the nonbasis. However, identifying which r variables to remove from the basis would require testing for linear dependence. This linear dependence check might be more expensive than the savings gained by fewer pivots.

1.5 Iterative Lifting

The benefit of the method described in Section 1.4 is that an optimal solution to the linear program can be found in the aggregated space. Crossover itself, however, is done in the extended space. By performing an iterative lifting procedure, we can do many of the crossover operations in lower dimensions. Recall that for *any* EP $\mathcal{P}$, $RLP(A, b, c, \mathcal{P})$ will have the same optimal solution value as $LP(A, b, c)$. Obviously, we'd prefer to find the coarsest partition to allow for the most aggregation (and smallest dimension of $ALP(A, b, c, \mathcal{P})$). Let $\mathcal{P}^1$ and $\mathcal{P}^2$ be two EPs of $LP(A, b, c)$ where $\mathcal{P}^2$ is a refinement of $\mathcal{P}^1$. We can use the above methods to lift a vertex solution of $ALP(A, b, c, \mathcal{P}^1)$ to $ALP(A, b, c, \mathcal{P}^2)$ and as long as $\mathcal{P}^2$ is not the discrete partition (That is, $|P_k| = 1, \forall P_k \in \mathcal{P}^2$), this lifting will be done in a smaller dimension than the original extended formulation.

Example 1.3. Lifting a solution from $ALP(A, b, c, \mathcal{P})$ to $ALP(A, b, c, \mathcal{P}^1)$.

Consider again the linear program given in (1.6). The coarsest EP, $\mathcal{P}^1$, after putting the problem in standard form, is $\mathcal{P} = \{(x_1, x_2, x_3), (s_1, s_2, s_3), (s_4), (c_1, c_2, c_3), (c_4)\}$. Isolating x_1 gives the refinement, $\mathcal{P}^1 = \{(x_1), (x_2, x_3), (s_1, s_3), (s_2), (s_4), (c_1, c_3), (c_2), (c_4)\}$, which is also an EP of (1.6). The formulation for $ALP(A, b, c, \mathcal{P}^1)$ in standard form is:

$$\max\ x_1 + 2x_2 \tag{1.12a}$$

$$\text{s.t. } x_1 + x_2 + s_1 = 1 \qquad (c_1) \tag{1.12b}$$

$$2x_2 + s_2 = 1 \qquad (c_2) \tag{1.12c}$$

$$x_1 + 2x_2 + s_4 = 1 \qquad (c_4) \tag{1.12d}$$

$$x, s \geq 0. \tag{1.12e}$$

As before, we can lift the solution from $ALP(A, b, c, \mathcal{P})$ to $ALP(A, b, c, \mathcal{P}^1)$. In this case, we have $x_1 = x_2 = x_3 = \frac{1}{3}$, $s_1 = s_2 = \frac{1}{3}$, and $s_4 = 0$. While not a vertex to (1.12), it is a vertex to $ELP(A, b, c, \mathcal{P}^1)$:

$$\max x_1 + 2x_2 \tag{1.13a}$$

$$\text{s.t. } x_1 + x_2 + s_1 = 1 \qquad (c_1) \tag{1.13b}$$

$$2x_2 + s_2 = 1 \qquad (c_2) \tag{1.13c}$$

$$x_1 + 2x_2 + s_4 = 1 \qquad (c_4) \tag{1.13d}$$

$$r_{2,1} = x_2 - x_1 \tag{1.13e}$$

$$x, s, r \geq 0. \tag{1.13f}$$

Pivoting on $r_{2,1}$ would result in a vertex to $ALP(A, b, c, \mathcal{P}^1)$. Note that (1.13) is a smaller formulation than that of (1.11).

We generate our set of EPs as follows. Let $\mathcal{P}$ be the the coarsest possible EP with respect to $\mathcal{F}$. We let $\mathcal{P}^1$ be the EP generated by $\mathcal{F}^1 = iso(x_1, \mathcal{F})$. Similarly, we let $\mathcal{P}^i$ be the coarsest EP generated by $\mathcal{F}^i = iso(x_i, \mathcal{F}^{i-1})$. We use the EP $\mathcal{P}$ to construct $ALP(A, b, c, \mathcal{P})$ and solve it for an initial aggregate solution. From here, we use the EPs $\mathcal{P}^1, \ldots, \mathcal{P}^d$ to construct $ELP(A, b, c, \mathcal{P}^1), \ldots, ELP(A, b, c, \mathcal{P}^d)$ and lift the solution from $ALP(A, b, c, \mathcal{P})$ to $ALP(A, b, c, \mathcal{P}^d)$. $\mathcal{P}^d$ is the discrete partition and thus an optimal feasible solution to $ALP(A, b, c, \mathcal{P}^d)$ is an optimal feasible solution to $LP(A, b, c)$. We formalize this solution technique in Algorithm 1.

Algorithm 1 Iterative Orbital Crossover Implementation

function IterativeOrbitalCrossover($LP(A, b, c),\ f$)
  $l \leftarrow 0$
  $k \leftarrow 0$
  $n \leftarrow$ the number of columns in $LP(A, b, c)$
  $m \leftarrow$ the number of rows in $LP(A, b, c)$
  $\mathcal{P}^k \leftarrow \mathbf{EP}(LP(A, b, c))$
  $ALP(A, b, \mathcal{P}^k) \leftarrow \mathbf{Fold}(LP(A, b, c), \mathcal{P}^k)$
  $(V^a, S^a, B^a) \leftarrow \mathbf{Solve}(ALP(A, b, c, \mathcal{P}^k))$
  while $|\mathcal{P}^k| < n + m$ **do**
    $\mathcal{P}^{k+1} \leftarrow \mathbf{Refine}(\mathcal{P}^k)$
    $l \leftarrow l + |\mathcal{P}_x^{k+1}| - |\mathcal{P}_x^k|$
    if $l < f$ **then**
      $k \leftarrow k + 1$
    else
      $ELP(A, b, c, \mathcal{P}^{k+1}) \leftarrow \mathbf{Extend}(ALP(A, b, c, \mathcal{P}^{k+1}), \mathcal{P}^k, \mathcal{P}^{k+1}, (V^a, S^a, B^a))$
      $(V^a, S^a, B^a) \leftarrow \mathbf{OC}(ELP(A, b, c, \mathcal{P}^{k+1})$
      $l \leftarrow 0$
      $k \leftarrow k + 1$
    end if
  end while
end function

(V^a, S^a, B^a) represent the solution and basis to the previously solved instance of $ALP(A, b, c, \mathcal{P})$ or $ELP(A, b, c, \mathcal{P})$. Let $|\mathcal{P}_x^k|$ be the number of non-slack sets in $\mathcal{P}^k$. The algorithm allows the user to control the minimum amount of new non-slack sets in the partition before an iterative lift and orbital crossover occur by the argument f in Algorithm 1. Each time a refinement occurs, the change in the number of non-slack sets is added to the value l. Once $l \geq f$ or $|\mathcal{P}^k| = n + m$, an iteration of lifting and orbital crossover occurs and l is set to 0. Appendix A.2 gives a description of each bold font function in Algorithm 1 and their components.

1.6 Benchmarks

1.6.1 Implementation

Orbital crossover was implemented using HiGHS version 1.2. HiGHS is a high performance serial and parallel dual simplex solver for large scale sparse linear programming problems (see [22] and https://github.com/ERGO-Code/HiGHS for more details). Although the HiGHS authors market it as a high-performance dual simplex solver, it possesses a well-implemented version of the primal simplex algorithm and an IPM solver with crossover. This makes HiGHS a capable candidate for implementing OC and for comparing against the dual simplex and IPM solvers.

A modified version of **saucy** [10] was used to compute the EPs. **saucy** was chosen over other symmetry-detection software [31, 40], as it has an extension [43] that allows it to accept LP files. It was integrated directly into the HiGHS source files as a class object callable by the main HiGHS objects.

1.6.2 Test Sets, Methods, and setup

Benchmark tests are set up using two different LP data sets. The first data set is a group of LP relaxations for a subset of instances in the **MIPLIB 2017 Benchmark** repository at https://miplib.zib.de/tag_benchmark.html. We preprocess the instances in this set

three times. First, using **saucy**, we remove instances containing no symmetry. Second, using our EP code, we remove instances whose symmetry is the result of duplicate columns or duplicate rows. However, we note that using our EP code, if an instance has both duplicate columns and duplicate rows, then an instance will not be removed. Finally, we remove any instance that takes less than ten seconds to solve across all methods listed below. We refer to this data set as **MIPLIB 2017**.

The second data set is a large collection of LP relaxations of highly symmetric MILP instances. We preprocess these instances one time removing instances that solve in less than ten seconds across all methods listed below. Instances beginning with `cod` are used to compute maximum cardinality binary error correcting codes [25]. Instances beginning with `codbt` are used to compute minimum dominating sets in Hamming graphs. Instances beginning with `cov` are covering design problems [33]. Many of these instances used to be available on Francois Margot's website `http://wpweb2.tepper.cmu.edu/fmargot/lpsym.html`, however, this website does not appear to be functional anymore. We refer to this data set as **HS-COV-COD**.

We use five different solving strategies across both data sets above. We list these strategies below.

- **HDS:** HiGHS serial dual simplex solver,
- **HIP:** HiGHS IPM solver with HiGHS crossover,
- **HIP ALP:** HiGHS IPM solver to solve $ALP(A, b, c, \mathcal{P})$, lift the solution to $LP(A, b, c)$, and obtain a optimal BFS with HiGHS crossover,
- **OCDS:** OC with iterative lifting using HiGHS serial dual simplex solver to solve $ALP(A, b, c, \mathcal{P})$,
- **OCIP:** OC with iterative lifting using HiGHS IPM solver and HiGHS crossover to solve $ALP(A, b, c, \mathcal{P})$.

1.6.3 Results

The computer used for all benchmarks is a Dell Lattitude 5521 with an Intel i7-11850H at 2.50GHz and 16 GB of RAM. All results for the **HS-COV-COD** test suite can be found in Table 1.1. The difference between OCDS and OCIP is minimal in the highly symmetric instances, as they likely are lifting the same solution to the the aggregate problem. Interestingly, the times reported for HIP, HIP ALP, OCDS, and OCIP are almost entirely the time spent in crossover. The small size of the aggregate allows HIP ALP, OCDS, and OCIP to solve the aggregate problem instantaneously, while HIP spends all but two seconds (for the entire test suite) in crossover.

When looking at **MIPLIB 2017**, we remove the HDS column from Table 1.2 as it is widely outperformed. However, these problems have more significant aggregated problems, and we wish to separate the impact of the solving the aggregate versus time spent specifically in crossover. We've added four columns titled HC, HAC, OC (DS), and OC (IP) which describe the time spent in crossover for HiGHS crossover from HIP, HiGHS crossover from HIP ALP, OC from OCDS, and OC from OCIP, respectively. We perform the same pair-wise comparison for these instances. HIP is the fastest solver for two instances. HIP ALP is the fastest solver for ten instances. OCDS is the fastest solver for six instances. OCIP is the fastest solver for one instance.

Note that HC performs much better than in the highly symmetric instances. This is likely due to the fact that the lifted solution to the aggregated problem sometimes is a vertex in the original problem (or only a few pivots away). The OC methods would still perform degenerate pivots if the lifted solution was a vertex.

Most modern commercial solvers use multiple methods to solve a given LP instance by passing each method to a single core in a multi-core machine and taking the solution from the solver that returns first. We simulate this method by taking the minimum solve times across each instance and analyzing the total solve time. For the instances in **HS-COV-COD**, the sum of the minimum solve times is 1,230.46 seconds. For the instances in **MIPLIB 2017**, the sum of the minimum solve times is 274.42 seconds. This is an 8.74% improvement

Table 1.1: Benchmark Results for the **HS-COV-COD** Instances

Instance	HD	HIP	HIP ALP	OCDS	OCIP
cod733	30.35	1.95	2.12	1.74	**1.73**
codbt161	22.88	**0.75**	1.07	0.84	0.84
codbt162	37.65	**0.90**	2.41	3.27	3.21
codbt163	77.56	**2.01**	5.00	4.78	4.80
codbt164	82.17	**2.90**	6.37	3.64	3.60
codbt171	3601.87	**5.42**	14.92	24.63	24.33
codbt172	3611.58	**188.32**	288.13	216.73	212.74
codbt173	3607.37	**61.98**	194.39	100.84	80.92
codbt174	3615.73	**94.84**	193.55	116.49	95.15
codbt261	464.03	**3.11**	8.01	6.51	6.56
codbt262	2601.94	65.62	65.60	**48.92**	49.35
codbt451	3603.75	**15.97**	30.89	27.47	26.98
codbt452	3626.21	96.47	156.73	90.38	**88.94**
codbt453	3620.45	**40.11**	81.68	47.67	47.96
codbt731	411.99	**8.84**	14.62	16.30	16.13
codbt732	2304.69	**12.71**	23.91	21.08	21.13
codbt821	363.48	**0.76**	2.27	3.67	3.76
codbt822	236.31	**2.94**	6.40	5.40	5.32
codbt831	3626.37	154.34	199.01	95.31	**94.89**
codbt832	3645.83	979.85	1524.01	212.35	**210.42**
cov1385	12.42	**0.23**	0.58	0.72	0.72
cov1496	114.54	4.72	11.30	**3.79**	3.88
cov1497	49.66	3.17	6.48	**2.29**	2.43
cov15107	967.88	322.61	275.93	**19.13**	19.16
cov15108	282.21	99.24	90.47	**9.98**	10.00
cov15109	28.46	1.96	3.05	**0.80**	0.91
cov15118	13.37	28.44	32.83	1.31	**1.29**
cov161110	114.61	7.33	9.53	**1.81**	1.93
cov16118	3604.33	1956.83	2041.95	53.52	**53.24**
cov16119	1451.97	477.84	482.37	**15.01**	15.05
cov161210	23.78	21.43	22.73	1.19	**1.17**
cov16129	40.89	109.67	122.78	**3.75**	3.85
cov171210	3632.91	2208.61	2164.76	**38.00**	38.18
cov171211	444.56	24.37	31.18	2.41	**2.31**
cov17129	3632.81	3602.68	3659.56	**179.51**	182.04
cov171310	151.74	443.44	392.68	10.55	**10.52**
cov171311	129.75	65.65	70.72	3.02	**2.93**
	53888.09	11118.00	12239.96	1394.81	**1348.36**

Table 1.2: Benchmark Results for the **MIPLIB 2017** Instances

	Solve Time (Secs)				Crossover Time (Secs)			
Instance	HIP	HIP ALP	OCDS	OCIP	HC	HAC	OC (DS)	OC (IP)
atlanta-ip	12.80	12.14	**6.65**	12.50	**0.09**	0.13	0.30	0.34
bab6	27.47	19.19	**9.05**	19.12	**0.07**	0.23	0.23	0.23
chromatic-index1024-7	14.21	**2.05**	5.67	6.36	3.08	**1.31**	5.52	5.61
cryptanalysis-kb128n5obj14	5.58	**4.51**	16.11	5.16	0.79	0.55	**0.47**	0.52
cryptanalysis-kb128n5obj16	5.70	**4.46**	11.66	5.04	0.79	0.71	**0.47**	0.52
k1mushroom	34.34	22.96	**13.03**	21.37	**1.53**	5.22	2.81	2.94
map10	166.74	36.34	**17.20**	46.97	**0.31**	0.69	10.43	11.37
map16715-04	213.93	49.65	**19.10**	59.78	**0.29**	0.65	10.69	11.80
neos-3555904-turama	12.69	10.08	**6.18**	8.55	0.26	2.03	**0.20**	0.43
neos-4722843-widden	10.40	**0.72**	2.13	3.57	**0.07**	0.20	2.08	3.06
neos-631710	354.08	175.57	18.36	**14.33**	132.22	175.27	16.44	**13.87**
neos-873061	13.57	**11.92**	31.16	12.35	**0.17**	0.34	0.84	0.76
physician-sched6-2	62.11	**34.53**	80.97	38.67	2.65	**0.69**	2.76	3.95
rail01	36.16	**28.59**	85.01	33.57	7.50	1.34	**0.68**	0.75
rail02	93.16	**75.52**	972.41	94.00	13.19	**2.66**	6.26	8.84
satellites2-40	9.95	**9.18**	44.91	10.10	2.67	1.98	**0.09**	0.23
satellites2-60-fs	**6.22**	6.23	30.35	6.69	2.09	1.53	**0.13**	0.23
uccase12	14.84	**3.82**	8.48	4.84	**0.17**	0.32	1.43	1.31
uccase9	**7.36**	7.94	13.57	8.04	0.10	0.13	**0.08**	0.10
	1101.27	515.40	1391.98	**410.99**	168.03	195.99	**61.89**	66.85

over OCIP for the **HS-COV-COD** data set and a 33.23% improvement over OCIP for the **MIPLIB 2017** data set.

1.7 Discussion

OC proves to be an effective method of exploiting symmetry. For the instances in **HS-COV-COD**, we show that the OCDS and OCIP strategies are approximately 87% faster than the next best strategy not using OC. The reason for this is the instances in this data set have an immense amount of symmetry which allows OC to further exploit the structure of the problems. This allows OC to perform much of the pivoting work in smaller dimensions via iterative lifting.

The benchmark results for the instances in **MIPLIB 2017** show different results. These instances do have symmetry, but in general, not nearly as much symmetry as the instances in **HS-COV-COD**. We see for this data set that the OCIP strategy is the best performer in terms of time required to solve all instances by 20.26%.

The decrease in effectiveness for OC in the **MIPLIB 2017** data set can be partly explained by the instances having less symmetric structure. On the other hand, the standard method of determining superbasic variables to perform pushes on during HiGHS crossover may result in less pivots than what OC would require. We see that the addition of OCDS and OCIP strategies in the porfolio of solvers results in reduction of total solve time for both **HS-COV-COD** and **MIPLIB 2017** data sets making OC a valuable strategy.

Going forward, we see this as a tool that can be used to help solve highly-symmetric integer programs. A trouble when using cut-generation schemes to solve highly-symmetric integer programs is that including all symmetric-equivalent cuts can quickly increase the size of the formulation. The proposed approach will help solve these large instances.

Chapter 2

Orbital Cut Generation

2.1 Nomenclature

2.1.1 Symmetries

I^n	The set of elements $\{1, 2, \ldots, n\}$.
Π^n	The symmetric group of I^n.
$\mathcal{G}$	Symmetry group of an integer linear program.
G	Formulation group of an integer linear program.
P_π	Permutation matrix acting on the columns of an integer linear program.
P_σ	Permutation matrix acting on the rows of an integer linear program.
$\mathcal{S}^n$	Set of permutation matrices of dimension $n \times n$.
$\mathcal{S}^m$	Set of permutation matrices of dimension $m \times m$.
$orb(v, G)$	Operator giving the orbit of a vector v with respect to a group G.
$stab(v, G)$	Operator giving the stabilizer subgroup of a group G with respect to a vector v.

2.1.2 Partitions

$\mathcal{O}$	Orbital partition of the variables (and constraints) of an integer linear program.
$\mathcal{V}$	Partition of the variables of a linear or integer linear program.

$\mathcal{C}$	Partition of the constraints of a linear or integer linear program.
O_k	The k^{th} set of an orbital partition $\mathcal{O}$.
V_k	The k^{th} set of a partition of the variables of a linear or integer linear program.
C_k	The k^{th} set of a partition of the constraints of a linear or integer linear program.
$\mathcal{R}(\mathcal{O})$	Set of all representatives in a partition $\mathcal{O}$.
$R(x_i, \mathcal{O})$	Representative of x_i in a partition $\mathcal{O}$.

2.1.3 Linear and Integer Linear Programs

A	Matrix of constraint coefficients for a linear or integer linear program.
b	Vector of constraint right-hand sides for a linear or integer linear program.
c	Vector of objective coefficients for a linear or integer linear program.
$ILP(A, b, c)$	Integer linear program with respect to A, b, and c.
$LP(A, b, c)$	Natural linear relaxation of $ILP(A, b, c)$.
$AILP(A, b, c, \mathcal{O})$	Aggregate integer linear program with respect to A, b, c, and an orbital partition $\mathcal{O}$.
$ALP(A, b, c, \mathcal{O})$	Aggregate linear program with respect to A, b, c, and an orbital partition $\mathcal{O}$.
$\mathcal{F}_{ILP}$	Feasible region of an integer linear program.
$\mathcal{F}_{LP}$	Feasible region of a linear program.
$\mathcal{F}_{AILP}$	Feasible region of an aggregate integer linear program.
$\mathcal{F}_{ALP}$	Feasible region of an aggregate linear program.

2.2 Introduction

We consider integer linear programs $ILP(A, b, c)$ of the following form:

$$\max\ c^t x \tag{2.1a}$$

$$\text{s.t. } Ax \leq b \tag{2.1b}$$

$$x \in \mathbb{Z}_{\geq 0}, \tag{2.1c}$$

with $c^t \in \mathbb{R}^n$ representing the objective coefficients and $x \in \mathbb{Z}^n_{\geq 0}$ representing the decision variables. We have that b is in $\mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}$.

One method of solving problems in the form of (2.1) is cutting plane algorithms [7, 8, 30, 14, 15]. The first version of cutting plane algorithms for integer programs was proposed by Ralph Gomory in the 1960s [16]. However, the algorithm seemed to produce weak cutting planes leading to slow convergence in practice. This caused a general lack of interest in cutting plane algorithms from integer programming community for quite some time after. The development of polyhedral theory and families of strong valid cuts in the 1980s led to a renewed interest in cutting plane techniques [34]. Early research on the success of strong valid cuts is shown for the *traveling salesman problem* in [39, 18]. Further work on the use of strong valid cuts in a variety of problem types can be found in [48, 23, 5].

Now, the state of the art method for solving problems in the form of (2.1) is the branch and cut (B&C) algorithm. The B&C algorithm is a composition of the branch and bound (B&B) algorithm and the cutting plane algorithm. We assume the reader is familiar with B&C and B&B, but excellent introductions can be found in [7, 11, 24, 39, 47]. In general, several rounds of cutting plane generation are applied at the root node of the B&C enumeration tree based on factors such as density and previous success, but many fewer or no rounds of cut generation are applied later in the enumeration tree [7].

Even with the success of strong valid cuts used in B&C, symmetry in $ILP(A, b, c)$ continues to wreak havoc. $ILP(A, b, c)$ is said to be symmetric if its variables can be

permuted without changing the structure of the problem [29]. Symmetry in $ILP(A, b, c)$ can massively increase the search space in the enumeration tree. Many symmetric optimal solutions to the natural linear relaxation $LP(A, b, c)$ of $ILP(A, b, c)$ may be evaluated with no actual progress in the dual bound. Research found in [1, 20, 32] focuses on auxiliary and (or) extended formulations that reduce the amount of symmetry in $ILP(A, b, c)$. The collection of work found in [26, 41, 45] shows valid cuts that can exclude feasible symmetric solutions.

The work in [27, 28] establishes a different idea known as *isomorphism pruning* within the context integer linear programming and is used to prune isomorphic subproblems in the enumeration tree. Ostrowski et al. [37, 38] introduces *orbital branching* and *constraint orbital branching*, respectively. Orbital branching uses knowledge of the orbital partitioning of variables based on the symmetry group of $ILP(A, b, c)$ to prune equivalent subproblems in the enumeration tree. Constraint orbital branching extends the ideas of orbtial branching into constraint based branching rules to prune equivalent subproblems from the enumeration tree.

In this work, we introduce the technique of *orbital cut generation*, which exploits the symmetry group of $ILP(A, b, c)$ to generate symmetrically equivalent cutting planes. We demonstrate the usefulness of this technique in improving the dual bound compared more traditional cut generation for $ILP(A, b, c)$ at the root node for a suite of highly symmetric instances. We compare our technique to that of standard cut generation, that is, cut generation that does not exploit symmetry and provide initial and final dual bounds for both techniques applied to $ILP(A, b, c)$ at the root node. Our results show the potential of our technique to improve the best dual bound at the root node, and in some cases, even provide the optimal objective value.

We organize the remainder of this chapter as follows. Section 2.3 provides a brief discussion of preliminaries on symmetry. In Section 2.4, we formally present orbital cut generation and prove its validity. Section 2.5 provides an extension of orbital cut generation and shows how to apply it iteratively using aggregate formulations of $ILP(A, b, c)$. Section 2.6 provides details on the implementation of orbital cut generation. Section 2.7 discusses the

methodology and provides results for the computational comparison of orbital cut generation and standard cut generation. Finally, Section 2.8 rounds the paper out with a discussion of the effectiveness and implications of orbital cut generation.

2.3 Symmetry in $ILP(A, b, c)$

2.3.1 Symmetries

In this section we mention some basic ideas in group theory. An in-depth review can be found in [42, 19, 4]. First, let Π^n represent the set of permutations of $I^n = \{1, 2, \ldots, n\}$, that is, Π^n is the symmetric group of I^n. For a vector $a \in \mathbb{R}^n$, the permutation $\pi \in \Pi^n$ acts on a by permuting its coordinates [38]. We denote the permutation of the coordinates of a as follows:

$$\pi(a) = (a_{\pi_1}, a_{\pi_2}, \ldots, a_{\pi_n}).$$

Throughout this work, we express permutations in *cycle notation*. that is, the expression $(p_1, p_2, \ldots, p_k)$ is the permutation that sends the entry p_i to p_{i+1} for $i = 1, 2, \ldots, k-1$ and the entry p_k to p_1.

The symmetry group of $ILP(A, b, c)$ is defined to be permutations that map feasible solutions to $ILP(A, b, c)$ to feasible solutions with the same objective value [35]. Let $\mathcal{F}_{ILP}$ represent the feasible region and $\mathcal{G}$ the symmetry of group of $ILP(A, b, c)$, then $\mathcal{G}$ is formally defined as:

$$\mathcal{G} \stackrel{\text{def}}{=} \{\pi \in \Pi^n \mid \pi(x) \in \mathcal{F}_{ILP},\ \forall x \in \mathcal{F}_{ILP}\}.$$

Computing $\mathcal{G}$ is not practical, as it requires the knowledge of all feasible solutions. The *formulation group*, G, is used to approximate $\mathcal{G}$ in practice. Note that $G \subset \mathcal{G}$. The formulation group of (2.1) contains the set of permutations $\pi \in \Pi^n$ such that there exists $\sigma \in \Pi^m$ where the permutation matrices $(P_\pi,\ P_\sigma)$ satisfy:

$$cP_\pi = c, \tag{2.2a}$$

$$P_\sigma b = b, \tag{2.2b}$$

$$AP_\pi = P_\sigma A, \tag{2.2c}$$

$$P_\pi \in \mathcal{S}^n,\ P_\sigma \in \mathcal{S}^m, \tag{2.2d}$$

where $\mathcal{S}^n$ and $\mathcal{S}^m$ are the sets of all permutation matrices of appropriate size [29]. Note that P_π represents permutations of the columns (variables) while P_σ represents permutations of the rows (constraints). Thus, we can think of constraints being equivalent with respect to the group generated by all feasible P_σ permutations.

2.3.2 Orbits and Orbital Partitions

The symmetric group Π^n can be used to define equivalence classes on the vectors $v \in \mathbb{R}^n$. We say two vectors are symmetrically equivalent with respect to Π^n if there exists a permutation in Π^n that permutes one vector onto another. We call all vectors symmetrically equivalent with respect to Π^n to a vector $v \in \mathbb{R}^n$ the orbit of v with respect to Π^n. This is denoted as $orb(v, \Pi^n)$ and formally we have

$$orb(v, \Pi^n) \stackrel{\text{def}}{=} \{\pi(v) \mid \pi \in \Pi^n\}.$$

In the context of $ILP(A, b, c)$, the problem's formulation group G can be used to define equivalence classes on both solutions and constraints [29]. We can also consider orbits of variables by calculating the orbit of their corresponding unit vectors. That is, We have that $x_j \in orb(x_i, G)$ if and only if $e_j \in orb(e_i, G)$. We say that two variables in the same orbit are equivalent. The variable orbits can be used to define a natural partition of the variables as $x_j \in orb(x_i, G)$ implies that $x_i \in orb(x_j, G)$. We let $\mathcal{O} = \{O_1, \ldots, O_k\}$ represent the *orbital partition* of the variables. As a final note on equivalence classes, we refer to symmetrically equivalent cuts throughout the remainder of this paper. We say two cuts (or constraints) $\lambda x \leq \beta$, $\gamma x \leq \beta$ are symmetrically equivalent if there exists a permutation $\pi \in G$ such that

$$\pi(\lambda) = \gamma.$$

We say that the *representative* of $O_i \in \mathcal{O}$ for $i = 1, \ldots, k$ is the lexigraphically lowest index member of O_i. Let $\mathcal{R}(\mathcal{O})$ be the set of all representatives associated with a partition $\mathcal{O}$. Further, we let $R(x_i, \mathcal{O})$ be the function that identifies the representative of x_i given partition $\mathcal{O}$, that is, the representative $x_j \in \mathcal{R}(\mathcal{O})$ such that x_i and x_j are in the same set of $\mathcal{O}$. We overload the R notation to act on the sets $O_i \in \mathcal{O}$ as well, where $R(O_i)$ returns the representative of the set O_i, that is $R(O_i) = O_i \cap \mathcal{R}(\mathcal{O})$.

2.3.3 Stabilizers

The *stabilizer* of a group Π^n with respect to an element v, $stab(v, \Pi^n)$ is defined to be:

$$stab(v, \Pi^n) \stackrel{\text{def}}{=} \{\pi \in \Pi^n \mid \pi(v) = v\}.$$

Note that this definition allows for v to be either a vector or a variable in the context of $ILP(A, b, c)$. Using the constraints in (2.2), this can be thought of adding the constraint $P_\pi v = v$, or, in the case where v represents a variable, say x_i, by fixing $P_\pi(i, i)$ to one.

2.4 Orbital Cut Generation

Standard cut generation produces valid cuts that remove solutions from $LP(A, b, c)$ but not $ILP(A, b, c)$. However, if $ILP(A, b, c)$ has symmetry then there may be many remaining solutions that are symmetrically equivalent to those removed. These remaining solutions might hinder standard cut generation's ability to improve the dual bound.

For example, in Figure 2.1, x^1 is an optimal vertex solution removed by the valid cut $\lambda^1 x \leq \beta$. However, there are other symmetrically equivalent solutions not removed by the valid cut $\lambda^1 x \leq \beta$. It is possible that resolving $LP(A, b, c)$ with the valid cut will produce a new solution that is one of the symmetrically equivalent vertices such as x^2 (shown in Figure 2.2). Standard cut generation may produce the symmetrically equivalent cut $\lambda^2 x \leq \beta$. This process may continue until a large number of symmetrically equivalent valid cuts are generated. For the example in Figure 2.1, all symmetric valid cuts required are shown in

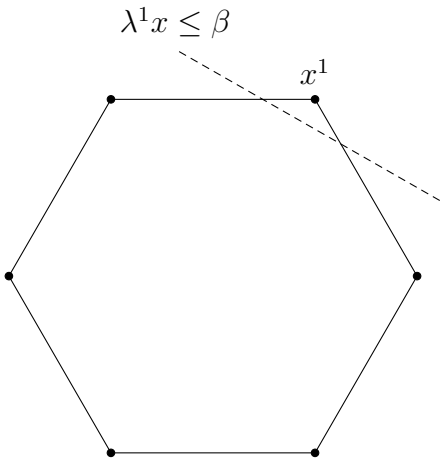

Figure 2.1: A Cutting Plane on the Optimal Face of $LP(A, b, c)$

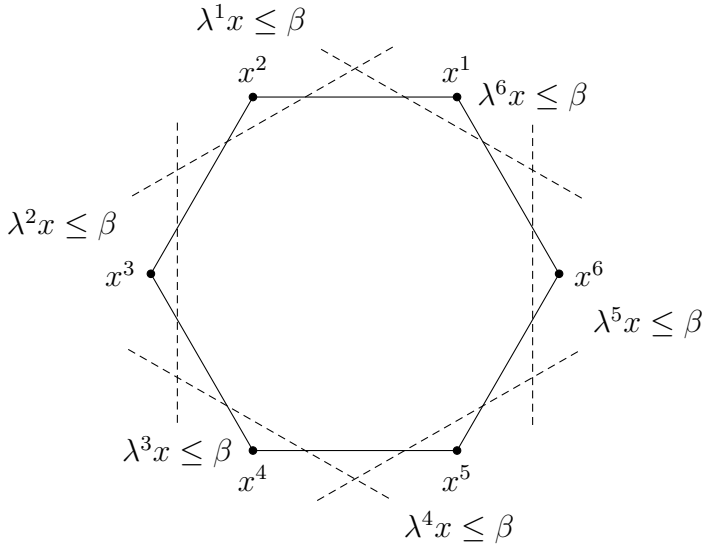

Figure 2.2: Symmetric Cutting Planes on the Optimal Face of $LP(A, b, c)$

Figure 2.2. However, we can exploit the symmetry in $ILP(A, b, c)$ to produce all of these symmetrically equivalent valid cuts without requiring multiple solves of $LP(A, b, c)$. In this section, we show how to exploit the symmetry in $ILP(A, b, c)$ to generate all symmetric valid cuts for a given valid cut $\lambda x \leq \beta$.

2.4.1 Generating Symmetric Valid cuts for $ILP(A, b, c)$

Let $\lambda x \leq \beta$ a valid cut with respect to $\mathcal{F}_{ILP}$. Using permutations in G, we can permute the coordinates of the coefficient vector λ. That is, we obtain a coefficient vector λ_π where

$$\lambda_\pi = \pi(\lambda) = (\lambda_{\pi_1}, \lambda_{\pi_2}, \ldots, \lambda_{\pi_n}).$$

The permuted coefficient vector then gives the symmetrically equivalent cut

$$\lambda_\pi x = \lambda_{\pi_1} x_1 + \lambda_{\pi_2} x_2 + \cdots + \lambda_{\pi_n} x_n \leq \beta.$$

In fact, we fully exploit G by computing

$$orb(\lambda, G) \stackrel{\text{def}}{=} \{\pi(\lambda) \mid \pi \in G\}.$$

This allows us to generate all symmetrically equivalent cuts

$$\lambda_\pi x \leq \beta, \ \forall \lambda_\pi \in orb(\lambda, G).$$

Theorem 2.1. *Let G' be a symmetry group of $ILP(A, b, c)$ and $\lambda x \leq \beta$ a valid cut for $\mathcal{F}_{ILP}$ of $ILP(A, b, c)$. Then $\lambda_\pi x \leq \beta$ is a valid cut for $\mathcal{F}_{ILP}$, $\forall\ \lambda_\pi \in orb(\lambda, G')$.*

Proof. Suppose that $\lambda x \leq \beta$ is a valid cut for $\mathcal{F}_{ILP}$. Choose a $\pi \in G'$ and let $\lambda_\pi x \leq \beta$ be the cut obtained by permuting the coordinates of λ by π, so $\lambda_\pi = \pi(\lambda)$. Now, suppose $\lambda_\pi x \leq \beta$ is not a valid cut for $\mathcal{F}_{ILP}$. Then, $\exists\ x^1 \in \mathcal{F}_{ILP}$ such that

$$\lambda_\pi x^1 > \beta.$$

Because G' is a group, π^{-1} is in G'. Permuting x^1 by π^{-1} gives

$$\pi^{-1}(x^1) = x^2.$$

Since $\pi^{-1} \in G'$, x^2 is feasible. However,

$$\lambda x^2 = \lambda \pi^{-1}(x^1) = \pi(\lambda) x^1 = \lambda_\pi x^1 > \beta.$$

Thus, $\lambda x \leq \beta$ is not valid.

□

Obviously, the addition of the symmetric valid cuts $\lambda_\pi x \leq \beta$ changes the formulation group of $ILP(A, b, c)$ because additional rows have been added to the constraint matrix. However, adding all symmetrically equivalent cuts preserves the symmetries acting on the variables of $ILP(A, b, c)$.

Theorem 2.2. *Let $ILP(A_\lambda, b_\lambda, c)$ be $ILP(A, b, c)$ with the valid cuts $\lambda_\pi x \leq \beta$, $\forall\ \lambda_\pi \in orb(\lambda, G)$. Let G and G^λ be the formulation group for $ILP(A, b, c)$ and $ILP(A_\lambda, b_\lambda, c)$, respectively. Let $G_\pi \subseteq G$ be the permutations acting on the variables of $ILP(A, b, c)$ and $G^\lambda_\pi \subseteq G^\lambda$ be the permutations acting on the variables of $ILP(A_\lambda, b_\lambda, c)$. Then $G_\pi \subseteq G^\lambda_\pi$.*

Proof. Choose any $(\pi_i, \sigma_i) \in G$ such that (π_i, σ_i) satisfies 2.2 for $ILP(A, b, c)$. Then we have the following for $ILP(A_\lambda, b_\lambda, c)$.

i) Obviously,

$$P_{\pi_i} c = c.$$

ii) Let

$$b_\beta = \begin{bmatrix} \beta \\ \vdots \\ \beta \end{bmatrix}$$

be a $|orb(\lambda, G)|$-dimensional vector. Note that

$$b_\lambda = \begin{bmatrix} b \\ b_\beta \end{bmatrix}$$

and let

$$P_{\sigma_i\gamma_i} = \begin{bmatrix} P_{\sigma_i} & 0 \\ 0 & P_{\gamma_i} \end{bmatrix}$$

where P_{γ_i} is any permutation on the elements $\{1, \ldots, |orb(\lambda, G)|\}$. We have that

$$P_{\sigma_i\gamma_i} b_\lambda = \begin{bmatrix} P_{\sigma_i} & 0 \\ 0 & P_{\gamma_i} \end{bmatrix} \begin{bmatrix} b \\ b_\beta \end{bmatrix} = \begin{bmatrix} P_{\sigma_i} b \\ P_{\gamma_i} b_\beta \end{bmatrix} = \begin{bmatrix} b \\ b_\beta \end{bmatrix} = b_\lambda.$$

iii) Let B be the matrix of coefficients in the valid cuts $\lambda_\pi x \leq \beta,\ \forall \lambda_\pi \in orb(\lambda, G)$. Then,

$$A_\lambda P_{\pi_i} = \begin{bmatrix} A \\ B \end{bmatrix} \begin{bmatrix} P_{\pi_i} \end{bmatrix} = \begin{bmatrix} AP_{\pi_i} \\ BP_{\pi_i} \end{bmatrix} = \begin{bmatrix} P_{\sigma_i} A \\ BP_{\pi_i} \end{bmatrix}.$$

P_{π_i} permutes the columns of B, but by construction of B, permuting the columns will produce a one-to-one mapping on the rows of B. That is, for $\lambda_\pi x \leq \beta,\ \forall \lambda_\pi \in orb(\lambda, G)$,

$$\lambda_\pi P_{\pi_i} = \begin{bmatrix} \lambda_\pi \end{bmatrix} \begin{bmatrix} P_{\pi_i} \end{bmatrix} = \pi_i(\lambda_\pi) = \lambda_{\pi'}$$

and $\lambda_{\pi'} \in orb(\lambda, G)$. Let P_{γ_i} be the permutation that permutes λ_π to $\lambda_{\pi'}$ for all $\lambda_\pi \in orb(\lambda, G)$. Then we have

$$\begin{bmatrix} P_{\sigma_i} A \\ BP_{\pi_i} \end{bmatrix} = \begin{bmatrix} P_{\sigma_i} A \\ P_{\gamma_i} B \end{bmatrix} = \begin{bmatrix} P_{\sigma_i} & 0 \\ 0 & P_{\gamma_i} \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} = P_{\sigma_i\gamma_i} A_\lambda.$$

Thus, the formulation group of $ILP(A_\lambda, b_\lambda, c)$ includes the set of permutations π_i and permutations $\sigma_i\gamma_i$ above. Note that we need not know the exact permutations $\sigma_i\gamma_i$, but only that they exist.

□

Example 2.1. Constructing symmetrically equivalent valid cuts

Consider the following example of $ILP(A, b, c)$:

$$\max\ x_1 + x_2 + x_3 + x_4 \tag{2.3a}$$

$$\text{s.t. } x_1 + x_2 + x_3 \leq 2 \qquad (c_1) \tag{2.3b}$$

$$x_2 + x_3 + x_4 \leq 2 \qquad (c_2) \tag{2.3c}$$

$$x_1 + x_3 + x_4 \leq 2 \qquad (c_3) \tag{2.3d}$$

$$x_1 + x_2 + x_4 \leq 2 \qquad (c_4) \tag{2.3e}$$

$$x_i \in \{0, 1\},\ \forall i. \tag{2.3f}$$

The formulation group G of (2.3) can be represented by the following group generators

$$(x_1, x_2)(c_2, c_3),$$
$$(x_1, x_4)(c_1, c_2),$$
$$(x_1, x_3)(c_2, c_4).$$

Each product of cycles above is one generator for G.

We note that the optimal solution to $LP(A, b, c)$ of (2.3) is

$$(x_1, x_2, x_3, x_4) = \left(\frac{2}{3}, \frac{2}{3}, \frac{2}{3}, \frac{2}{3}\right)$$

with an objective function value of 2.67. A valid cut that separates the above solution from $\mathcal{F}_{ILP}$ is

$$3x_1 + 6x_2 + 6x_3 + 6x_4 \leq 12. \tag{2.4}$$

Adding (2.4) to (2.3) and resolving $LP(A, b, c)$ gives a new optimal solution

$$(x_1, x_2, x_3, x_4) = \left(1, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right)$$

with an objective function value of 2.5.

Let λ be the vector of coefficients in (2.4), that is $\lambda = (3, 6, 6, 6)$. The orbit of λ with respect to G is

$$orb(\lambda, G) = \{(3, 6, 6, 6), (6, 3, 6, 6), (6, 6, 3, 6), (6, 6, 6, 3)\}$$

and the resulting valid cuts are

$$3x_1 + 6x_2 + 6x_3 + 6x_4 \leq 12 \qquad (c_5), \tag{2.5a}$$

$$6x_1 + 3x_2 + 6x_3 + 6x_4 \leq 12 \qquad (c_6), \tag{2.5b}$$

$$6x_1 + 6x_2 + 3x_3 + 6x_4 \leq 12 \qquad (c_7), \tag{2.5c}$$

$$6x_1 + 6x_2 + 6x_3 + 3x_4 \leq 12 \qquad (c_8). \tag{2.5d}$$

Adding all cuts in (2.5) to (2.3) and resolving $LP(A, b, c)$ results in the new optimal solution

$$(x_1, x_2, x_3, x_4) = \left(\frac{4}{7}, \frac{4}{7}, \frac{4}{7}, \frac{4}{7}\right)$$

with an objective function value of 2.29.

2.5 Aggregation

2.5.1 Reducing the Dimension of $ILP(A, b, c)$

The benefit of the method described in Section 2.4 is that all symmetric solutions can be separated from $\mathcal{F}_{ILP}$ of $ILP(A, b, c)$ using knowledge from G in conjunction with a valid cut. However, if $ILP(A, b, c)$ has a lot of symmetry, generating these symmetric valid cuts can become cumbersome. In the worst case, we may need to add one cut for every permutation in G. It is not uncommon to have instances of $ILP(A, b, c)$ whose symmetry groups are of order ranging from 10^6 to 10^{10}. In this section, we discuss how to restrict our search to valid cuts with small orbit sizes.

Recall from Section 2.3 that we can naturally partition the variables of $ILP(A, b, c)$ according to their orbits. From this point forward we let $\mathcal{O}^{G'} = \{\mathcal{V}, \mathcal{C}\}$ be an orbital partition of the variables and constraints of $ILP(A, b, c)$ with respect to G'. Here $\mathcal{V}$ and $\mathcal{C}$ are partitions of sets containing co-orbital variables and co-orbital constraints, respectively.

We can aggregate the variables and constraints of $ILP(A, b, c)$ via $\mathcal{O}^{G'}$. The resulting aggregate integer linear program $AILP(A, b, c, \mathcal{O}^{G'})$ will have $|R(\mathcal{V})|$-many variables and $|R(\mathcal{C})|$-many constraints. Note that this aggregation can be thought of as a relaxation of $ILP(A, b, c)$. The aggregation routine is stated formally in Algorithm 2. As a consequence of this procedure, an aggregate variable $x^a_{R(V_j)}$ in $AILP(A, b, c, \mathcal{O}^{G'})$ represents the sum of the variables x_i such that $x_i \in V_j$. That is,

$$x^a_{R(V_j)} = \sum_{i \in V_j} x_i, \ \forall V_j \in \mathcal{V}. \tag{2.6}$$

As a result of Theorem 2.2, adding $\lambda_\pi x \leq \beta, \ \forall \lambda_\pi \in orb(\lambda, G)$ does not change symmetries acting on the variables of $ILP(A, b, c)$. It does, however, add only one new constraint set to $\mathcal{O}^G$, meaning that regardless of the size of $orb(\lambda, G)$, only one constraint is added to $AILP(A, b, c, \mathcal{O}^G)$.

2.5.2 Lifting a Valid cut

Even though $AILP(A, b, c, \mathcal{O}^{G'})$ is a relaxation of $ILP(A, b, c)$, we can generate valid cuts in the aggregate space that are valid in the lifted space. Now, we show how to lift a valid cut in the aggregate problem to the original problem.

Let

$$\lambda^a x^a = \lambda^a_1 x^a_{R(V_1)} + \cdots + \lambda^a_k x^a_{R(V_k)} \leq \beta$$

be a valid cut for the feasible region $\mathcal{F}_{AILP}$ of $AILP(A, b, c, \mathcal{O}^{G'})$. Since

$$x^a_{R(V_j)} = \sum_{j \in V_j} x_j \ \forall V_j \in \mathcal{V},$$

Algorithm 2 Aggregate $ILP(A, b, c)$

Input: $ILP(A, b, c)$, $\mathcal{O}^{G'}$
Output: $AILP(A, b, c, \mathcal{O}^{G'})$

```
procedure AGGREGATE(ILP(A, b, c), O^{G'})
    A^a ← |R(C)| × |R(V)|-dimensional zero matrix
    b^a ← an |R(C)|-length zero vector
    c^a ← an |R(V)|-length zero vector
    l^a, u^a ← an |R(V)|-length zero vector
    for j ∈ R(V) do                          ▷ Loop over each representative variable
        for i = 1, 2, ..., m do
            k ← R(c_i, O^{G'})
            A^a_{k,j} ← A^a_{k,j} + a_{i,j}   ▷ a_{i,j} the coefficient of x_j in constraint i of A
            b^a_k ← b^a_k + b_i
        end for
    end for
    for j = 1, 2, ..., n do
        k ← R(x_i, O^{G'}).
        l^a_k ← l^a_k + l_j                   ▷ l_j is the lower bound of x_j
        u^a_k ← u^a_k + u_j                   ▷ u_j is the upper bound of x_j
    end for
    for j ∈ R(V) do
        c^a_j ← c_j                           ▷ c_j is the objective coefficient of x_j in c
    end for
end procedure
```

we have that

$$\begin{aligned}\lambda^a x^a &= \lambda^a_1 x^a_{R(V_1)} + \cdots + \lambda^a_k x^a_{R(V_k)} \\ &= \lambda^a_1 \sum_{j \in V_1} x_j + \cdots + \lambda^a_k \sum_{j \in V_k} x_j \leq \beta.\end{aligned} \tag{2.7}$$

Theorem 2.3. *Let G' be a symmetry group of $ILP(A, b, c)$, $\mathcal{O}^{G'}$ an orbital partition of the variables and constraints of $ILP(A, b, c)$ with respect to G', and $\lambda^a x^a \leq \beta$ a valid cut for $\mathcal{F}_{AILP}$ of $AILP(A, b, c, \mathcal{O}^{G'})$. Then*

$$\lambda^a_1 \sum_{j \in V_1} x_j + \cdots + \lambda^a_k \sum_{j \in V_k} x_j \leq \beta$$

is a valid cut for $\mathcal{F}_{ILP}$.

Proof. Suppose $\lambda^a x^a \leq \beta$ is a valid cut for $\mathcal{F}_{AILP}$. Now assume that

$$\lambda^a_1 \sum_{j \in V_1} x_j + \cdots + \lambda^a_k \sum_{j \in V_k} x_j \leq \beta$$

is not valid for $\mathcal{F}_{ILP}$. Then $\exists\ y^d \in \mathcal{F}_{ILP}$ such that

$$\lambda^a_1 \sum_{j \in V_1} y^d_j + \cdots + \lambda^a_k \sum_{j \in V_k} y^d_j > \beta.$$

Let y^a be the aggregate solution of y^d according to (2.6). Obviously, $y^a \in \mathbb{Z}_{\geq 0}$. Now, if we sum the co-orbital constraints of $ILP(A, b, c)$ substituting y^d_j for x_j, we obtain the following

$$\sum_{i \in C_k} \sum_{j=1}^{n} a_{i,j} y^d_j \leq \sum_{i \in C_k} b_i,\ \forall C_k \in \mathcal{C}. \tag{2.8}$$

Note that co-orbital variables will have like coefficients in (2.8), otherwise they would not be co-orbital.

Let $a^a_{R(C_k),1}, \ldots, a^a_{R(C_k),l}, \ \forall C_k \in \mathcal{C}$ be the coefficients on variables belonging to the same set $V_1, \ldots, V_l$ in (2.8), respectively. Further, the right-hand sides of each constraint in a set $C_k \in \mathcal{C}$ are the same. Therefore, the right-hand sides of (2.8) can be written as $|C_k| b_{R(C_k)}, \ \forall C_k \in \mathcal{C}$. Now, the left-hand sides and right-hand sides in (2.8) can be expressed as

$$\begin{aligned} \sum_{i \in C_k} \sum_{j=1}^{n} a_{i,j} y^d_j \leq \sum_{i \in C_k} b_i &= a^a_{R(C_k),1} \sum_{j \in V_1} y^d_j + \cdots + a^a_{R(C_k),l} \sum_{j \in V_l} y^d_j, \ \forall C_k \in \mathcal{C} \\ &= a^a_{R(C_k),1} y^a_{R(V_1)} + \cdots + a^a_{R(C_k),l} y^a_{R(V_l)}, \ \forall C_k \in \mathcal{C} \\ &= a^a_{R(C_k)} y^a, \ \forall C_k \in \mathcal{C} \\ &\leq |C_k| b_{R(C_k)}, \ \forall C_k \in \mathcal{C}. \end{aligned} \tag{2.9}$$

The row sums in (2.9) are the aggregate constraints of $AILP(A, b, c, \mathcal{O}^{G'})$. Now, we know $y^a \in \mathcal{F}_{AILP}$. Thus,

$$\begin{aligned} \lambda^a y^a &= \lambda^a_1 y^a_{R(V_1)} + \cdots + \lambda^a_k y^a_{R(V_k)} \\ &= \lambda^a_1 \sum_{j \in V_1} y^d_j + \cdots + \lambda^a_k \sum_{j \in V_k} y^d_j \\ &\leq \beta. \end{aligned} \tag{2.10}$$

The statement in (2.10) is a contradiction.

□

Example 2.2. Aggregating $ILP(A, b, c)$ and a valid cut in $AILP(A, b, c, \mathcal{O}^{G'})$.

Recall Example 2.1 and let $ILP(A_\lambda, b_\lambda, c)$ be (2.3) with the valid cuts in (2.5) and let G^λ be the formulation group of $ILP(A_\lambda, b_\lambda, c)$. We proved in Theorem 2.2 that the addition of all co-orbital valid cuts preserves the permutations $\pi \in G$ acting on the variables of $ILP(A, b, c)$. The orbital partition of the variables and constraints in $ILP(A_\lambda, b_\lambda, c)$ with respect to G^λ is

$$\mathcal{O}^{G^\lambda} = \{(x_1, x_2, x_3, x_4), (c_1, c_2, c_3, c_4), (c_5, c_6, c_7, c_8)\}.$$

$AILP(A_\lambda, b_\lambda, c, \mathcal{O}^{G^\lambda})$ is

$$\max\ x_1^a \tag{2.11a}$$

$$\text{s.t. } 3x_1^a \leq 8 \qquad (c_1^a) \tag{2.11b}$$

$$21x_1^a \leq 48 \qquad (c_5^a) \tag{2.11c}$$

$$0 \leq x_1^a \leq 4 \tag{2.11d}$$

$$x_1^a \in \mathbb{Z}_{\geq 0}, \tag{2.11e}$$

and the optimal objective value of $ALP(A_\lambda, b_\lambda, c, \mathcal{O}^{G^\lambda})$ is 2.29. $ALP(A_\lambda, b_\lambda, c, \mathcal{O}^{G^\lambda})$ is the linear relaxation of $AILP(A, b, c, \mathcal{O}^{G^\lambda})$. The four valid cuts c_5, c_6, c_7, and c_8 aggregate down into only one valid cut c_5^a in $AILP(A_\lambda, b_\lambda, c, \mathcal{O}^{G^\lambda})$.

A valid cut for $\mathcal{F}_{AILP}$ of (2.11) is $x_1^a \leq 2$. This gives a new optimal objective value of 2. From (2.7), the valid cut when lifted to $ILP(A, b, c)$ is

$$x_1^a = 2\,(x_1 + x_2 + x_3 + x_4) \leq 2 \qquad (c_9) \tag{2.12}$$

giving $LP(A, b, c)$ of (2.3) with the cuts in (2.5) and (2.12) and optimal objective of 2. Note that the addition of (2.12) to (2.3) results in

$$\mathcal{O}^{G^\lambda} = \{(x_1, x_2, x_3, x_4), (c_1, c_2, c_3, c_4), (c_5, c_6, c_7, c_8), (c_9)\}$$

as

$$orb((1, 1, 1, 1), G) = \{(1, 1, 1, 1)\}.$$

2.5.3 Iterative Lifting

Once an optimal feasible solution is found for $AILP(A, b, c, \mathcal{O}^G)$, no more useful valid cuts can be generated for $ILP(A, b, c)$ using $AILP(A, b, c, \mathcal{O}^G)$. We can then use the stabilizer chain of G to find a tighter relaxation. Let $G^1 = stab(x_1, G)$ and $\mathcal{O}^1$ be an orbital partition of the variables and constraints of $ILP(A, b, c)$ with respect to G^1. $\mathcal{O}^1$ is a *refinement* of

$\mathcal{O}^G$. That is, for $O_i \in \mathcal{O}^1$, $\exists\ O_j \in \mathcal{O}^G$ where $O_i \subseteq O_j$. We define $G^i = stab(x_i, G^{i-1})$, $i = 2, 3, \ldots, n$ and $\mathcal{O}^i$ the orbital partition of the variables and constraints of $ILP(A, b, c)$ with respect to G^i.

For each orbital partition $\mathcal{O}^i$, we can use the methods above to first construct $AILP(A, b, c, \mathcal{O}^i)$ and then generate valid cuts $\lambda^a x^a \leq \beta$ for $\mathcal{F}_{AILP}$. When using G, adding the cut $\lambda_\pi x \leq \beta$ for all π only resulted in one additional constraint in the aggregate. Because we are using G^i, a subgroup of G, this is no longer the case. We need to be able to efficiently compute all unique aggregations with respect to $\mathcal{O}^i$, that is, for a given orbit O with respect to G, we wish to compute the set of representatives for the orbital partition of O with respect to G^i. To do this, we refer the reader to [21], and note that the command `orbitsDomain` in GAP does this computation.

Lastly, recall that once an optimal feasible solution is found for $AILP(A, b, c, \mathcal{O}^i)$, we lift and continue cut generation in $AILP(A, b, c, \mathcal{O}^{i+1})$. However, it is possible that a solution to $ALP(A, b, c, \mathcal{O}^{i+1})$ may not be integral when aggregated according to $\mathcal{O}^j$, $j < i$. We exploit this situation by "swapping" stabilizer levels back to G^j to generate more cutting planes in $AILP(A, b, c, \mathcal{O}^j)$. This swapping allows us to extend the time spent generating cuts with the smaller orbits. We compute $\mathcal{O}^j$ and aggregate $ILP(A, b, c)$ with respect to $\mathcal{O}^j$ when we swap stabilizer levels. Upon finding a new optimal feasible solution to $AILP(A, b, c, \mathcal{O}^j)$, we compute the new orbital partition $\mathcal{O}^{i+1}$, aggregate $ILP(A, b, c)$ with respect to $\mathcal{O}^{i+1}$, and resume cut generation in $AILP(A, b, c, \mathcal{O}^{i+1})$. Algorithm 3 formalizes iterative orbital cut generation.

Subroutines in Algorithm 3 without formal descriptions in this text are explained below.

- **ComputeOrbs**: This method computes the orbital partition of the variables and constraints of $ILP(A, b, c)$ with respect to a group.

- **CutGen**: This method generates valid cuts for $AILP(A, b, c, \mathcal{O}^i)$, lifts them to $ILP(A, b, c)$, and generates all symmetrically equivalent valid cuts for $ILP(A, b, c)$. Then, the symmetrically equivalent valid cuts are aggregated down to $AILP(A, b, c, \mathcal{O}^i)$

Algorithm 3 Orbital Cut Generation

Input: $ILP(A, b, c),\ G$

Output: z_b ▷ Best dual bound for $ILP(A, b, c)$

procedure OrbitalCutGen($ILP(A, b, c),\ G$)

$k \leftarrow 0$

$\mathcal{O}^G \leftarrow$ ComputeOrbs(G)

$AILP(A, b, c, \mathcal{O}^G) \leftarrow$ Aggregate($ILP(A, b, c),\ \mathcal{O}^G$)

$VIQ, z_b \leftarrow$ CutGen($AILP(A, b, c, \mathcal{O}^G),\ ILP(A, b, c)$) ▷ List of valid cuts, dual bound

$G^k \leftarrow G$

$\mathcal{O}^k \leftarrow \mathcal{O}^G$

while IsNotDiscrete($\mathcal{O}^k$) **do**

$k \leftarrow k + 1$

$G^k \leftarrow stab(x_k, G^{k-1})$

$\mathcal{O}^k \leftarrow$ ComputeOrbs(G^k)

$AILP(A, b, c, \mathcal{O}^k) \leftarrow$ Aggregate($ILP(A, b, c),\ \mathcal{O}^k$)

$VIQ, z_b \leftarrow$ CutGen($AILP(A, b, c, \mathcal{O}^k),\ ILP(A, b, c)$)

end while

end procedure

to improve the dual bound. This method continues until all aggregate variables are integral.

- **IsNotDiscrete**: This method determines whether or not the orbital partition is discrete. That is, each set in the partition is a singleton set.

2.6 Implementation

Orbital cut generation is implemented using a Python driver. It is a mix of of pure Python, CPython, and C code. The formulation group, stabilizer groups, and orbits are computed using `saucy` and `GAP` [43, 13]. All solving is completed using `CPLEX` due to the access it provides users to the simplex tableau and LU factorization.

CPython is used for more intense methods that aggregate $ILP(A, b, c)$ and add valid cuts to $ILP(A, b, c)$ and $AILP(A, b, c, \mathcal{O}^i)$. Simpler routines such as verifying whether or not an aggregate solution is close to integral are implemented in pure Python. Python was chosen as a driver due to the use of `SageMath` [46] for its library access to `GAP`.

2.6.1 Generating Symmetric Cutting Planes

`CPLEX` has a method `binvarow` that produces the optimal simplex tableau for a linear program. Although, to obtain the columns of the tableau corresponding to the slack variables, one must use the `binvrow` method to obtain the inverse basis matrix rows and use linear algebra to compute the reduced slack columns. We use both the `binvarow` and `binvrow` methods in the `CPLEX` Python API to compute the full optimal simplex tableau of $ALP(A, b, c, \mathcal{O}^G)$.

We construct a Gomory mixed integer cut for each reduced row with a non-slack basic variable and fractional right-hand side in the optimal simplex tableau of $ALP(A, b, c, \mathcal{O}^G)$. We then lift the cut using (2.7) to $ILP(A, b, c)$ and generate $orb(\lambda, G)$ for the lifted cut coefficient vector λ. We use the `GAP` `orbit` method that takes a permutation group, a list of

objects, and an action as arguments. The `sg.lg.Permuted` action tells `GAP` to compute the orbit of the coefficient vector by applying all permutations in G to the vector.

Each coefficient vector in $orb(\lambda, G)$ is assigned an index and the indices are added to a list and appended to $\mathcal{O}^G$. The new constraints resulting from $orb(\lambda, G)$ are added to $ILP(A, b, c)$, aggregated back down to $AILP(A, b, c, \mathcal{O}^G)$, and $ALP(A, b, c, \mathcal{O}^G)$ is resolved with the aggregate cutting planes. This continues until each integer required variable in $AILP(A, b, c, \mathcal{O}^G)$ is integral. In practice, we measure closeness to integrality and consider variables in $AILP(A, b, c, \mathcal{O}^G)$ to be integer if they are within 0.01 of integrality.

2.6.2 Exploiting Stabilizer Subgroups

Once $AILP(A, b, c, \mathcal{O}^G)$ is close to integrality, we start to exploit the stabilizer chain of G. We examine $\mathcal{O}^G$, choose the lowest index variable i that is not in a singleton set in $\mathcal{O}^G$, and compute the stabilizer subgroup G^i of G with respect to x_i. G^i will produce a new orbital partition $\mathcal{O}^i$. The new orbital partition $\mathcal{O}^i$ does not contain the orbital partition of the valid cuts that have been added to $ILP(A, b, c)$ at this point. We must compute the new orbital partition of the valid cuts under the action of G^i and append them to $\mathcal{O}^i$.

We do so by maintaining a list of lists of the coefficient vectors corresponding to co-orbital valid cuts in the lifted space. We compute the new orbital partition of the co-orbital coefficient vectors under the action of G^i using `orbitsDomain`. Once the new orbital partition of all valid cuts is known, it is appended to $\mathcal{O}^i$ and $AILP(A, b, c, \mathcal{O}^i)$ is constructed. Now, the process of generating cutting planes in the aggregate space, lifting them to the disaggregated space, generating the symmetrically equivalent cutting planes, and aggregating them back down to the aggregate space continues. The entire process is allowed to continue until we arrive at a discrete orbital partition. Like with $AILP(A, b, c, \mathcal{O}^G)$, we move to the next stabilizer subgroup in the stabilizer chain when all variables in $AILP(A, b, c, \mathcal{O}^i)$ are close to integral.

We provide a final note on the implementation of orbital cut generation. For stabilizer swapping from G^i to G^j where $j < i$, it is possible that $i - j > 1$. However, once no

more cuts can be generated at level j, we resume cut generation at level i. In practice, our implementation of orbital cut generation does not generally make it past the third level of the stabilizer chain before timing out, so resuming cut generation at $j+1, \ldots, i-1$ is not necessary. The interested reader is directed to the source code found at https://github.com/edeakins/MIPSymmetryCuts.

2.7 Benchmarks

2.7.1 Test Sets, Methods, and setup

The instances chosen for testing in this paper are combinatorial optimization problems with application in coding theory and statistical design. Instances beginning with `cod` are used to compute maximum cardinality binary error correcting codes [25]. Instances beginning with `codbt` are used to compute minimum dominating sets in Hamming graphs. Instances beginning with `cov` are covering design problems [33]. Finally, instances beginning with `sts` compute the incidence width of Steiner-triple systems [12]. These families of instances have been used as tests for many papers on symmetry in integer programming such as [27, 28, 36].

We compare orbital cut generation against the standard cutting plane method [7]. In this section, we refer to orbital cut generation as OCG and the standard cutting plane method as SCG. The methods are described in detail below.

- **OCG**: The contribution described in this paper and displayed in Algorithm 3 using Gomory's mixed integer cuts.
- **SCG**: The standard cutting plane method. This method iteratively generates Gomory's mixed integer cuts and adds them to $ILP(A, b, c)$ until the variables of $ILP(A, b, c)$ are integral.

For both OCG and SCG we generate a Gomory cut for every non-integer x variable at each $ALP(A, b, c, \mathcal{O}^i)$ solve. Note that we refer to solving the natural linear programming relaxation of $ILP(A, b, c)$ without any cuts as LPR.

Each run of OCG and SCG is given a two-hour time limit. We record the best dual bound achieved by both methods and compare against one another. Further, we compare against the dual bounds given by a rank-3 Sherali-Adams closure [44], as well as a slightly modified closure (by adding *counting constraints* [36]). We denote the rank-3 Sherali-Adams dual bound with and without counting constraints as SA and SA$^+$, respectively. We show the results from the Sherali-Adams closure to compare the relative strength of the proposed cuts, but note that computing the Sherali-Adams dual bounds can take several hours.

2.7.2 Results

Tables 2.1 and 2.2 provide data on the results of orbital cut generation benchmarks. Table 2.1 shows the best dual bounds across each method, the optimal objective value (column seven), and some statistics on cut generation for each instance in the test suite. Overall, OCG performs very well compared to LPR and only one instance did not see an improvement in best dual bound. OCG is able to achieve a better dual bound than SCG for 24 of the 30 instances. OCG matches or outperforms SA on 22 of the 30 instances. SA$^+$ outperforms OCG for 20 of the 30 instances, however, OCG does produce a dual bound for five instances that SA$^+$ could not produce in a twenty-four-hour time limit. Both SA$^+$ and OCG are able to achieve the optimal objective value for five instances. However, the five instances vary by one across both methods. That is, SA$^+$ produces the optimal objective value for `codbt33` and OCG does not, where as OCG produces the optimal objective value for `cov943` and SA$^+$ does not. Interestingly, certain instances seem to benefit most from OCG. The largest improvements in best dual bound from LPR to OCG occur in the `sts` instances with an average improvement of 58%. The `codbt` instances see the least improvement over LPR with an average improvement of 4%.

Columns eight, nine, and ten of Table 2.1 show the number of cuts added to $ILP(A, b, c)$ by SCG, number of calls to the cut generator function in orbital cut generation, and number of cuts added to $ILP(A, b, c)$ by OCG, respectively. Note that the number of calls to the cut generator function in SCG is equal to the number of cuts added to $ILP(A, b, c)$. Many

Table 2.1: Bounds for SA, SA$^+$, SCG, and OCG

Name	LPR	SA	SA$^+$	SCG	OCG	OPT	SCG Cuts	OCG Calls	OCG Cuts
Maximization									
cod103	93	90	85	92	91	72	3620	99	965635
cod105	18	18	15	18	14	12	2818	71	906755
cod83	28	26	23	27	24	20	15106	79	2171141
cod93	51	48	45	50	48	40	8398	95	1363460
Minimization									
codbt05	23	24	26	23	24	27	15291	5538	1333344
codbt15	41	42	*	41	44	54	8424	6807	3016119
codbt24	30	31	*	30	31	36	121274	8208	2518454
codbt33	22	23	**24**	22	23	24	17161	8932	1792910
codbt34	54	56	*	54	57	72	6075	3875	2389178
codbt42	16	18	19	17	17	20	23436	10325	1391041
codbt43	40	41	*	40	41	48	9207	6090	2481410
codbt52	29	31	*	29	30	36	13604	7677	2083682
codbt61	22	23	**24**	22	**24**	24	19195	62	10083
codbt71	39	41	43	39	40	48	10142	2353	810244
codbt80	29	31	**32**	30	**32**	32	14764	27	67587
codbt90	52	54	56	52	52	62	8283	60	750594
cov1053	12	13	14	12	14	17	15971	188	825321
cov1054	42	45	46	45	45	51	15560	168	638192
cov1075	12	14	16	13	14	20	25326	21478	193118
cov1076	30	39	39	32	36	45	25554	3754	7733103
cov1174	10	10	12	16	16	17	18395	148	825002
cov743	9	11	**12**	10	**12**	12	37575	198	41080
cov832	10	10	**11**	**11**	**11**	11	3114	6	114
cov943	21	22	23	23	**25**	25	21189	102	397779
cov954	26	28	29	28	28	30	24674	155	1039251
sts15	5	8	8	7	8	9	40047	137	2328
sts27	9	15	16	12	16	18	38398	109	467129
sts45	15	24	24	19	21	30	36394	654	37862
sts63	21	34	37	27	34	45	33806	493	592492
sts81	27	44	48	32	40	61	31821	10761	872493

Table 2.2: IP Gap (%) for LPR, SCG, and OCG

Name	LPR Gap (%)	SCG Gap (%)	OCG Gap (%)
Maximization			
cod103	23	22	21
cod105	33	33	14
cod83	29	26	17
cod93	22	20	17
Minimization			
codbt05	17	17	13
codbt15	32	32	23
codbt24	20	20	16
codbt33	9	9	4
codbt34	33	33	26
codbt42	25	18	18
codbt43	20	20	17
codbt52	24	24	20
codbt61	9	9	0
codbt71	23	23	20
codbt80	10	7	0
codbt90	19	19	19
cov1053	42	42	21
cov1054	21	13	13
cov1075	67	54	43
cov1076	50	41	25
cov1174	70	6	6
cov743	33	20	0
cov832	10	0	0
cov943	19	9	0
cov954	15	7	7
sts15	80	29	13
sts27	100	50	13
sts45	100	58	43
sts63	114	67	32
sts81	126	91	53
	40	27	17

more cuts are added using OCG, however with fewer calls to the cut generator. On average, for every one call to the cut generator in OCG, 3,827 symmetrically equivalent cuts are generated. Further, OCG averages 103 times as many cuts generated in the two-hour time limit.

Table 2.2 shows the gap between the best dual bound achieved across LPR, SCG, and OCG and the optimal objective value for each instance. We compute the gap for each method using the standard formula for mixed integer programming (MIP) gap used by solvers. The last row in Table 2.2 gives the average of each column. The average gap for LPR is 40% showing that there is plenty of room for improvement by strengthening the natural relaxation of $ILP(A, b, c)$. SCG decreases the average final gap to 27% and OCG furthers this trend pushing the average final gap down to 17%. The average gap using OCG is least for the `cov` instances at 13% and greatest for the `sts` instances at 31%. We note that the `sts` instances have notoriously bad relaxations and the greatest average gap for LPR as well.

2.8 Discussion

OCG proves to be an effective method at improving the dual bound at the root node compared to SCG. All instances, except one, see an improvement in the dual bound for LPR vs. OCG with an average improvement of 19% across all instances. As for SCG vs. OCG, all instances, except five, see an improvement in the dual bound. The instances with the largest improvement are `sts` instances. The `sts` family of instances is well known for having very weak relaxations. This may provide some insights as to why these instances see the most improvement.

With the exception of `cod103`, the `codbt` family of instances seems to be the hardest to improve. Many of the instances that see the least improvement using OCG tend to start with a tighter dual bound, although, this is not always the case (see `codbt34` in Table 2.1). OCG is able to find the optimal objective value for five of the thirty instances in the test suite. Four of these five instances are the same instances that SA^+ achieves the optimal objective

value. However, OCG is able to achieve the optimal objective value for one instance that SA^+ is not and vice versa.

We note that a very naive cut selection routine was implemented for OCG and SCG. We use Gomory mixed integer cutting planes and select all rows having a fractional right-hand side unless the corresponding basic variable is a slack variable. With a better selection rule for which cuts and how many cuts to generate, the results of OCG would likely improve. Further, using various types of cutting planes would most likely produce an improvement compared to the naive implementation of OCG.

Bibliography

[1] Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh, and Pamela H Vance. Column generation for solving huge integer programs, 1996. 26

[2] Christoph Berkholz, Paul Bonsma, and Martin Grohe. Tight lower and upper bounds for the complexity of canonical colour refinement. *Theory of Computing Systems*, 60(4):581–614, 2017. 8

[3] Robert E. Bixby and Matthew J. Saltzman. Recovering an optimal lp basis from an interior point solution. *Operations Research Letters*, 15(4):169–178, 1994. 4

[4] Peter J Cameron et al. *Permutation groups*. Number 45. Cambridge University Press, 1999. 6, 27

[5] Alberto Caprara and Matteo Fischetti. Branch-and-cut algorithms. *Annotated bibliographies in combinatorial optimization*, pages 45–64, 1997. 25

[6] Christopher Maes, Edward Rothberg, Zonghao Gu, Robert Bixby. Initial basis selection for lp crossover. URL: https://cerfacs.fr/wp-content/uploads/2016/01/maes.pdf, 6 2014. 4

[7] Michele Conforti, Gérard Cornuéjols, Giacomo Zambelli, et al. *Integer programming*, volume 271. Springer, 2014. 25, 45

[8] George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954. 25

[9] George B Dantzig and Mukund N Thapa. *Linear programming 1: introduction*. Springer Science & Business Media, 2006. 4

[10] Paul T. Darga, Mark H. Liffiton, Karem A. Sakallah, and Igor L. Markov. Exploiting structure in symmetry detection for cnf. In *Proceedings of the 41st Annual Design Automation Conference*, DAC '04, page 530–534, New York, NY, USA, 2004. Association for Computing Machinery. 17, 59

[11] Matthias Elf, Carsten Gutwenger, Michael Jünger, and Giovanni Rinaldi. Branch-and-cut algorithms for combinatorial optimization and their implementation in abacus. In *Computational Combinatorial Optimization*, pages 157–222. Springer, 2001. 25

[12] DR Fulkerson, George L Nemhauser, and LE Trotter Jr. Two computationally difficult set covering problems that arise in computing the 1-width of incidence matrices of steiner triple systems. Technical report, WISCONSIN UNIV-MADISON MATHEMATICS RESEARCH CENTER, 1974. 45

[13] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.12.2*, 2022. 43

[14] Ralph Gomory. An algorithm for the mixed integer problem. Technical report, RAND CORP SANTA MONICA CA, 1960. 25

[15] Ralph E Gomory. Solving linear programming problems in integers. *Combinatorial Analysis*, 10:211–215, 1960. 25

[16] Ralph E Gomory. An algorithm for integer solutions to linear programs. *Recent advances in mathematical programming*, 64(260-302):14, 1963. 25

[17] Martin Grohe, Kristian Kersting, Martin Mladenov, and Erkal Selman. Dimension reduction via colour refinement. In *European Symposium on Algorithms*, pages 505–516. Springer, 2014. 4, 7, 9, 11

[18] Martin Grötschel and Olaf Holland. Solution of large-scale symmetric travelling salesman problems. *Mathematical Programming*, 51(1-3):141–202, 1991. 25

[19] Larry C Grove and Clark T Benson. *Finite reflection groups*, volume 99. Springer Science & Business Media, 1996. 6, 27

[20] Søren Holm and Michael Malmros Sørenson. *The Optimal Graph Partitioning Problem: Solution Method Based on Reducing Symmetric Nature and Combinatorial Cuts*. Aarhus School of Business, 1992. 26

[21] Derek F Holt, Bettina Eick, and Eamonn A O'Brien. *Handbook of computational group theory*. Chapman and Hall/CRC, 2005. 41

[22] Qi Huangfu and JA Julian Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018. 17

[23] M Junger, G Reinelt, and S Thienel. Practical problem solving with cutting plane algorithms in combinatorial optimization, dimacs series in discrete mathematics and theoretical computer science. *American Mathematical Society*, 111, 1995. 25

[24] Michael Jünger and Denis Naddef. *Computational combinatorial optimization: optimal or provably near-optimal solutions*, volume 2241. Springer, 2003. 25

[25] Simon Litsyn. An update table of the best binary codes known. *Handbook of Coding Theory*, 1998. 18, 45

[26] Elder M Macambira, Nelson Maculan, and CC de Souza. Reducing symmetry of the sonet ring assignment problem using hierarchical inequalities. Technical report, Technical Report ES-636/04, Programa de Engenharia de Sistemas e Computaçao . . . , 2004. 26

[27] François Margot. Pruning by isomorphism in branch-and-cut. *Mathematical Programming*, 94(1):71–90, 2002. 26, 45

[28] François Margot. Exploiting orbits in symmetric ilp. *Mathematical Programming*, 98(1):3–21, 2003. 7, 26, 45

[29] François Margot. *Symmetry in Integer Linear Programming*, pages 647–686. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. 6, 26, 28

[30] Harry M Markowitz and Alan S Manne. On the solution of discrete programming problems. *Econometrica: journal of the Econometric Society*, pages 84–110, 1957. 25

[31] Brendan D McKay. Nauty user's guide (version 2.4). *Computer Science Dept., Australian National University*, pages 225–239, 2007. 17

[32] Isabel Méndez-Díaz and Paula Zabala. A branch-and-cut algorithm for graph coloring. *Discrete Applied Mathematics*, 154(5):826–847, 2006. 26

[33] WH Mills and RC Mullin. *Coverings and packings*. New York, NY: Wiley, 1992. 18, 45

[34] John E Mitchell. Branch-and-cut algorithms for combinatorial optimization problems. *Handbook of applied optimization*, 1(1):65–77, 2002. 25

[35] James Ostrowski. *Symmetry in integer programming*. Lehigh University, 2009. 6, 27

[36] James Ostrowski. Using symmetry to optimize over the sherali–adams relaxation. *Mathematical Programming Computation*, 6:405–428, 2014. 45, 46

[37] James Ostrowski, Jeff Linderoth, Fabrizio Rossi, and Stefano Smriglio. Orbital branching. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 104–118. Springer, 2007. 26

[38] James Ostrowski, Jeff Linderoth, Fabrizio Rossi, and Stefano Smriglio. Constraint orbital branching. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 225–239. Springer, 2008. 26, 27

[39] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991. 25

[40] Peter Dobsan. Pynauty pypi web page. URL: https://pypi.org/project/pynauty/, 2022. 17

[41] E Rothberg. Using cuts to remove symmetry. In *17th International Symposium on Mathematical Programming*, 2000. 26

[42] Joseph J Rotman. *An introduction to the theory of groups*, volume 148. Springer Science & Business Media, 2012. 6, 27

[43] Jonathan David Schrock. Symmetry detection in integer linear programs. 2015. 17, 43, 59

[44] Hanif D Sherali and Warren P Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3(3):411–430, 1990. 46

[45] Hanif D Sherali and J Cole Smith. Improving discrete model representations via symmetry considerations. *Management Science*, 47(10):1396–1407, 2001. 26

[46] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.7)*, 2023. `https://www.sagemath.org`. 43

[47] Laurence A Wolsey. *Integer programming*. John Wiley & Sons, 2020. 25

[48] Laurence A Wolsey and George L Nemhauser. *Integer and combinatorial optimization*, volume 55. John Wiley & Sons, 1999. 25

Appendices

Appendix A

Orbital Crossover Implementation

A.1 Nomenclature

A.1.1 Input Data

n	Number of columns in $LP(A, b, c)$.
n^k_{col}	Number of sets in $\mathcal{V}^k$.
n^k_{row}	Number of sets in $\mathcal{C}^k$.
P^k_{part}	Array tracking which set $P^k_i \in \mathcal{P}^k$ each variable/constraint belongs to.
Ai^p	Array of $P^k_{part}[i], \forall i \in Ai^o$.
P^k_{size}	Array of the size of each set $P^k_i \in \mathcal{P}^k$.
P^k_{set}	Array of the sets $P^k_i \in \mathcal{P}^k$.
B^a	Array of basis statuses for all variables in solution $ALP(A, b, c, \mathcal{P}^k)$.

A.1.2 Output Data

F	Temporary array for a new aggregate row entering Ai^{temp}.
Av^a	Array of constraint matrix coefficients in $ALP(A, b, c, \mathcal{P}^k)$.
Av^{temp}	Temporary array of constraint matrix coefficients in $ALP(A, b, c, \mathcal{P}^k)$.
Ai^a	Array of constraint matrix indices in $ALP(A, b, c, \mathcal{P}^k)$.

Ai^{temp}	Temporary Array of constraint matrix indices in $ALP(A, b, c, \mathcal{P}^k)$.
As^a	Array of column starts in constraint matrix in $ALP(A, b, c, \mathcal{P}^k)$.
(V^a, S^a)	Solution to $ALP(A, b, c, \mathcal{P}^k)$.
Av^e	Array of constraint matrix coefficients in $ELP(A, b, c, \mathcal{P}^k)$.
Ai^e	Array of constraint matrix indices in $ELP(A, b, c, \mathcal{P}^k)$.
As^e	Array of column starts in constraint matrix in $ELP(A, b, c, \mathcal{P}^k)$.
Av^o	Array of constraint matrix coefficients in $LP(A, b, c)$.
Ai^o	Array of constraint matrix indices in $LP(A, b, c)$.
As^o	Array of column starts in constraint matrix in $LP(A, b, c)$.
Obj^a	Array of objective coefficients in $ALP(A, b, c, \mathcal{P}^k)$.
Obj^e	Array of objective coefficients in $ELP(A, b, c, \mathcal{P}^k)$.
Obj^o	Array of objective coefficients in $LP(A, b, c, \mathcal{P}^k)$.
L^a	Array of variable lower bounds in $ALP(A, b, c, \mathcal{P}^k)$.
U^a	Array of variable upper bounds in $ALP(A, b, c, \mathcal{P}^k)$.
L^e	Array of variable lower bounds in $ELP(A, b, c, \mathcal{P}^k)$.
U^e	Array of variable upper bounds in $ELP(A, b, c, \mathcal{P}^k)$.
L^o	Array of variable lower bounds in $LP(A, b, c, \mathcal{P}^k)$.
U^o	Array of variable upper bounds in $LP(A, b, c, \mathcal{P}^k)$.
Rhs^a_L	Array of constraint right-hand side lower bounds in $ALP(A, b, c, \mathcal{P}^k)$.
Rhs^a_U	Array of constraint right-hand side upper bounds in $ALP(A, b, c, \mathcal{P}^k)$.
Rhs^e_L	Array of constraint right-hand side lower bounds in $ELP(A, b, c, \mathcal{P}^k)$.
Rhs^e_U	Array of constraint right-hand side upper bounds in $ELP(A, b, c, \mathcal{P}^k)$.
Rhs^o_L	Array of constraint right-hand side lower in $LP(A, b, c, \mathcal{P}^k)$.
Rhs^o_U	Array of constraint right-hand side upper in $LP(A, b, c, \mathcal{P}^k)$.
B^e	Array of basis statuses for all variables in initial solution to $ELP(A, b, c, \mathcal{P}^k)$.
r^k_{links}	Array of the linking variables in $ELP(A, b, c, \mathcal{P}^k)$.

A.2 Implementation Details

Note that data containers that are associated with $ALP(A, b, c, \mathcal{P}^k)$ and $ELP(A, b, c, \mathcal{P}^k)$ have their entries set to zero before each major iteration k. This includes temporary arrays that house data for a short period during Algorithm 4. $R(\cdot)$ retains it definition from Section 1.3. The basis statuses mentioned in the definitions of B^a and B^e are: 0 (nonbasic at lower bound), 1 (basic), 2 (nonbasic at upper bound), 3 (zero), and 4 (nonbasic with no bound information). These values comes from HiGHS, but many commercial and other open sources solvers use a similar system for classifying the basis/nonbasic status of variables in a LP solution. Finally, we let $\mathcal{P}^0 = \mathcal{P}$ and note that all arrays containing constraint matrix data for $ALP(A, b, c, \mathcal{P})$ and $ELP(A, b, c, \mathcal{P})$ are CSC formatted. The following sections give a detailed description of the methods that make up the bold font sub-functions from Algorithm 1.

Equitable Partitions (**EP** and **Refine**):

For the purposes of this paper, we leave out the direct implementation of our EP routine. There is a large amount of components that integrate to compute and refine EPs in our OC implementation. The backbone of EP implementation comes from **saucy** and details for saucy can be seen in [10, 43]. It suffices to explain that **EP** accepts $LP(A, b, c)$ as input and computes an initial EP. If the EP is not discrete and **Refine** is called, the representative of the largest non-slack set is isolated and a new EP is computed afterwards. We refer the interested reader to the source code at https://github.com/edeakins/EQLPSolver/tree/stable/HiGHS-1-2-1/src/presolve in the `OCEquitable.cpp` and `OCEquitable.h` files.

Aggregation (**Fold**):

Algorithm 4 is used to construct the aggregate A matrix for $ALP(A, b, c, \mathcal{P}^k)$. The aggregate A matrix is constructed via nested for-loops. The outer loop iterates across each variable set $P_i^k \in \mathcal{V}^k$ recording $x_{R(P_i^k)}$. Once $x_{R(P_i^k)}$ is known, the first inner for-loop calculates the column wise entries for the aggregate column $x^a_{R(P_i^k)}$ using the coefficients of $x_{R(P_i^k)}$ in $LP(A, b, c)$. The second inner for-loop adds the coefficients and aggregate row indices of $x^a_{R(P_i^k)}$ into Av^a and Ai^a, respectively. Lastly, the second inner for-loop clears the

values from temporary data containers. Once Algorithm 4 exists both inner for-loops, As^a is updated to reflect the number of nonzero entries in aggregate column $x^a_{R(P_i^k)}$. The value of *start* is updated with the current number of nonzero entries in the aggregate A matrix. Note that $R(P_i^k)$ is a naming index on the aggregate variable $x^a_{R(P_i^k)}$. The actual internal index for the aggregate column $x^a_{R(P_i^k)}$ is i and this is why As^a is updated via $As^a[i+1]$.

Algorithm 5, aggregates the objective function from $LP(A, b, c)$ down to an objective function for $ALP(A, b, c, \mathcal{P}^k)$. For each $P_i^k \in \mathcal{V}^k$, the objective coefficient of $x_{R(P_i^k)}$ in $LP(A, b, c)$ is extracted and then multiplied by $|P_i^k|$. Similar to Algorithm 4, multiplying the coefficient entry of $x_{R(P_i^k)}$ in $ALP(A, b, c, \mathcal{P}^k)$ means only the representatives of each set in $\mathcal{V}^k$ must be considered. Next, the constraint right-hand sides must be aggregated as well. Algorithm 6 shows this routine.

Finally, Algorithm 7 is used to construct the upper and lower bounds for each aggregate variable in $ALP(A, b, c, \mathcal{P}^k)$. For each $P_i^k \in \mathcal{V}^k$, the upper and lower bound of $x_{R(P_i^k)}$ in $LP(A, b, c)$ is extracted and recorded as the upper and lower bound of the aggregate variable $x^a_{R(P_i^k)}$. No multiplication is performed in this function because each aggregated variable represents the average value of each variable in its respective set in $\mathcal{P}^k$. Thus, the average of the aggregate variable $x_{R(P_i^k)}$ representing P_i^k must be within bounds of the variables in P_i^k. The **Fold** sub-function in Algorithm 1 comprises Algorithms 4 - 7 to construct $ALP(A, b, c, \mathcal{P}^k)$.

Algorithm 4 Construct $ALP(A, b, c, \mathcal{P}^k)$ Constraint Matrix

function FOLDMATRIX($LP(A, b, c)$, $\mathcal{P}^k = (\mathcal{V}^k, \mathcal{C}^k)$)
  $start \leftarrow 0$
  $nnz \leftarrow 0$
  $As^a[0] \leftarrow 0$
  for $i \leftarrow 0$, $n^k_{col} - 1$ **do**
    $q \leftarrow 0$
    $P^k_i \leftarrow P^k_{set}[i]$
    $r \leftarrow R(P^k_i)$
    $s \leftarrow P^k_{size}[i]$
    for $j \leftarrow As^o[r]$, $As^o[r+1]$ **do**
      $c \leftarrow Ai^p[j]$
      $w \leftarrow Av^o[j]$
      if $F[c] = 0$ **then**
        $F[c] \leftarrow F[c] + 1$
        $Ai^{temp}[nnz] = c$
        $nnz \leftarrow nnz + 1$ ▷ Number of nonzeros in ALP
      end if
      $Av^{temp}[c] \leftarrow Av^{temp}[c] + w \times s$
    end for
    for $j \leftarrow start$, nnz **do**
      $c \leftarrow Ai^{temp}[j]$
      $w \leftarrow Av^{temp}[c]$
      $Av^a[nnz] \leftarrow w$
      $Ai^a[nnz] \leftarrow c$
      $Av^{temp}[c] = 0$
      $Ai^{temp}[j] = 0$
      $F[c] = 0$
    end for
    $As^a[i+1] \leftarrow nnz$
    $start \leftarrow nnz$
  end for
end function

Algorithm 5 Construct $ALP(A, b, c, \mathcal{P}^k)$ Objective Function

function FOLDOBJECTIVE($LP(A, b, c)$, $\mathcal{P}^k = (\mathcal{V}^k, \mathcal{C}^k)$)
  for $i \leftarrow 0$, $n^k_{col} - 1$ **do**
    $P^k_i \leftarrow P^k_{set}[i]$
    $r \leftarrow R(P^k_i)$
    $s \leftarrow P^k_{size}[i]$
    $v \leftarrow Obj^o[r]$
    $Obj^a[i] \leftarrow v \times s$
  end for
end function

Algorithm 6 Construct $ALP(A, b, c, \mathcal{P}^k)$ Right-hand Sides

function FOLDRHS($LP(A, b, c)$, $\mathcal{P}^k = (\mathcal{V}^k, \mathcal{C}^k)$)
  for $i \leftarrow 0$, $n^k_{row} - 1$ **do**
    $P^k_i \leftarrow P^k_{set}[i + n^k_{col}]$
    $r \leftarrow R(P^k_i)$
    $s \leftarrow P^o_{size}[i]$
    $l \leftarrow Rhs^o_L[r - n]$
    $u \leftarrow Rhs^o_U[r - n$
    $Rhs^a_L[i] \leftarrow l \times s$
    $Rhs^a_L[i] \leftarrow u \times s$
  end for
end function

Algorithm 7 Construct $ALP(A, b, c, \mathcal{P}^k)$ Variable Bounds

function FOLDBNDS($LP(A, b, c)$, $\mathcal{P}^k = (\mathcal{V}^k, \mathcal{C}^k)$)
  for $i \leftarrow 0$, $n^k_{col} - 1$ **do**
    $P^k_i \leftarrow P^k_{set}[i]$
    $r \leftarrow R(P^k_i)$
    $l \leftarrow L^o[r]$
    $u \leftarrow U^o[r]$
    $L^a[i] \leftarrow l$
    $U^a[i] \leftarrow u$
  end for
end function

Solving $ALP(A, b, c, \mathcal{P}^0)$ (**Solve**):

$ALP(A, b, c, \mathcal{P}^0)$ is solved using either using HiGHS serial dual simplex solver or HiGHS IPM with HiGHS crossover. The choice is left to the user and specified via options when calling the modified HiGHS code containing OC.

Lifting (**Extend**):

Note that the **Extend** sub-function from Algorithm 1 uses Algorithm 4 as well to build the initial A matrix of $ELP(A, b, c, \mathcal{P}^k)$ and linking variables/constraints are added later. Recall that after OC, the solution to $ALP(A, b, c, \mathcal{P}^{k-1})$ is the solution to $ELP(A, b, c, \mathcal{P}^{k-1})$ without the r variables. If the variable bound for $x^a_{R(P_i^{k-1})}$ is active in the solution to $ALP(A, b, c, \mathcal{P}^{k-1})$, then the variable bound(s) for $x^a_{R(P_j^k)}$ $\forall P_j^k \subseteq P_i^{k-1}$ will be active in $ELP(A, b, c, \mathcal{P}^k)$ Aggregate variable bounds in $ELP(A, b, c, \mathcal{P}^k)$ are determined via Algorithm 8. The same logic is applied to determine the right-hand sides of aggregate constraints in Algorithm 9. Note in Algorithm 9 that S^a contains the row values from the solution to $ALP(A, b, c, \mathcal{P}^{k-1})$.

Next, the basis for $ALP(A, b, c, \mathcal{P}^{k-1})$ is lifted similarly to how the variable bounds and right-hand sides are. Algorithm 10 explains this process. B^e initially has all entries set to a value of 4 (nonbasic without bound information). Non-slack aggregate variables $x^a_{R(P_j^k)}$ are given the same basis status as $x^a_{R(P_i^{k-1})}$ so long as $P_j^k \subseteq P_i^{k-1}$. If $R(P_i^{k-1}) = R(P_j^k)$ and $s^a_{R(P_i^{k-1})}$ is anything but basic, then $s^a_{R(P_i^k)}$ is given the same basis status as $s_{R(P_i^{k-1})}$. Otherwise, $s_{R(P_i^k)}$ is set to basic. This follows from our basis construction method in the proof for Theorem 1.2 and the method for reducing the number of r variables from Section 1.4.

Algorithm 8 Fix Active Variables in $ELP(A, b, c, \mathcal{P}^k)$

function FIXBNDS($LP(A, b, c)$, $\mathcal{P}^{k-1} = (\mathcal{V}^{k-1}, \mathcal{C}^{k-1})$, $\mathcal{P}^k = (\mathcal{V}^k, \mathcal{C}^k)$, V^a)
  for $i \leftarrow 0$, $n^k_{col} - 1$ **do**
    $P^k_i \leftarrow P^k_{set}[i]$
    $r \leftarrow R(P^k_i)$
    $j \leftarrow P^{k-1}_{part}[r]$
    $l \leftarrow L^o[r]$
    $u \leftarrow U^o[r]$
    if $V^a[j] = l$ **then**
      $L^e[i] \leftarrow l$
      $U^e[i] \leftarrow l$
    else if $V^a[j] = u$ **then**
      $L^e[i] \leftarrow u$
      $U^e[i] \leftarrow u$
    else
      $L^e[i] \leftarrow l$
      $U^e[i] \leftarrow u$
    end if
  end for
end function

Algorithm 9 Fix Active Rows in $ELP(A, b, c, \mathcal{P}^k)$

function FIXRHS($LP(A, b, c)$, $\mathcal{P}^{k-1} = (\mathcal{V}^{k-1}, \mathcal{C}^{k-1})$, $\mathcal{P}^k = (\mathcal{V}^k, \mathcal{C}^k)$, S^a)
  for $i \leftarrow 0$, n_{row}^{k+1} **do**
    $P_i^k \leftarrow P_{set}^k[i + n_{col}^k]$
    $r \leftarrow R(P_i^k)$
    $j_1 \leftarrow P_{part}^k[r]$
    $j_2 \leftarrow P_{part}^{k-1}[r]$
    $s^k \leftarrow P_{size}^k[j_1]$
    $s^{k-1} \leftarrow P_{size}^{k-1}[j_2]$
    $l \leftarrow Rhs_L^o[r]$
    $u \leftarrow Rhs_U^o[r]$
    if $S^a[j - n_{col}^{k-1}] = l \times s^{k-1}$ **then**
      $Rhs_L^e[i] \leftarrow l \times s^k$
      $Rhs_U^e[i] \leftarrow l \times s^k$
    else if $S^a[j - n_{col}^{k-1}] = u \times s^{k-1}$ **then**
      $Rhs_L^e[i] \leftarrow u \times s^k$
      $Rhs_U^e[i] \leftarrow u \times s^k$
    else
      $Rhs_L^e[i] \leftarrow l \times s^k$
      $Rhs_U^e[i] \leftarrow u \times s^k$
    end if
  end for
end function

Algorithm 10 Lift the Basic/Nonbasic Columns to $ELP(A, b, c, \mathcal{P}^k)$

function LIFTBASIS($LP(A, b, c)$, $\mathcal{P}^{k-1} = (\mathcal{V}^{k-1}, \mathcal{C}^{k-1})$, $\mathcal{P}^k = (\mathcal{V}^k, \mathcal{C}^k)$, B^a)
  for $i \leftarrow 0$, $n^k_{col} + n^k_{row} - 1$ **do**
    $B^e[i] \leftarrow 4$
  end for
  for $i \leftarrow 0$, $n^k_{col} - 1$ **do**
    $P^k_i \leftarrow P^k_{set}[i]$
    $r \leftarrow R(P^k_i)$
    $j \leftarrow P^{k-1}_{part}[r]$
    $b \leftarrow B^a[j]$
    $B^e[i] \leftarrow b$
  end for
  for $i \leftarrow 0$, $n^k_{row} - 1$ **do**
    $P^k_i \leftarrow P^k_{set}[i + n^k_{col}]$
    $r_2 \leftarrow R(P^k_i)$
    $P^{k-1}_i \leftarrow P^{k-1}_{set}[r]$
    $r_1 \leftarrow R(P^{k-1}_i)$
    $j \leftarrow P^{k-1}_{part}[r_1]$
    $b \leftarrow B^a[j]$
    if $b \neq 1$ **and** $r_1 = r_2$ **then**
      $B^e[i + n^k_{col}] \leftarrow b$;
    end if
  end for
end function

Next, given a partition $\mathcal{P}^k$, the linking r variables and their corresponding rows must be constructed. This process is detailed in Algorithm 11. For each new set $P_j^k \in \mathcal{V}^k$, a linking pair $x_{R(P_j^k)}, x_{R(P_i^{k-1})}$ such that $P_j^k \subset P_i^{k-1}$ is passed to the sub-function **addLinks** that adds the linking variables and rows to $ELP(A, b, c, \mathcal{P}^k)$. The details of **addlinks** are left out of this paper as it is comprised of resizing LP data arrays, moving data within these arrays, and inserting new data into these arrays.

Orbital Crossover Pivoting (**OC**):

Finally, Algorithm 12 revises typical entering column selection procedure within HiGHS for the primal simplex algorithm so that only the linking r variables are selected as entering columns. As the linking variables are selected to enter the basis, their lower and upper bounds are set to $-\infty$ and ∞, respectively. Once the entering column is identified, the primal simplex method continues by choosing the row for the leaving column, updating/re-computing primal LP values, and (if necessary) re-factoring the LU factorization of the inverse basis matrix B^{-1}.

The functions **primalChooseRow**, **primalUpdate**, and **primalRebuild** are internal functions of HiGHS and choose the leaving row, update the primal problem, and rebuild the LU factorization of B^{-1}, respectively. Further, the value of *update* is determined within HiGHS primal and dual simplex code that require a LU re-factorization.

Algorithm 11 Construct Linking Variables and Constraints for $ELP(A, b, c, \mathcal{P}^k)$

```
function createLinks(LP(A,b,c), P^{k-1} = (V^{k-1}, C^{k-1}), P^k = (V^k, C^k), B^a)
    for i ← n_col^{k-1}, n_col^k do
        P_i^k ← P_set^k[i]
        r ← R(P_i^k)
        j ← P_part^{k-1}[r]
        if B^a[j] = 1 then
            addLinks(j, i)
        end if
    end for
end function
```

Algorithm 12 Primal Simplex Iteration with Revised Pivoting Rules

function PIVOTONLINKS($ELP(A, b, \mathcal{P}^k), V^{k-1}, V^k$)
  $n^k_{links} \leftarrow |V^k| - |V^{k-1}|$
  for $i \leftarrow 0$, n^k_{links} **do**
    $c_{in} \leftarrow r^k_{links}[i]$
    $U^e[c_{in}] \leftarrow \infty$
    $L^e[c_{in}] \leftarrow -\infty$
    primalChooseRow()
    primalUpdate()
    if $update > 0$ **then**
      primalRebuild()
    end if
  end for
end function

Vita

Ethan Jedidiah Deakins was born on October 6, 1995, to Tim and Melissa Deakins. He attended Heritage High School in Maryville, Tennessee. Upon graduation, he enrolled at the University of Tennessee, Knoxville and in the spring of 2014 he completed a undergraduate degree in Industrial Engineering.

In the Summer of 2018, Ethan enrolled at the University of Tennessee, Knoxville, to pursue his PhD in Industrial Engineering under the guidance of James Ostrowski. He was supported in part by the university's chancellor's fellowship his first four years. Ethan completed his PhD in Industrial Engineering in March 2023.