12-2017

Almost Symmetries and the Unit Commitment Problem

Bernard Albert Knueven
University of Tennessee, bknueven@vols.utk.edu

To the Graduate Council:

I am submitting herewith a dissertation written by Bernard Albert Knueven entitled "Almost Symmetries and the Unit Commitment Problem." 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:

John E. Kobza, Michael A. Langston, Oleg Shylo

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

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

Almost Symmetries and the Unit Commitment Problem

A Dissertation Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Bernard Albert Knueven

December 2017

Acknowledgments

I would like to express my deepest gratitude to my advisor, Professor James Ostrowski, for all his support and guidance. He and Professor Olga Brezhneva played a pivotal role in my returning to graduate school. I would like to thank Dr. Jean-Paul Watson for the opportunity he provided me to work at Sandia National Laboratories, and for ensuring my time in Albuquerque, NM was both productive and enjoyable. I would also like to thank Professors Oleg Shylo, John E. Kobza, and Michael A. Langston for serving on my thesis committee.

I would like to express my appreciation to the University of Tennessee, the National Science Foundation, and the U.S. Department of Energy, for providing funding for my graduate studies and the work in this thesis.

Finally, I would like to thank all the wonderful friends I have made these past four years: Mike Hare, Tony Rodriguez, Whitney Forbes, Amelia and Derek McIlvenna, Bryan Arguello, Alegria Salazar, Bismark Singh, Austin Love, Magen Shedden, Terry Hendersen, Christopher Muir, Ethan Deakins, Jonathan Schrock, and Jillian Blueford. I have benefited greatly from their friendship and encouragement.

Abstract

This thesis explores two main topics. The first is almost symmetry detection on graphs. The presence of symmetry in combinatorial optimization problems has long been considered an anathema, but in the past decade considerable progress has been made. Modern integer and constraint programming solvers have automatic symmetry detection built-in to either exploit or avoid symmetric regions of the search space. Automatic symmetry detection generally works by converting the input problem to a graph which is in exact correspondence with the problem formulation. Symmetry can then be detected on this graph using one of the excellent existing algorithms; these are also the symmetries of the problem formulation.

The motivation for detecting almost symmetries on graphs is that almost symmetries in an integer program can force the solver to explore nearly symmetric regions of the search space. Because of the known correspondence between integer programming formulations and graphs, this is a first step toward detecting almost symmetries in integer programming formulations. Though we are only able to compute almost symmetries for graphs of modest size, the results indicate that almost symmetry is definitely present in some real-world combinatorial structures, and likely warrants further investigation.

The second topic explored in this thesis is integer programming formulations for the unit commitment problem. The unit commitment problem involves scheduling power generators to meet anticipated energy demand while minimizing total system operation cost. Today, practitioners usually formulate and solve unit commitment as a large-scale mixed integer linear program.

The original intent of this project was to bring the analysis of almost symmetries to the unit commitment problem. Two power generators are almost symmetric in the unit commitment problem if they have almost identical parameters. Along the way, however, new formulations for power generators were discovered that warranted a thorough investigation of their own. Chapters 4 and 5 are a result of this research.

Thus this work makes three contributions to the unit commitment problem: a convex hull description for a power generator accommodating many types of constraints, an improved formulation for time-dependent start-up costs, and an exact symmetry reduction technique via reformulation.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, we introduce some foundational concepts, and discuss the motivations for the work presented in subsequent chapters. We present some background on integer programming and methods for symmetry handling in integer programming. We then turn to the study of almost symmetries and their potential applications in optimization. Finally, we consider the preceding in the context of a real-world scheduling problem, namely the unit commitment problem.

1.1 Integer Programming

Integer programming (IP) is a broad modeling paradigm for optimization problems which allows the practitioner to model objects that by their nature come in discrete quantities. Such objects could be the number of workers a company hires to complete a certain task, or whether a power generator is on or off. Modeling flexibility can be further extended by mixing integer and continuous variables, for example, if a power generator is on, its power output may be allowed to vary between two specified quantities.

Put most broadly, an integer program is an optimization problem of the following form:

$$\begin{aligned} &\min\ f(x,y) \\ &\text{subject to } g_1(x,y) \leq 0 \\ &\qquad\vdots \\ &\qquad g_m(x,y) \leq 0 \\ &\qquad x \in \mathbb{R}^n,\ y \in \mathbb{Z}^p, \end{aligned} \tag{1.1}$$

where $f, g_1, \ldots, g_m : \mathbb{R}^{n+p} \to \mathbb{R}$ are arbitrary functions. When $n = 0$, i.e., there are no continuous variables, this is sometimes called a *pure integer program*, otherwise it is called a *mixed integer program*. If one of the functions $f, g_1, \ldots, g_m$ is nonlinear, then (1.1) is called a *mixed integer nonlinear program* (MINLP). If all the functions $f, g_1, \ldots, g_m$ are linear (or really, affine), then (1.1) is called a *mixed integer linear program* (MILP). Conforti et al. [21] provide a thorough overview on integer linear programming; for a recent overview on MINLP, see [150]. A historical perspective on integer programming is given by Jünger et al. [72].

This dissertation will only concern itself with mixed integer linear programs. When all the functions in (1.1) are linear, we usually write the objective function and constraints in a matrix-vector form:

$$\begin{aligned} \min\ & c^T x + h^T y \\ \text{subject to } & Ax + Gy \leq b \\ & x \in \mathbb{R}^n,\ y \in \mathbb{Z}^p, \end{aligned} \tag{1.2}$$

where $c \in \mathbb{R}^n$, $d \in \mathbb{R}^p$, A is an $m \times n$ matrix, G is an $m \times p$ matrix, and $b \in \mathbb{R}^m$. We further assume that all entries in these vectors and matrices are rational.

Assuming the feasible region is bounded in the y variables, an obvious solution method for (1.2) is to enumerate the space in the y variables. For fixed y, (1.2) is a linear program, so it is known to be solvable in polynomial time [80, 76]. (See Bertsimas and Tsitsiklis [10] for a thorough introduction to linear programming.) Notice, however, that even if y is restricted to be a vector in $\{0, 1\}^p$ (that is, y is restricted to be a binary vector), this involves solving 2^p many linear programs.

A slightly more sophisticated enumeration technique, known as *branch-and-bound*, relies on the some simple observations. Consider the problem (1.2), where we have relaxed the integer y variables to be continuous:

$$\begin{aligned} \min\ & c^T x + h^T y \\ \text{subject to } & Ax + Gy \leq b \\ & x \in \mathbb{R}^n,\ y \in \mathbb{R}^p. \end{aligned} \tag{1.3}$$

Problem (1.3) is said to be the *linear programming relaxation* (*LP relaxation*) for problem (1.2). Let $\hat{z}^R$ be the optimal objective value to the relaxed problem (1.3) and z^* be the optimal objective

value to the original problem (1.2). Since (1.3) is a relaxation of (1.2), we know $\hat{z}^R \leq z^*$. Further, any integer feasible solution to (1.2) with objective value z^I has the property $z^* \leq z^I$.

A few things become apparent from this paradigm. First, we may get very lucky and the optimal solution we found for (1.3), $(\hat{x}^R, \hat{y}^R)$, may have the property that $\hat{y}^R \in \mathbb{Z}^p$. In this case we know $(\hat{x}^R, \hat{y}^R)$ is the optimal solution for (1.2). If conversely the solution $(\hat{x}^R, \hat{y}^R)$ has some component of y, $\hat{y}_i^R$, that is not integer, we can build a "divide and conquer" strategy for solving (1.2). We create two subproblems (called *branching*), one with $y_i \geq \lceil \hat{y}_i^R \rceil$ added to the constraints and the other with $y_i \leq \lfloor \hat{y}_i^R \rfloor$ added to the constraints. (Here $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$ are the ceiling and floor functions, respectively.) This process is then iteratively repeated, the which builds the so-called branch-and-bound tree. Whenever a subproblem has an integer solution for its relaxation, we update the incumbent value z^I (if it's improving), and by the earlier bounding logic we need not branch anymore at this subproblem. Similarly, if for some subproblem we have $\hat{z}^R > z^I$, no further branching on this subproblem can result in a better solution. The hope in using the branch-and-bound procedure is that complete enumeration in the y variables will not be required to find and prove an optimal solution. This however is not guaranteed, and perhaps more importantly, an exponential number of subproblems (in p) may have to be considered.

While today, with cheap parallel computing power that can be purchased on-demand, the branch-and-bound approach at least seems feasible, it was surely not sixty years ago, when transistor computers were just starting to supplant vacuum tube computers. It was in this environment that Ralph Gomory developed his cutting plane method for integer linear programs [63, 64], which today are known as Gomory cuts. Gomory demonstrated that given an optimal solution $(\hat{x}^R, \hat{y}^R)$ to (1.3), if said optimal solution had some fractional y components, one can generate a *cut*, which is a new constraint $\alpha^T x + \gamma^T y \leq \beta$ which "cuts off" the current fractional solution $(\hat{x}^R, \hat{y}^R)$, while not cutting off any integer solutions. That is, $\alpha^T \hat{x}^R + \gamma^T \hat{y}^R > \beta$, and the sets $\{x \in \mathbb{R}^n,\ y \in \mathbb{Z}^p \mid Ax + Gy \leq b\}$ and $\{x \in \mathbb{R}^n,\ y \in \mathbb{Z}^p \mid Ax + Gy \leq b,\ \alpha^T x + \gamma^T y \leq \beta\}$ are equal. The problem (1.3) can then be re-solved with the addition of this new constraint. (Note that both in this and the branch-and-bound context, adding a new constraint and re-solving a linear program is usually much easier than the original problem. This is because the dual simplex method can be used with the previous basis to find a new solution.) This process can be iteratively repeated until the solution is integer in y.

Gomory showed that his cutting plane approach terminated in finite time, and certainly did not have the potential issue of subproblem explosion that branch-and-bound poses. Much of the subsequent research in integer programming has been focused on efficient and new cut generation.

That being said, Gomory's cutting plane algorithm is not a polynomial-time algorithm for general integer linear programs (an exponential number of cuts may have to be generated); further, after several iterations, numerical stability issues often arise when implemented using finite precision arithmetic [22, 24].

However, the branch-and-bound paradigm and the cutting-plane approach are not mutually exclusive. Both commercial (CPLEX [69], Gurobi [66], Xpress [37]) and non-commercial (SCIP [93], CBC [43]) MILP solvers implement an algorithmic framework know as *branch-and-cut*. This combines the branch-and-bound and cutting-plane approaches by generating cuts as branch-and-bound progresses, with the hope of tightening the LP relaxations for the subproblems. Modern MILP solvers often expend significant computational effort at the initial LP relaxation generating cuts before beginning to branch.

Thus, advances in integer programming the past sixty years have generally fallen into one of the following categories: (i) valid cut generation, so as to tighten the LP relaxation bound, (ii) heuristics, so as to find good incumbent solutions early, (iii) improved branching strategies, with the hope of limiting the number of subproblem solves, (iv) reformulations, either to strengthen the LP relaxation or eliminate redundancies in the search space. Note that these can be application specific, as the techniques developed in this dissertation are, or general. A complete overview of the history on integer linear programming is impossible to give here, so the interested reader is again referred to the excellent texts [72, 21].

Often the goal when attempting to reformulate a problem is to find a reformulation such that $\hat{z}^R = z^*$ for arbitrary objective functions. Put another way, to find new variables and constraints such that $\text{conv}\{(x, y) \in \mathbb{R}^n \times \mathbb{Z}^p \mid Ax + Gy \leq b\} = \{(x, y) \in \mathbb{R}^n \times \mathbb{R}^p \mid Ax + Gy \leq b\}$, that is, the extremal points of the polytope are integer. If the all the extreme points of the polytope are integer in the y variables, we describe it as an *integer polytope*. A reformulation with extra variables and/or constraints is usually called an *extended formulation*. Such a reformulation is always possible, for example, all of the integer points my be enumerated and mapped to a simplex of appropriate dimension. Hence the goal with reformulations is to find a *compact* reformulation, i.e., one that does not add too many additional variables and constraints. More precisely, a compact extended formulation is one that is polynomial in the size of the problem's input.

However, if the problem is known to be NP-hard, then finding a compact extended formulation is equivalent to proving P = NP. (In fact, this technique has often been used for flawed proofs of P = NP [137].) A common technique therefore is to find some substructure whose formulation can

be tightened without spending one's time on a Millennium Prize Problem. Such is the approach applied to the unit commitment problem (UC) in Chapters 3 and 4. (We will outline the unit commitment problem in Section 1.4.) Chapter 5 also proposes a reformulation approach, however its technique exploits symmetries that exist in the problem structure, a topic we now turn to.

1.1.1 Symmetry in Integer Programming

The presence of symmetry in integer programs has long been an issue for practitioners, though much progress has been made in recent years. Symmetry occurs when two or more sets variables in the problem (1.2) are equivalent – that is, they can be interchanged without changing the problem. Such situations arise naturally in problems of practical interest. For example, in a scheduling problem, there may be identical machines or identical jobs, which will create symmetry in ILP formulations of the problem. Such symmetries confound the branch-and-bound process by forcing the consideration of identical solutions, as they cannot be pruned by bound nor with cuts that tighten the convex hull. We give a high-level review of this topic here, for a more complete overview the interested reader is referred to the excellent monographs [118, 97].

One approach to symmetry handling in ILP is reformulating or modifying the formulation of the problem in such a way as to eliminate or at least mitigate the symmetry. Sometimes, alternative formulations of a problem may avoid symmetry entirely [59, 31, 32, 153, 154, 155, 105]. Other times, it may be possible to add inequalities to the problem formulation that break the symmetry. In the simplest case, it may be possible to impose hierarchical constraints on the equivalent integer variables. That is, if y_1 and y_2 are equivalent, the constraint $y_1 \geq y_2$ cuts off all solutions where $y_2 > y_1$. So long as interchanging y_1 and y_2 has no effect on the feasibility or objective value, such a cut is valid – in the sense that while it may cut off *some* optimal solutions, it leaves some equivalent optimal solution in place.

Usually the symmetries in a problem are not so simple. Interchanging the value of two variables may necessitate interchanging the value of other variables. Such relationships may be more difficult to describe using linear inequalities. For specific problems, it may be possible to define relatively simple inequalities to break symmetry [132, 106, 142, 107, 30, 87]. All known methods of general symmetry breaking using linear inequalities, however, require inequalities that are exponential in quantity to completely remove the problem symmetry. Further, the simplest of these, the *lexicographic constraints*, require coefficients that are exponential in scope when expressed as linear constraints. Kaibel and Pfetsch [75] investigate the polyhedral structure for some

classes of symmetry-breaking inequalities, the associated polytopes of which are called *orbitopes*. Further, they provide an exponential class of symmetry-breaking inequalities, with a polynomial-time separating algorithm. Kaibel et al. [74] extends this by considering these inequalities implicitly in the branch-and-bound tree, as opposed to in the problem formulation, to fix additional variables during search.

Another approach for symmetry handling is borrowed from convex optimization. If we have K identical variables $w_1, \ldots, w_K$, we create a new aggregate variable $W = w_1 + \ldots + w_K$, replace each w_i with W/K, and eliminate some now redundant constraints. This procedure is known as *orbital shrinking*. For a convex problem, this reformulation is exact, and optimal solution for the original problem can be constructed by considering $w_i^* = W^*/K$, for optimal W^* in the aggregated model. For models with discrete variables, such a reformulation is not usually exact, but is a relaxation. Fischetti and Liberti [41] introduce orbital shrinking for discrete problems and investigate the strength of the derived bounds. Fischetti et al. [42] develop a framework that allows one to use orbital shrinking to solve a discrete problem exactly using a benders-type approach verify the feasibility of solutions proposed by the shrunk model. However, while their approach works well on pure-integer problems, it is not clear it can be used as a practical method for mixed integer problems. Chapter 5 shows that under certain conditions, orbital shrinking can be done exactly for MILP formulations of the unit commitment problem. As we will see, having strong formulations is a necessary condition to do exact orbital shrinking in the mixed-integer case.

For the remainder of this section we will assume the reader has some basic knowledge of group theory and graph theory. Herstein [68] and Dummit and Foote [33] are excellent references for group theory; West [156] for graph theory. Godsil and Royle [60] provides a concise treatment of both topics.

As their names suggest, both orbitopes and orbital shrinking rely on the idea of an *orbit* from group theory. For the purposes of exposition it will be useful to constrain ourselves to the pure integer setting. Consider a pure integer program of the form

$$\begin{aligned} &\min c^T x \\ &\text{subject to } Ax \leq b \\ &x \in \mathbb{Z}^n, \end{aligned} \tag{1.4}$$

where $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, and A is an $n \times m$ matrix. We consider the so-called *ground set* $I_n = \{1, \ldots, n\}$, that is, the indices of the vector x. A *permutation* π is a bijective function $\pi : I_n \to I_n$; the set of all such permutations form the *symmetric group* S_n under function composition. That is, for two elements $\pi_1, \pi_2 \in S_n$, $\pi_1 \circ \pi_2 \in S_n$ is defined by $(\pi_1 \circ \pi_2)(i) = \pi_1(\pi_2(i))$. A permutation $\pi \in S_n$ *acts* on a vector $a \in \mathbb{R}^n$ by permuting its components by index. As an example, let $n = 3$, and let $\pi(1) = 2$, $\pi(2) = 1$, and $\pi(3) = 3$. Then the action of π on the vector $a = [a_1, a_2, a_3]^T$, is denoted $\pi(a)$, and

$$\pi\left(\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}\right) = \begin{bmatrix} a_2 \\ a_1 \\ a_3 \end{bmatrix}. \tag{1.5}$$

In a similar fashion, for $\sigma \in S_m$ and $\pi \in S_n$, define $A(\pi, \sigma)$ to be the matrix formed by permuting the rows of A by σ and the columns of A by π. The *formulation group* $\mathcal{G}_{LP}$ for the problem (1.4) is

$$\mathcal{G}_{LP} = \{\pi \in S_n \mid \pi(c) = c,\ \exists \sigma \in S_m \text{ such that } \sigma(b) = b \text{ and } A(\pi, \sigma) = A\}. \tag{1.6}$$

The group $\mathcal{G}_{LP}$ does not contain all the symmetries of the problem, but it does have the distinct advantage of being (relatively) easily computable. (In general, $\mathcal{G}_{LP}$ is a subgroup of the problem symmetries; i.e., there may be redundant variables or constraints in the formulation that break the formulation symmetry.) The generators and orbits of the group $\mathcal{G}_{LP}$ can be calculated via a bijective transformation to a graph with colored edges and vertices [139]. Programs such as `nauty` [103] or `saucy` [29] can then be used to calculate the generators and orbits for the associated graph. In this way formulation symmetry can be detected automatically based upon the input, as opposed to in an ad-hoc manner than relies on the modeler.

The two main methods for exploiting general formulation symmetries are *isomorphism pruning* [95, 96] and *orbital branching* [123]. Isomorphism pruning relies on the formulation symmetry to derive isomorphism classes of subproblems, and then ensures that only one subproblem of each class is explored. The numerically unstable lexicographic constraints mentioned before are enforced implicitly in the tree; subproblems which violate a lexicographic inequality are pruned. In Margot's original formulation, this requires rigid branching rules to ensure validity. In the context of constraint optimization, this restriction is not a problem, but modern ILP solvers have sophisticated strategies for selecting a branching variable, often with the hope of changing the bound so as to cut off more of the search space. Flexible isomorphism pruning [121] overcomes this

issue by imposing a lexicographic ordering based on the previous branching decisions, allowing for the solver to chose the branching variable based on other criteria.

As its name suggests, orbital branching exploits the group theory concept of an orbit. In general, for a set $S \subseteq I_n$, the *orbit* of S under the group $\mathcal{G}$ is

$$\text{orb}(S, \mathcal{G}) = \{S' \subseteq I_n \mid \exists \pi \in \mathcal{G} \text{ such that } \pi(S) = S'\}. \tag{1.7}$$

In particular, if $j \in \text{orb}(\{i\}, \mathcal{G}_{LP})$, then this implies $i \in \text{orb}(\{j\}, \mathcal{G}_{LP})$, and we describe the associated variables x_i and x_j as *co-orbital*. That is, two objects (in general) are said to be co-orbital is there is a mapping from one to the other under the group in question. In this sense the orbits of the group $\mathcal{G}_{LP}$ encode the variables of (1.4) that are equivalent.

Orbital branching then relies on a relatively simple observation. Suppose for a moment all the variables in (1.4) are binary. Then for some set of indices $O \subseteq I_n$,

$$\sum_{i \in O} x_i \geq 1 \qquad \text{or} \qquad \sum_{i \in O} x_i \leq 0 \tag{1.8}$$

is a valid branch. If O happens to be an orbit of $\mathcal{G}_{LP}$, then all the variables x_i for $i \in O$ are equivalent. A representative index i^* can be chosen arbitrarily, and this branch can be strengthened to

$$x_{i^*} = 1 \qquad \text{or} \qquad \sum_{i \in O} x_i = 0. \tag{1.9}$$

A basic implementation of orbital branching just relies on the root node symmetries $\mathcal{G}_{LP}$, but Ostrowski et al. [123] also explore updating the formulation group as branching decisions are made and variables are fixed. Constraint orbital branching [122] extends the idea of branch strengthening to arbitrary branching constraints, again exploiting the orbits to select only one representative for the left branch.

Chapter 2 explores on the idea of *almost symmetry* in graphs. This is a natural place to begin the exploring the idea of almost symmetry in integer programs – that is, mappings π and σ that *almost* preserve optimality and feasibility. It is likely that the presence of such near symmetries has a similar affect on the branch-and-bound process as exact symmetries, but they are undetectable, in general, by current software. Chapter 2 takes a small step in this direction by considering almost symmetries on simple, undirected graphs. In particular, we seek to modify the given graph no more than a fixed amount, such that the number of orbits is minimized. In certain contexts, such

modifications amount to creating a relaxed version of (1.4) with fewer orbits. The orbital branches for this relaxed problem then are stronger than the unmodified problem. With this in mind we now turn to the question of finding symmetries on graphs, to properly place in context finding almost symmetries on graphs.

1.2 Graph Automorphism

The problem of finding the symmetries of a graph is the *graph automorphism* problem. First, we need to dispense with a few definitions. A *graph* G is an ordered pair comprising of a set of *vertices* V and a set of *edges* E, which are unordered two-element subsets of V. At times we will refer to this as the graph $G(V, E)$. Elements of E will be denoted $\{u, v\}$, for $u, v \in V$, and we say that u and v are *adjacent*. The *degree* of a vertex $v \in V$ is the number of edges incident to v, and we write $d(v)$. All the graphs considered in this thesis are *simple*, that is, they have no loops or multiple edges.

Two graphs $G(V, E)$, $G'(V', E')$ are *isomorphic* if there is a bijection $\pi : V \to V'$ such that $\{u, v\} \in E$ if and only if $\{\pi(u), \pi(v)\} \in E'$. We write $G \cong G'$. Put another way, two graphs are isomorphic if there is a bijection between their vertices that preserves adjacency.

An isomorphism from a graph onto itself is called an *automorphism*. For a graph G, we denote the set of all automorphism as $\mathrm{Aut}(G)$. The set $\mathrm{Aut}(G)$ is a group under function composition, and when $V = \{1, \ldots, n\}$, we have that $\mathrm{Aut}(G)$ is a subgroup of S_n. The set $\mathrm{Aut}(G)$ is always non-empty, as the identity map which just sends vertices to themselves is a valid automorphism.

A graph $G(V, E)$ is said to be *colored* if it has associated functions $c_V : V \to X$, and $c_E : E \to Y$, where X and Y are some set. For each $v \in V$, we say v is colored $c_V(v)$, and similarly for each $e \in E$, we say e is colored with $c_E(e)$. For a graph $G(V, E)$ with associated coloring c_V and c_E, $\mathrm{Aut}(G(V, E), c_V, c_E)$ is a subgroup of $\mathrm{Aut}(G(V, E))$ containing the automorphisms that map vertices to vertices of like color and edges to edges of like color.

Additionally, for this section we will assume the reader is familiar with some foundational ideas from complexity theory. West [156, Appendix B] provides a high-level overview.

1.2.1 Computational Complexity

The *graph isomorphism problem* (GI), that is, determining whether an isomorphism exists between to given graphs, is obviously in NP. However, GI is also not know to be in P, and neither it is

know to be NP-complete. The NP-completeness of GI is considered unlikely. Recently, Babai [3] presented a quasi-polynomial time algorithm for GI. The following result of Mathon [100] shows checking and counting are polynomial-time reducible to each other.

Theorem 1.1. *The following problems are polynomial-time equivalent:*

- *isomorphism recognition for two graphs G_1 and G_2 (GI),*
- *isomorphism map from G_1 onto G_2, if it exists,*
- *number of isomorphisms from G_1 onto G_2,*
- *order of the automorphism group of G,*
- *generators of the automorphism group of G,*
- *automorphism partition of G.*

Further, Goldreich et al. [61] show that GI is not NP-complete under a widely-believed computational complexity assumption (namely, that the *polynomial-time hierarchy* is infinite, see [138] for an overview). Since GI is not known to be in P nor known to be NP-complete, the problems listed in Mathon's theorem above are said to be *GI-complete*. Further we can say a problem is *GI-hard* if there is a polynomial-time reduction from GI to that problem. An example of a related problem that is GI-hard but not known to be GI-complete is *graph canonization*, which is the problem of finding a *canonical form* for a given graph G. Formally, let $\mathcal{G}$ denote a class of graphs closed under isomorphisms. The function $CF : \mathcal{G} \to \mathcal{G}$ is a *canonical form* for $\mathcal{G}$ if:

- For $G \in \mathcal{G}$, $CF(G) \cong G$;
- For $G, G' \in \mathcal{G}$, $G \cong G'$ if and only if $CF(G) = CF(G')$.

Notice the usefulness of this definition. For a database of graphs stored in a canonical form, testing to see if a new graph $\hat{G}$ is already in the database is as "easy" as computing its canonical form and testing for equality, as opposed to testing for isomorphism between $\hat{G}$ and each graph in the database. Similarly a canonical form can be used to remove isomorphs from a collection of graphs.

No polynomial-time algorithm is know for GI, and the fastest running time algorithm provided by Babai et al. [4] has stood for the last three decades at $e^{\mathcal{O}(\sqrt{n \log n})}$. As mentioned before, Babai [3] claims to have a quasi-polynomial time algorithm for GI, but at the time of writing this result

has not been peer-reviewed. It is known that finding a canonical labeling of a uniformly chosen n-vertex random graph can be done in $\mathcal{O}(n^2)$ time [5]. This helps to illuminate the fact that many algorithms for computing graph canonization, isomorphism, and/or automorphism seem to behave very well in practice; see `nauty` and `Traces` [103], `saucy` [77], `Bliss` [73], and `conauto` [89], among others.

1.2.2 GI – A brief algorithmic history

The proceeding is not meant to be a complete overview of proposed algorithms to solve GI. Indeed for many decades it has been a favorite area of research for algorithm designers, to the extent that even as early as 1976 it was described as a "disease" [135]. To date there have been at least a few hundred published algorithms [103]. However, this does lay the groundwork for the successful algorithms that would follow the "individualization-refinement" paradigm, namely `nauty` and its compatriots mentioned above.

The graph isomorphism problem became of practical interest to chemists in the 1960s as a way of comparing two chemical structures [152, 112, 148, 91]. To test two n-vertex graphs G and G' for isomorphism, using a "dumb" enumerating algorithm would require checking $|S_n| = n!$ possibilities. Sussenguth [148] states a general principle for reducing the number of possibilities, which underlies the research at the time:

> If graph G is isomorphic to graph G', the subset of nodes of G which exhibit some property must correspond to that subset of nodes of G' which exhibit the same property.

This principle provides a powerful method for eliminating potential isomorphisms. The "properties" Sussenguth refers to became known in the literature as *vertex invariants*. A *vertex invariant* is any property of a vertex that does not change under automorphisms of G. A simple vertex invariant is degree, but just about any vertex property will do. For example, Sussenguth uses the shortest cycle containing the vertex.

However, this method for partitioning the vertices is often not enough to sufficiently narrow the search space. To address this Parris and Read [126] introduced and Corneil and Gotlieb [23] further developed a procedure for *refining* the partitions. Of particular importance is that any refinement of a partition is also invariant under automorphisms. Notice that a partition of vertices can be defined using a coloring of the vertices, whereby each partition is assigned a different color. Corneil and Gotlieb [23] give an algorithm for what would become known in the literature as *color*

refinement. Put simply, the logic of color refinement is this: if two vertices i and j currently have the same color, assign them different colors if for some color k, i and j have a different number of neighbors of color k. We stop when no further refinement is possible, that is, when every two vertices of the same color have the same number of neighbors of each color. Such a coloring is called *stable* or *equitable*. Notice that if all vertices start with the same color (i.e., are just partitioned trivially), then the color refinement algorithm will at least partition the vertices by degree. They further prove that such a refinement scheme is invariant under automorphism, provided the initial partition (or coloring) is invariant under automorphism.

Corneil and Gotlieb [23] then provide necessary and sufficient conditions for isomorphism using this refinement technique. In particular, given a coloring which is equitable, choose a color k with more than one vertex and "individualize" a vertex i by giving it a new color and "refining" until the coloring is equitable again. This process can be continued until the coloring is *discrete*, that is, each vertex has its own color. This is the heart of the individualization-refinement paradigm that would be most successful at providing efficient solutions to GI.

Corneil and Gotlieb [23] conjectured that the algorithm they provide for graph isomorphism produces a canonical form. Unfortunately this turned out to be incorrect. Their algorithm individualizes each vertex in a given color class k, but only uses this information to further partition that color class, and conjecture that the resulting partition form the orbits of $\mathrm{Aut}(G)$. In general this is not true, see [99]. All successful algorithms that followed employed some sort of branching at the individualization stage.

Arlazarov et al. [1] provide a correct algorithm for finding a canonical labeling of a graph; further, their algorithm performs the strongest on strongly regular graphs. They do not consider vertex-invariants in their study; however their algorithm could be extended to use invariants. Their method to find a canonical labeling uses the adjacency matrix A and constructs a new adjacency matrix $\tilde{A}$ which is lexicographically maximal relative to permutations of the matrix, that is

$$\widetilde{A} = \operatorname*{lex\,max}_{\pi \in S_n} \pi A \pi^{-1}.$$

It is clear that $\widetilde{G}$, the graph whose adjacency matrix is $\widetilde{A}$, is a canonical form for G. The authors prove the maximal form (as defined above) for an adjacency matrix is monotonic, and use this fact to recursively construct $\widetilde{A}$ from the top left corner, proceeding down the diagonal using a tree. They note that in the case when G is complete so that $S_n = \mathrm{Aut}(G)$ (here $|G| = n$), the tree

constructed needs $n!$ leaves to compute $\widetilde{A}$. To combat this the authors define a "heuristic" which computes elements of $\mathrm{Aut}(G)$ as the tree is explored, so that symmetric regions of the search space can be pruned. They show that in the case when G is complete this heuristic allows the reduction of the search tree to n leaves, though they note that in the case when $|\mathrm{Aut}(G)| = 1$ time is wasted in futilely attempting to find elements of $\mathrm{Aut}(G)$. Finally it is shown in this process computes the orbits of $\mathrm{Aut}(G)$ as a side effect. Having defined a valid algorithm for graph canonization, it could be used to test two graphs for isomorphism, in addition to finding the orbital partitions of both graphs. Finally, it is worth noting that this approach (finding a lexicographically maximal incidence matrix) for the graph canonization problem is NP-hard [25].

In the next section we will see how `nauty` was able to leverage the best ideas in the two algorithms outlined above.

1.2.3 `nauty` – No AUTomorphisms, Yes?

Despite the algorithmic flurry of the preceding decade and a half, `nauty` [101], was the first algorithm that could successfully handle both structurally regular graphs and graphs with large automorphism groups. When introduced `nauty` could successfully handle some graphs on thousands or more vertices on the hardware of the time. The algorithm finds a canonical labeling of a vertex-colored graph and finds the generators of its automorphism group.

The refinement procedure used by `nauty` is nearly identical to the one presented by Corneil and Gotlieb [23]. The main difference is that McKay [101] provides a proof that the refinement procedure used by `nauty` produces a *canonical coloring*; that is, two isomorphic graphs $G(V, E)$ and $G'(V', E')$ are colored so that if ϕ is an isomorphism from G to G', for every $v \in V$, $\phi(v)$ has the same color as v. Very broadly, this is achieved by ensuring that the creation of new colors and selecting the current refinement color are done in a way that is *label invariant* or *isomorphism invariant*, namely by using degree information.

Having a label invariant refinement procedure in hand allows us to write down an algorithm for canonical labeling. For each vertex not in its own color class, individualize that vertex by giving it a new color. Refine until an equitable coloring is achieved and repeat until all vertices are in their own color class (that is, the coloring is *discrete*). The leaves of this search tree will then be canonically labeled versions of the graph, and by returning the lexicographically smallest, we have a canonical labeling procedure! Moreover, for many graphs for which $|\mathrm{Aut}(G)|$ is small, the search tree will be of a manageable size.

To address the cases when $|\mathrm{Aut}(G)|$ may be larger, like the algorithm of Arlazarov et al. [1], `nauty` exploits automorphisms of the graph found during the search to prune. The tree is constructed depth-first so that as automorphisms are discovered at the leaves other parts of the search space may be pruned by automorphism.

It should be noted that for some graphs, namely those that have a high degree of regularity, the refinement procedure defined above is not very powerful. To that end `nauty` provides no less than a dozen vertex-invariants to provide an initial partition. The choice of which vertex-invariant to use, if any, is left up to the user. For more details the reader is referred to [102].

Finally, it has been shown that there exists class of graphs for which `nauty` has exponential running time [108]. However, in most circumstances `nauty` performs exceptionally well. Because of this, after its introduction `nauty` would be the program of choice for computing canonical forms and automorphisms for the the next several decades. For a deeper overview of `nauty` the reader is referred to [67, 103].

1.2.4 `saucy` – A worthy competitor emerges

In 2004, motivated by the study of symmetry in *boolean satisfiability*, `saucy` was introduced by Darga et al. [28]. Unlike `nauty`, `saucy` simply finds the generators of the automorphism group for the input graph; it does not compute canonical forms. The first enhancement of `saucy` was the use of sparse data structures. The performance improvement from this alone prompted McKay to release a version of `nauty` for sparse graphs. The major enhancement, however, in the initial release of `saucy` is in the refinement procedure. Since the authors of `saucy` were interested in sparse graphs, they take advantage of this fact to avoid unnecessary work during refinement. Specifically, for each vertex v they keep track of the colors it is connected to. Thus when attempting to refine a color, they need only attempt to use the other colors its vertices are connected to. If the graph is sufficiently sparse this can have a dramatic effect on run time – in their experiments the refining operation was over 80% of the execution time for symmetry detecting engines. As expected this tweak to the refinement procedure does produce impressive speedups on sparse graphs; however, on dense graphs the resulting overhead is prohibitive.

Another enhancement to `saucy` was introduced in [29] which delinked it algorithmically from `nauty` and other canonical labeling tools. The modification to `saucy` in Darga et al. [29] can sometimes detect symmetries at non-leaf nodes by assuming that most symmetries are sparse themselves, that is, they move very few vertices. This change and subsequent improvements made

`saucy`'s search for symmetries resemble that of the search of a SAT solver for satisfying assignments. The details of this are beyond the scope of this thesis; the interested reader is referred to [78].

1.2.5 `Bliss`, `Traces`, and `conauto`

The introduction of `saucy` set off renewed interest in research in developing efficient software for graph automorphism. Soon after `saucy`'s appearance `Bliss` was introduced in Junttila and Kaski [73]. `Bliss` is algorithmically similar to `nauty`, with some enhancements in data structures and a heuristic which incrementally computes leaf certificates. In some cases this allows pruning of the tree before the refinement procedure is complete. `Traces`, which also computes canonical labels and automorphisms, introduced a new search paradigm by using a modified breadth-first strategy. Since automorphisms can only be detected at leaf nodes, `Traces` forms and "experimental path" to a leaf so automorphisms can still be used to prune the search space. The interested reader is referred to [127, 103]. The initial release of `conauto` only tested two graphs for isomorphism, and used automorphisms detected along the way to prune the search space [90]. It did not compute the full automorphism group of a graph nor the canonical form; however a subsequent release added the ability to compute the full automorphism group [89].

1.3 Almost Symmetries in Graphs

Chapter 2 attempts to use the essential idea of vertex invariants and refinement to narrow the search space for almost symmetries. To be specific, *k-almost symmetries* are "almost-automorphisms" which are permitted to map up to k edges to non-edges. One simple vertex invariant for k-almost symmetries is the degree difference must be no more than k. Another is that automorphisms must map neighbors to neighbors, so a matching problem is formulated and solved to determine if this is possible with no more than k edge deletions. By doing this process iteratively we can eliminate certain k-almost symmetries.

Chapter 2 then poses the following problem: given a graph G and budget k, compute a subgraph of G by removing no more than k edges with the most symmetry. Specifically, we seek to minimize the size of the orbital partition of G's vertices after modification. In certain contexts this corresponds to a relaxation of the integer program (1.4), and techniques such as orbital branching or isomorphism pruning could be used to quickly solve the resulting relaxed problem and provide a valid lower bound.

However, as we will see, the problem of finding such a subgraph is very computationally challenging, though we were able to compute almost symmetries for graphs with a few hundred vertices and a low budget k. This does not bode well as a procedure for quickly finding lower bounds for difficult combinatorial problems, but perhaps some clever heuristics could be developed in the future. Promisingly, almost all of the graphs examined exhibited some non-trivial almost symmetry, so it is certainly a phenomenon that occurs in real-world problems.

1.4 The Unit Commitment Problem

Chapters 3, 4, and 5 are concerned with the *unit commitment problem* (UC). The unit commitment problem is that of scheduling (i.e., committing) power generating units so as to meet energy demand while minimizing operational costs. Unit commitment is a problem faced by electrical systems operators daily, with the potential for huge economic impacts. O'Neill [117] estimates that a 1% savings in electrical energy production is worth $10 billion annually. Further, the necessity of bringing increasing levels of renewable energy online in the coming decades will create new challenges for system operators. By their nature, most renewable energy sources are non-dispatchable and not fully predicable, which necessitates using more sophisticated optimization methods. We introduce the unit commitment problem here, but postpone a full literature review for Chapter 3.

Put more formally, for a set of generators $\mathcal{G}$ and a discretized time horizon $\mathcal{T} = \{1, \ldots, T\}$, the unit commitment problem is

$$\min \sum_{g \in \mathcal{G}} c^g(p^g) \tag{1.10}$$

$$\text{subject to } \sum_{g \in \mathcal{G}} p_t^g = D_t \qquad \forall t \in \mathcal{T} \tag{1.11}$$

$$p^g \in \Pi^g \qquad \forall g \in \mathcal{G}, \tag{1.12}$$

where $c^g(\cdot)$ is the production cost function for a power vector p^g for generator g, D_t is the power demand at time t, and Π^g is the set of feasible dispatch vectors for generator g. Unit commitment is often formulated and solved as a MILP. Generators usually have technical constraints and non-convex startup costs which necessitate introducing integer variables to describe Π^g and c^g. In general, unit commitment is an NP-hard problem [151].

Beside the system demand constraints (1.11), the unit commitment problem is totally decomposable by generator. One avenue of research in MILP formulations of UC has been in reformulating Π^g and c^g to tighten their linear relaxations. Chapters 3 and 4 contribute to this line of research. Chapter 3 provides a large, but polynomial, extended formulation for Π^g and c^g that is integral for any reasonable generator parameters. Because the formulation is so large, Chapter 3 exploits the decomposability of UC by considering a cut-generating linear program for each generator g to tighten-up the description of Π^g in the master MILP. Chapter 4 introduces a tighter formulation for time-dependent start-up costs; for reasonable objective functions the formulation introduced is experimentally as tight as the larger ideal formulation.

The structure of UC also makes exact and almost symmetry detection straightforward. Two generators can create formulation (almost) symmetry for UC whenever they have (almost) identical parameters. If generators g_1 and g_2 are identical, one way to exploit this symmetry is using orbital shrinking. That is, instead of representing the output of identical g_1 and g_2 using the sets Π^{g_1} and Π^{g_2}, we represent their combined output using one set Π^{g_1,g_2}, using half the variables and constraints. We can treat the cost functions similarly. Chapter 5 explores this idea.

One interesting consequence is that the strong formulations developed in Chapters 3 and 4 are required to perform this reduction in an exact way. Note that Π^g, in general, is a mixed-integer set. Consider then the feasible region from problem (1.2)

$$S = \{x \in \mathbb{R}^n,\ y \in \mathbb{Z}^p \mid Ax + Gy \leq b\}, \tag{1.13}$$

and its continuous relaxation

$$P = \{x \in \mathbb{R}^n,\ y \in \mathbb{R}^p \mid Ax + Gy \leq b\}. \tag{1.14}$$

We say that the mixed-integral polytope P has the *mixed-integer decomposition property* (MIDP) if for any positive integer k and for any $(x, y) \in kP$ with $y \in \mathbb{Z}^p_+$ (i.e., $(x, y) \in kS$), there exists $(x_1, y_1), \ldots, (x_k, y_k) \in P$ with $y_1, \ldots, y_k \in \mathbb{Z}^p_+$ such that $(x, y) = (x_1, y_1) + \cdots + (x_k, y_k)$.

The MIDP is interesting the context of UC, because when the formulations for Π^g and c^g have the MIDP, we know identical generators can be represented using orbital shrinking. This makes the mixed-integer formulation for UC smaller, and as we will see in Chapter 5, also allows the representation of many feasible solutions in one shrunk solution. In turn, the resulting branch-and-bound tree may be smaller, as certain classes of mutually non-dominating solutions can be

considered in one subproblem. Hence, even with the sophisticated automatic symmetry-detection techniques described earlier, the reduced model can perform significantly better computationally. We have the following remark.

Remark 1.1. If the polytope P has the MIDP, then P is integer in y.

The proof of Remark 1.1 is exactly as the pure-integer case, which can be found in Schrijver [140, §22.10].

Remark 1.1 demonstrates the need strong formulations for generators in order express them exactly using aggregation, and more generally, we need strong formulations when hoping to use the orbital shrinking technique exactly. Even then it may not be sufficient to ensure a decomposition exists. In the case of UC though, Chapter 5 shows the formulation presented in Chapter 3 has the MIDP, and under certain conditions the new start-up cost formulation introduced in Chapter 4 does as well. This implies we can exploit the generator symmetry in UC using the MIDP of these formulations to construct (possibly many) exact optimal solutions to the original problem. Additionally, it should be possible to exploit generator almost symmetry in a similar fashion, using orbital shrinking and the decomposability of these new formulations to arrive at almost optimal and/or almost feasible solutions to large-scale UC instances.

Chapter 2

Detecting Almost Symmetries of Graphs

This chapter and Appendices A and B are based on a paper published by Ben Knueven, Jim Ostrowski, and Sebastian Pokutta:

> Knueven, B., Ostrowski, J., and Pokutta, S. (2017). Detecting Almost Symmetries of Graphs. *Mathematical Programming Computation*, to appear.

Authors Ostrowski and Pokutta posed the question. Authors Knueven and Ostrowski developed the algorithmic framework. Author Knueven developed the software and conducted all computational experiments. Author Knueven wrote the manuscript and created all the tables and figures. Authors Ostrowski and Pokutta edited the manuscript.

In this chapter we present a branch-and-bound framework to solve the following problem: Given a graph G and an integer k, find a subgraph of G formed by removing no more than k edges that minimizes the number of vertex orbits. We call the symmetries on such a subgraph "almost symmetries" of G. We implement our branch-and-bound framework in PEBBL to allow for parallel enumeration and demonstrate good scaling up to 16 cores. We show that the presented branching strategy is much better than a random branching strategy on the tested graphs. Finally, we consider the presented strategy as a heuristic for quickly finding almost symmetries of a graph G. The software that has been written as part of this chapter has been issued the Digital Object Identifier DOI: 10.5281/zenodo.840558.

2.1 Introduction

Two graphs are *isomorphic* if there is a bijection between their vertices that preserves adjacency; such a bijection is called an *isomorphism*. An isomorphism from a graph onto itself is called an *automorphism*, and the set of all automorphisms of a given graph G, denoted $\mathrm{Aut}(G)$, forms a group under composition.

The Graph-Isomorphism problem is that of determining the existence (or not) of an isomorphism between two input graphs. It is a notorious problem in complexity theory, as no polynomial time algorithm is known, and at the same time Graph-Isomorphism is generally not believed to be NP-complete [61]. Babai recently presented a quasi-polynomial time algorithm for Graph-Isomorphism [3]. It is well-known that Graph-Isomorphism and the problem of determining the orbits of $\mathrm{Aut}(G)$ (Automorphism-Partition) are polynomial time equivalent [135, 100].

Recent results have demonstrated the hardness of various robust or approximate versions of Graph-Isomorphism [88, 2, 116]. All these are either NP-hard [88, 2] or believed to be NP-hard [116]. However, there has been little study of the computational feasibility of such problems. In this chapter we propose and implement a branch-and-bound algorithm for solving a robust version of Automorphism-Partition.

2.1.1 Preliminaries

We need to first lay out some notation that we will use in this chapter. Given an undirected graph $G = (V, E)$ (later we always write $G(V, E)$), for some set of edges $F \subset E$, we define the graph $(G - F) := (V, E \setminus F)$, that is, $(G - F)$ is the graph G with the edges in F removed. We may also use the notion $V(G)$ to refer to the vertices of the graph G and $E(G)$ to refer to the edges of G.

Definition 2.1. For an n-vertex graph G, a permutation $\pi\colon V(G) \to V(G)$ is an automorphism of G if for every $\{u, v\} \in E(G)$, $\{\pi(u), \pi(v)\} \in E(G)$ and for every $\{u, v\} \notin E(G)$, $\{\pi(u), \pi(v)\} \notin E(G)$.

Definition 2.2. For a graph $G(V, E)$, the mapping $\sigma : V(G) \to V(G)$ is a *k-almost symmetry* of G if there exists a set of edges $E_D \subseteq E$ with $|E_D| \leq k$ such that σ is an automorphism for the graph $(G - E_D)$. We denote the set of k-almost symmetries of G as $\mathcal{AS}_k(G)$.

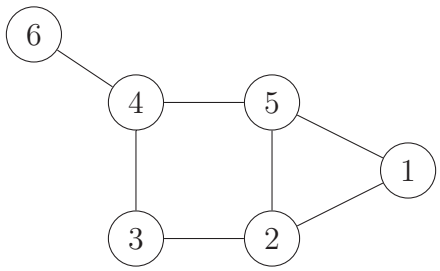


Figure 2.1: A graph with almost symmetry

The 0-almost symmetries are exactly the automorphisms of G. We have the following motivating result courtesy of [35]; an examination of the proof of Theorem 1 in [35] shows that only edge deletions are used.

Theorem 2.1. *For an n-vertex graph G and $k \geq \lfloor \frac{n-1}{2} \rfloor$, it holds $|\mathcal{AS}_k(G)| > 1$, that is, there exists a non-trivial k-almost symmetry.*

Example 2.1. Consider the graph shown in Figure 2.1 from [52]. The permutation (25)(34) is a 1-almost symmetry of the graph because it is an automorphism of $G - \{\{4, 6\}\}$. The permutations (16)(24) and (35) are also 1-almost symmetries, as seen by removing edge $\{1, 5\}$.

From this example we also obtain the following remark.

Remark 2.1. For $k \geq 1$, the set of k-almost symmetries is not necessarily a group.

To this end, consider the permutation $(35) \circ (25)(34) = (2345)$. Enumerating all 7 possibilities shows that (2345) is not a 1-almost symmetry of the graph. Since there is no group structure to be exploited during the search for almost symmetries, intuitively the problem of finding almost symmetries seems much harder than that of finding symmetries. Indeed, one of the main contributions of McKay's venerable `nauty` program [101] for graph isomorphism and automorphism is its ability to prune symmetric regions of the search space *as* symmetries on the graph are detected. However, without a group structure the underlying theory developed by McKay falls apart.

It will be helpful to tie our definition of k-almost symmetry to the notion of α-automorphism from [116]. First consider α-isomorphism between two graphs G and H:

Definition 2.3. For non-empty n-vertex graphs G and H, and $0 \leq \alpha \leq 1$, a permutation $\pi \colon V(G) \to V(H)$ is an α-isomorphism if

$$\frac{|\{\{u, v\} \in E(G) : \{\pi(u), \pi(v)\} \in E(H)\}|}{\max\{|E(G)|, |E(H)|\}} \geq \alpha. \tag{α-ISO}$$

In this definition, α is a lower bound on the ratio of edges that get mapped to edges. O'Donnell et al. [116] proved, assuming Feige's R3XOR hypothesis [38], that given two $(1-\epsilon)$-isomorphic graphs, finding a $(1-r(\epsilon))$-isomorphism is NP-hard (for some function $r(\epsilon) \to 0$ as $\epsilon \to 0^+$). Arvind et al. [2] have shown that finding a permutation π which maximizes α is NP-hard. An *α-automorphism* is an α-isomorphism from a graph to itself:

Definition 2.4. For a non-empty n-vertex graph G, and $0 \leq \alpha \leq 1$, a permutation $\pi\colon V(G) \to V(G)$ is an α-automorphism of G if

$$\frac{|\{\{u,v\} \in E(G) : \{\pi(u), \pi(v)\} \in E(G)\}|}{|E(G)|} \geq \alpha. \tag{α-AUT}$$

While it is beyond the scope of this chapter to address the complexity question of k-almost symmetries, we do note that the k-almost symmetries of G are $\left(1 - \frac{k}{|E(G)|}\right)$-automorphisms of G, since at most k edges get mapped to non-edges under a k-almost symmetry. Similarly any α-automorphism of G is a $\lfloor(1-\alpha)|E|\rfloor$-almost symmetry of G.

2.1.2 Contribution

We present an algorithm capable of finding the k-almost symmetries of a graph G. However, we will be more interested in the following related problem for a given graph $G(V,E)$ and budget k:

$$\gamma_k^G = \min\{|\mathrm{orb}_{\mathrm{Aut}(G')}(V(G'))| : G' = (G - E_D), E_D \subseteq E, |E_D| \leq k\}, \tag{2.1}$$

where $\mathrm{orb}_\Gamma(X)$ for some set X and group Γ is the orbital or automorphism partition of X under Γ's action on X, and $|\mathrm{orb}_\Gamma(X)|$ is the number of orbits. Notice in this framework $\mathrm{Aut}(G')$ will be a subset of the k-almost symmetries on G; further it will be a subset with a group structure. Put another way problem (2.1) is that of finding the subgraph G by removing at most k edges in such a way that the number of orbits is minimized. It is in this sense that we provide an algorithm for a generalized version of Automorphism-Partition. We present a branch-and-bound algorithm for solving (2.1) with a branching strategy that we show is much better than random, and is in fact by one measure often the best branching choice available. Additionally we show that with some modifications the branch-and-bound strategy presented can be used as an effective heuristic, and we demonstrate the robustness of our branching strategy.

2.1.3 Motivation

In integer programming (IP), it is well known that the presence of symmetry, if not properly addressed, can confound the branch-and-bound process. The authors hypothesize that a large amount of almost symmetry can have a similar effect by causing "almost symmetric" regions of the search space to be considered. While we will not address the question of almost symmetry in IP directly, it is worth noting that all the methods in the literature for dynamic symmetry breaking in IP rely on insights from symmetry detection on graphs [95, 96, 123]. We restrict ourselves to edge-deletions because they have a natural IP corollary. Suppose we have the following IP:

$$\max_{x \in \{0,1\}^n} \left\{ \mathbb{1}^T x \mid Ax \leq \mathbb{1} \right\}, \tag{2.2}$$

where $A \in \{0,1\}^{m \times n}$ and $\mathbb{1}$ is the appropriately-sized vector of 1's. Then there is a one-to-one correspondence between the symmetries of the formulation of (2.2) and the bipartite graph $G(A) = (N, M, E)$ where the partite set N represents the columns of A and the other partite set M represents the rows of A, and an edge is between a vertex $i \in N$ and $j \in M$ if and only if $a_{ij} = 1$ (see [123] for more details). An edge deletion in $G(A)$ therefore represents changing a 1 to a 0 in the constraint matrix A, leading to a relaxation of (2.2). Similarly an edge addition would result in a restriction of (2.2). Our method presented here also works for edge additions by simply considering the complement of G. Therefore, a natural starting point for the study of almost symmetry in IP is the detection of almost symmetries on graphs.

2.1.4 Literature Review

There have been several efforts for detecting near, fuzzy, or almost symmetries in graphs. Buchheim and Jünger present an integer programming approach in [15] which allows for the possibility of edge deletions and additions. They consider rotational and reflective symmetries separately and attempt to find a rotational or reflective symmetry that minimizes the number of edge modifications. Computational results are presented on graphs with no more than two dozen vertices. With advancements in MIP solvers since publication their method may be applicable to larger graphs today.

Markov [98] considers almost-symmetries on colored graphs, where some "chameleon" vertices are allowed to be mapped to vertices of any color. They extend the common graph automorphism

algorithm used by `nauty` [103] and `saucy` [29] to consider these chameleon vertices. However, no computational results are presented.

Fox, Long, and Porteous provide a heuristic method for finding near symmetries under edge contractions [44]. They modify color refinement to look for possible edge contractions as `nauty` individualizes vertices. Additionally, their heuristic looks to minimize the number of fixed vertices under some near symmetry group, with online heuristic detection for when a vertex may be axial under a reflective symmetry. The heuristic is implemented and shown effective for graphs with a hundred vertices and edges and no more than 5 flaws.

The remainder of this chapter is outlined as follows. In Section 2.2 we discuss the details of the branch-and-bound framework for detecting almost symmetries in graphs. Section 2.3 gives some implementation details. Accompanying computational results are in Section 2.4, along with some natural extensions. Finally we draw some conclusions in Section 2.5.

2.2 Algorithmic Overview

We will assume throughout that we are solving the problem of finding almost symmetries on an n-vertex graph $G(V, E)$ for a budget k of edge deletions. We maintain two sets throughout:

- A set E^D of deleted edges;
- A set E^F of fixed edges.

We construct a search tree $\mathcal{T}$, where each node $A \in \mathcal{T}$ is uniquely represented as a tuple (E_A^D, E_A^F). Disjunctions are created by taking an edge $e \in E(G) \setminus (E_A^F \cup E_A^D)$ and in one branch adding e to E_A^D, and in the other adding e to E_A^F. We reach a leaf whenever $|E_A^D| = k$ or $E_A^F = E(G) \setminus E_A^D$. Let $G_A = G - E_A^D$ for some node A. We see then, if we completely expand the tree $\mathcal{T}$ to its leaves, $\mathcal{AS}_k(G) = \bigcup_{A \in \mathcal{T}} \mathrm{Aut}(G_A)$. We say A' is a *child* of A if $E_A^D \subseteq E_{A'}^D$ and $E_A^F \subseteq E_{A'}^F$. While the tuple (E_A^D, E_A^F) completely describes the node A of the tree, for convenience we will describe a node as the 4-tuple (G_A, P_A, E_A^F, k_A), where $k_A = k - |E_A^D|$ (the residual budget), and P_A is an n-vertex graph. P_A encodes those pairwise permutations or mappings that have not been proved impossible in A's children. Specifically for an edge $\{i, j\} \in P_A$ and some child A' of A, a permutation mapping vertex i to vertex j in $V(G)$ *may* exist in $\mathrm{Aut}(G_{A'})$.

It is worth noting that the lack of group structure (Remark 2.1) necessitates storing possible k-almost symmetries as an n-vertex graph as opposed to a vertex partition. In particular, for

vertices u, v, w, the existence of k-almost symmetries σ, π such that $\sigma(u) = v$ and $\pi(v) = w$ does not guarantee the existence of a k-almost symmetry mapping u to w. Notice, however, that each k-almost symmetry is in *some* group, and hence has an inverse which will also be a k-almost symmetry, allowing us to use an undirected graph. At the root node R we initialize $R = (\emptyset, \emptyset)$ and P_R to be the complete graph with self-loops, representing that no edges have been deleted, no edges have been fixed, and we have not yet eliminated any permutations as not being k-almost symmetries, respectively. We note also that P_A allows us to write down an additional stopping condition: If at some node A all that is left in P_A are the self-loops, then no additional k-almost symmetries can be found.

Our strategy for controlling the size of the tree is essentially this: at each node A we check necessary conditions for a k_A-almost symmetry mapping i to j for each $\{i, j\} \in E(P_A)$. Any such $\{i, j\}$ that does not satisfy the given necessary conditions is removed. However, since we will only test local consistency, the remaining edges in P_A need not represent actual k_A-almost symmetries of the graph G_A. As laid out in Section 2.2.3, P_A will also provide us a lower bound on the number of possible orbits in any of A's children. Finally, sufficiency is captured by `nauty`, which at node A computes the automorphisms on the graph G_A, and hence the k-almost symmetries at this node.

2.2.1 Eliminating mappings

We present three approaches for proving that two vertices of G are not k-almost symmetric. The first two are an extension of simple vertex invariants to the k-almost symmetry case, and the third makes use of the fact that neighbors must be mapped to neighbors, and dominates the other two, at an additional computational cost. For the discussion of all three, suppose we are at some node in the tree denoted by (G_A, P_A, E_A^F, k_A), as described above.

The first approach, listed in Algorithm 2.1, is based on the following simple fact.

Fact 2.1. If two vertices' degrees differ by more than k_A, then they are not symmetric in any of node A's children.

If the graph has a good amount of irregularity then Algorithm 2.1 will be rather effective. Also note that as edges are deleted and k_A is decreased this becomes more powerful, and hence this is run at the root node and after every edge deletion.

Algorithm 2.1 (DegreeDiffElim) Eliminates mappings between vertices whose degree difference is more than k_A.

procedure DegreeDiffElim(G_A, P_A, k_A)
  for all $\{i, j\} \in E(P_A)$ **do**
    if $|d_{G_A}(i) - d_{G_A}(j)| > k_A$ **then**
      remove edge $\{i, j\}$ from $E(P_A)$

Table 2.1: DegreeDiffElim for graph in Figure 2.1 at root node

vertex v	$d_{G_R}(v)$	$P_R(v)$
1	2	{1, 2, 3, 4, 5, 6}
2	3	{1, 2, 3, 4, 5, ~~6~~}
3	2	{1, 2, 3, 4, 5, 6}
4	3	{1, 2, 3, 4, 5, ~~6~~}
5	3	{1, 2, 3, 4, 5, ~~6~~}
6	1	{1, ~~2~~, 3, ~~4~~, ~~5~~, 6}

Example 2.2. Consider the graph from Example 2.1, and suppose $k = 1$ and we are at the root node R, so P_R is the complete graph with self-loops. See Table 2.1 for how Algorithm 2.1 updates P_R.

The second approach, listed in Algorithm 2.2 is based on another simple observation.

Fact 2.2. For $i \in V$, define $d_{E_A^F}(i)$ to be the number of fixed edges incident to i. For $i, j \in V$, if $d_{E_A^F}(i) > d_{G_A}(j)$, then i and j are not symmetric in any of node A's children.

Since i has more fixed neighbors than j has neighbors, this is just a restatement of the fact that vertices of different degrees cannot be symmetric. While this is useless at the root node, it becomes more and more powerful as edges are fixed, and so is run after every edge fixing.

Algorithm 2.2 (FixedDegElim) Eliminates mappings between vertices where one's fixed degree exceeds the other's degree.

procedure FixedDegElim(G_A, P_A, E_A^F)
  for all $\{i, j\} \in E(P_A)$ **do**
    if $d_{E_A^F}(i) > d_{G_A}(j)$ **then**
      remove edge $\{i, j\}$ from $E(P_A)$

Example 2.3. Picking up where Example 2.2 left off, suppose we are at the node A, where $E_A^D = \emptyset$ and $E_A^F = \{(1, 2), (1, 5)\}$, represented by the graph in Figure 2.2. In this case since $d_{E_A^F}(1) = 2$ and $d_{G_A}(6) = 1$, Algorithm 2.2 allows us to rule out a mapping between vertex 1 and vertex 6 and updates P_A accordingly.

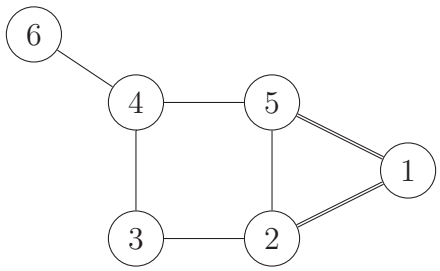


Figure 2.2: FixedDegElim example: Edges (1,2) and (1,5) are fixed

The last, and most powerful method for testing the feasibility of a mapping comes from the observation below.

Fact 2.3. For graph G, if a permutation $\pi \in \mathrm{Aut}(G)$ maps vertex i to vertex j, it must map neighbors of i to neighbors of j.

Its extension to the almost-symmetry case is outlined in Algorithms 2.3 and 2.4. We use these to check the feasibility of such a mapping with k_A edge deletions.

Let us begin with Algorithm 2.3. We construct a bipartite graph as follows. First, create two partite sets, the left one being $N_{G_A}(i) \setminus \{j\}$ and the right one being $N_{G_A}(j) \setminus \{i\}$, and label their associated vertices u_i for $u \in N_{G_A}(i) \setminus \{j\}$ and v_j for $v \in N_{G_A}(j) \setminus \{i\}$ to distinguish them from the vertices in G_A. (We leave j and i out of these partite sets since a mapping from $i \to j$ implies a mapping from $j \to i$.) Add to the left partite set $k_A + \max\{d_{G_A}(j) - d_{G_A}(i), 0\}$ vertices labeled $\times_j$, and to the right partite set add $k_A + \max\{d_{G_A}(i) - d_{G_A}(j), 0\}$ vertices labeled $\times_i$, so as to create two equal-sized partite sets. The weight of the edge between u_i and v_j is the minimum number of edges that need to be deleted so that $u \to v$. If we have already determined $u \nrightarrow v$, we set this to $+\infty$ to represent that the matching cannot happen (it suffices to set it to $(2k_A+1)$). It is important to note here that in actuality this weight is only considered to be half the degree difference between u and v (since everything else is multiplied by 2 – which is done so all values remain integer). This is because any edge deletion also lowers *some other* vertex w's degree by 1. If w is a neighbor of i or j this may "help" it with its matching. Since an edge deletion only has two endpoints it suffices to consider $\frac{1}{2}$ the degree difference. Suppose WLOG $d_{G_A}(u) > d_{G_A}(v)$ and u is independent of $(N(i) \setminus \{j\}) \cup (N(j) \setminus \{i\})$ (that is, u has no neighbors in $(N(i) \setminus \{j\}) \cup (N(j) \setminus \{i\})$). Then we know such a w does not exist, so any edge deletions from u will count only once. Hence we multiply by 2 in this case (recalling everything is doubled). Second, each vertex u_i is connected to at least k_A vertices of the type $\times_i$. These edges represent the deletion of edge $\{i, u\}$ and so get weight 2

Algorithm 2.3 (BuildCostMatrix) Creates the bipartite graph for testing the map between neighbors.

function BuildCostMatrix(i, j, G_A, P_A, E_A^F, k_A)
  for all $u \in N(i) \setminus \{j\}$ **do**
    Create vertex u_i
  for all $v \in N(j) \setminus \{i\}$ **do**
    Create vertex v_j
  if $d_{G_A}(i) < d_{G_A}(j)$ **then**
    Exchange i and j
  degDiff $\leftarrow d_{G_A}(i) - d_{G_A}(j)$
  Create $k_A +$ *degDiff* copies of vertices $\times_i$ and k_A copies of $\times_j$
  for all $u \in N_{G_A}(i) \setminus \{j\}$, $v \in N_{G_A}(j) \setminus \{i\}$ **do**
    Draw an edge between u_i and v_j,
    if $\{u, v\} \in P_A$ **then**
      if $d_{G_A}(u) > d_{G_A}(v)$ and u independent of $(N(i) \setminus \{j\}) \cup (N(j) \setminus \{i\})$ **then**
        $w_{u_i,v_j} \leftarrow 2(d_{G_A}(u) - d_{G_A}(v))$ ▷ Cost of mapping $u \to v$
      else if $d_{G_A}(v) > d_{G_A}(u)$ and v indep. of $(N(i) \setminus \{j\}) \cup (N(j) \setminus \{i\})$ **then**
        $w_{u_i,v_j} \leftarrow 2(d_{G_A}(v) - d_{G_A}(u))$
      else
        $w_{u_i,v_j} \leftarrow |d_{G_A}(u) - d_{G_A}(v)|$
    else
      $w_{u_i,v_j} \leftarrow +\infty$ ▷ We have already determined $u \nrightarrow v$
  for all $u \in N_{G_A}(i) \setminus \{j\}$, $\times_i$ **do**
    Draw an edge between u_i and $\times_i$,
    if $\{i, u\} \in E_A^F$ **then**
      $w_{u_i,\times_i} \leftarrow +\infty$ ▷ Edge $\{i, u\} \in E(G_A)$ cannot be deleted
    else
      $w_{u_i,\times_i} \leftarrow 2$ ▷ Edge $\{i, u\} \in E(G_A)$ can be deleted
  for all $v \in N_{G_A}(j) \setminus \{i\}$, $\times_j$ **do**
    Draw an edge between v_j and $\times_j$,
    if $\{j, v\} \in E_A^F$ **then**
      $w_{v_j,\times_j} \leftarrow +\infty$ ▷ Edge $\{j, v\} \in E(G_A)$ cannot be deleted
    else
      $w_{v_j,\times_j} \leftarrow 2$ ▷ Edge $\{j, v\} \in E(G_A)$ can be deleted
  for all $\times_i$, $\times_j$ **do**
    Draw an edge between $\times_i$ and $\times_j$ with $w_{\times_i,\times_j} \leftarrow 0$.
  return the constructed graph as an assignment matrix

Algorithm 2.4 (RefineByMatching) Eliminates mappings by attempting to map neighbors to neighbors.

```
function RefineByMatching(G_A, P_A, E_A^F, k_A)
    for all e ∈ E(G_A) do
        edgeUse(e) = 0
    for all {i, j} ∈ E(P_A) do
        CostMatrix ← BuildCostMatrix(i, j, G_A, P_A, E_A^F, k_A)
        cost, deleteEdges ← HungarianSolve(CostMatrix)
        if cost > 2k_A then
            remove edge {i, j} from E(P_A)
        else
            for all e ∈ deletedEdges do
                edgeUse(e) ← edgeUse(e) + 1
    return edgeUse
```

or $+\infty$ if through branching the edge $\{i, u\}$ has become fixed. A similar process occurs for each v_j. Finally these "deletion" nodes $\times_i$ and $\times_j$ all have weight 0 between them since *not* deleting an edge from i or j is free.

Now we turn to Algorithm 2.4. For each edge $\{i, j\} \in E(P_A)$ we use Algorithm 2.3 to construct a weighted bipartite graph based on the neighbors of i and j, and the number of edge deletions allowed. The Hungarian Algorithm [85, 114] is used to a find minimum cost perfect matching. From the solution we determine when an edge $\{u_i, \times_i\}$ or $\{v_j, \times_j\}$ is in the optimal assignment. The former corresponds to the deletion of edge $\{i, u\}$ in G_A, the later to the deletion of edge $\{j, v\}$. Finally we determine if the cost is more than $2k_A$, in which case we eliminate the mapping $i \to j$, and if not, we increment *edgeUse* based on the deleted edges.

Based on the preceding discussion, we arrive at the following:

Theorem 2.2. *Algorithm 2.4 is valid, that is, for a given node (G_A, P_A, E_A^F, k_A), if Algorithm 2.4 deletes edge $\{i, j\}$ from P_A, then i and j are not symmetric in any of its children. Further, Algorithm 2.4 dominates Algorithms 2.1 and 2.2, in that any mapping deleted by either Algorithm 2.1 or 2.2 is also deleted by Algorithm 2.4.*

Proof. Assume there exists a permutation $\pi \in \mathcal{AS}_k(G)$ such that $\pi(i) = j$. Thus in some child node we must have $d(i) = d(j)$, with fewer than k_A edge deletions. Since this implies the existence of π^{-1} such that $\pi^{-1}(j) = i$, we exclude j from $N(i)$ and i from $N(j)$ to avoid double counting; we represent these edge deletions explicitly with the nodes $\times_i$ and $\times_j$. Since any permutation must move neighbors to neighbors, for some $I \subseteq N(i) \backslash \{j\}$ and $J \subseteq N(j) \backslash \{i\}$, $\pi : I \to J$ bijectively. Such subsets are given to us by a feasible assignment; we need now consider the edge deletions implicit in

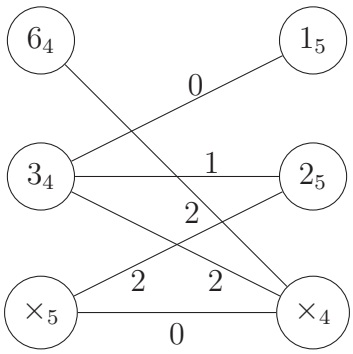


Figure 2.3: RefineByMatching example: Edges with weight $+\infty$ have been excluded for clarity.

such an assignment. For each $u \in I$ let R_u be a set of edge removals needed so that $d(u) = d(\pi(u))$ (note: $|R_u| \geq |d_{G_A}(u) - d_{G_A}(\pi(u))|$). Let $\mathcal{R} = \{R_{u_1}, R_{u_2}, \ldots, R_{u_{|I|}}\}$. We need to prove then that any edge e occurs at most twice in such a list; further if $d_{G_A}(u) > d_{G_A}(\pi(u))$ and u is independent of $(N(i) \setminus \{j\}) \cup (N(j) \setminus \{i\})$, any edge occurs at most once. For contradiction suppose such an edge e occurred m times in $\mathcal{R}$. Since I has distinct vertices and J has distinct vertices, if $m > 2$ then e has more than two endpoints, a contraction. Similarly suppose $d_{G_A}(u) > d_{G_A}(\pi(u))$ and u independent of $(N(i) \setminus \{j\}) \cup (N(j) \setminus \{i\})$. At least $(d_{G_A}(u) - d_{G_A}(\pi(u)))$ edges in R_u have an endpoint at u. Therefore at least $(d_{G_A}(u) - d_{G_A}(\pi(u)))$ many edges appear only once in $\mathcal{R}$ by the independence of u. Hence, assuming a minimum matching, line 6 of Algorithm 2.4 will return at least twice the minimal number of edge removals needed for $i \to j$.

The last part follows from noting that any feasible solution will be at least $2|d_{G_A}(i) - d_{G_A}(j)|$ and that if $d_{E_A^F}(i) > d_{G_A}(j)$ then in a perfect matching one of the fixed neighbors of i must be matched to one of the vertices $\times_i$. □

Example 2.4. Continuing where Example 2.3 left off, we have the graph shown in Figure 2.2. Suppose $k_A = 1$ and P_A is as follows:

1 : {1, 2, 3, 5}

2 : {1, 2, 3, 4, 5}

3 : {1, 2, 3, 5, 6}

4 : {2, 3, 4, 5}

5 : {1, 2, 4, 5}

6 : {3, 6}

and suppose we enter the **for all** loop on line 5 of Algorithm 2.4 with $\{i, j\} = \{4, 5\} \in P_A$. Algorithm 2.3 constructs the bipartite graph shown in Figure 2.3. We see by inspection that the

minimum cost perfect matching has cost $4 > 2k_A$, so mapping $\{4, 5\}$ can be deleted.

Finally, we note that in spite of Theorem 2.2, Algorithms 2.1 and 2.2 are still useful. A worst-case complexity bound on both is $\mathcal{O}(n^2)$, whereas assuming HungarianSolve is implemented efficiently (i.e., $\mathcal{O}(n^3)$), Algorithm 2.4 may need $\mathcal{O}(n^2(n + k_A)^3)$ time.

2.2.2 Branching

As mentioned above, at a given node (G_A, P_A, E_A^F, k_A), an edge $e \in E(G_A) \setminus E_A^F$ is selected for branching. Two children are created based on the selection of e, one where e is deleted (added to E_A^D) and k_A is decreased, and the other where e is added E_A^F.

We use the following rule for selecting e using the *edgeUse* array collected in Algorithm 2.4. Our first choice for a branching edge e will be a maximal element of this array. The hope is that in the deletion child, an edge that is "getting in the way" of symmetry is deleted. Conversely, in the child where the edge is fixed it is expected that such an edge will cause P_A to lose several edges at the next pass of Algorithm 2.4. Computational experiments (Section 2.4) show this to be a much better rule than random branching, and is in fact often among the best edges to branch on. If no such maximal edge is found, we fall back to the rule of branching on the first edge found in $E(G_A) \setminus E_A^F$. If it happens that $E(G_A) \setminus E_A^F = \emptyset$, this node is pruned.

We know that any automorphism maps edges to edges and non-edges to non-edges. Therefore if we have determined that two vertices are *fixed* with respect to automorphisms in all this node's children, we get the following helpful observation.

Observation 2.1. If $N_{P_A}(i) = \{i\}$ and $N_{P_A}(j) = \{j\}$, we need not branch on edge $\{i, j\}$.

Since i can only be mapped to i and j to j, then $\{i, j\}$ will always be mapped to itself, whether it is an edge or non-edge. The preceding discussion is summarized in Algorithm 2.5.

Algorithm 2.5 (FindBranchEdge) Selects an edge to branch on.

```
function FindBranchEdge(G_A, P_A, E_A^F, edgeUse)
    if max(edgeUse) > 0 then
        return argmax(edgeUse)
    for all {i, j} ∈ E(G_A) \ E_A^F do
        if N_{P_A}(i) ≠ {i} or N_{P_A}(j) ≠ {j} then
            return {i, j}
    return prune                    ▷ If here either E(G_A) \ E_A^F = ∅ or all edges satisfy Remark 2.1
```

2.2.3 Bounding

Bounding is done in two ways. First, after an edge is deleted we compute the symmetries of the modified graph G_A. The number of orbits in the orbital partition gives an upper bound on the solution value, since this is a feasible solution for (2.1). Lower bounding is done using the information in P_A. For a lower bound using P_A, we can partition $V(= V(G) = V(G_A) = V(P_A))$ using the following rule:

> Two vertices in V can belong to the same partition if there exists an edge between them in P_A.

The set of all such partitionings of V represent all possible orbital partitions at this node. Therefore in order to generate a valid lower bound we would need the minimum of such partitionings. This leads to the following observation.

Remark 2.2. Partitioning V as described above is the same as vertex coloring P_A's complement, $\overline{P_A}$.

Since finding the chromatic number of a graph [55] and approximating the chromatic number of a graph [39] are both NP-hard, we settle for a "bad" lower bound on the partition size, namely we use the size of a greedily constructed independent set for P_A. This has the merit though of only requiring $\mathcal{O}(n)$ time. Defining $\chi(G)$, $\alpha(G)$, and $\omega(G)$ to be the chromatic number, independence number, and clique number of G, respectively, we know from basic graph theory $\chi(\overline{P_A}) \geq \omega(\overline{P_A}) = \alpha(P_A) \geq |I|$, for any independent set I. Therefore we have the following bounding procedure:

> After any call to Algorithms 2.1, 2.2, or 2.4, compute the size of a maximal independent set of P_A. If this is greater than or equal the current incumbent, we prune.

2.2.4 An Algorithm

We now tie together the preceding discussion in Algorithm 2.6 that solves (2.1) using a depth-first search. The routines HungarianSolve, ComputeAutomorphisms and GreedyIndependentSetSize are treated as a black-box. We similarly assume PopFromStack and AppendToStack manage *NodeStack*.

Theorem 2.3. *Algorithm 2.6 is valid.*

Algorithm 2.6 (FindAlmostSymmetry) Solves (2.1)

function FindAlmostSymmetry(G, k)
  initialize $P_R \leftarrow$ {complete graph with self-loops}; $E_R^D \leftarrow \emptyset$; $E_R^F \leftarrow \emptyset$
  initialize *delChild* $\leftarrow$ **true**
  initialize *incumbentValue* $\leftarrow |V(G)|$; *incumbentSolution* $\leftarrow \emptyset$
5: **initialize** *NodeStack* $\leftarrow$ { (P_R, E_R^D, E_R^F, *delChild*) }
  while *NodeStack* $\neq \emptyset$ **do**
    (P_A, E_A^D, E_A^F, *delChild*) $\leftarrow$ PopFromStack
    $G_A \leftarrow G - E_A^D$.
    $k_A \leftarrow k - |E_A^D|$
10: **if** *delChild* **then** ▷ This node is either root or has a new edge in E_A^D
      orbitNum $\leftarrow$ ComputeAutomorphisms(G_A)
      if *orbitNum* $<$ *incumbentValue* **then**
        incumbentValue $\leftarrow$ *orbitNum*; *incumbentSolution* $\leftarrow E_A^D$
      if $k_A = 0$ **or** $|E_A^F| = |E(G_A)|$ **then**
15: **prune** ▷ (Delete this node and go to line 6)
      DegreeDiffElim(G_A, P_A, k_A)
    else ▷ This node has a new edge in E_A^F
      if $|E_A^F| = |E(G_A)|$ **then**
        prune
20: FixedDegElim(G_A, P_A, E_A^F)
    lowerBound $\leftarrow$ GreedyIndependentSetSize(P_A)
    if *lowerBound* $\geq$ *incumbentValue* **then**
      prune
    while P_A is changed by RefineByMatching() **do**
25: *edgeUse* $\leftarrow$ RefineByMatching(G_A, P_A, E_A^F, k_A)
      lowerBound $\leftarrow$ GreedyIndependentSetSize(P_A)
      if *lowerBound* $\geq$ *incumbentValue* **then**
        prune
    branchEdge $\leftarrow$ FindBranchEdge(G_A, P_A, E_A^F, *edgeUse*)
30: ▷ Once we have arrived here all the bounding we can do is done
    if *branchEdge* = prune **then**
      prune
    ▷ Else we will create the children
    AppendToStack(P_A, E_A^D, E_A^F + *branchEdge*, **false**) ▷ Edge fixing child
    AppendToStack(P_A, E_A^D + *branchEdge*, E_A^F, **true**) ▷ Edge deletion child
35: **return** *incumbentValue*, *incumbentSolution*

Proof. This follows from Theorem 2.2. If we exclude the logic which prunes by bound (lines 21-23, 26-28) and collected the permutations computed in line 11 we find all the k-almost symmetries of G. □

2.3 Implementation

In this section we discuss some of the implementation choices made and libraries used; however, the interested reader is referred directly to the source code for details. C++ is used throughout.

`PEBBL` is a general-purpose parallel branch-and-bound framework written in C++ [34]. `PEBBL` allows the easy implementation of a parallel branch-and-bound algorithm provided the user already has a serial implementation in mind (such as Algorithm 2.6). `PEBBL` is therefore used for tree management. `PEBBL` has many user-configurable options, in particular the user can specify the search order to be breadth-first, depth-first, or the default best-first, which in the minimization case will select a problem with the lowest bound. One particularly powerful feature of `PEBBL` is its ability to exploit parallelism during the *ramp-up* phase. Ramping-up occurs when the number of active nodes in the search tree is smaller than the number of available processors. During ramp-up, before parallel enumeration begins, `PEBBL` has all threads synchronously explore the same nodes of the branch-and-bound tree near the root node. This allows the programmer to exploit parallelism that may be present within each node using MPI communication. Since the running time of the Hungarian Algorithm is $\mathcal{O}(n^3)$, a parallel version of Algorithm 2.4 is used during ramp-up. Lines 5-6 are parallelized; we do an MPI reduce operation to collect P_A *or edgeUse*. Notice in RefineByMatching we only use *edgeUse* if P_A does not change, so it does not need to be reduced, and we only need to reduce P_A if it does change, in which case we will call RefineByMatching again and not use *edgeUse*. Since we would expect P_A to be fairly dense high in the tree this is helpful. Once *cross-over* occurs and `PEBBL` is doing parallel enumeration the serial version of Algorithm 2.4 is used.

A C implementation of the Hungarian Method from [146] is used for HungarianSolve, which itself is an enhancement of the implementation provided by the Stanford GraphBase [83]. This code was slightly modified to avoid redundant memory allocation/deallocation in the second **for all** loop of Algorithm 2.4, and thus is included with the source code.

Finally, `nauty` 2.5r9 [103] is used as the implementation of ComputeAutomorphisms in Algorithm 2.6, with canonical labeling turned off. `nauty`'s packed graph format is used for graph

representation. GreedyIndependentSetSize uses the standard greedy procedure to construct an independent set and returns the cardinality of the constructed set. OpenMPI was the MPI implementation used for all tests.

2.4 Computational Results

All computational experiments were done on a Dell PowerEdge T620 with 2 Intel Xeon E5-2670 processors and 256GB of memory running Ubuntu 14.04.2. Hyper-threading was enabled for a total of 16 cores and 32 threads. Random graph instances from [104] and DIMACS coloring instances from [26] were analyzed. Coloring instances that already exhibited large amounts of symmetry (i.e., $\gamma_0^G \leq \frac{1}{2}|V(G)|$) were excluded from testing. The test set was further reduced by only selecting a subset of the Leighton graphs. All times are wall-clock times reported by `PEBBL`.

The results of the main experiment are reported in Table 2.2. γ_k^G (equation (2.1)) was computed for each graph, incrementing k. A wall clock limit of 60 minutes was used, and k was no longer incremented after hitting the time limit. We also report on the size of the automorphism group for the optimal subgraph G', labeled $|\mathrm{Aut}(G')|$. Note that for presentation reasons, we sometimes scale this value by a constant. Since `PEBBL`'s parallel search is non-deterministic, each experiment is repeated five times, and the average search time is reported. If every run timed-out we report that with a †. On instances where at least one repetition timed-out we report the best objective value of a solution found across all five. An asterisk indicates that we were not able to prove the objective value optimal in any of the five repetitions. For conciseness some levels of k are excluded. All 32 available threads were used for this test, and `PEBBL`'s best-first search was used to explore the tree.

From the objective value in Table 2.2 we can see that for many of the structured graphs, many fewer edges than Theorem 2.1 requires need be removed to induce a non-trivial k-almost symmetry. Additionally there is often a commensurate increase in group size as the number of orbits decreases. Inducing symmetry on random graphs is more difficult, as is to be expected given Theorem 2 in [35] (the bound from Theorem 2.1 is tight in the limit for random graphs).

We also examined how the algorithm scales. Given the results in Table 2.2 some easier instances were selected and ran with a varying number of threads up to 32. Note that the system used only has 16 cores, so linear speedup cannot be expected past that even in the ideal case. The results

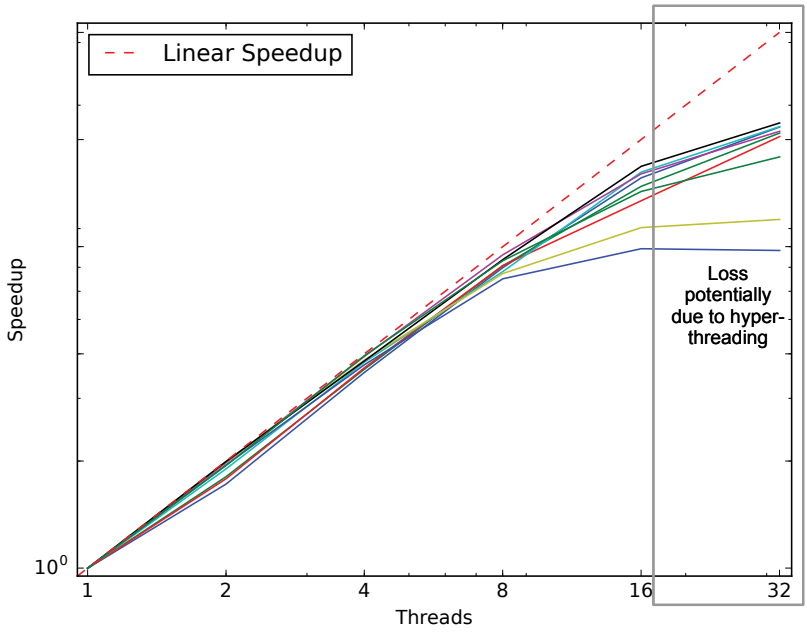


Figure 2.4: Scaling on various problems

are in Figure 2.4. We can see good scaling through 8 cores on the test examples, starting to taper off at 16 cores. Individual test cases are detailed in Appendix A.

2.4.1 Branching Strategy

We compare the branching strategy dictated by *edgeUse* (*edgeUse* branching) against selecting a random eligible edge to branch on. We also consider a "local branching" rule, in which after we have done all the valid refining that can be done at a node, we select the branching edge by doing an additional single pass at RefineByMatching with $k = 1$. Several easy instances from above were tested using a single thread. Each random branching trial was replicated 50 times, and compared to the *edgeUse* branching strategy and the local branching strategy. The results are summarized in the graphs in Figure 2.5, where the box-plots are the random branching trials, the red square with the text on the right represents the *edgeUse* branching strategy, and the blue $\times$ with the text on the left represents the local branching strategy. We show four examples here, the remaining are available in Appendix A. The *edgeUse* branching strategy laid out in Section 2.2.2 was always at least as good (and often much better) than random in the examples tested, and in addition was always better than the local branching strategy. While the local branching strategy was sometimes

Table 2.2: Computational Results

games120.col										$n = 120$ $e = 638$
k	0	1	2	3	4	5	6	7	8	9
γ_k^G	119	118	117	114	113	112	112	111	111	112*
$\lvert\mathrm{Aut}(G')\rvert$	2	4	8	8	16	48	48	16	16	48*
seconds	0.0	0.0	0.1	0.2	0.2	0.4	2.3	27.8	896.9	†

miles250.col								$n = 128$ $e = 387$
k	0	1	2	3	4	5	6	7
γ_k^G	108	106	104	102	100	99	97	97*
$\lvert\mathrm{Aut}(G')\rvert/10^7$	0.26542	3.1851	6.3701	12.740	203.84	1019.2	815.37	6115.3*
seconds	0.0	0.2	0.3	0.8	3.1	57.6	844.8	†

miles500.col									$n = 128$ $e = 1170$
k	0	1	2	3	4	5	6	7	8
γ_k^G	114	113	111	110	109	108	107	106	107*
$\lvert\mathrm{Aut}(G')\rvert/10^6$	0.82944	1.6580	3.3178	6.6355	46.449	92.897	185.79	53.084	371.59*
seconds	0.0	0.1	0.2	0.4	1.5	14.7	186.1	1868	†

miles750.col										$n = 128$ $e = 2113$
k	0	1	2	3	4	5	6	7	8	
γ_k^G	122	121	120	119	118	117	116	115	115*	
$\lvert\mathrm{Aut}(G')\rvert$	96	288	576	1152	2304	4608	9216	18432	18432*	
seconds	0.0	0.1	0.3	0.5	1.4	9.1	86.3	849.1	†	

miles1000.col									$n = 128$ $e = 3216$
k	0	1	2	3	4	5	6	7	
γ_k^G	123	122	121	120	119	118	118	119*	
$\lvert\mathrm{Aut}(G')\rvert$	72	144	288	576	1152	2304	2304	1152*	
seconds	0.0	0.1	0.2	0.5	2.2	33.5	416.9	†	

miles1500.col							$n = 128$ $e = 5198$
k	0	1	2	3	4	5	
γ_k^G	102	101	100	99	98	97*	
$\lvert\mathrm{Aut}(G')\rvert/10^{12}$	0.11466	0.80263	3.2105	4.5865	19.263	57.790*	
seconds	0.0	0.8	2.6	40.0	913.5	†	

le450_5b.col									$n = 450$ $e = 5734$
k	0	13	14	15	16	17	18	19	
γ_k^G	450	450	450	450	450	450	450	449*	
$\lvert\mathrm{Aut}(G')\rvert$	1	1	1	1	1	1	1	2*	
seconds	0.0	1.1	1.2	1.6	2.0	5.0	564.1	†	

le450_15b.col														$n = 450$ $e = 8169$
k	0	1	2	4	5	7	8	10	11	14	15	16	17	
γ_k^G	450	450	449	449	448	448	447	447	446	446	445	445	446*	
$\lvert\mathrm{Aut}(G')\rvert$	1	1	2	2	6	6	24	24	120	120	720	720	120*	
seconds	0.0	0.1	0.2	0.4	0.5	0.8	0.9	1.1	1.2	8.3	19.7	93.2	†	

le450_25b.col												$n = 450$ $e = 8263$
k	0	1	2	3	4	5	6	7	8	9	10	
γ_k^G	450	450	449	449	449	448	448	447	447	447	447*	
$\lvert\mathrm{Aut}(G')\rvert$	1	1	2	2	2	6	6	12	12	12	12*	
seconds	0.0	0.1	0.2	0.3	0.6	0.7	0.9	1.6	7.1	43.8	†	

Table 2.2: (continued)

ran10_100_a.bliss									$n = 100$ $e = 502$
k	0	6	7	8	9	10	11	12	
γ_k^G	100	100	99	99	99	99	99	100*	
$\lvert\mathrm{Aut}(G')\rvert$	1	1	2	2	2	2	2	1*	
seconds	0.0	0.0	0.1	0.4	2.8	22.8	372.0	†	
ran10_100_b.bliss									$n = 100$ $e = 464$
k	0	3	4	7	8	9	10	11	12
γ_k^G	100	100	99	99	99	99	98	98	100*
$\lvert\mathrm{Aut}(G')\rvert$	1	1	2	2	2	2	3	3	1*
seconds	0.0	0.0	0.0	0.2	4.0	11.1	116.6	1924.3	†
ran10_100_c.bliss									$n = 100$ $e = 525$
k	0	5	6	7	8	9	10	11	12	13
γ_k^G	100	100	99	99	99	99	99	98	98	100*
$\lvert\mathrm{Aut}(G')\rvert$	1	1	2	2	2	2	2	6	6	1*
seconds	0.0	0.0	0.0	0.1	0.5	2.3	9.1	117.5	1712	†
ran10_100_d.bliss									$n = 100$ $e = 514$
k	0	1	7	8	9	10	11	12	
γ_k^G	100	100	100	99	99	99	99	100*	
$\lvert\mathrm{Aut}(G')\rvert$	1	1	1	2	2	2	2	1*	
seconds	0.0	0.0	0.1	0.9	5.2	25.8	560.3	†	

competitive with *edgeUse* branching, in other cases it does worse (in a few instances significantly so) than the random branching mean.

In order to test the strength of *edgeUse* branching, we test it against strong branching at the root node. Strong branching is implemented at the root node by considering the number of permutations refined in the "edge fixing" child for every edge in the graph G. We then rank the potential edge branches by the number of permutations removed by RefineByMatching for the candidate edge in the edge fixing child node. The results are presented in Table 2.3, where we present the ranks in percentiles. Instances which are solved at the root node are discarded. In particular, in the column labeled "*edgeUse* branch percentile rank" we give the percentage of edges with a worse score than the *edgeUse* branch. In the column labeled $\frac{\#\text{ refined by } edgeUse \text{ branch}}{\#\text{ refined by strong branch}}$ we present the ratio of permutations refined by the *edgeUse* branch against the permutations refined by the strongest branch. Hence, if this is 1, that means the *edgeUse* branch is as good as the strong branch. First, we can see based on the *edgeUse* branch percentile rank that it is often in the top 1% of possible edge choices based on our strong branching score, and in only 6 of the 99 cases is it below the 80th percentile. Turning to the ratio, we see that in 59 of the 99 cases *edgeUse* branching selects an edge with the highest rank. However, in 18 of the 99 cases the *edgeUse* branch is 80% worse by score than the strong branch. Note that deciding an *edgeUse* branch is a linear check after RefineByMatching, whereas strong branching is a potentially $\mathcal{O}(m \cdot n^2(n+k)^3)$ check at the root node (where n is the number of vertices and m is the number of edges in the graph

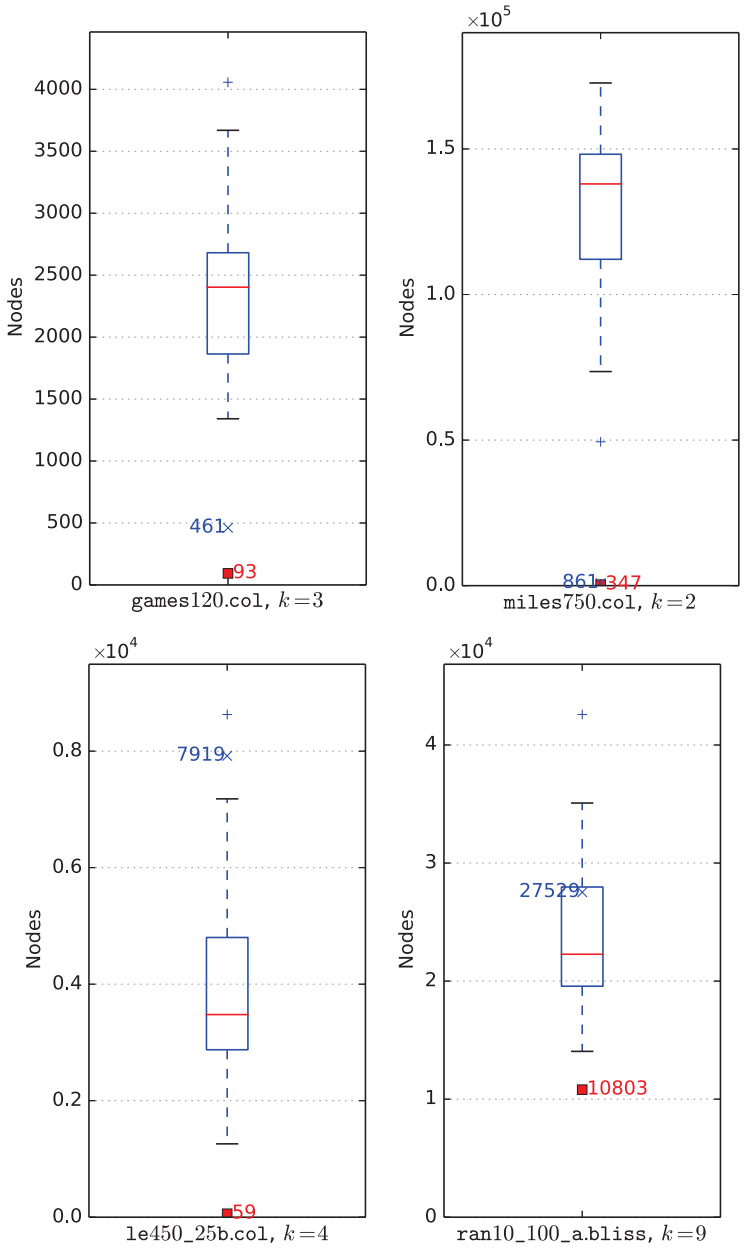


Figure 2.5: Random branching (box plot) vs. *edgeUse* branching (■, number of nodes on right) vs. local branching (×, number of nodes on left), selected instances

G). In this light, *edgeUse* branching seems both practical and effective for selecting a disjunction. Additionally, these results (along with the results in Section 2.4.2) verify the notion that the edges that end up being deleted in the matchings are often those that are "in the way" of symmetry.

We also demonstrate the robustness of *edgeUse* branching. Recall, for a given node, *edgeUse* branching selects the most frequently deleted edge in the refinement. To test the robustness we examine how the ranking changes after branching and refining. In particular, how often is the second place edge chosen for branching in the child problems? If it has a high rank, this suggests that (1) our choice of edge is somewhat insensitive to changes in the graph, and (2) we may be able to get away with refining less frequently, while still maintaining the strength of our branching strategy. If its rank is 1, then this is the exact edge we branched on in the child problems. The tests were all done on one thread, and the results are reported in Table 2.4. We report the rank in the child (Rank) of the second most frequently deleted edge in the parent refinement, and the percentage of nodes in the tree where it has a given rank (% of Nodes) for selected instances. (The other instances tests are reported in Appendix A.) As we can see, it is often the case that the next edge selected for branching is that with second rank in the parent node. Note that as implemented, we refine as much as possible at each node to attempt to lift the bound, so it may be computationally advantageous to refine less frequently, at the possible cost of more nodes, while not losing much in the strength of our branching decisions. In Appendix A we present additional computational results that attempt to exploit this using a modified branching strategy.

2.4.2 Heuristics

To exploit almost symmetries in a problem such as (2.2), it is not necessary to compute the optimal group of k-almost symmetries. Toward that end, we examine how well both *edgeUse* branching and local branching work as a heuristic for the problems in Table 2.2 by just diving left and never creating the edge fixing child. That is, we consider Algorithm 2.6 with line 33 excluded.

All heuristic tests were done on a single thread. The main result is in Table 2.5, where we compare the optimality gap closed by both the *edgeUse* branching and local branching rule, where the optimality gap closed is defined as

$$\text{optimality gap closed} := \frac{\gamma_0^G - \lambda_k^G}{\gamma_0^G - \gamma_k^G}, \tag{2.3}$$

Table 2.3: *edgeUse* branching vs. strong branching at root node

`games120.col`									$n = 120$ $e = 638$
k	1	2	3	4	5	6	7	8	9
edgeUse branch percentile rank	99.9	49.4	99.8	100	100	99.2	1.0	99.2	94.9
$\frac{\text{\# refined by } edgeUse \text{ branch}}{\text{\# refined by strong branch}}$	1.0	0.0	0.50	1.0	1.0	0.185	1.0	0.033	0.020
`miles250.col`									$n = 128$ $e = 387$
k	1	2	3	4	5	6	7		
edgeUse branch percentile rank	99.8	99.4	94.8	83.0	94.9	85.5	82.6		
$\frac{\text{\# refined by } edgeUse \text{ branch}}{\text{\# refined by strong branch}}$	1.0	0.812	0.342	0.094	0.345	0.214	0.279		
`miles500.col`									$n = 128$ $e = 1170$
k	1	2	3	4	5	6	7	8	
edgeUse branch percentile rank	99.9	99.9	100.0	99.9	99.9	99.7	99.9	100	
$\frac{\text{\# refined by } edgeUse \text{ branch}}{\text{\# refined by strong branch}}$	1.0	1.0	1.0	1.0	0.692	0.850	0.857	1.0	
`miles750.col`									$n = 128$ $e = 2113$
k	1	2	3	4	5	6	7	8	
edgeUse branch percentile rank	99.9	99.8	99.7	1.0	99.9	99.9	96.7	99.8	
$\frac{\text{\# refined by } edgeUse \text{ branch}}{\text{\# refined by strong branch}}$	1.0	1.0	1.0	1.0	1.0	0.900	0.500	0.916	
`miles1000.col`									$n = 128$ $e = 3216$
k	1	2	3	4	5	6	7		
edgeUse branch percentile rank	99.9	99.9	99.8	99.9	100	99.9	99.9		
$\frac{\text{\# refined by } edgeUse \text{ branch}}{\text{\# refined by strong branch}}$	1.0	1.0	1.0	1.0	1.0	1.0	1.0		
`miles1500.col`									$n = 128$ $e = 5198$
k	1	2	3	4	5				
edgeUse branch percentile rank	99.9	100	99.9	99.9	99.9				
$\frac{\text{\# refined by } edgeUse \text{ branch}}{\text{\# refined by strong branch}}$	1.0	1.0	1.0	1.0	1.0				
`le450_5b.col`									$n = 450$ $e = 5734$
k	18	19							
edgeUse branch percentile rank	87.5	78.8							
$\frac{\text{\# refined by } edgeUse \text{ branch}}{\text{\# refined by strong branch}}$	0.030	0.019							
`le450_15b.col`									$n = 450$ $e = 8169$
k	2	3	4	5	6	7	8	9	
edgeUse branch percentile rank	99.9	99.9	99.9	99.9	99.9	99.9	99.9	99.9	
$\frac{\text{\# refined by } edgeUse \text{ branch}}{\text{\# refined by strong branch}}$	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	
k	10	11	12	13	14	15	16	17	
edgeUse branch percentile rank	99.9	99.9	99.9	99.9	99.9	100	100	99.9	
$\frac{\text{\# refined by } edgeUse \text{ branch}}{\text{\# refined by strong branch}}$	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	
`le450_25b.col`									$n = 450$ $e = 8263$
k	2	3	4	5	6	7	8	9	10
edgeUse branch percentile rank	99.9	100	99.9	100	99.9	99.9	100	100	100
$\frac{\text{\# refined by } edgeUse \text{ branch}}{\text{\# refined by strong branch}}$	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Table 2.3: (continued)

ran10_100_a.bliss							$n = 100$ $e = 502$
k	7	8	9	10	11	12	
edgeUse branch percentile rank	100	96.4	84.7	91.2	93.0	91.6	
$\frac{\#\text{ refined by } edgeUse \text{ branch}}{\#\text{ refined by strong branch}}$	1.0	0.130	0.250	0.276	0.219	0.093	

ran10_100_b.bliss									$n = 100$ $e = 464$
k	4	5	6	7	8	9	10	11	12
edgeUse branch percentile rank	99.6	99.6	100	100	95.0	68.5	72.7	83.1	96.7
$\frac{\#\text{ refined by } edgeUse \text{ branch}}{\#\text{ refined by strong branch}}$	1.0	1.0	1.0	1.0	0.297	0.102	0.102	0.095	0.179

ran10_100_c.bliss									$n = 100$ $e = 525$
k	6	7	8	9	10	11	12	13	
edgeUse branch percentile rank	99.5	99.8	92.0	97.2	79.7	89.5	97.1	95.6	
$\frac{\#\text{ refined by } edgeUse \text{ branch}}{\#\text{ refined by strong branch}}$	1.0	1.0	0.036	0.666	0.101	0.098	0.343	0.238	

ran10_100_d.bliss						$n = 100$ $e = 514$
k	8	9	10	11	12	
edgeUse branch percentile rank	99.2	85.8	69.6	97.7	99.2	
$\frac{\#\text{ refined by } edgeUse \text{ branch}}{\#\text{ refined by strong branch}}$	0.364	0.175	0.081	0.454	0.692	

where λ_k^G is the objective value of the heuristic solution, and γ_k^G, γ_0^G are the optimal values from Table 2.2. Recall that γ_0^G is the number of vertex orbits in the graph G, so we measure performance relative to this trivial solution. That is, a value of 0% means $\lambda_k^G = \gamma_0^G$, i.e., no improvement in objective value, and a value of 100% means $\lambda_k^G = \gamma_k^G$, i.e., the heuristic found a globally optimal solution. For k such that $\gamma_0^G = \gamma_k^G$ we show a "–". Timing results are reported in Table 2.6. As we can see, for `games120.col` and the `miles` graphs, the performance of the *edgeUse* branching dive is not impressive, especially for larger values of k. That being said, for the `le450` and random graphs it perpforms better, and usually either finds the optimal objective value or is only one away. Turning to the local branching heuristic, we see the situation is exactly reversed from before. The local branching dive is able to do well with `games120.col` and the `miles` graphs, whereas it is not as successful finding the little almost symmetry in the `le450` and random graphs. Both heuristics can be run usually in under a few seconds for graphs of this size, with the exception being `le450_5b.col` for large values of k, which is a graph with least almost symmetry examined. Overall the *edgeUse* branching heuristic performs the best, usually capturing a solution that is within 50% of optimal with only a few seconds of computational effort on a single thread.

2.5 Conclusion

In this chapter we presented and tested a branch-and-bound algorithm for solving a generalized version of Automorphism-Partition. We provide a branching strategy which is much more effective at controlling the size of the tree, compared to random branching, even on random graphs,

Table 2.4: *edgeUse* branching robustness, selected instances

`games120.col`, $k = 7$

Rank	% of Nodes
1	56.54%
2	16.02%
3	4.70%
4	3.24%
5	2.51%
6	2.05%
7	1.71%
8	1.33%
9	1.41%
10	1.00%
11+	9.48%

`miles750.col`, $k = 5$

Rank	% of Nodes
1	85.56%
2	1.88%
3	0.09%
4	0.10%
5	0.12%
6	0.63%
7	1.42%
8	0.98%
9	0.66%
10	0.64%
11+	7.92%

Table 2.4: (continued)

`le450_25b.col`,
$k = 9$

Rank	% of Nodes
1	72.43%
2	0.31%
3	0.14%
4	0.05%
5	0.10%
6	0.09%
7	0.08%
8	0.04%
9	0.11%
10	0.22%
11+	26.41%

`ran10_100_a.bliss`,
$k = 10$

Rank	% of Nodes
1	76.81%
2	0.37%
3	0.17%
4	0.28%
5	1.24%
6	3.07%
7	3.26%
8	3.15%
9	0.92%
10	0.04%
11+	10.70%

and provide computational evidence that it is relatively robust. The branching strategy presented can often be used heuristically to find a non-trivial set of almost-symmetries (i.e., not $\mathrm{Aut}(G)$) in a relatively short time period by diving left. Finally, we demonstrated that parallel enumeration is effective in speeding up the wall-clock solution times for our implementation.

Table 2.5: % gap reduced for heuristics

edgeUse branching heuristic:																		
Graph / k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
games120.col	100%	100%	100%	17%	14%	14%	0%	13%										
miles250.col	0%	75%	33%	25%	22%	0%												
miles500.col	0%	67%	50%	60%	50%	43%	38%											
miles750.col	0%	50%	67%	75%	60%	50%	43%											
miles1000.col	0%	0%	33%	50%	40%	60%												
miles1500.col	0%	50%	33%	25%														
le450_5b.col	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
le450_15b.col	–	100%	100%	100%	100%	100%	100%	100%	100%	100%	75%	100%	100%	100%	80%	80%		
le450_25b.col	–	100%	100%	100%	50%	50%	67%	67%	67%									
ran10_100_a.bliss	–	–	–	–	–	–	100%	100%	100%	100%	100%							
ran10_100_b.bliss	–	–	–	100%	100%	100%	100%	100%	100%	50%	50%							
ran10_100_c.bliss	–	–	–	–	–	100%	100%	100%	100%	100%	50%	50%						
ran10_100_d.bliss	–	–	–	–	–	–	–	0%	0%	0%	0%							

Local branching heuristic:																		
Graph / k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
games120.col	100%	100%	40%	100%	86%	86%	75%	75%										
miles250.col	0%	25%	50%	38%	44%	45%												
miles500.col	0%	0%	25%	40%	33%	29%	25%											
miles750.col	0%	50%	67%	50%	40%	50%	57%											
miles1000.col	0%	0%	33%	50%	60%	80%												
miles1500.col	0%	50%	67%	75%														
le450_5b.col	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
le450_15b.col	–	100%	100%	100%	50%	100%	100%	67%	67%	67%	50%	50%	50%	50%	40%	40%		
le450_25b.col	–	0%	0%	0%	0%	0%	0%	0%	0%									
ran10_100_a.bliss	–	–	–	–	–	–	0%	0%	0%	0%	0%							
ran10_100_b.bliss	–	–	–	0%	0%	100%	0%	0%	0%	0%	0%							
ran10_100_c.bliss	–	–	–	–	–	0%	100%	100%	100%	100%	50%	50%						
ran10_100_d.bliss	–	–	–	–	–	–	–	0%	0%	0%	0%							

Table 2.6: Time in seconds for heuristics

edgeUse branching heuristic:																		
Graph / k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
games120.col	0.1	0.1	0.1	0.2	0.7	0.9	1.1	1.2										
miles250.col	0.0	0.0	0.1	0.1	0.1	0.3												
miles500.col	0.0	0.0	0.1	0.1	0.1	0.2	0.2											
miles750.col	0.0	0.1	0.1	0.1	0.2	0.3	0.3											
miles1000.col	0.0	0.1	0.2	0.2	0.3	0.4												
miles1500.col	0.2	0.5	0.7	1.0														
le450_5b.col	0.6	0.9	1.3	1.4	1.8	1.9	2.2	2.4	2.4	2.7	2.9	3.2	3.3	3.8	4.7	6.3	19.6	51.0
le450_15b.col	0.4	0.8	0.9	1.2	1.2	1.4	1.7	1.8	1.9	2.2	2.4	2.4	2.8	2.9	3.2	3.2		
le450_25b.col	0.3	0.7	0.8	0.9	1.2	1.2	1.3	1.7	1.7									
ran10_100_a.bliss	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.4	0.7	0.9	0.7							
ran10_100_b.bliss	0.0	0.0	0.0	0.0	0.1	0.1	0.3	0.6	0.7	0.9	0.8							
ran10_100_c.bliss	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.5	0.8	0.8	0.8	1.1						
ran10_100_d.bliss	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.4	0.8	0.8	0.8							

Local branching heuristic:																		
Graph / k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
games120.col	0.1	0.1	0.1	0.2	0.6	1.0	1.1	1.2										
miles250.col	0.0	0.0	0.1	0.1	0.2	0.3												
miles500.col	0.0	0.0	0.1	0.1	0.1	0.2	0.2											
miles750.col	0.0	0.1	0.1	0.2	0.2	0.3	0.4											
miles1000.col	0.0	0.1	0.2	0.2	0.3	0.5												
miles1500.col	0.2	0.5	0.9	1.0														
le450_5b.col	0.7	1.0	1.4	1.6	1.8	2.0	2.0	2.4	2.5	2.5	3.4	3.1	3.3	3.8	4.6	6.4	19.5	35.3
le450_15b.col	0.4	0.7	0.9	1.0	1.1	1.4	1.5	1.9	2.0	2.0	2.4	2.5	2.8	2.5	3.0	3.2		
le450_25b.col	0.3	0.6	0.9	0.9	1.0	1.1	1.4	1.5	1.6									
ran10_100_a.bliss	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.4	0.6	1.0	0.9							
ran10_100_b.bliss	0.0	0.0	0.0	0.0	0.1	0.1	0.3	0.6	0.9	1.0	1.0							
ran10_100_c.bliss	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.5	0.9	0.9	1.0	0.9						
ran10_100_d.bliss	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.4	0.8	0.9	1.0							

Chapter 3

The Ramping Polytope and Cut Generation for the Unit Commitment Problem

This chapter and Appendix C are based on a manuscript prepared for publication by Ben Knueven, Jim Ostrowski, and Jianhui Wang:

> Knueven, B., Ostrowski, J., and Wang, J. (2017). The Ramping Polytope and Cut Generation for the Unit Commitment Problem. *Submitted*.

Authors Ostrowski and Wang discovered the extended formulation. Authors Knueven and Ostrowski developed the proof of Theorem 3.1. Author Knueven proposed and developed the cut-generation procedure. Author Knueven drafted the manuscript and conducted all computational experiments. Author Ostrowski edited the manuscript. A preprint of this paper is available at http://www.optimization-online.org/DB_FILE/2015/09/5099.pdf.

In this chapter we present a perfect formulation for a single generator in the unit commitment problem, inspired by the dynamic programming approach taken by Frangioni and Gentile. This generator can have characteristics such as ramp up/down constraints, time-dependent start-up costs, and start-up/shut-down limits. To develop this perfect formulation we extend the result of Balas on unions of polyhedra to present a framework allowing for flexible combinations of polyhedra using indicator variables. We use this perfect formulation to create a cut-generating linear program, similar in spirit to lift-and-project cuts, and demonstrate computational efficacy of these cuts in a utility-scale unit commitment problem.

3.1 Introduction

The unit commitment problem (UC) is that of scheduling generators to meet power demand, and has been one of the great successes of mixed-integer programming models. The Midwest Independent Transmission System Operator (MISO), recipient of the Edelman Award in 2011, reports annual savings of over $500 million by using integer programming to optimize UC in place of Lagrangian relaxation [16]. Because of the scales involved, a 1% savings in energy markets results in translates to a $10 billion annual savings [117].

The unit commitment problem is nearly decomposable as generators are only linked through the demand constraint. Therefore most improvements in unit commitment models are a result of studying the properties of an individual generator's feasible region. Frangioni and Gentile [46] provide a dynamic programming model for an individual generator with ramping constraints. Inspired by their dynamic programming model, we construct a compact extended formulation for a single generator, which can be used to model generators within a unit commitment MIP model. During the drafting of this chapter, we learned that Frangioni and Gentile independently discovered a similar perfect formulation for a single thermal generator [48, 49]. The formulation developed can be used to model any properties of a generator that are polyhedrally representable when the commitment status is fixed. We use this extended formulation to create a cut-generating linear program that can be used to strengthen the linear programming (LP) relaxation for MIP formulations of UC and/or as a callback in the MIP solver.

The rest of the chapter is outlined as follows. In Section 3.2 we review the current state of the unit commitment problem, including the typical 3-binary formulation used in state-of-the-art models, and introduce the extended formulation. In Appendix C, we prove the integrality of this extended formulation by revisiting a classic result of Balas [6, 7], and do so in more general terms than we strictly require. In Section 3.3 we use the results of the preceding sections to develop a cut-generating linear program for a ramping-constrained generator, and we present computational experiments based on a utility-scale unit commitment problem. Finally, in Section 3.4 we draw conclusions and discuss possible directions for future research.

3.2 The Unit Commitment Problem

We begin by providing an overview of the unit commitment problem. For a set of generators $\mathcal{G}$ and T time steps, we formulate the unit commitment problem as follows:

$$\min \sum_{g \in \mathcal{G}} c^g(p^g) \tag{3.1a}$$

$$\text{subject to } \sum_{g \in \mathcal{G}} p_t^g \geq L_t, \ \forall t \in [T] \tag{3.1b}$$

$$\sum_{g \in \mathcal{G}} \bar{p}_t^g \geq L_t + R_t, \ \forall t \in [T] \tag{3.1c}$$

$$p^g, \bar{p}^g \in \Pi^g \subset \mathbb{R}^T, \ \forall g \in \mathcal{G}, \tag{3.1d}$$

p^g is the power output vector of generator g, $c^g(p^g)$ is the cost of the vector p^g, and $\bar{p}_t^g$ is the maximum power available from generator g at time t. L_t is the electricity load at time t, while R_t is the spinning reserve requirement at time t. For convenience let $[T] := \{1, \ldots, T\}$. Π^g represents the often non-convex technical constraints on production and commitment of generator g, such as minimum up/down times, ramping rates, time-dependent start-up costs, etc. As mentioned above, most research in improving unit commitment models has focused improving the modeling of individual generators, i.e., (3.1d) above. The justification for this line of research is that tighter MIP formulations for Π^g increase the linear programming bound for (3.1), which will in turn decrease the enumeration necessary to solve UC.

Most of the literature modeling individual generators followed Garver's [56] general structure, using three different types of binary variables to describe the status of a generator at a given time: one variable indicating if the generator is on, another indicating if is turned on, and the last indicating if generator is turned off. Models of this type are referred to as 3-binary models, or *3-bin* models. An alternative 1-binary model (e.g., see [17]), or *1-bin* model, considers the variables indicating if the generator is turned on/off as superfluous, rewriting each constraint using only variables that represent if the generator is on at a given time period. The hope in this reduction is that fewer binary variables will lead to smaller branch-and-bound trees and smaller computation times. However, moving to a smaller formulation comes with a cost of weaker inequalities. A convex hull description for a simplified generator using the 1-bin formulation is given by Lee et al. [86], showing that the convex hull has exponentially many constraints. Yet, the same simplified generator's production region has a linearly-sized convex hull when using the 3-bin model [94, 133].

The simplified generator considered by Rajan and Takriti [133] only models on/off status, minimum/maximum power, and minimum up/down times. For this reason, several recent results have strengthened the 3-bin model with additional generator characteristics. A common extension is to the case when *ramping constraints* are considered. Ramping constraints represent the fact that generators, in general, cannot vary their power output dramatically from one time period to the next. Polynomial classes of strengthening inequalities for the 3-bin model with ramping are given by Ostrowski et al. [119]; Damcı-Kurt et al. [27] build on this by providing exponential classes of such strengthening inequalities along with a polynomial separation algorithm. Additionally, Damcı-Kurt et al. [27] provide a convex hull description for the ramp-up and ramp-down polytopes in two time periods. Pan and Guan [125] extend this by intersecting these two polytopes into an integrated ramping polytope for three time periods.

A convex hull description for the 3-bin model with the addition of start-up and shut-down power is proved by Gentile et al. [58]. The same authors extend this in Morales-España et al. [109] by separating power from energy (often assumed to be the same since almost all UC models operate on a one-hour time interval). This allows for the modeling of a generator's output below economic minimum when starting up and shutting down, while still maintaining integrality.

Our model moves away from the standard 3-bin polytope by considering on-off intervals. To demonstrate the relationship between the proposed formulation and the classical formulations, we first review the standard 3-bin model typically used to represent Π^g and $c^g(\cdot)$. Then we turn our focus toward the dynamic programming method for optimizing over a single generator laid out by Frangioni and Gentile [46]. Indeed using their dynamic programming procedure allows one to optimize a convex function over the unit commitment polytope (with ramping constraints) in polynomial time. It should not be surprising then to find polynomial-sized extended formulations for Π^g (although this is by no means guaranteed, see [137]). Additionally, the formulation derived can be trivially extended to model other generator characteristics, provided the constraints on the generator are represented with a polytope when the commitment status is fixed.

3.2.1 3-bin Formulation

We now describe the typical 3-bin formulation for the feasible region Π^g with cost function $c^g(\cdot)$. Consider the binary vectors $u^g, v^g, w^g \in \{0,1\}^T$, where u^g_t is the commitment status of the generator at time t, v^g_t indicates if the generator was started up at time t, and w^g_t indicates if the generator was shut down at time t. Suppose UT and DT are the minimum up and down time for the generator.

We first consider the logical constraints [56]:

$$u_t^g - u_{t-1}^g = v_t^g - w_t^g, \ \forall t \in [T], \tag{3.2}$$

and the minimum up/down time constraints [133]:

$$\sum_{i=t-UT+1}^{t} v_i^g \leq u_t^g, \ \forall t \in [UT^g, T] \tag{3.3}$$

$$\sum_{i=t-DT+1}^{t} w_i^g \leq 1 - u_t^g, \ \forall t \in [DT^g, T]. \tag{3.4}$$

Rajan and Takriti [133] showed that (3.2 – 3.4) along with the variable bound constraints give a convex hull description for the minimum up/down time polytope.

Next we consider constraints on the generation limits. Let $\underline{P}$ and $\overline{P}$ represent the minimum and maximum feasible power output when on, RD and RU represent the maximum ramp-down and ramp-up rates, and SD and SU represent the maximum shut-down and start-up levels. First we note that when a generator is on it must be operating within its specified limits

$$\underline{P}^g u_t^g \leq p_t^g \leq \bar{p}_t^g \leq \overline{P}^g u_t^g, \ \forall t \in [T]. \tag{3.5}$$

We note if $RU^g, RD^g \geq (\overline{P}^g - \underline{P}^g)$ and $SU^g, SD^g \geq \overline{P}^g$, that is, there are no real ramping and start-up shut-down constraints, then the formulation given by (3.2 – 3.5) is perfect for this simple generator. However, most generators are not so simple, and have ramp-up constraints:

$$\bar{p}_t^g - p_{t-1}^g \leq RU^g u_{t-1}^g + SU^g v_t^g, \ \forall t \in [T], \tag{3.6}$$

and ramp-down constraints:

$$\bar{p}_{t-1}^g - p_t^g \leq RD^g u_t^g + SD^g w_t^g, \ \forall t \in [T]. \tag{3.7}$$

For reference later, define:

$$\Pi^g_{\text{3-bin}} := \{(p^g, \bar{p}^g, u^g, v^g, w^g) \in \mathbb{R}^{5T}_+ | \ (3.2 - 3.7); (u^g, v^g, w^g) \in \{0,1\}^{3T}\} \tag{3.8}$$

and

$$^{R}\Pi^{g}_{\text{3-bin}} := \{(p^g, \bar{p}^g, u^g, v^g, w^g) \in \mathbb{R}^{5T}_{+} |\ (3.2 - 3.7); (u^g, v^g, w^g) \in [0,1]^{3T}\}. \tag{3.9}$$

That is, $\Pi^{g}_{\text{3-bin}}$ is the feasible set for the technical constraints for generator g, and $^{R}\Pi^{g}_{\text{3-bin}}$ is its continuous relaxation. We will colloquially refer to $\Pi^{g}_{\text{3-bin}}$ and $^{R}\Pi^{g}_{\text{3-bin}}$ as being in "3-bin space", dropping the g when it is implied by context.

Now we consider the cost function $c^g(\cdot)$. Typically $c^g(p^g) = c^g_f(u^g) + \sum_{t\in[T]} c^g_p(p^g_t)$, where $c^g_p(\cdot)$ is convex and either quadratic or piecewise linear in the power output, and $c_f(\cdot)$ is the fixed commitment costs and start-up/shut-down costs, and as such is a function of the indicator variables. First, we consider $c^g_p(\cdot)$. We assume that $c^g_p(\cdot)$ is convex and piecewise linear where $^{1}\overline{P}^{g}, \ldots, {}^{L}\overline{P}^{g}$ represent the upper breakpoints for power available at marginal costs $^{1}c^g, \ldots, {}^{L}c^g$ with $^{1}c^g < \ldots < {}^{L}c^g$. Define $^{0}\overline{P}^{g} = 0$. We use the standard convex piecewise formulation by introducing new variables $^{l}p^g_t$, representing the power generator g produces at time t at marginal cost $^{l}c^g$, along with the constraints

$$0 \le {}^{l}p^g_t \le {}^{l}\overline{P}^{g} - {}^{l-1}\overline{P}^{g},\ \forall l \in [L], \forall t \in [T] \tag{3.10a}$$

$$p^g_t = \sum_{l=1}^{L} {}^{l}p^g_t,\ \forall t \in [T]. \tag{3.10b}$$

We can then represent $c^g_p(\cdot)$ linearly as $\sum_{t\in[T]}\sum_{l\in[L]} {}^{l}c^g\, {}^{l}p^g_t$. Now consider $c^g_f(\cdot)$. Typically the start-up cost is an increasing function of how long the generator has been off. For simplicity we will only consider two start-up types, hot (H) and cold (C). A start-up is said to be hot if the generator has been off for less than DT^g_C time periods. We formulate the start-up costs as in Morales-España et al. [111]; namely, let $^{H}\delta^g_t, {}^{C}\delta^g_t \in \{0,1\}$ represent a hot and cold start-up, respectively. Then we may write $^{H}\delta^g_t, {}^{C}\delta^g_t$ in terms of the start-up and shut-down variables v^g_t, w^g_t:

$${}^{H}\delta^g_t \le \sum_{i=DT}^{DT^g_C - 1} w^g_{t-i},\ \forall t \in [DT^g_C, T] \tag{3.11a}$$

$${}^{H}\delta^g_t + {}^{C}\delta^g_t = v^g_t,\ \forall t \in [T]. \tag{3.11b}$$

Thus, if $^{U}c^g$ is the fixed cost of running the generator, and $^{D}c^g$ is the cost of shutting down the generator, then we can represent $c^g_f(\cdot)$ linearly as $\sum_{t\in[T]}({}^{C}c^g\, {}^{C}\delta^g_t + {}^{H}c^g\, {}^{H}\delta^g_t + {}^{U}c^g u^g_t + {}^{D}c^g w^g_t)$.

If the ramping constraints (3.6) and (3.7) are irredundant, then it is well known that $\text{conv}(\Pi^{g}_{\text{3-bin}}) \neq {}^{R}\Pi^{g}_{\text{3-bin}}$ (where $\text{conv}(S)$ is the convex hull of the set S). Recently, Damcı-Kurt

et al. [27] characterized separately the ramp-up and ramp-down polytopes for when $T = 2$, and Pan and Guan [125] fully characterized $\text{conv}(\Pi^g_{\text{3-bin}})$ for $T = 3$. In the next section we will develop a new extended formulation for Π^g, which can be used to generate valid inequalities for $\text{conv}(\Pi^g_{\text{3-bin}})$.

3.2.2 The Feasible Dispatch Polytope

The feasible dispatch polytope describes the possible generator outputs given that the generator's on/off status has been fixed. Let $D^{[a,b]} \subset \mathbb{R}^{2T}$ represent the set of all feasible production schedules assuming that the generator is only (and continuously) on during the time interval $[a, b]$. For any $(p^{[a,b]}, \bar{p}^{[a,b]}) \in D^{[a,b]}$, $p_t^{[a,b]}$ represents the power produced by the generator at time t (note that $p_t^{[a,b]} = 0$ for all t not in the interval $[a, b]$), and $\bar{p}_t^{[a,b]}$ represents the maximum power available at time t. We can write $D^{[a,b]}$ as

$$D^{[a,b]} = \{(p^{[a,b]}, \bar{p}^{[a,b]}) \in \mathbb{R}^{2T}_+ | A^{[a,b]} p^{[a,b]} + \bar{A}^{[a,b]} \bar{p}^{[a,b]} \leq \bar{b}^{[a,b]}\} \tag{3.12}$$

for $A^{[a,b]}, \bar{A}^{[a,b]} \in \mathbb{R}^{m \times T}$ and $\bar{b}^{[a,b]} \in \mathbb{R}^m$. For our purposes, $D^{[a,b]}$ is defined by max/min power and ramping constraints, and is obviously bounded. The methods described in this chapter then can be used to extend $D^{[a,b]}$ to accommodate any number of services so long as $D^{[a,b]}$ remains a bounded polyhedron.

To demonstrate, consider the most common description of $D^{[a,b]}$ found in the power systems literature, which deals with the following types of constraints: minimum/maximum output, maximum ramping, and start-up/shut-down levels. The constraints defining the polytope $D^{[a,b]}_{typical}$ are:

$$p_t^{[a,b]} \leq 0 \qquad \forall t < a \text{ and } t > b \tag{3.13a}$$

$$\bar{p}_t^{[a,b]} \leq 0 \qquad \forall t < a \text{ and } t > b \tag{3.13b}$$

$$-p_t^{[a,b]} \leq -\underline{P} \qquad \forall t \in [a, b] \tag{3.13c}$$

$$p_t^{[a,b]} \leq \bar{p}_t^{[a,b]} \qquad \forall t \in [a, b] \tag{3.13d}$$

$$\bar{p}_t^{[a,b]} \leq \min(\overline{P}, SU + (t-a)RU, SD + (b-t)RD) \qquad \forall t \in [a, b] \tag{3.13e}$$

$$\bar{p}_t^{[a,b]} - p_{t-1}^{[a,b]} \leq \min(RU, SD + (b-t)RD - \underline{P}) \qquad \forall t \in [a+1, b] \tag{3.13f}$$

$$\bar{p}_{t-1}^{[a,b]} - p_t^{[a,b]} \leq \min(RD, SU + (t-a)RU - \underline{P}) \qquad \forall t \in [a+1, b]. \tag{3.13g}$$

Constraints (3.13a) and (3.13b) specify that the generator does not output power nor provide reserves while off; (3.13c) specifies the minimum level of power output when the generator is on. (3.13d) ensures the power available is at least the power committed. Constraint (3.13e) enforces the upper bound on the power output at time t. This ensures the generator does not produce more power than its maximum output $\overline{P}$, the power level it could ramp up to by time t $(SU+(t-a)RU)$, or ramp down from at time t $(SD+(b-t)RD)$ to reach shut-down status. The ramp up constraint (3.13f) ensures the power jump between times $t-1$ and t is no more than RU or that which we could ramp back down to in the remaining time $(SU+(b-t)RD-\underline{P})$. The ramp down constraints (3.13g) work symmetrically.

We will let $\mathcal{T}$ be the set of all feasible continuous operating intervals for the generator. Recalling UT and DT are the minimum up and down time for the generator, $\mathcal{T}$ contains all intervals $[a,b]$ where $1 \leq a \leq a+UT \leq b \leq T$. $\mathcal{T}$ also contains cases when the generator has been turned on prior to time one and cases where the generator will be on past time T. To account for this, we let the interval $[0,b]$ represent cases where the generator was already on before the planning period and is turned off at time b. It is not necessary for $b+1$ to be larger than UT. Similarly, we let the interval $[a,T+1]$ represent the case where the generator continues to be on after the planning period, where the actual shut-down time is undetermined. Note all polytopes $D^{[a,b]}_{typical}$ are nonempty for $[a,b] \in \mathcal{T}$. Frangioni and Gentile [46] develop a dynamic-programming approach for scheduling a single generator in polynomial time by combining the polytopes $D^{[a,b]}_{typical}$ such that the intervals only overlap in feasible combinations. We will use the polytopes $D^{[a,b]}_{typical}$ in a similar fashion to develop an extended formulation for the ramping polytope.

3.2.3 Packing Dispatch Polytopes

To develop the extended formulation, we construct an interval graph from $\mathcal{T}$, where two intervals $[a,b]$, $[c,d]$ are defined to overlap if $[a,b+DT] \cap [c,d+DT] \neq \emptyset$. That is, $G=(V,E)$ has $V=\mathcal{T}$ and edges between two vertices if they overlap, is an interval graph by construction, and hence G is a line graph. We now consider packing the vertices of G, that is, selecting a subset of $V_P \subseteq V$ such that for any $u,v \in V_P$, $(u,v) \notin E$. If we use variables $\gamma \in \{0,1\}^{|\mathcal{T}|}$ to indicate whether a vertex (interval) is in the packing or not, then it is well known (since G is a line graph) that the clique inequalities (along with non-negativity) give a convex hull description of the vertex packing

problem. That is, the vertices of

$$\Gamma = \begin{cases} \sum_{\{[a,b]\in\mathcal{T} \mid t\in[a,b+DT]\}} \gamma_{[a,b]} \leq 1 & t \in [T] \\ \gamma_{[a,b]} \geq 0 & \forall [a,b] \in \mathcal{T} \end{cases} \tag{3.14}$$

are binary and represent all feasible vertex packings. Using the dispatch polytopes developed in Section 3.2.2, we can write down an extended formulation for a ramping-constrained generator.

Theorem 3.1. *The polytope*

$$D := \begin{cases} A^{[a,b]}p^{[a,b]} + \bar{A}^{[a,b]}\bar{p}^{[a,b]} \leq \gamma_{[a,b]}\bar{b}^{[a,b]} & \forall [a,b] \in \mathcal{T} & \text{(3.15a)} \\ \sum_{[a,b]\in\mathcal{T}} p^{[a,b]} = p & & \text{(3.15b)} \\ \sum_{[a,b]\in\mathcal{T}} \bar{p}^{[a,b]} = \bar{p} & & \text{(3.15c)} \\ (p^{[a,b]}, \bar{p}^{[a,b]}) \in \mathbb{R}^{2T}_{+} & \forall [a,b] \in \mathcal{T} & \text{(3.15d)} \\ \sum_{\{[a,b]\in\mathcal{T} \mid t\in[a,b+DT]\}} \gamma_{[a,b]} \leq 1 & t \in [T] & \text{(3.15e)} \\ \gamma_{[a,b]} \geq 0 & \forall [a,b] \in \mathcal{T} & \text{(3.15f)} \end{cases}$$

is a compact (polynomial-sized in T) formulation for a ramping-constrained generator, and the vertices of D have integer γ.

Proof. Proof. See Appendix C. □

Remark 3.1. Not dispatching the generator in time period $[0, T+1]$ corresponds to having $\gamma_{[a,b]} = 0$ for all $[a,b] \in \mathcal{T}$.

Linear generation costs $c \in \mathbb{R}^T$ and fixed start-up (and shut-down) costs $w \in \mathbb{R}^{|\mathcal{T}|}$ can be modeled by optimizing the linear function $c^\top p + w^\top \gamma$ over D. This formulation does not concern itself with time-dependent start-up costs. These can be easily added by considering additional indicator variables $\zeta_{[c,d]}$ which represent the generator being off from time c to d, and construct a set of feasible off intervals $\mathcal{T}'$ similar to the construction of $\mathcal{T}$. Recall ${}^{C}c$ and ${}^{H}c$ are the cost of a cold and hot start, respectively. When ${}^{H}c \geq {}^{C}c/2$, we can replace (3.15e) above with

$$\sum_{\{[a,b]\in\mathcal{T} \mid t\in[a,b+DT]\}} \gamma_{[a,b]} + \sum_{\{[c,d]\in\mathcal{T}' \mid t\in[c+DT,d]\}} \zeta_{[c,d]} = 1 \qquad t \in [T] \tag{3.16a}$$

$$\zeta_{[c,d]} \geq 0 \qquad \forall [c,d] \in \mathcal{T}', \tag{3.16b}$$

and the $\zeta_{[c,d]}$ variables are in the objective function with the appropriate objective value.

However, when ${}^{H}c < {}^{C}c/2$, a formulation discovered by Frangioni and Gentile [48, 49] must be considered. It uses the following shortest path formulation in place of the packing formulation in (3.15e) and (3.15f) above

$$\sum_{\{[c,d]\in\mathcal{T}' \mid t=d+1\}} \zeta_{[c,d]} = \sum_{\{[a,b]\in\mathcal{T} \mid t=a\}} \gamma_{[a,b]} \qquad t \in [T] \tag{3.17a}$$

$$\sum_{\{[a,b]\in\mathcal{T} \mid t=b+1\}} \gamma_{[a,b]} = \sum_{\{[c,d]\in\mathcal{T}' \mid t=c\}} \zeta_{[c,d]} \qquad t \in [T] \tag{3.17b}$$

$$\sum_{\{[a,b]\in\mathcal{T} \mid a=0\}} \gamma_{[a,b]} + \sum_{\{[c,d]\in\mathcal{T}' \mid c=0\}} \zeta_{[c,d]} = 1 \tag{3.17c}$$

$$\sum_{\{[a,b]\in\mathcal{T} \mid b=T+1\}} \gamma_{[a,b]} + \sum_{\{[c,d]\in\mathcal{T}' \mid d=T+1\}} \zeta_{[c,d]} = 1 \tag{3.17d}$$

$$\gamma_{[a,b]} \geq 0,\ \forall [a,b] \in \mathcal{T},\ \zeta_{[c,d]} \geq 0,\ \forall [c,d] \in \mathcal{T}'. \tag{3.17e}$$

Call the resulting polytope D', that is, equations (3.15a - 3.15d) with (3.17). We note that Frangioni and Gentile [49] provide a proof of the integrality of D', building it up one dispatch polytope at a time, which also proves Theorem 3.1. It essentially relies on the fact that each of the combined polytopes are integer in their respective variables, and share only one variable with the other polytopes. In a similar fashion the results outlined in Appendix C prove the integrality of D' in the γ, ζ variables. Both proofs rely on the underlying integrality of the polytopes relating the indicator variables; the one presented in this chapter also provides an interesting geometric interpretation of the result, whereas that in Frangioni and Gentile [49] is a bit more generalizable.

Both convex hull descriptions are large, and are unlikely to be computationally effective within the problem (3.1). Preliminary computational experiments in Frangioni and Gentile [49] bear this out. As such, instead of using (3.15) directly to represent Π^g in the unit commitment problem (3.1), we will use (3.15) to develop a procedure for generating cuts based on the polytope D similar in spirit to lift-and-project cuts [8].

3.3 A Cutting-Plane Procedure for the 3-bin Formulation

The basic logic of the proposed approach is as follows. Let D^g be the feasible dispatch polytope for generator g. The LP relaxation for (3.1) when the extended formulation is used to represent

each generator is

$$
\begin{aligned}
{}^{LP}x^*_{EF} = \min \ & \sum_{g\in\mathcal{G}} c^g(p^g) && \text{(3.18a)}\\
\text{subject to } & \sum_{g\in\mathcal{G}} p^g_t \geq L_t, \ \forall t \in [T] && \text{(3.18b)}\\
& \sum_{g\in\mathcal{G}} \bar{p}^g_t \geq L_t + R_t, \ \forall t \in [T] && \text{(3.18c)}\\
& (p^g, \bar{p}^g) \in D^g, \ \forall g \in \mathcal{G}. && \text{(3.18d)}
\end{aligned}
$$

On the other hand, define the LP relaxation for (3.1) with the typical 3-bin formulation as

$$
\begin{aligned}
{}^{LP}x^*_{\text{3-bin}} = \min \ & \sum_{g\in\mathcal{G}} c^g(p^g) && \text{(3.19a)}\\
\text{subject to } & \sum_{g\in\mathcal{G}} p^g_t \geq L_t, \ \forall t \in [T] && \text{(3.19b)}\\
& \sum_{g\in\mathcal{G}} \bar{p}^g_t \geq L_t + R_t, \ \forall t \in [T] && \text{(3.19c)}\\
& (p^g, \bar{p}^g) \in {}^{R}\Pi^g_{\text{3-bin}}, \ \forall g \in \mathcal{G}. && \text{(3.19d)}
\end{aligned}
$$

Because the extended formulation is at least as tight as 3-bin, we have ${}^{LP}x^*_{EF} \geq {}^{LP}x^*_{\text{3-bin}}$. However, as mentioned above, because representing D^g requires $\mathcal{O}(T^3)$ variables whereas ${}^{R}\Pi^g_{\text{3-bin}}$ needs only $\mathcal{O}(T)$ variables, the problem (3.18) is likely to be much more computationally difficult than (3.19). In the MIP context, this not only slows down the root-node solve time, but also subsequent node resolves in the branch-and-bound tree.

We propose a cutting-plane procedure attempts to ameliorate the computational issues of solving (3.18) while still maintaining the strength of its LP bound. In particular, given a solution to (3.19), for each $g \in \mathcal{G}$ we can lift generator schedules $(p^{g*}, \bar{p}^{g*}) \in {}^{R}\Pi^g_{\text{3-bin}}$ to the "D^g-space." If $(p^{g*}, \bar{p}^{g*}) \in D^g$, then we do nothing. On the other hand, if $(p^{g*}, \bar{p}^{g*}) \notin D^g$, we can calculate a separating cut, project the cut back into 3-bin space, add it to the problem (3.19), and then resolve. Repeating this process iteratively until $(p^{g*}, \bar{p}^{g*}) \in D^g$ for all $g \in \mathcal{G}$ allows us to calculate ${}^{LP}x^*_{EF}$ while decomposing the difficult constraints D^g into individual easier separation subproblems for each $g \in \mathcal{G}$. Further, we can stop at any point and possibly obtain a better LP bound than ${}^{LP}x^*_{\text{3-bin}}$. That is, for each $g \in \mathcal{G}$ let C^g be the feasible region for a (possibly empty) set of cuts generated

from D^g. Consider the following linear program:

$$
\begin{aligned}
{}^{LP}x^*_{\text{3-bin}+C} = \min \ & \sum_{g\in\mathcal{G}} c^g(p^g) && \text{(3.20a)}\\
\text{subject to } & \sum_{g\in\mathcal{G}} p^g_t \geq L_t, \ \forall t \in [T] && \text{(3.20b)}\\
& \sum_{g\in\mathcal{G}} \bar{p}^g_t \geq L_t + R_t, \ \forall t \in [T] && \text{(3.20c)}\\
& (p^g, \bar{p}^g) \in {}^{R}\Pi^g_{\text{3-bin}} \cap C^g, \ \forall g \in \mathcal{G}. && \text{(3.20d)}
\end{aligned}
$$

Then we have ${}^{LP}x^*_{EF} \geq {}^{LP}x^*_{\text{3-bin}+C} \geq {}^{LP}x^*_{\text{3-bin}}$, where we achieve equality on the left if we add every possible separating cut, and we have equality on the right if we add no cuts. As is well known, in practice is it often not desirable to add every possible cut, so we add cuts heuristically (in this work, we do only one round of cuts for a given LP relaxation). Finally, note that the discussion above not only applies to the root node of the branch-and-bound tree, but also at any subsequent node subproblem.

3.3.1 From Dispatch Polytope Space to 3-bin Space

Recalling the typical 3-bin formulation described in Section 3.2.1 and the new extended formulation developed in Sections 3.2.2 and 3.2.3, we see how these formulations can be "connected" through a linear transformation, which will be the basis for our cut-generation routine.

Dropping the superscript g for a moment to focus on one generator, recall equation (3.15). Although it is clear from the formulation of D how to project it on to the space of $(p, \bar{p})$ variables, by using a linear transformation we can project D into 3-bin space, that is, the space of the $(p, \bar{p}, u, v, w)$ variables from Section 3.2.1. Of course, p and $\bar{p}$ remain the same, and we can link γ and (u, v, w) as follows:

$$
\begin{aligned}
\sum_{\{[a,b]\in\mathcal{T} \ | \ t\in[a,b]\}} \gamma_{[a,b]} &= u_t && t \in [T] && \text{(3.21a)}\\
\sum_{\{[a,b]\in\mathcal{T} \ | \ t=a\}} \gamma_{[a,b]} &= v_t && t \in [T] && \text{(3.21b)}\\
\sum_{\{[a,b]\in\mathcal{T} \ | \ t=b+1\}} \gamma_{[a,b]} &= w_t && t \in [T]. && \text{(3.21c)}
\end{aligned}
$$

Notice by adding the constraints (3.21) to the formulation of D (3.15), for a given 3-bin solution $(p^*, \bar{p}^*, u^*, v^*, w^*) \in {}^R\Pi_{\text{3-bin}}$, either the system of equations defined by (3.15), (3.21), $p = p^*$, $\bar{p} = \bar{p}^*$, $u = u^*$, $v = v^*$, and $w = w^*$, will be feasible, in which case this 3-bin solution is in the ramping polytope, or this system of equations will not be feasible, in which case this 3-bin solution is not in the ramping polytope. In the latter case we can use the Farkas certificate for the system of equations to generate a cut for the 3-bin space which cuts-off this infeasible solution. Our cut-generating linear program picks, in some sense, the best such infeasibility certificate so as to get the deepest cut.

3.3.2 A Cut-Generating Linear Program

For ease we will consider the dual form of the cut-generating LP, which is derived from (3.15) and (3.21) above. Let e be the appropriately sized vector of 1's and suppose $z \in \mathbb{R}_+$. Let $(p^*, \bar{p}^*, u^*, v^*, w^*)$ be a solution vector in 3-bin space. Consider the following linear program:

$$z^* = \min\ z \tag{3.22a}$$

subject to

$$\pi \qquad A^{[a,b]}p^{[a,b]} + \bar{A}^{[a,b]}\bar{p}^{[a,b]} \leq \gamma_{[a,b]}\bar{b}^{[a,b]} + ze \qquad \forall [a,b] \in \mathcal{T} \tag{3.22b}$$

$$\delta \qquad \sum_{\{[a,b]\in\mathcal{T}\ \mid\ t\in[a,b+DT]\}} \gamma_{[a,b]} \leq 1 + z \qquad t \in [T] \tag{3.22c}$$

$$\epsilon \qquad \sum_{[a,b]\in\mathcal{T}} p^{[a,b]} = p^* \tag{3.22d}$$

$$\mu \qquad \sum_{[a,b]\in\mathcal{T}} \bar{p}^{[a,b]} = \bar{p}^* \tag{3.22e}$$

$$\xi \qquad \sum_{\{[a,b]\in\mathcal{T}\ \mid\ t\in[a,b]\}} \gamma_{[a,b]} = u_t^* \qquad t \in [T] \tag{3.22f}$$

$$\alpha \qquad \sum_{\{[a,b]\in\mathcal{T}\ \mid\ t=a\}} \gamma_{[a,b]} = v_t^* \qquad t \in [T] \tag{3.22g}$$

$$\sigma \qquad \sum_{\{[a,b]\in\mathcal{T}\ \mid\ t=b+1\}} \gamma_{[a,b]} = w_t^* \qquad t \in [T] \tag{3.22h}$$

$$z \in \mathbb{R}_+;\ \ p^{[a,b]}, \bar{p}^{[a,b]} \in \mathbb{R}_+^T, \gamma_{[a,b]} \in \mathbb{R}_+, \qquad \forall [a,b] \in \mathcal{T}, \tag{3.22i}$$

where $\pi \in \mathbb{R}_-^{m|\mathcal{T}|}$ is the set of dual variables for constraints (3.22b), $\delta \in \mathbb{R}_-^T$ is the set of dual variables for (3.22c), and $\epsilon, \mu, \xi, \alpha, \sigma \in \mathbb{R}^T$ are the sets of dual variables for constraints (3.22d -

3.22h), respectively. We observe if z^* is 0, then $(p^*, \bar{p}^*)$ is a feasible solution to D, and if not, we can use the optimal dual vector to cut off the 3-bin solution $(p^*, \bar{p}^*, u^*, v^*, w^*) \in {}^R\Pi_{\text{3-bin}}$. To demonstrate, suppose $z^* > 0$ and we have an optimal dual vector $\pi^*, \delta^*, \epsilon^*, \mu^*, \xi^*, \alpha^*, \sigma^*$. Then by strong duality $z^* = (\delta^*)^T e + (\epsilon^*)^T p^* + (\mu^*)^T \bar{p}^* + (\xi^*)^T u^* + (\alpha^*)^T v^* + (\sigma^*)^T w^* > 0$, and so the cut $(\delta^*)^T e + (\epsilon^*)^T p + (\mu^*)^T \bar{p} + (\xi^*)^T u + (\alpha^*)^T v + (\sigma^*)^T w \leq 0$ cuts off the solution $(p^*, \bar{p}^*, u^*, v^*, w^*)$ in 3-bin space (that is, it is a valid separating hyperplane between $(p^*, \bar{p}^*, u^*, v^*, w^*)$ and $\text{conv}(\Pi_{\text{3-bin}})$).

We maximize the depth of the cut by choosing the optimal such cut, with respect to the 1-norm normalization, instead of any dual feasible solution to (3.22) [8]. In particular, note in the dual of (3.22), the constraint associated with z is $-e^T\delta - e^T\pi \leq 1$. When the proposed solution $(p^*, \bar{p}^*, u^*, v^*, w^*)$ is infeasible for ${}^R\Pi_{\text{3-bin}}$, this limits the 1-norm of these otherwise unbounded rays.

3.3.3 Implementation

To test the efficacy of these cuts, we implement them as a callback for a utility-scale unit commitment problem based on the set of FERC generators. These consist of two sets of generators, a "summer" and "winter" set of generators, which are based on market data provided from the PJM Interconnection and other sources [84]. We use the standard 3-bin formulation for the master unit commitment MIP, as discussed in Section 3.2.1, that is:

$$\min \sum_{g\in\mathcal{G}} \sum_{t\in[T]} \left(\sum_{l\in[L]} ({}^l c^g \, {}^l p_t^g) + {}^C c^g \, {}^C \delta_t^g + {}^H c^g \, {}^H \delta_t^g + {}^R c^g u_t^g + {}^D c^g w_t^g \right) \tag{3.23a}$$

$$\text{s.t.} \sum_{g\in\mathcal{G}} p_t^g \geq L_t, \ \forall t \in [T] \tag{3.23b}$$

$$\sum_{g\in\mathcal{G}} \bar{p}_t^g \geq L_t + R_t, \ \forall t \in [T] \tag{3.23c}$$

$$(3.10), (3.11) \ \forall g \in \mathcal{G}, \tag{3.23d}$$

$$(p^g, \bar{p}^g, u^g, v^g, w^g) \in \Pi_{\text{3bin}}^g, \ \forall g \in \mathcal{G}. \tag{3.23e}$$

We only consider cuts on a subset of the generators, namely for those that have irredundant ramping constraints while operating and those that have a minimum run time of at least 2. That is, we consider cuts on $\mathcal{G}^C := \{g \in \mathcal{G} \mid (\overline{P}^g - \underline{P}^g) > \min\{RD^g, RU^g\} \text{ and } UT^g \geq 2\}$. We do this because the 3-bin formulation is tight for generators with no ramping constraints and the problem (3.22) becomes impractically large for generators with $UT^g = 1$.

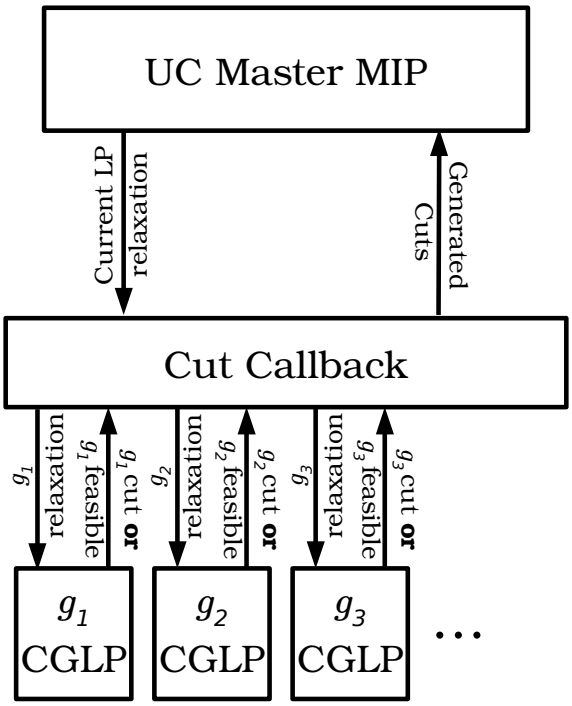


Figure 3.1: Visualization of the cut-generation procedure

The cuts are implemented in a callback, namely, given the current LP relaxation for (3.23), for each generator in $\mathcal{G}^C$ with fractional status variables, we use (3.22) to determine if $(p^*, \bar{p}^*, u^*, v^*, w^*) \in \text{conv}(\Pi_{\text{3-bin}})$, and if not we add the violated ramping inequality given by the optimal dual solution of (3.22). We generate the cuts using the "bundling" approach of Balas et al. [8], that is, at the current master LP relaxation we try to generate a cut for each $g \in \mathcal{G}^C$ and give these to the solver together. This is visualized in Figure 3.1, where we see for each unit $g \in \mathcal{G}^C$ the cut-generation LP (CGLP) is an independent problem. For a given LP relaxation to the UC Master MIP (3.23), we separate the solution by generator g, and then check the feasibility of this relaxed solution in D^g by solving the CGLP (3.22). If we find the relaxed solution infeasible, we pass the generated cut to the UC Master MIP, along with the cuts generated for other units in $\mathcal{G}^C$. Though we do not do so for this experiment, it would be trivial to parallelize the cut-generation procedure.

Next, we discuss some computational enhancements to this general outline. First, to mitigate numerical issues, we only add cuts for which the solution to (3.22), z^*, is greater than 10^{-2}. Second, after the first round of calls to the cut-generating LPs we use the existing basis information if and only if we did not generate a cut from it in the previous pass. The intuition is that if we did not

generate a cut from this generator previously then the current 3-bin vector is probably close to the previous one. On the other hand, if we did generate a cut, then (we hope) the current 3-bin vector is far away from the previous one, so we discard the previous basis information. Third, we make an enhancement based on symmetry by observing that if $g_1, g_2 \in \mathcal{G}^C$ have identical parameters (including initial conditions), a cut generated for g_1 is valid for g_2, and vice versa. That is, if we denote generators identical to g as $\text{orb}(g)$, for every cut generated for g we add the associated cut for every $\hat{g} \in \text{orb}(g)$. Finally, we choose an aggressive branch-and-cut strategy, generating cuts at the root node, for the first 50 nodes, and then every 100 nodes thereafter.

While this is heavy machinery for easy UC instances, we will see that for hard UC instances having this machinery available results in a noticeable improvement.

3.3.4 Computational Experiments

All computational experiments were performed on a Dell PowerEdge T620 with 2 Intel Xeon E5-2670 processors and 256GB of RAM running Ubuntu 14.04.2. Gurobi 6.5.0 was used as the MIP and LP solver for all problems, and the callback routine was implemented using Gurobi's Python interface. For all problems the number of available threads was set to 1. For the MIP unit commitment problem, the parameter PreCrush was set to 1 to facilitate adding cuts in callbacks along with a time limit of 1800 seconds. All other parameters were preserved at default. A dummy callback was used for instances where cuts were not added.

To generate a diverse set of unit commitment test instances, real-time load, day-ahead reserves, and wind generation for 2015 were obtained from PJM's website [128, 129]. For each day in 2015 a 24-hour unit commitment problem was formulated, with wind generation accounted for as negative demand in (3.23). For ease, the daylight savings days of 08 Mar and 01 Nov were excluded. 31 Dec was excluded for lack of available data. For the months of April – September the set of summer generators was used, and the winter generators were considered for the remaining six months. Generators with missing cost curves were excluded, and generators with missing up/down time data were given $UT^g = DT^g = 1$. Generators marked as wind powered were dropped as wind generation is considered separately. In total then 935 generators were considered for the winter system and 978 generators were considered for the summer system. Given the selection criteria above for $\mathcal{G}^C$, $|\mathcal{G}^C| = 459$ for the winter system and $|\mathcal{G}^C| = 492$ for the summer system. As no data on start-up or shut-down ramp rates are provided, $SU^g = SD^g = \underline{P}^g$ for all $g \in \mathcal{G}$. Additionally, no data on cool down is provided, so we assume all generators cool down in twice their minimum

Table 3.1: 3-bin UC Formulation Problem Size

	# cont. vars	# binary vars	# constraints	# nonzeros
Winter System	102048	112220	288478	963303
Summer System	103584	117360	300108	983799

Table 3.2: Winter System: Cut Generation LP Sizes

	# variables	# constraints	# nonzeros
Mean	7382.1	12247.2	46819.3
Min	1177	2148	7788
Median	8674	14251	53971
Max	14701	22572	84701

down time period, i.e., $DT_C^g = 2DT^g$. We use the data provided on initial status and assume all generators currently on are available to be turned off and operating at minimum power.

In Table 3.1 we specify the size of the base UC formulation (3.23) for both the winter and summer set of generators, reporting the number of constraints, continuous variables, binary variables, and non-zero elements in the constraints matrix, respectively. These form the template for our test set, as we vary only the demand and reserve data. As we can see, both generator sets yield MIPs of significant size.

In Table 3.2 we report some summary statistics for the 459 cut generation LPs for the winter system, and similarly in Table 3.3 we report summary statistics for the 492 cut generation LPs for the summer system, based on the formulation (3.22). As we can see, most of the LPs are of modest size, with the variation dependent on the parameters UT^g and DT^g. Namely, the larger UT^g and DT^g are, the fewer valid time intervals in $\mathcal{T}$ for generator g, so the smaller the formulation (3.22) is. As an example, many of the generators in the test sets are large coal units with $UT^g = 15$ and $DT^g = 9$. These units only have 2059 variables in their cut generation LPs. Some of the nuclear units in the test sets only have 1177 variables in their cut generation LPs because once started (stopped) they must stay on (off) for the rest of the time horizon. Conversely, there are some

Table 3.3: Summer System: Cut Generation LP Sizes

	# variables	# constraints	# nonzeros
Mean	7568.3	12537.3	47918.5
Min	1177	2148	7788
Median	8674	14778.5	56977
Max	14701	22572	84701

Table 3.4: Initial Computational Results

	No User Cuts		Ramping Polytope Cuts			
	Time (s)	Nodes	Time (s)	Nodes	Cuts Added	Cut Time (s)
Geometric Mean	172.52	44.01	163.39	44.08	97.16	10.50
Min	85.48	0	82.13	0	16	1.24
Median	168.97	0	153.82	0	99	11.83
Max	557.82	715	812.32	640	636	77.13
# inst. better	130	37	231	26		

Table 3.5: High Wind Computational Summary, Solved Instances

	No User Cuts		Ramping Polytope Cuts			
	Time (s)	Nodes	Time (s)	Nodes	Cuts Added	Cut Time (s)
Geometric Mean	204.47	60.84	196.50	48.03	153.99	8.34
Min	58.40	0	54.78	0	2	0.19
Median	197.41	0	192.63	0	173	10.32
Max	1523.65	9875	1030.74	10976	804	58.50
# inst. better	138	57	218	73		

small gas units in both test sets which have $UT^g = DT^g = 2$. These small units have larger cut generation LPs with 14701 variables.

3.3.5 Initial Results

Of the 362 instances tested, 361 were feasible, and the summary statistics for these instances with and without the cuts developed above are given in Table 3.4. As we can see, none of the problems in the test set devised are particularly difficult for a modern MIP solver. Though there may be some slight benefit to adding the cuts, most instances are solved at the root node, and no instance takes more than about 15 minutes or 1000 nodes. It is worth noting for these instances that Gurobi needs quite a bit of time (usually 60-120 seconds) to solve the root relaxation, and then spends quite a bit of time at the root node generating cuts and applying heuristics. In the last row we report the number of instances for which that method is strictly better. Here we see the cuts usually result in a better time, though just slightly, and in most cases ($n = 298$) both methods need the same number of nodes to prove optimality to the default tolerance of 0.01% (as most are solved at the root).

Table 3.6: High Wind, Harder Instances

	No User Cuts		Ramping Polytope Cuts			
Date	Time (s)	Nodes	Time (s)	Nodes	Cuts Added	Cut Time (s)
04 Jan	601.93	2841	543.98	1935	208	7.58
15 Mar	988.85	4129	751.59	1533	222	8.77
01 Apr	704.16	554	458.90	216	525	21.18
03 Apr	644.06	689	540.69	544	367	13.84
12 Apr	742.06	1310	435.95	556	125	13.45
15 Apr	1523.65	627	615.81	374	742	26.00
25 Apr	1152.09	1440	1030.74	1220	413	15.89
24 May	988.10	686	377.98	162	505	14.19
23 Jun	723.91	583	447.34	575	298	40.49
02 Oct	620.03	5030	911.74	10976	108	7.65
24 Oct	556.26	934	608.30	772	21	7.49
28 Oct	411.65	1134	701.46	3764	133	6.59
21 Nov	1193.51	4125	940.53	3130	187	9.30
25 Nov	567.15	2648	810.62	3910	288	13.08
16 Dec	993.68	5946	758.29	3214	192	8.54
23 Dec	793.52	9875	570.36	3688	94	6.47
Geometric Mean	780.74	1742.70	629.51	1255.49	210.66	11.88

3.3.6 High Wind Instances

To create more difficult test instances, we again used the 2015 data from PJM, but considered increased wind penetration. In 2015, wind energy accounted for approximately 2% of energy demanded. A recent study conducted for PJM suggested that the interconnection could handle renewable penetration as high as 30%, which may be coming online as soon as 2026 [57]. Therefore, to create high-wind penetration instances, we multiplied the 2015 wind data by a factor of 15 to get to 30% wind energy. Note that our model implicitly allows for the possibility of curtailment (as we consider wind as negative load); further, if the wind is greater than load at a given hour it may also provide reserves. Given the greater swings in the net-load curve that the extra wind generation causes, we would expect these instances to be much harder than the base-case instances, and indeed, we find this to be true.

Of the 362 instances tested, 6 timed out for both methods, and 356 solved for both methods (all instances were feasible). The summary statistics for the instances which did not time out are reported in Table 3.5. As we can see, there are modest reductions in geometric mean solve time and geometric mean nodes. To see the impact on more interesting instances, those for which either method took more than 10 minutes to solve (but did not time out) are detailed in Table 3.6. For these harder instances we can see that for the most part the cuts are effective at reducing the enumeration necessary to arrive at and prove an optimal solution. We have a geometric mean

Table 3.7: High Wind, Timed Out Instances

	No User Cuts		Ramping Polytope Cuts			
Date	MIP Gap	Nodes	MIP Gap	Nodes	Cuts Added	Cut Time (s)
27 Oct	0.0102%	10202	0.0116%	8024	400	15.69
12 Nov	0.1287%	2613	0.0871%	3470	447	13.50
14 Nov	0.0111%	10202	0.0104%	8939	561	17.88
17 Nov	0.1015%	2242	0.1144%	1917	729	13.29
26 Nov	0.2110%	4778	0.2128%	4498	225	9.83
20 Dec	0.0232%	10202	0.0188%	9955	243	12.49

reduction in run time of about 150 seconds, such that the typical hard instance went from taking approximately 13 minutes to 10.5 minutes to solve. As these problems are usually solved in a 10 or 15 minute time window, this is a significant improvement. Additionally, there is a 28% reduction in geometric mean nodes for these instances, suggesting that strengthening the feasible region for the ramping-constrained generators with cuts from the ramping polytope eliminates some enumeration. Lastly in Table 3.7 we summarize the 6 instances which timed out, reporting the final MIP gap in place of computational time. There do not seem to be any conclusions that can be safely drawn from these 6 instances.

3.3.7 Observed Cuts

To better understand the cuts generated from (3.22), we examined the generated cuts for a subset of the high-wind test instances. The vast majority of the cuts were variable upper-bound inequalities. Specifically, those of the form

$$\bar{p}_t^g \leq \sum_{i \in [T]} \left(\xi_i^g u_i^g + \alpha_i^g v_i^g + \sigma_i^g w_i^g \right), \tag{3.24}$$

for some $t \in [T]$, where the coefficients ξ_i^g, α_i^g, and σ_i^g are the normalized optimal dual values from (3.22) (so that the coefficient on $\bar{p}_t^g$ is 1).

In a similar fashion two-period ramping inequalities were observed, i.e.

$$\bar{p}_t^g - p_{t-j}^g \leq \sum_{i \in [T]} \left(\xi_i^g u_i^g + \alpha_i^g v_i^g + \sigma_i^g w_i^g \right), \tag{3.25}$$

$$\bar{p}_{t-j}^g - p_t^g \leq \sum_{i \in [T]} \left(\xi_i^g u_i^g + \alpha_i^g v_i^g + \sigma_i^g w_i^g \right), \tag{3.26}$$

where j was most often 1, but sometimes 2, and on one occasion 3 for the ramp-up inequality (3.25). Three-period ramping inequalities were also common

$$-p^g_{t-j} + \bar{p}^g_t - p^g_{t+k} \leq \sum_{i\in[T]} \left(\xi^g_i u^g_i + \alpha^g_i v^g_i + \sigma^g_i w^g_i\right), \tag{3.27}$$

$$\bar{p}^g_{t-j} - p^g_t + \bar{p}^g_{t+k} \leq \sum_{i\in[T]} \left(\xi^g_i u^g_i + \alpha^g_i v^g_i + \sigma^g_i w^g_i\right), \tag{3.28}$$

with (3.28) occurring much more often than (3.27), and both usually having $j = k = 1$. A few inequalities of the form (3.28) were observed with $j = 1$, $k = 2$ and $j = 2$, $k = 1$, and at least one instance with $j = 1$, $k = 3$.

Occasionally more exotic inequalities would be generated. The four-period ramping inequalities

$$-p^g_{t-j} + \bar{p}^g_t - p^g_{t+k} + \bar{p}^g_{t+l} \leq \sum_{i\in[T]} \left(\xi^g_i u^g_i + \alpha^g_i v^g_i + \sigma^g_i w^g_i\right), \tag{3.29}$$

$$\bar{p}^g_{t-j} - p^g_t + \bar{p}^g_{t+k} - p^g_{t+l} \leq \sum_{i\in[T]} \left(\xi^g_i u^g_i + \alpha^g_i v^g_i + \sigma^g_i w^g_i\right), \tag{3.30}$$

with consecutive time periods ($j = 1$, $k = 1$, $l = 2$) were most common, but four-period inequalities with $j = 2$, $k = 1$, and $l = 2$ were observed as well. The five-period ramping inequality

$$\bar{p}^g_{t-2} - p^g_{t-1} + \bar{p}^g_t - p^g_{t+1} + \bar{p}^g_{t+2} \leq \sum_{i\in[T]} \left(\xi^g_i u^g_i + \alpha^g_i v^g_i + \sigma^g_i w^g_i\right), \tag{3.31}$$

was also generated on several occasions. In a similar vain, a seven-, nine-, and ten-period ramping inequalities were observed once.

In all cases the generated inequalities had varying degrees of sparsity in the generator's status variables (u^g, v^g, w^g). Some cuts were generated with only two non-zeros on the right-hand side, these were always involving the end of the time horizon. Several inequalities only had a few non-zeros in the right-hand side. Many more however spanned the generator's production horizon (i.e., when the u^g variables are non-zero), though in most cases these inequalities had about one-third to one-half non-zeros on the right-hand side. This is because with fractional status variables, the cut generated usually spans several of the polytopes $D^{[a,b]}$.

3.3.8 Reflections

First, we note that it is somewhat surprising that we were able to separate so many cuts in a reasonable amount of time. However, with $T = 24$ we see from Tables 3.2 and 3.3 that many of the LPs are of a manageable size, and most can be solved quickly with a modern commercial LP solver. Further, after the first cut pass the most of the generators have the same solution in the UC LP relaxation, so there's nothing for Gurobi to do in the LP separation problem. (As mentioned above, we leave the LP separation problems loaded in memory.) Additionally, across all the test instances, 33% of the cuts we add are additional valid cuts added by symmetry. (Recall that if we compute a cut for a generator we also add that cut for all generators in its orbit.)

Even though in practice we were able to solve the cut-generating problem for $T = 24$, obviously the problem (3.22) increases super-linearly in T. One way to improve the performance could be to more carefully screen which generators we generate cuts on. It may only make sense with longer time horizons to generate cuts on those "large" generators (with large UT^g and DT^g) for which the super-linear explosion in formulation size is more manageable. Additionally, it is clear from the formulation of (3.22) that most of the columns and rows (specifically those from (3.22b)) probably never enter the basis, and could perhaps be generated on the fly, with an initial basis constructed based on the 3-bin solution. However, the implementation of such a method would be non-trivial. Another possibility to improve the performance on longer time horizons would be to solve a "rolling" separation problem, where for each time period t we solve a small (e.g. 7-period) version of (3.22) centered around time t. That being said, the density of the cuts observed suggests that such a procedure may be less effective at generating quality cuts. Alternatively, problem (3.22) could be generated on the fly based on the LP relaxation values for u^g. Notice (3.22) decomposes when $u^*_t = 0$ for some t, and though adding time-dependent start-up costs complicates this picture, such a decomposition could be done heuristically.

Finally, it is worth taking a moment to bridge the gap between the computation results presented in this section and those reported on 2 and 3-period ramping inequalities recently, namely Damcı-Kurt et al. [27] and Pan and Guan [125]. We note that the cuts given by (3.22) are a superset of those presented in these two papers. The "slow-start" generators in Damcı-Kurt et al. [27] take an average of 4 time periods to ramp from SU to $\overline{P}$, and the "fast-start" generators need an average of 3 time periods. Similarly, for the instances used in Pan and Guan [125], every generator in the test set needs 4 time periods to ramp up to $\overline{P}$. This test set also contains large amounts of

symmetry, which for unit commitment is not perfectly encoded in the formulation symmetry, and hence cannot be exploited by the MIP solver [120]. In the systems we test here, for both the winter and summer generator sets, the generators in $\mathcal{G}^C$ take an average of just 2 time periods to ramp from SU to $\overline{P}$, and half (which we do not generate inequalities for) do not have ramping constraints at all. As far as the authors are aware this is the first experiment to test any ramping inequalities based on real-world generator data. The computational results presented here demonstrate that valid inequalities from the ramping polytope are beneficial for difficult unit commitment instances, and do not detract from unit commitment instances which are easy to solve.

3.4 Conclusion

We have presented a compact extended formulation for a ramping-constrained generator and a cut-generating linear program based upon the extended formulation. We demonstrated that the these cuts are computational beneficial for high-wind unit commitment instances based on the FERC generator set and data from PJM. Finally, the slight generalization of Balas's result [6, 7] presented in Appendix C may be of use in developing new extended formulations.

Chapter 4

A Novel Matching Formulation for Startup Costs in Unit Commitment

This chapter and Appendices D and E are based on a manuscript prepared for publication by Ben Knueven, Jim Ostrowski, and Jean-Paul Watson:

> Knueven, B., Ostrowski, J., and Watson, J. P. (2017). A Novel Matching Formulation for Startup Costs in Unit Commitment. *Submitted*.

Authors Knueven and Ostrowski discovered the proposed matching formulation. Author Knueven posed Theorem 4.1; authors Knueven and Ostrowski proved Theorem 4.1. Authors Knueven and Watson set-up the computational experiments; author Knueven performed the computational experiments and collected the data. Author Knueven drafted the initial manuscript; authors Ostrowski and Watson edited and enhanced the manuscript. A preprint of this paper is available at http://www.optimization-online.org/DB_FILE/2017/03/5897.pdf.

In this chapter, we present a novel formulation for startup cost computation in the unit commitment problem (UC). Both our proposed formulation and existing formulations in the literature are placed in a formal, theoretical dominance hierarchy based on their respective linear programming relaxations. Our proposed formulation is tested empirically against existing formulations on large-scale unit commitment instances drawn from real-world data. While requiring more variables than the current state-of-the-art formulation, our proposed formulation requires fewer constraints, and is empirically demonstrated to be as tight as a perfect formulation for startup costs. This tightening can reduce the computational burden in comparison to existing

formulations, especially for UC instances with large reserve margins and high penetration levels of renewables.

4.1 Nomenclature

4.1.1 Indices and Sets

$g \in \mathcal{G}$	Thermal generators
$l \in \mathcal{L}_g$	Piecewise production cost intervals for generator g: $1, \ldots, L_g$.
$s \in \mathcal{S}_g$	Startup categories for generator g, from hottest (1) to coldest (S_g).
$t \in \mathcal{T}$	Hourly time steps: $1, \ldots, T$.

4.1.2 Parameters

c_g^l	Cost coefficient for piecewise segment l for generator g (\$/MWh).
c_g^R	Cost of generator g running and operating at minimum production $\underline{P}_g$ (\$/h).
c_g^s	Startup cost in category s for generator g (\$).
$D(t)$	Load (demand) at time t (MW).
DT_g	Minimum down time for generator g (h).
$\overline{P}_g$	Maximum power output for generator g (MW).
$\overline{P}_g^l$	Maximum power available for piecewise segment l for generator g (MW).
$\underline{P}_g$	Minimum power output for generator g (MW).
$R(t)$	Spinning reserve at time t (MW).
RD_g	Ramp-down rate for generator g (MW/h).
RU_g	Ramp-up rate for generator g (MW/h).
SD_g	Shutdown rate for generator g (MW/h).
SU_g	Startup rate for generator g (MW/h).
TC_g	Time down after which generator g goes absolutely cold, i.e., enters state S_g.
$\underline{T}_g^s$	Time offline after which the startup category s is available ($\underline{T}_g^1 = DT_g$, $\underline{T}_g^{S_g} = TC_g$).
$\overline{T}_g^s$	Time offline after which the startup category s is no longer available ($= \underline{T}_g^{s+1}$, $\overline{T}_g^{S_g} = +\infty$).
UT_g	Minimum run time for generator g (h).
$W(t)$	Aggregate wind (renewables, more generally) generation available at time t (MW).

4.1.3 Variables

$p_g(t)$ Power above minimum for generator g at time t (MW).

$p_W(t)$ Aggregate wind generation used at time t (MW).

$p_g^l(t)$ Power from piecewise interval l for generator g at time t (MW).

$r_g(t)$ Spinning reserves provided by generator g at time t (MW), ≥ 0.

$u_g(t)$ Commitment status of generator g at time t, $\in \{0, 1\}$.

$v_g(t)$ Startup status of generator g at time t, $\in \{0, 1\}$.

$w_g(t)$ Shutdown status of generator g at time t, $\in \{0, 1\}$.

$c_g^{SU}(t)$ Startup cost for generator g at time t (\$), ≥ 0.

$\delta_g^s(t)$ Startup in category s for generator g at time t, $\in \{0, 1\}$.

$x_g(t, t')$ Indicator arc for shutdown at time t, startup at time t', uncommitted for $i \in [t, t')$, for generator g, $\in \{0, 1\}$.

$y_g(t, t')$ Indicator arc for startup at time t, shutdown at time t', committed for $i \in [t, t')$, for generator g, $\in \{0, 1\}$.

4.2 Introduction

The unit commitment problem (UC) concerns the scheduling of thermal generators to meet projected demand while minimizing system operations cost [160]. Here, we propose a new formulation for representing thermal generator startup costs, which leads to a tightening of the linear programming (LP) relaxation of the mixed-integer linear programming (MILP) UC problem. We then empirically demonstrate that the tighter LP relaxation can translate into reduced run-times to solve the MILP UC using commercial branch-and-cut solvers.

MILP formulations for UC have been of interest since Garver's original formulation [56]. These are extremely difficult problems to solve in practice at the scale, e.g., at the scale of the Midcontinent ISO in the United States. Many practical problems involve hundreds of generators on a transmission system with thousands of buses with a time horizon of at least 48 hours. Further, solutions must be completed in tens of minutes at most. As a consequence, system operators often have to use substantially suboptimal solutions to comply with the time limit, i.e., with optimality gaps that are sometimes tens of a percent [18].

There are a few approaches for reducing run-times to an optimal solution. One approach is via decomposition. The intuition is that loosely-connected parts of the UC problem can be

decomposed into easier subproblems, and a solution to the original can be discovered through an iterative process. One way to decompose UC is by generators – splitting the generator set $\mathcal{G}$ into subsets (by location or some other criteria). Classical decomposition methods (such as ADMM) can then be used to force convergence between subproblems [40, 134]. Another possible decomposition is on the time horizon – the principle here being that after a sufficiently long period decisions made previously do not have much affect on decisions made now. Such an approach is explored in [81].

An orthogonal approach for reducing run times is stronger formulations for UC, and this research has found its way into practice. Most of this work has focused on tightening the polyhedral description of a single generator's dispatch. In [86] an exponential convex hull description for minimum up and down times in terms of a generator's status variables is given; Rajan and Takriti [133] uses the startup and shutdown status variables to describe the same set using only a linear number of inequalities. This result is extended in [58] to generators with startup and shutdown power constraints. Inequalities to tighten the formulation of the ramping process are considered in [119, 27, 125], as well as Chapter 3.

A formulation for time-dependent startup costs based on generator commitment variables appears in [115]; Carrion and Arroyo [17] considers the same formulation in the context of a MILP approach to UC. Startup cost categories together with associated indicator variables are introduced in [113]. Morales-España et al. [110] improves on the indicator formulation from [145] and demonstrates empirically that the use of startup category indicators results in a tighter formulation than those described in [115] and [17], both of which use generator commitment variables to model startup costs. Morales-España et al. [111] uses this same approach to model generator start-up and shut-down energy production. Brandenberg et al. [14] shows that the epigraph for concave increasing startup costs modeled using generator status variables has an exponential number of facets. However, [14] provides a linear-time separation algorithm for computing these facets. Finally, a restrictive temperature-based model for startup cost is presented in [144].

In this chapter we introduce a novel matching formulation for time-dependent startup costs in UC. We theoretically analyze the strength of our formulation relative to existing formulations in the literature, and introduce an additional formulation as an intermediary to ease the comparison between existing formulations. We then empirically analyze the impact of our new formulation, both in an absolute sense and relative to other formulations, on the ability of commercial branch-and-cut software packages to solve utility-scale UC problems.

The remainder of this chapter is organized as follows. We begin in Section 4.3 with a discussion of the base UC formulation, without startup cost components. Section 4.4 then details both existing and two novel startup cost formulations for UC. In Section 4.5, we establish a provable dominance hierarchy concerning the relative tightness of the LP relaxations of the different startup cost formulations. We empirically compare the performance of the various startup cost formulations in Section 4.6, using large-scale UC instances based on industrial data. We discuss the implications of our results in Section 4.7. Finally, we conclude with a summary of our contributions in Section 4.8.

4.3 Unit Commitment Formulation

We present a MILP UC formulation based on [110] that we will use as the baseline for the comparison between startup cost formulations. We assume that the production cost is piecewise linear convex in $p_g(t)$, where L_g is the number of piecewise intervals and $\overline{P}_g^0 = \underline{P}$ is the start of the first interval. Let $\mathcal{G}^1$ be the generators that have $UT^g = 1$ and $\mathcal{G}^{>1}$ be the generators with $UT^g > 1$. We then formulate the UC problem as follows:

$$\min \sum_{g\in\mathcal{G}} \sum_{t\in\mathcal{T}} \left(\sum_{l\in\mathcal{L}_g} (c_g^l \, p_g^l(t)) + c_g^R \, u_g(t) + c_g^{SU}(t) \right) \tag{4.1a}$$

subject to:

$$\sum_{g\in\mathcal{G}} \big(p_g(t) + \underline{P}_g u_g(t)\big) + p_W(t) = D(t) \qquad \forall t \in \mathcal{T} \tag{4.1b}$$

$$\sum_{g\in\mathcal{G}} r_g(t) \geq R(t) \qquad \forall t \in \mathcal{T} \tag{4.1c}$$

$$p_g(t) + r_g(t) \leq (\overline{P}_g - \underline{P}_g)u_g(t) - (\overline{P}_g - SU_g)v_g(t) \qquad \forall t \in \mathcal{T}, \forall g \in \mathcal{G}^1 \tag{4.1d}$$

$$p_g(t) + r_g(t) \leq (\overline{P}_g - \underline{P}_g)u_g(t) - (\overline{P}_g - SD_g)w_g(t{+}1) \qquad \forall t \in \mathcal{T}, \forall g \in \mathcal{G}^1 \tag{4.1e}$$

$$\begin{aligned} p_g(t) + r_g(t) \leq{}& (\overline{P}_g - \underline{P}_g)u_g(t) \\ &- (\overline{P}_g - SU_g)v_g(t) - (\overline{P}_g - SD_g)w_g(t{+}1) \end{aligned} \qquad \forall t \in \mathcal{T}, \forall g \in \mathcal{G}^{>1} \tag{4.1f}$$

$$p_g(t) + r_g(t) - p_g(t{-}1) \leq RU_g \qquad \forall t \in \mathcal{T}, \forall g \in \mathcal{G} \tag{4.1g}$$

$$p_g(t{-}1) - p_g(t) \leq RD_g \qquad \forall t \in \mathcal{T}, \forall g \in \mathcal{G} \tag{4.1h}$$

$$p_g(t) = \sum_{l\in\mathcal{L}_g} p_g^l(t) \qquad \forall t \in \mathcal{T}, \forall g \in \mathcal{G} \tag{4.1i}$$

$$p_g^l(t) \leq (\overline{P}_g^l - \overline{P}_g^{l-1}) \qquad \forall t \in \mathcal{T},\ \forall l \in \mathcal{L}_g,\ \forall g \in \mathcal{G} \qquad (4.1j)$$

$$u_g(t) - u_g(t-1) = v_g(t) - w_g(t) \qquad \forall t \in \mathcal{T},\ \forall g \in \mathcal{G} \qquad (4.1k)$$

$$\sum_{i=t-UT_g+1}^{t} v_g(i) \leq u_g(t) \qquad \forall t \in [UT_g, T],\ \forall g \in \mathcal{G} \qquad (4.1l)$$

$$\sum_{i=t-DT_g+1}^{t} w_g(i) \leq 1 - u_g(t) \qquad \forall t \in [DT_g, T],\ \forall g \in \mathcal{G} \qquad (4.1m)$$

$$p_W(t) \leq W(t) \qquad \forall t \in \mathcal{T} \qquad (4.1n)$$

$$p_g^l(t) \in \mathbb{R}_+ \qquad \forall t \in \mathcal{T},\ \forall l \in \mathcal{L}_g,\ \forall g \in \mathcal{G} \qquad (4.1o)$$

$$p_g(t), r_g(t) \in \mathbb{R}_+ \qquad \forall t \in \mathcal{T},\ \forall g \in \mathcal{G} \qquad (4.1p)$$

$$p_W(t) \in \mathbb{R}_+ \qquad \forall t \in \mathcal{T} \qquad (4.1q)$$

$$u_g(t), v_g(t), w_g(t) \in \{0,1\} \qquad \forall t \in \mathcal{T},\ \forall g \in \mathcal{G}. \qquad (4.1r)$$

Constraints (4.1b – 4.1r) are standard in UC formulations without time-varying startup costs, see [110] or Chapter 3.

We will take the formulation above as given, and for the remainder of the chapter we will focus on the formulation of the startup cost $c_g^{SU}(t)$.

4.4 Startup Cost Formulations

In this section, we introduce the formulations for startup cost $c_g^{SU}(t)$ examined in this chapter. (We provide the full specification for each formulation in Appendix D.) For notational ease, since in all cases we are referencing a single generator, we will drop the subscript g on all variables and parameters in this section and in the following section.

4.4.1 Formulations from the Literature

One Binary Formulation (1-bin)

The typical formulation for startup costs using only the status variable u is [115, 17]

$$c^{SU}(t) \geq c^s \left(u(t) - \sum_{i=1}^{\underline{T}^s} u(t-i) \right) \qquad \forall s \in \mathcal{S},\ \forall t \in \mathcal{T} \qquad (4.2a)$$

$$c^{SU}(t) \geq 0 \qquad \forall t \in \mathcal{T}. \qquad (4.2b)$$

This formulation has the advantage of only needing as many constraints as startup types, and no additional variables.

Strengthened One Binary Formulation (1-bin*)

As pointed out in [144], the 1bin formulation above can be strengthened by increasing the coefficients on the $u(t-i)$ variables

$$c^{SU}(t) \geq c^s \left(u(t) - \sum_{i=1}^{DT} u(t-i) \right) - \sum_{k=1}^{s-1} \left((c^s - c^k) \sum_{i=\underline{T}^k+1}^{\overline{T}^k} u(t-i) \right) \qquad \forall s \in \mathcal{S},\ \forall t \in \mathcal{T} \tag{4.3a}$$

$$c^{SU}(t) \geq 0 \qquad \forall t \in \mathcal{T}. \tag{4.3b}$$

Startup Type Indicator Formulation (STI)

The formulation proposed in [110] introduces binary indicator variables for each startup type. Specifically, for each startup type s, we have $\delta^s(t)$, $\forall t \in \mathcal{T}$ which is 1 if the generator has a type s startup in time t and 0 otherwise. The corresponding constraints are

$$\delta^s(t) \leq \sum_{i=\underline{T}^s}^{\overline{T}^s-1} w(t-i) \qquad \forall s \in \mathcal{S} \setminus S,\ \forall t \in \mathcal{T} \tag{4.4a}$$

$$v(t) = \sum_{s=1}^{S} \delta^s(t) \qquad \forall t \in \mathcal{T}. \tag{4.4b}$$

We can replace the objective function variables $c^{SU}(t)$ using the substitution

$$c^{SU}(t) = \sum_{s=1}^{S} c^s \delta^s(t) \qquad \forall t \in \mathcal{T}. \tag{4.4c}$$

Extended Formulation (EF)

Pochet and Wolsey [131] propose an extended formulation for startup and shutdown sequences, which provides a perfect formulation for startup costs. Let $y(t,t') = 1$ if there is a startup in time t and a shutdown in time t' and 0 otherwise, for $t' \geq t + UT$. Similarly let $x(t,t') = 1$ if there is a shutdown in time t and a startup in time t' and 0 otherwise, for $t' \geq t + DT$. The constraints are

$$\sum_{\{t' | t' > t\}} y(t,t') = v(t) \qquad \forall t \in \mathcal{T} \tag{4.5a}$$

$$\sum_{\{t'|t'<t\}} y(t',t) = w(t) \qquad \forall t \in \mathcal{T} \tag{4.5b}$$

$$\sum_{\{t'|t'<t\}} x(t',t) = v(t) \qquad \forall t \in \mathcal{T} \tag{4.5c}$$

$$\sum_{\{t'|t'>t\}} x(t,t') = w(t) \qquad \forall t \in \mathcal{T} \tag{4.5d}$$

$$\sum_{\{\tau,\tau'|\tau\leq t<\tau'\}} y(\tau,\tau') = u(t) \qquad \forall t \in \mathcal{T}. \tag{4.5e}$$

Note that with constraints (4.5a–4.5e), constraints (4.1k–4.1m) become redundant. Hence, the u, v, and w variables may be projected out. The startup costs are calculated by placing the appropriate coefficient on the x variables

$$c^{SU}(t) = \sum_{s=1}^{S} c^s \left(\sum_{t'=t-\overline{T}^s+1}^{t-\underline{T}^s} x(t',t) \right) \qquad \forall t \in \mathcal{T}, \tag{4.5f}$$

where the inside summation is understood to be taken over valid t'.

From integer programming theory [159], we know this formulation to be integral because it is a network flow model, where the vertices are two partite sets, one for startups and the other for shutdowns, and the arcs y connect startups to feasible shutdowns and the arcs x connect shutdowns to feasible startups. By putting a flow of one unit through the network, we arrive at a feasible generator schedule. Note integrality comes at the cost of needing $\mathcal{O}(|\mathcal{T}|^2)$ additional variables to model startup costs.

4.4.2 Novel Formulations

Here we present two new formulations for startup costs. The first can be seen as a relaxation of EF, and the second as 1-bin* with the inequalities strengthened by using the turn on and turn off indicators v and w.

Matching Formulation (Match)

Similar to EF, for $t \in \mathcal{T}$ let $x(t',t) = 1$ if there is a shutdown in time t' and a startup in time t and 0 otherwise, for $t' \in \mathcal{T}$ such that $t - TC < t' \leq t - DT$. Note that this is only $(TC - DT)|\mathcal{T}|$

additional variables. The associated constraints are

$$\sum_{t'=t-TC+1}^{t-DT} x(t',t) \le v(t) \qquad \forall t \in \mathcal{T} \tag{4.6a}$$

$$\sum_{t'=t+DT}^{t+TC-1} x(t,t') \le w(t) \qquad \forall t \in \mathcal{T}, \tag{4.6b}$$

(where again the sums are understood to be taken over valid t') and the objective function is

$$c^{SU}(t) = c^S v(t) + \sum_{s=1}^{S-1} (c^s - c^S) \left(\sum_{t'=t-\overline{T}^s+1}^{t-\underline{T}^s} x(t',t) \right) \qquad \forall t \in \mathcal{T}. \tag{4.6c}$$

Let us examine constraints (4.6a) and (4.6b). Note that if $v(t)$ and $w(t)$ are already determined, these equations serve to match shutdowns with startups. That is, if $v(t) = 1$ and $w(t') = 1$, then in any optimal solution $x(t',t) = 1$ since $c^s - c^S < 0$. We arrive at this formulation by eliminating the arcs y from EF and the arcs $x(t',t)$ such that $t - t' \ge TC$.

Three Binary Formulation (3-bin)

This formulation is similar in spirit to the 1-bin* formulation, only instead of using the status variables u, we use the on/off variables v and w to keep track of the different types of startups

$$c^{SU}(t) \ge c^s v(t) - \sum_{k=1}^{s-1} \left((c^s - c^k) \sum_{i=\underline{T}^k}^{\overline{T}^k-1} w(t-i) \right) \qquad \forall s \in \mathcal{S},\ \forall t \in \mathcal{T} \tag{4.7a}$$

$$c^{SU}(t) \ge 0 \qquad \forall t \in \mathcal{T}. \tag{4.7b}$$

Equation (4.7a) works analogously to equation (4.3a). That is, if we did not turn off in the last $\overline{T}^s$ time periods, (4.7a) ensures that we pay at least c^s for a startup in time t. Note that when $s = 1$, the second term is an empty sum, and hence is 0. Equation (4.7b) ensures that the startup cost is never negative. Note that over STI, this formulation needs the same number of constraints and has fewer variables (only the additional $|\mathcal{T}|$ variables for $c^{SU}(t)$, which can be eliminated for STI, while not needing the indicator variables δ). This formulation is presented to ease the comparison between STI and the 1-bin formulations.

4.5 Dominance Hierarchy of Startup Cost Formulations

In this section we establish the relationships between the six formulations presented in Section 4.4. First, we consider the relationship between the tightness of each formulation. Let z_{EF}, z_{Match}, z_{STI}, z_{3bin}, z_{1bin^*}, and z_{1bin} be the linear programming relaxation values for their respective formulations. We have the following:

Theorem 4.1.

$$z_{1bin} \leq z_{1bin^*} \leq z_{3bin} \leq z_{STI} \leq z_{Match} \leq z_{EF},$$

that is, EF is the tightest formulation, 1-bin is the weakest formulation, with the relationship above amongst the others.

Proof. Since the EF formulation is the convex hull description, it is clear that it is the tightest formulation, implying $z_{Match} \leq z_{EF}$. Furthermore, [144] shows that 1bin* is a tighter formulation than 1bin, implying that $z_{1bin} \leq z_{1bin^*}$. As a result, we only need to prove the inner three relationships. To prove these relationships, i.e. that $z_A \leq z_B$, it is sufficient to show that there is a linear mapping from the polytope associated with B onto the polytope associated with A that preserves objective value and that through this linear mapping, every constraint in formulation A is implied by constraints in formulation B. This is sufficient to show that $z_A \leq z_B$ as it shows that *all* feasible solutions for B can be mapped to solutions feasible for A with the same objective value.

$\mathbf{z_{STI} \leq z_{Match}}$: We proceed by demonstrating that all the inequalities in STI are implied by the inequalities in Match. First, consider the linear transformation from Match to STI

$$\delta^s(t) = \sum_{t'=t-\overline{T}^s+1}^{t-\underline{T}^s} x(t',t) \qquad \forall s \in \mathcal{S} \setminus S,\ \forall t \in \mathcal{T} \tag{4.8}$$

$$\delta^S(t) = v(t) - \sum_{t'=t-TC+1}^{t-DT} x(t',t) \qquad \forall t \in \mathcal{T}. \tag{4.9}$$

The equality constraints (4.4b) follow directly from the sum of (4.8) and (4.9). To see (4.4a), notice that by (4.6b),

$$x(i,t) \leq w(i) \qquad \forall t \in \{i+DT, \ldots, i+TC-1\}, \forall i \in \mathcal{T}. \tag{4.10}$$

By (4.8) and (4.10), we have

$$\delta^s(t) \leq \sum_{i=t-\overline{T}^s+1}^{t-\underline{T}^s} w(i) = \sum_{i=\underline{T}^s}^{\overline{T}^s-1} w(t-i) \qquad \forall s \in \mathcal{S} \setminus S,\ \forall t \in \mathcal{T}, \tag{4.11}$$

which is just (4.4a).

$\mathbf{z_{3bin}} \leq \mathbf{z_{STI}}$: This follows by eliminating the indicators δ from the objective function using (4.4a) and (4.4b). As $c^k \geq c^s$ for all $k \in \mathcal{S}$ such that $k > s$, we have

$$\begin{aligned} c^{SU}(t) &= \sum_{k=1}^{S} c^k \delta^k(t) \\ &\geq \sum_{k=1}^{s-1} c^k \delta^k(t) + c^s \sum_{k=s}^{S} \delta^k(t) \\ &= c^s \sum_{k=1}^{S} \delta^k(t) - \sum_{k=1}^{s-1} (c^s - c^k) \delta^k(t) \\ &\geq c^s v(t) - \sum_{k=1}^{s-1} \left((c^s - c^k) \sum_{i=\underline{T}^k}^{\overline{T}^k-1} w(t-i) \right) \qquad \forall s \in \mathcal{S},\ \forall t \in \mathcal{T}, \end{aligned} \tag{4.12}$$

which is (4.7a). (4.7b) follows from the non-negativity of c^s and $\delta^s(t)$.

$\mathbf{z_{1bin^*}} \leq \mathbf{z_{3bin}}$: (4.3b) and (4.7b) are the same, so we need show that (4.7a) implies (4.3a). Consider the inequality (4.7a), noting $v(t) \geq u(t) - u(t-1)$ by (4.1k) and $-w(t) \geq -u(t-1)$ by (4.1k) and (4.1l)

$$\begin{aligned} c^{SU}(t) &\geq c^s v(t) - \sum_{k=1}^{s-1} \left((c^s - c^k) \sum_{i=\underline{T}^k}^{\overline{T}^k-1} w(t-i) \right) \\ &\geq c^s (u(t) - u(t-1)) - \sum_{k=1}^{s-1} \left((c^s - c^k) \sum_{i=\underline{T}^k}^{\overline{T}^k-1} u(t-1-i) \right) \\ &\geq c^s (u(t) - u(t-1)) - \sum_{k=1}^{s-1} \left((c^s - c^k) \sum_{i=\underline{T}^k+1}^{\overline{T}^k} u(t-i) \right) \\ &\geq c^s \left(u(t) - \sum_{i=1}^{DT} u(t-i) \right) - \sum_{k=1}^{s-1} \left((c^s - c^k) \sum_{i=\underline{T}^k+1}^{\overline{T}^k} u(t-i) \right) \qquad \forall s \in \mathcal{S},\ \forall t \in \mathcal{T}, \end{aligned} \tag{4.13}$$

which is (4.3a).

□

Table 4.1: Size of the Formulations

Formulation	# variables	# constraints
1-bin	$\mathcal{O}(\|\mathcal{T}\|)$	$\mathcal{O}(\|\mathcal{S}\|\|\mathcal{T}\|)$
1-bin*	$\mathcal{O}(\|\mathcal{T}\|)$	$\mathcal{O}(\|\mathcal{S}\|\|\mathcal{T}\|)$
3-bin	$\mathcal{O}(\|\mathcal{T}\|)$	$\mathcal{O}(\|\mathcal{S}\|\|\mathcal{T}\|)$
STI	$\mathcal{O}(\|\mathcal{S}\|\|\mathcal{T}\|)$	$\mathcal{O}(\|\mathcal{S}\|\|\mathcal{T}\|)$
Match	$\mathcal{O}((TC - DT)\|\mathcal{T}\|)$	$\mathcal{O}(\|\mathcal{T}\|)$
EF	$\mathcal{O}(\|\mathcal{T}\|^2)$	$\mathcal{O}(\|\mathcal{T}\|)$

We note that Theorem 4.1 is not demonstrating strict dominance. We only guarantee that, for example, EF is no worse than Match in its linear programming relaxation. As we will see from the computational experiments in Section 4.6, some of these relationship often hold with equality.

Table 4.1 compares the size as a function of the problem parameters for each startup formulation. Note we only consider the variables needed in addition to the baseline formulation (4.1).

4.6 Computational Experiments

The dominance hierarchy for various startup cost formulations introduced in Section 4.5 establishes their relative tightness. We quantify tightness as the optimal objective function value for the LP relaxation of the UC problem with a given startup cost formulation. In the context of a MILP, tighter LP relaxations can lead to more efficient branch-and-cut search, due to increased fathoming opportunities. However, the size and structure of the underlying LP varies across startup cost formulations, and reductions in branch-and-cut search time (measured in terms of number of tree nodes explored) may be offset by the cost of solving the LP relaxations at each node. Further, formulation details interact with heuristics and other features of MILP solvers, often in unpredictable ways.

In this context, we now experimentally compare the performance of the range of startup cost formulations for UC, using two state-of-the-art commercial MILP solvers. We consider two sets of problem instances. The first set of instances are realistic instances derived from publicly available market and regulatory data obtained from the CAISO system operator in the US. The second set is the FERC generator set [84] (which itself is based on data from the PJM independent system operator in the US), with demand, reserve, and wind scenarios based on publicly-available data obtained from PJM for 2015 [128, 129].

The "CAISO" instances have 610 thermal generators, of which 410 are schedulable, i.e., not forced to run. Generators with quadratic cost curves were approximated using $L_g = 2$. Five 48-hour demand scenarios were examined; demands were taken directly from CAISO historical data. Four of the demand scenarios are based on historical information, while "Scenario400" is a hypothetical scenario where wind supply is on average 40% of demand; the wind profile is constructed based on actual CAISO data, scaled appropriately. For each instance the reserve level was varied from 0%, 1%, 3%, and 5% of demand, resulting in a total of 20 test instances. We allow for the possibility of curtailment of wind generation by (4.1b) and (4.1n). Each generator has only two startup categories, i.e., $S_g = 2$.

The "FERC" instances are based on two generator sets provided by (and publicly available from) the US Federal Energy Regulatory Commission (FERC): a "Summer" set of generators and a "Winter" set of generators [84]. We use the Summer set of generators for dates in April - September and the Winter set for the remaining dates. After (i) excluding generators with missing or negative cost curves, (ii) letting $UT_g = DT_g = 1$ for generators g with missing up/down time data, and (iii) eliminating generators marked as wind (we consider wind power separately), the Summer and Winter sets respectively contain 978 and 934 generators. No data on startup or shutdown power limits was provided by FERC, so we assume $SU_g = SD_g = \underline{P}^g$. Similarly, FERC provided no data for cool-down times, so we set $TC_g = 2DT_g$. All generators had at most two startup types, i.e., $S_g \leq 2$, and the piecewise production cost curves are based on market bids, such that $1 \leq L_g \leq 10$.

For the FERC instances, we consider twelve 48-hour demand, reserve, and wind scenarios from 2015, one from each month. In 2015, wind generation accounted for 2% of the electricity supplied in PJM, so we created twelve additional "high-wind" scenarios by multiplying the wind data for 2015 by a constant factor of 15 to increase mean wind energy supply for the year to 30% of load. A recent study conducted for PJM suggests that in less than a decade renewables could achieve 30% penetration rates in the interconnection [57]. Like the CAISO instances, we allow for the curtailment of wind generation.

The two test instance sets represent vastly different systems. The CAISO instances consist of mostly small, flexible generators. Of the 410 schedulable generators, only 20 have irredundant ramping constraints (i.e., $RU_g \geq (\overline{P}_g - \underline{P}_g)$ and $RU_g \geq (\overline{P}_g - \underline{P}_g)$). Therefore, for 390 of the generators (95% of the total), EF, together with the equations from (4.1), is a convex hull description of each generator's dispatch. These flexible generators account for 75% of schedulable capacity. For

Table 4.2: Summary of computational experiments for CAISO instances using Gurobi. For time (s) and number of branch-and-cut (B&C) nodes we report the geometric mean across the 20 instances, including those which reach the wall-clock limit of 600 seconds.

Formulation	EF	Match	STI	3-bin	1-bin*	1-bin
Time (s)	370.5	**43.12**	52.84	91.43	600	600
# of times best	0	**11**	8	1	0	0
# of times 2nd	0	8	**11**	1	0	0
Max. time (s)	600	**130**	243	600	600	600
# of time outs	7	**0**	**0**	1	20	20
# B&C nodes	1.510	**13.50**	20.22	39.09	5914	5827

Table 4.3: Summary of computational experiments for CAISO Instances using CPLEX. For time (s) and number of branch-and-cut (B&C) nodes we report the geometric mean across the 20 instances, including those which hit the wall-clock limit of 600 seconds.

Formulation	EF	Match	STI	3-bin	1-bin*	1-bin
Time (s)	261	48.0	**40.8**	62.9	600	600
# of times best	0	5	**11**	4	0	0
# of times 2nd	0	6	**8**	6	0	0
Max. time (s)	600	**110**	223	423	600	600
# of time outs	2	**0**	**0**	**0**	20	20
# of B&C nodes	3.60	**2.83**	4.75	32.6	15442	15210

both the Summer and Winter FERC generator sets, such flexible generators only account for 50% of the fleet, and approximately 30% of schedulable capacity.

Computational experiments were conducted on a Dell PowerEdge T620 with two Intel Xeon E5-2670 processors, for a total of 16 cores and 32 threads, and 256GB of RAM, running the Ubuntu 14.04.5 Linux operating system. The Gurobi 7.0.1 MILP solver was used for the experiments labeled "Gurobi", while the CPLEX 12.7.1.0 MILP solver was used for the experiment labeled "CPLEX". Both solvers were allowed to use all 32 threads in each experimental trial. Here we present summaries of the computational experiments; the full results are available in Appendix E.

4.6.1 CAISO Instances

We first consider the experimental results for the CAISO instances. For both Gurobi and CPLEX, we impose a wall-clock time limit of 600 seconds; all other settings were left at their defaults. In Tables 4.2 and 4.3, we summarize the computational experiments for these instances. For each UC formulation we report the geometric mean time to an optimal solution (Time (s)), the number of

instances for which that method did best (# of times best), the number of instances for which that method did second best (# of times 2nd), the longest run time across the 20 instances (Max. time (s)), and the number of instances for which that method hit the 600 second time limit (# of time outs). When a solver times out for an instance, we substitute the time limit in the calculation for the geometric mean, leading to an underestimation when an instance fails to solve for a given UC formulation. In the last row, we report the shifted geometric mean number of branch-and-cut tree nodes explored by the solver, substituting the number of nodes explored when the solver hits the time limit. To compute the shifted geometric mean, we add 1 to each node count, so as to avoid multiplying by 0 when the solver identifies a solution at the root node. A bold-faced entry in a row denotes the startup cost variant that performed best for the given measure.

We immediately see that both of the 1-bin variants are not competitive, and in no case identify an optimal solution within the time limit, even after exploring a considerable number of branch-and-cut nodes. This is consistent with results reported recently in the UC literature. Gurobi identifies optimal solutions to the EF variant in approximately half the cases, and CPLEX identifies optimal solutions in all but two cases. However, for both solvers, the EF variant exhibits significantly larger run times – presumably due to the size of the LP formulation – than those observed for the Match, STI, or 3-bin variants.

Overall, 3-bin variant is not competitive with the Match and STI variants, and Gurobi times out for one instance. Using CPLEX, the 3-bin variant is often the best or second-best, but when it performs poorly 3-bin often takes much longer than the Match and STI variants.

Comparing the Match and STI variants, we can see that overall Gurobi performs better using the Match variant while CPLEX performs better using the STI variant. However, for both solvers, the Match variant has the lowest maximum time across the CAISO test instances, suggesting it may be a more robust UC formulation in practice. Additionally, for both the Match and STI variants, solution times generally grow with increases in reserve level; we refer to the detailed results in Appendix E. The latter observation has significant potential impact on stochastic unit commitment solvers, as we discuss further below in Section 4.7.

Turning to the number of branch-and-cut nodes explored, in the case of the 1-bin variants, the large number of nodes explored is consistent with the inability of the solver to identify optimal solutions within the specified time limit. Interestingly, Gurobi and CPLEX typically did not leave the root node processing phase within the 600 second time limit when considering the EF variant. Further, we note that the size of the EF formulation makes cut generation (and heuristics) at the

Table 4.4: Computational results for CAISO instances: Relative Integrality Gap (%), geometric mean across 20 instances.

Formulation	EF	Match	STI	3-bin	1-bin*	1-bin
Gap (%)	0.008	0.008	0.033	0.033	1.525	1.569

root node more difficult. In the case of the Match and STI variants, both Gurobi and CPLEX identify an optimal solution at the root node for instances with relatively low reserve levels, with CPLEX finding a root node solution more often. However, as reserve levels increase, the number of nodes explored increases. Finally, we observe that the relatively few number of tree nodes explored with the tighter Match and STI variants indicates relatively few opportunities for parallelism, at least in terms of accelerating the tree search process.

Finally, in Table 4.4 we report the relative integrality gap for each combination of startup cost formulation and instance. For each instance, we compute the relative integrality gap by taking the best integer solution objective value found across all six formulations and both solvers, denoted z^*_{IP}, and the objective value of the LP relaxation for each instance (as computed by Gurobi after relaxing the binary variables), denoted z^*_{LP}; we then report $(z^*_{IP} - z^*_{LP})/z^*_{IP}$ as a percentage. First, we observe that the results in Table 4.4 are consistent with and empirically verify the correctness of Theorem 4.1. The 1-bin variants are significantly weaker than the other variants, with the relative integrality gap typically exceeding 1%. We also note that in all instances, the relative integrality gap (and hence LP relaxation) for the EF and Match variants is identical; an analogous situation is observed for the STI and 3-bin variants. Lastly, we note that the Match formulation typically closes 50-90% of the integrality gap relative to STI (74% in geometric mean), which explains its computational benefit despite the additional variables required.

4.6.2 FERC Instances

Because the FERC instances are larger and therefore likely more difficult than the CAISO instances, we increased the wall-clock time limit to 900 seconds. Further, for Gurobi we set the `Method` parameter to 3 so Gurobi would use the non-deterministic concurrent optimizer to solve the root LP relaxations. The non-deterministic concurrent optimizer solves LPs by running primal and dual simplex on one thread each and a barrier plus crossover method on the remaining 14 threads, returning an optimal LP basis from whichever method returns first. All other settings for Gurobi were left at their defaults. CPLEX settings were preserved at their defaults. When describing the

Table 4.5: Summary of computational experiments for FERC Instances using Gurobi. For time (s) and number of branch-and-cut (B&C) nodes we report the geometric mean across the 12 instances, including those which hit the wall-clock limit of 900 seconds.

(a) 2% Wind Penetration

Formulation	EF	Match	STI	3-bin	1-bin*	1-bin
Time (s)	702	**154**	218	267	712	739
# of times best	0	**6**	4	2	0	0
# of times 2nd	0	**6**	5	1	0	0
Max. time (s)	900	**411**	491	841	900	900
# of time outs	4	**0**	**0**	**0**	7	7
# of B&C nodes	1.00	**1.38**	5.91	9.03	67.5	50.8

30% Wind Penetration

Formulation	EF	Match	STI	3-bin	1-bin*	1-bin
Time (s)	808	**215**	391	401	799	804
# of times best	0	**8**	2	2	0	0
# of times 2nd	2	1	**6**	3	0	0
Max. time (s)	900	**648**	900	900	900	900
# of time outs	6	**0**	2	3	10	10
# of B&C nodes	1.00	**4.66**	51.7	78.2	142	130

computational results below, we separate the instances into two categories: the results considering the 2% wind penetration levels observed in 2015 and hypothetical 30% wind penetration levels based on the same data.

In Tables 4.5 and 4.6 we summarize the computational experiments for both Gurobi and CPLEX for the FERC instances. Tables 4.5 and 4.6 report the same statistics for the FERC instances as Tables 4.2 and 4.3 did for the CAISO instances. First, we consider the 2% wind penetration instances, which are reported in part (a) of both tables. We observe that the 1-bin variants and EF are not competitive with the Match and STI variants. As was the case with the CAISO instances, Match performs best with Gurobi, whereas STI performs better with CPLEX. CPLEX does not find these instances difficult, solving all 12 problems using the Match and STI variant at the root node. The 3-bin variant is occasionally the fastest method for CPLEX for a given instance, but mirroring the CAISO instances it has a significantly inferior worse-case solve time than either Match or STI.

Next, we consider the FERC instances with 30% wind penetration levels. Here, we see that only the Match variant able to solve all 12 instances within the time limit on both solvers. Considering the solve time geometric mean, we observe that the Match variant reduces the solve time relative

Table 4.6: Summary of computational experiments for FERC Instances using CPLEX. For time (s) and number of branch-and-cut (B&C) nodes we report the geometric mean across the 12 instances, including those which hit the wall-clock limit of 900 seconds.

(a) 2% Wind Penetration

Formulation	EF	Match	STI	3-bin	1-bin*	1-bin
Time (s)	478	136	**114**	162	499	538
# of times best	0	2	**5**	**5**	0	0
# of times 2nd	0	4	**6**	2	0	0
Max. time (s)	900	222	**150**	737	900	900
# of time outs	1	**0**	**0**	**0**	4	5
# of B&C nodes	1.23	**1.00**	**1.00**	7.52	356	414

30% Wind Penetration

Formulation	EF	Match	STI	3-bin	1-bin*	1-bin
Time (s)	604	**185**	211	298	784	798
# of times best	0	**5**	2	**5**	0	0
# of times 2nd	1	3	**8**	0	0	0
Max. time (s)	900	**269**	900	900	900	900
# of time outs	2	**0**	1	3	10	10
# of B&C nodes	1.95	**1.94**	5.91	25.3	1171	1153

to the STI variant by 45% for Gurobi, with a more moderate reduction for CPLEX. Overall, the 30% wind penetration level instances are noticeably more difficult than the 2% wind penetration instances. However, for Gurobi, the Match variant requires only 40% more computational time on average to solve the former, while the STI variant requires more than 80% additional computational time. The situation is similar for CPLEX, where the Match variant only needs 36% more computational time on average for 30% wind instances, whereas STI variant needs 85% more computational time.

Next, we consider the number of branch-and-cut tree nodes explored when solving each instance. Looking at Table 4.5, we observe that for the Match variant, Gurobi typically locates an optimal solution at the root node, or at least relatively early in the tree search process. Overall, the number of tree nodes explored under the Match variant is significantly less than that under the STI variant; the latter in turn dominates, as expected, the 3-bin and 1-bin variants. Mirroring the results for CAISO instances, Gurobi does not exit root node processing on the EF variant – in all cases the root relaxation is solved, but the time limit is exhausted applying cuts and heuristics. For both 1-bin variants, Gurobi spends a significant amount of time during root node processing generating

Table 4.7: Computational results for FERC instances: Relative Integrality Gap (%), geometric mean across each of the 12 instances.

(a) 2% Wind Penetration

Formulation	EF	Match	STI	3-bin	1-bin*	1-bin
Gap (%)	0.068	0.068	0.075	0.075	0.911	0.911

(b) 30% Wind Penetration

Formulation	EF	Match	STI	3-bin	1-bin*	1-bin
Gap (%)	0.206	0.206	0.314	0.314	3.003	3.003

cuts, which is why for some instances a small number of nodes are explored before the time limit expires. Consistent with the increase in relative instance difficulty, Gurobi requires more nodes to identify an optimal solution in the case of 30% wind instances, but the increase is much less pronounced than for the STI or 3-bin variants.

Examining the node count summaries for CPLEX in Table 4.6, we observe that using the Match and STI variants the 2% wind instances are easy, never leaving the root node. Similar to the experience with Gurobi, for 30% wind instances there is only a modest increase in node count for the Match variant (only one instance does not solve at the root node), and larger but still modest increases for STI and 3-bin variants. When it solves, the EF variant does so at the root node, and the 1-bin variants need more enumeration, and usually hit the time limit while still exploring the tree.

Finally, we report the relative integrality gap for each combination of instance and variant in Table 4.7, calculated in the same manner as those reported in Table 4.4. Like the results for the CAISO instances, we again observe empirical verification of Theorem 4.1. Further, the EF and Match variants have identical integrality gaps, as do the STI and 3-bin variants. Unlike as was observed for the CAISO instances, the 1-bin* variant is not significantly tighter than the 1-bin variant, and the Match variant typically only closes 0%-40% of the root gap over STI. This result is partially explained by the fact that approximately half of the generators are ramp-constrained – and even in the EF case, we are not using an ideal formulation for ramp-constrained generators. However, for the January and February 30% wind penetration level instances the difference is significant (Match closes 95% of the root gap for January and 67% of the root gap for February over STI, see Appendix E), which is consistent with the result that only the Match and EF variants were able to solve these instances within the time limits on both solvers. For the 2% wind penetration

instances the Match variant only closes 8% of the relative integrality gap on average versus STI. Interestingly Table 4.5 shows the Match variant is still computationally competitive for Gurobi, but STI is able to outperform Match using CPLEX because the extra variables are not providing much in the way of additional tightness.

4.6.3 Statistical Analysis

A statistical analysis of the computational results was performed using the Wilcoxon signed-rank test [158] for both Gurobi and CPLEX. On Gurobi, across the entire test set (both CAISO and FERC), Match is superior to all the other formulations examined at the $\alpha = 0.01$ level. On the other hand, using CPLEX, though Match was faster than STI and 3-bin in mean solve time, these differences were not significant at the $\alpha = 0.05$ level. Further, on the "Low Wind" instances (those being the 2% Wind Penetration instances from FERC and the instances corresponding to historical dates from CAISO), the STI variant is able to outperform Match at the $\alpha = 0.01$ level, though the difference in magnitude is only 16.8 seconds. Finally, we note that for the "High Wind" instances (Scenario400 instances from CAISO and 30% Wind Penetration instances from FERC) Match is 60.4 seconds faster in mean solve time, but this difference was not significant at the $\alpha = 0.05$ level. This is likely due to this test being underpowered at $n = 16$. The full results of the statistical analysis are available in Appendix E.

4.7 Discussion

We now discuss the implications of the computational experiments described above. First, we note that both the CAISO and FERC test instances have $S_g \leq 2$ for all generators g, due to the data available. In real-world instances, a non-trivial number of generators may have $S_g > 2$. Our proposed Match formulation of startup costs in UC can model more startup categories – up to TC_g – by simply changing the objective coefficients. In contrast, with the exception of EF, all other startup cost formulations require additional variables and/or constraints.

As the experiments on the CAISO instances demonstrate, reserve requirements do have a significant impact on the difficulty of solving UC. For instances with a 0% or 1% reserve requirement, Gurobi is able to solve all instances using the Match formulation in under a minute. This is an interesting observation in the context of stochastic unit commitment [149], in which reserve levels for individual scenarios are minimal, as the scenarios themselves are intended to capture the range

of uncertainties that may be encountered. Further, we note that effective decomposition techniques for solving stochastic UC problems – including progressive hedging [20] – repeatedly solve individual (and thus deterministic) scenario problems. Thus, we expect our Match formulation to significantly accelerate the solution of stochastic UCs.

Our experiments also demonstrate that with a modern MILP solver, commodity workstation hardware, and tight formulations, we can quickly solve utility-scale UC problems to very small ($< 0.01\%$) optimality gaps. In fact, the results for the CAISO instances suggest they could be solved to even tighter gaps than the Gurobi and CPLEX defaults, within the imposed time limit. Reduced optimality gaps are important to guarantee market fairness, i.e., to ensure that a cheaper generator is scheduled in place of a more expensive one. The ability to run to very small optimality gaps is also important in the context of scenario-based decomposition approaches to stochastic UC. Deterministic UC scenarios often have feasible solutions which are far away from optimal. Thus, imposition of a tighter optimality gap can significantly improve convergence of algorithms such as progressive hedging, by providing strong initial solutions of individual scenarios – which are used to guide subsequent iterations of the algorithm. Further, the ability of progressive hedging to generate high-quality lower bounds for stochastic UC is dependent on how tightly the scenario UC problems are solved [20, 53].

In the context of stochastic UC – where renewable energy supply is the main driver of uncertainty – it is interesting to note that for both systems the high-wind scenarios ("Scenario400" for CAISO and the 30% wind penetration instances for FERC) are significantly more difficult, independent of startup formulation. However, these instances are also where our proposed Match formulation shows the most improvement over STI. Across all 16 high-wind scenarios, using Gurobi the Match variant exhibited a >44% improvement in geometric mean solve time, with a more modest >15% improvement using CPLEX, and a 57% improvement in geometric mean relative integrality gap. In comparison, on the other 28 instances, using Gurobi the Match variant only showed a 20% improvement in geometric mean solve time, with a 20% degradation using CPLEX, and a 50% geometric mean relative integrality gap closure over STI. This is not surprising, given that more variability in net-load implies there will be more switches in generator status.

Finally, we comment on the "synthetic" UC instances from [17], which are extended via replication in [119] and again in [110]. These originate from a now dated genetic algorithm UC paper [79], which has no indication that these were drawn from real-world data. Compared to the generator sets gathered from CAISO and FERC, these instances have much less flexible capacity

(less than 10% in all cases), which implies that the ramping process is a much bigger factor in adjusting to changes in demand than generator switching. Additionally, the replication of the same 8 or 10 generators induces artificial symmetry into the problem, which can confound the branch-and-cut process. Though modern commercial MILP solvers have sophisticated symmetry detection, they do not capture all the symmetry in UC [120]. These factors together imply that the synthetic instances are less likely to be impacted by improvements in startup cost formulations. Based on the instances in [119], we created twenty 48-hour unit commitment problems, and tested the six startup cost formulations on the platform described in Section 4.6 using Gurobi. After 1800 seconds of wall-clock time, the Match, STI, and 3-bin variants were able to solve only 6 of the 20 instances, the EF variant was able to solve 2 of the 20 instances, and the 1-bin variants only 1 of the 20 instances. In geometric mean Match was only able to close 5% of the relative integrality gap over STI. The confounding symmetry and inflexibility in these instances makes it difficult to draw a distinction between the Match, STI, and 3-bin variants, though they all out-perform the 1-bin, 1-bin*, and EF variants.

4.8 Conclusions

We have presented a novel matching formulation for time-dependent startup costs in UC, and an additional compact formulation for time-dependent startup costs as an intermediary between the STI and the 1-bin formulations. We have formally placed these two new formulations, in addition to existing alternatives, in a formal dominance hierarchy based on the corresponding LP relaxations. We examined the computational efficacy of the various alternative formulations for time-dependent startup costs on large-scale unit commitment instances based on real-world data from the PJM and CAISO independent system operators in the US using two commercial MILP solvers. We find that the proposed matching formulation is computationally as effective on average than the current state-of-the-art formulation, and is computationally more effective for high-wind penetration scenarios. Additionally, we empirically demonstrated that the proposed matching formulation is as tight as the ideal formulation while being more compact.

Chapter 5

Exploiting Identical Generators in Unit Commitment

This chapter and Appendices F and G are based on a manuscript prepared for publication by Ben Knueven, Jim Ostrowski, and Jean-Paul Watson:

> Knueven, B., Ostrowski, J., and Watson, J. P. (2017). Exploiting Identical Generators in Unit Commitment. *Submitted*.

Author Knueven proposed the exact aggregation approach. Authors Knueven and Ostrowski developed the theoretical results. Authors Knueven and Watson programmed the computational experiments; author Knueven performed all computational experiments and data analysis. Author Knueven wrote the initial draft of the manuscript; authors Ostrowski and Waston revised and edited the manuscript. A preprint of this paper is available at http://www.optimization-online.org/DB_FILE/2017/06/6112.pdf.

This chapter presents sufficient conditions under which thermal generators can be aggregated in mixed-integer linear programming (MILP) formulations of the unit commitment (UC) problem, while maintaining feasibility and optimality for the original disaggregated problem. Aggregating thermal generators with identical characteristics (e.g., minimum/maximum power output, minimum up/down-time, and cost curves) into a single unit reduces redundancy in the search space caused by both exact symmetry (permutations of generator schedules) and certain classes of mutually non-dominated solutions. We study the impact of aggregation on two large-scale UC instances, one from the academic literature and another based on real-world data. Our computational tests demonstrate that when present, identical generators can negatively affect the performance of modern MILP

solvers on UC formulations. Further, we show that our reformation of the UC MILP through aggregation is an effective method for mitigating this source of difficulty.

5.1 Nomenclature

5.1.1 Indices and Sets

$g \in \mathcal{G}$	Thermal generators.
$t \in \mathcal{T}$	Hourly time steps: $1, \ldots, \mathbf{T}$.
$[t, t') \in \mathcal{Y}^g$	Feasible intervals of operation for generator g with respect to its minimum uptime, that is, $[t, t') \in \mathcal{T} \times \mathcal{T}$ such that $t' \geq t + \mathbf{UT}^g$.

5.1.2 Parameters

$\mathbf{D}_t$	Load (demand) at time t (MW).
$\mathbf{DT}^g$	Minimum down time for generator g (h).
$\overline{\mathbf{P}}^g$	Maximum power output for generator g (MW).
$\underline{\mathbf{P}}^g$	Minimum power output for generator g (MW).
$\mathbf{RD}^g$	Ramp-down rate for generator g (MW/h).
$\mathbf{RU}^g$	Ramp-up rate for generator g (MW/h).
$\mathbf{SD}^g$	Shutdown ramp rate for generator g (MW/h).
$\mathbf{SU}^g$	Startup ramp rate for generator g (MW/h).
$\mathbf{TC}^g$	Time down after which generator g goes cold (h).
$\mathbf{UT}^g$	Minimum up time for generator g (h).

5.1.3 Variables

p_t^g	Power above minimum for generator g at time t (MW).
u_t^g	Commitment status of generator g at time t, $\in \{0, 1\}$.
v_t^g	Startup status of generator g at time t, $\in \{0, 1\}$.
w_t^g	Shutdown status of generator g at time t, $\in \{0, 1\}$.
$y_{[t,t')}^g$	Indicator arc for startup at time t, shutdown at time t', committed for $i \in [t, t')$, for generator g, $\in \{0, 1\}$, $[t, t') \in \mathcal{Y}^g$.

5.2 Introduction

Unit commitment (UC) is a core optimization problem in power systems operations, in which the objective is to determine an on/off schedule for thermal generating units that minimizes production costs while satisfying constraints related to generator performance characteristics and power flow physics [160]. UC is now widely modeled as a mixed-integer linear program (MILP) and solved using commercial branch-and-cut technologies, e.g., those available in the Gurobi [66] and CPLEX [69] software packages. Due to its criticality, significant research has been dedicated over the past 15 years toward improving the quality of MILP formulations of UC, specifically focusing on the strength of the associated LP relaxation – as this is strongly correlated with computational difficulty. Recent examples of the progress in state-of-the-art of UC formulations are reported in [119, 110, 58, 109] and Chapter 3.

Most of the UC research to date has focused on the analysis of generator ramping and startup cost polytopes. An alternative and orthogonal approach, however, considers the impact of symmetry in the UC MILP model induced by the presence of multiple generators with identical physical performance characteristics and other properties. Here, we consider two generators to be identical if they have identical performance parameters and cost coefficients. We will show that initial status can be ignored for our purposes. As we discuss subsequently, such identical generators are found in both academic UC instances and those based on real-world data. Further, developing an understanding of exact symmetry is a necessary first step toward developing formulations that consider partial symmetry, which is more pervasive in practice.

Aggregating thermal generators with identical characteristics into a single unit reduces redundancy in the search space caused by both exact symmetry (permutations of generator schedules) and certain classes of mutually non-dominated solutions – both of which can cause performance issues when commercial MILP solvers are employed to solve UC [120, 87]. Further, degeneracy in the solution space induced by symmetry is known to cause convergence difficulties for decomposition-based solvers for stochastic UC, e.g., see [19].

In the context of exact MILP-based solution methods for UC, one approach to addressing identical generator symmetry is through the introduction of advanced branching strategies when exploring the branch-and-bound search tree. For example, [120] introduces "modified orbital branching," which strengthens orbital branching for cases when a problem's symmetry group contains additional structure, as is the case with UC [123]. While such an approach has promise, its

implementation is non-trivial and their are issues in practice. Further, when using CPLEX callbacks many of the solver's advanced features are disabled, which can result in slower solve times overall. Further, the approach is not possible to implement in Gurobi as the callback interface does not allow the user to access or decide on a branching variable [66].

Another approach to addressing the presence of symmetry in UC MILP formulations involves the introduction of symmetry-breaking inequalities. For example, [87] adds static symmetry-breaking inequalities to a UC MILP formulation. Their approach eliminates some, but not all of, of the redundancy in the branch-and-bound tree induced by symmetry, leading to faster computational times overall for highly symmetric instances. However, symmetry-breaking inequalities have the disadvantage that they increase the size of the LP relaxation. We compare their inequalities to the method proposed in this chapter in Appendix G.

In this chapter we propose addressing identical generators in UC MILP formulations through a novel aggregation approach. While this idea is certainly not new (e.g., see [141, 54, 143, 124]), we introduce conditions under which such an aggregation can be done exactly – that is, after simple post-processing we have a provably optimal solution to the original disaggregated problem. Our results are a consequence of recent advances in convex hull formulations for commitment of a single generator, see [94, 86, 133, 58] and Chapter 3. We provide conditions under which "orbital shrinking" [41] can be performed exactly for UC. Finally, we show that our approach can significantly reduce the solve times required for large-scale UC instances, considering a state-of-the-art UC MILP formulation.

The remainder of this chapter is organized as follows. In Section 5.3 we review the UC problem, and demonstrate how naive aggregation can result in an infeasible solution to the original disaggregated UC problem. Section 5.4 explores sufficient conditions for feasible and optimal disaggregation and introduces simple algorithms for disaggregation. These base results are expanded for more cases in Appendix F. Section 5.5 considers how the presence of multiple identical generators can result in symmetric and non-symmetric solutions with the same objective function value, and how aggregation models can express these solutions concurrently. In Section 5.6 we present computational results for two sets of large-scale UC instances, one from the academic literature and another based on real-world data. We then conclude in Section 5.7 with a summary of our contributions.

5.3 Unit Commitment Formulation

UC MILP formulations reported in the literature are typically expressed in the general form

$$\min \sum_{g \in G} \mathbf{c}^g(p^g) \tag{5.1a}$$

subject to

$$\sum_{g \in \mathcal{G}} p_t^g = \mathbf{D}_t \qquad \forall t \in \mathcal{T} \tag{5.1b}$$

$$p^g \in \Pi^g \qquad \forall g \in \mathcal{G} \tag{5.1c}$$

where $\mathbf{c}^g(p^g)$ denotes the costs associated with thermal generator g producing a (vector) output of p^g over the scheduling horizon and Π^g denotes the set of feasible schedules for generator g. When it is clear from context that we are referencing a single generator, we drop the superscript g on parameters and variables. A significant portion of the UC literature focuses on how to model the feasible sets Π^g, where there is a trade-off between the model size versus the tightness of the formulation.

For the purposes of exposition we will focus on the simplified version of UC above, but in Appendix F we go into more detail, including spinning reserves, piecewise linear operating costs, and time-dependent startup costs.

The ability to aggregate identical generators is dependent on which UC MILP formulation is considered. For instance, consider the basic "3-bin" formulation for a generator [17, 119]:

$$\underline{\mathbf{P}} u_t \leq p_t \leq \overline{\mathbf{P}} u_t, \qquad \forall t \in \mathcal{T} \tag{5.2a}$$

$$p_t - p_{t-1} \leq \mathbf{RU} u_{t-1} + \mathbf{SU} v_t, \qquad \forall t \in \mathcal{T} \tag{5.2b}$$

$$p_{t-1} - p_t \leq \mathbf{RD} u_t + \mathbf{SD} w_t, \qquad \forall t \in \mathcal{T} \tag{5.2c}$$

$$u_t - u_{t-1} = v_t - w_t, \qquad \forall t \in \mathcal{T} \tag{5.2d}$$

$$\sum_{i=t-\mathbf{UT}+1}^{t} v_i \leq u_t, \qquad \forall t \in [\mathbf{UT}, \mathbf{T}] \tag{5.2e}$$

$$\sum_{i=t-\mathbf{DT}+1}^{t} w_i \leq 1 - u_t, \qquad \forall t \in [\mathbf{DT}, \mathbf{T}] \tag{5.2f}$$

$$p_t \in \mathbb{R}_+,\ u_t,\ v_t,\ w_t \in \{0,1\} \qquad \forall t \in \mathcal{T} \tag{5.2g}$$

where Constraints (5.2a) enforce minimum / maximum generator output, Constraints (5.2b, 5.2c) enforce ramping limits, Constraints (5.2d) enforce logical constraints on u, v, and w, and Constraints (5.2e, 5.2f) enforce minimum up / down times.

Consider the case of two generators with identical performance and cost parameters. We can always model this situation by treating each generator individually. However, it would be desirable if we could aggregate the generators to exploit the symmetric structure. Unfortunately, the above model does not present a straightforward way to accomplish this. Consider a 5 time period case where two generators each have $\underline{\mathbf{P}} = \mathbf{SU} = \mathbf{SD} = 100$, $\overline{\mathbf{P}} = 200$, and $\mathbf{RU} = \mathbf{RD} = 50$. Ideally, we would prefer to let the u, v, and w variables represent how many of the generators remain on, are turned on, and are turned off at a given time. Suppose, then, that we use formulation (5.2) but allow u_t, v_t, and w_t variables to take values in $\{0,1,2\}$. Then, consider the following feasible solution to this simple aggregated UC model:

$$U = \begin{pmatrix} 1 \\ 2 \\ 2 \\ 2 \\ 1 \end{pmatrix}, P = \begin{pmatrix} 200 \\ 300 \\ 400 \\ 300 \\ 200 \end{pmatrix}. \tag{5.3}$$

The solution to the aggregated model is clearly not feasible in the disaggregated model, as both generators must produce at full capacity in time period 3, but the generator that started up at time 2 cannot ramp to full capacity by time period 3. The problem with this naive aggregated formulation is that when one generator is operating at full capacity, the other is able to "steal" its ramping capability. Unfortunately, tighter descriptions in the 3-bin space do not overcome this problem. Hence a necessary condition for exact aggregation of the 3-bin model is for the generators to have redundant ramping constraints.

Some UC MILP formulations do allow for variable aggregation without violating the structure of the base problem. For example, consider the formulation

$$\mathbf{A}^{[a,b)} p^{[a,b)} \leq \mathbf{b}^{[a,b)} y_{[a,b)} \qquad \forall [a,b) \in \mathcal{Y} \tag{5.4a}$$

$$\sum_{[a,b)\in\mathcal{Y}} p_t^{[a,b)} = p_t \qquad \forall t \in \mathcal{T} \tag{5.4b}$$

$$\sum_{\{[a,b)\in\mathcal{Y} | t\in[a,b+\mathbf{DT})\}} y_{[a,b)} \leq 1 \qquad \forall t \in \mathcal{T} \tag{5.4c}$$

$$y_{[a,b)} \in \{0,1\} \qquad \forall [a,b) \in \mathcal{Y}, \tag{5.4d}$$

where the polytope $\mathcal{P}^{[a,b)} = \{p^{[a,b)} \in \mathbb{R}_+^T \mid \mathbf{A}^{[a,b)} p^{[a,b)} \leq \mathbf{b}^{[a,b)}\}$ describes the feasible production of the generator if it is turned on at time a, turned off at time b, and consistently on during the interval $[a,b)$. Constraints (5.4a) enforce the appropriate ramping and minimum/maximum power output given that the generator is on during the $[a,b)$ time interval. Constraints (5.4c) ensure that minimum up and downtime constraints are met. While the above formulation is large (it contains $O(|\mathcal{T}|^3)$ many variables and constraints), it is provably tight (see Chapter 3). In particular, so long as the dispatch of the generator operating in the interval $[a,b)$ can be described as a polytope, this formulation is integer in y. We will see this is also a sufficient condition for exact aggregation using this extended formulation.

As observed in Chapter 3, Constraints (5.4c) correspond to clique inequalities for an interval graph. Because interval graphs are totally unimodular [62], it follows that the matrix given by Constraints (5.4c) is totally unimodular, and thus has the integer decomposition property [9]. Therefore, we can model k-many identical generators by letting the Y variables be general integers in the range $[0,k]$ and rewriting Constraints (5.4c) as

$$\sum_{\{[a,b)\in\mathcal{Y} | t\in[a,b+\mathbf{DT})\}} Y_{[a,b)} \leq k \qquad \forall t \in \mathcal{T}. \tag{5.5}$$

In this context, $Y_{[a,b)}$ represents *how many* of the generators are on during the interval $[a,b)$.[1] Because there are separate power variables for each on-interval $[a,b)$, this formulation overcomes the problems seen in (5.3).

Note that the size of the above formulation is heavily dependent on generator minimum up and down times. For generators with small minimum up and down times, the above formulation is very large, so much so that the benefits of a tight model are outweighed by model size. However, for generators with moderate minimum up and downtimes (say 8 hours), the above model is quite tractable.

[1] We use capital letters to represent aggregated variables.

Given that we see many generators with small minimum up and downtimes (say 2 hours each), it is worthwhile to ask *when* the 3-bin model is decomposable. As the example in (5.3) suggests, the problem is with the ramping. Consider the traditional 3-bin formulation for generators with redundant ramping constraints, when $\mathbf{UT} \geq 2$ [110]:

$$\underline{\mathbf{P}} u_t \leq p_t, \qquad \forall t \in \mathcal{T} \tag{5.6a}$$

$$p_t \leq \overline{\mathbf{P}} u_t + (\mathbf{SU} - \overline{\mathbf{P}}) v_t + (\mathbf{SD} - \overline{\mathbf{P}}) w_{t+1} \qquad \forall t \in \mathcal{T} \tag{5.6b}$$

$$u_t - u_{t-1} = v_t - w_t \qquad \forall t \in \mathcal{T} \tag{5.6c}$$

$$\sum_{i=t-\mathbf{UT}+1}^{t} v_i \leq u_t \qquad \forall t \in [\mathbf{UT}, \mathbf{T}] \tag{5.6d}$$

$$\sum_{i=t-\mathbf{DT}+1}^{t} w_i \leq 1 - u_t \qquad \forall t \in [\mathbf{DT}, \mathbf{T}] \tag{5.6e}$$

$$p_t \in \mathbb{R}_+,\ u_t,\ v_t,\ w_t \in \{0,1\} \qquad \forall t \in \mathcal{T}. \tag{5.6f}$$

This formulation has the property that the constraint matrix defined by (5.6c, 5.6d, 5.6e) is totally unimodular [94], and so it too has the integer decomposition property [9]. We discuss the case when $\mathbf{UT} = 1$ in Appendix F for clarity of exposition here.

In any case, the total unimodularity of Constraints (5.4c) and (5.6c, 5.6d, 5.6e) only ensure that the on-off schedules can be decomposed, and tell us nothing about whether (and if so, how) the aggregated power output can be disaggregated into a feasible production schedule for each generator in the aggregation. In the following section we explore these issues.

5.4 Disaggregating Solutions

While the results from [9] give conditions for when a UC schedule can be decomposed, it does not suggest a constructive approach to performing the decomposition. We now outline how to decompose solutions to the aggregate UC formulation into individual generator schedules. We first provide theorems regarding the relationship between schedules and the power output of identical generators.

Theorem 5.1. *Consider identical generators $g_1, g_2 \in \mathcal{G}$, and assume their production costs are increasing and convex. Then there exists an optimal solution with $p_t^{g_1} = p_t^{g_2}$ or (inclusive) one of*

the nominal or startup/shutdown ramping constraints is binding for generator g_1 or g_2 for all times t for which they are both on.

Proof. By contradiction, consider an optimal schedule in which $p_t^{g_1} > p_t^{g_2}$ and with no binding nominal or startup/shutdown ramping constraints at time t. Then, we may decrease $p_t^{g_1}$ by epsilon and increase $p_t^{g_2}$ by epsilon without affecting feasibility. Furthermore, because production costs are increasing and convex, this new solution is no worse than the original. □

From Theorem 5.1, if two identical generators start up at time a and shut down at time b, then their power outputs in the interval $[a, b)$ are identical. Theorem 5.1 also applies to fast-ramping generators, i.e., generators that are not ramp-limited, as the lack of ramping constraints ensures if two identical generators are on in a given time period, then they must have the same power output. This result suggests that allowing u to be a general integer is also sufficient. The only exception to this rule is when there is a binding startup/shutdown rate.

Theorem 5.2. *Suppose generator g_1 is turned off at time t. If identical generator g_2 can also be turned off at time t, there exists an optimal solution where the generator that has been on for the least amount of time is turned off.*

Proof. Suppose identical $g_1, g_2 \in \mathcal{G}$ have been on at time t for at least $\mathbf{UT}(= \mathbf{UT}^{g_1} = \mathbf{UT}^{g_2})$ time periods, starting at time t_0, and that generator g_1 has been on longer. If generator g_1 is turned off at time t, then there exists a t' with $t_0 \leq t' \leq t$ such that $p_{t'-1}^{g_2} \leq p_{t'-1}^{g_1}$ and $p_{t'}^{g_1} < p_{t'}^{g_2}$. Notice that permuting g_1 and g_2 for all $t \geq t'$ does not affect the objective value, and does not change the power output. Finally, the permuted solution satisfies the ramping constraints since $p_{t'}^{g_1} - p_{t'-1}^{g_2} < p_{t'}^{g_2} - p_{t'-1}^{g_2}$ and $p_{t'}^{g_2} - p_{t'-1}^{g_1} \leq p_{t'}^{g_2} - p_{t'-1}^{g_2}$, satisfying ramp up, and $p_{t'-1}^{g_1} - p_{t'}^{g_2} < p_{t'-1}^{g_1} - p_{t'}^{g_1}$ and $p_{t'-1}^{g_2} - p_{t'}^{g_1} \leq p_{t'-1}^{g_1} - p_{t'}^{g_1}$, satisfying ramp down. □

Theorem 5.3. *Suppose generator g_1 is turned on at time t. If an identical generator g_2 can also be turned on at time t, and there are no time-dependent startup costs for g_1 and g_2, then there exists an optimal solution where g_2 is turned on at t.*

Proof. Suppose identical $g_1, g_2 \in \mathcal{G}$ at time t have been off for at least $\mathbf{DT}(= \mathbf{DT}^{g_1} = \mathbf{DT}^{g_2})$ time periods. Then for all $t' \in \mathcal{T}$ such that $t' \geq t$, we can permute the schedules for g_1 and g_2 without changing the objective value or the power output, in which case g_2 is turned on at time t. □

Theorems 5.2 and 5.3 illustrate how we can interchange parts of a UC schedule that involve identical generators, and not lose optimality. We explore this issue further in Section 5.5. The implications of the above theorems also provide some strong direction for the characteristics a disaggregation method for aggregated UC schedules that maintains both feasibility and optimality. Before formalizing this procedure, we first introduce some additional notation. Suppose $\mathcal{K} \subset \mathcal{G}$ such that all generators in $\mathcal{K}$ have identical properties, except for initial status. We again use capital letters to represent aggregated variables, and the superscript $\mathcal{K}$ to represent the parameters shared among the generators (e.g., $\mathbf{DT}^{\mathcal{K}} = \mathbf{DT}^g$, $\mathbf{SD}^{\mathcal{K}} = \mathbf{SD}^g$, etc., for $g \in \mathcal{K}$). In this context, we now illustrate how to decompose schedules for both the extended formulation and the 3-bin formulation for fast-ramping generators.

5.4.1 Extended Formulation (EF)

Let $Y = \sum_{g\in\mathcal{K}} y^g$, $P = \sum_{g\in\mathcal{K}} p^g$, and $P^{[a,b)} = \sum_{g\in\mathcal{K}} p^{g,[a,b)}$ $\forall [a,b) \in \mathcal{Y}^{\mathcal{K}}$. Consider the aggregated extended formulation for the generators in $\mathcal{K}$:

$$\mathbf{A}^{[a,b)} P^{[a,b)} \leq \mathbf{b}^{[a,b)} Y_{[a,b)} \qquad \forall [a,b) \in \mathcal{Y}^{\mathcal{K}} \tag{5.7a}$$

$$\sum_{[a,b)\in\mathcal{Y}^{\mathcal{K}}} P_t^{[a,b)} = P_t \qquad \forall t \in \mathcal{T} \tag{5.7b}$$

$$\sum_{\{[a,b)\in\mathcal{Y}^{\mathcal{K}} | t\in[a,b+\mathbf{DT})\}} Y_{[a,b)} \leq |\mathcal{K}| \qquad \forall t \in \mathcal{T} \tag{5.7c}$$

$$Y_{[a,b)} \in \{0,\ldots,|\mathcal{K}|\} \qquad \forall [a,b) \in \mathcal{Y}^{\mathcal{K}}. \tag{5.7d}$$

Algorithm 5.1 demonstrates how to construct feasible schedules given a solution (Y^*, P^*) to (5.7), by "peeling-off" a feasible solution (y^g, p^g) for generator g and leaving behind a feasible solution $(\hat{Y}, \hat{P})$ to (5.7) for $\mathcal{K} \setminus \{g\}$. The essential logic of the method is to always take feasible startups when available (line 10), thus ensuring the remaining aggregated solution is feasible. Theorem 5.3 ensures optimality of this approach. Additionally, Theorem 5.1 allows us to assign to each generator on during an interval $[a,b)$ the average of the power output across all generators on during interval $[a,b)$ (line 9) while maintaining optimality and feasibility.

Note that having different initial conditions is not an issue as long as we extend $\mathcal{Y}^{\mathcal{K}}$ back to include intervals when the generators in $\mathcal{K}$ last started or last ran (adding duplicate intervals if necessary for two generators that started at the same time period but have different outputs at

Algorithm 5.1 (Peel Off EF) Constructs feasible generator schedules from a solution of (5.7).

Initialize all y^g, $p^{g,[a,b)}$ to 0.
Initialize all $\hat{Y}$ to Y^*, $\hat{P}^{[a,b)}$ to $P^{*[a,b)}$ respectively.
$t \leftarrow \min\{i \mid Y^*_{[i,j)} \geq 1\}$
while $t \leq \mathbf{T}$ **do**
5: $t' \leftarrow \min\{j \mid Y^*_{[t,j)} \geq 1\}$
$y^g_{[t,t')} \leftarrow 1$; $\hat{Y}_{[t,t')} \leftarrow Y^*_{[t,t')} - 1$;
for $i \in [t,t') \cap \mathcal{T}$ **do**
$p_i^{g,[t,t')} \leftarrow P_i^{*[t,t')} / Y^*_{[t,t')}$
$\hat{P}_i^{[t,t')} \leftarrow P_i^{*[t,t')} - p_i^{g,[t,t')}$
10: $t \leftarrow \min\{i \geq t' + \mathbf{DT} \mid Y^*_{[i,j)} \geq 1\}$
$\hat{P}_t \leftarrow \sum_{[a,b)\in\mathcal{Y}} \hat{P}_t^{[a,b)}, \ \forall t \in \mathcal{T}$
$p^g_t \leftarrow \sum_{[a,b)\in\mathcal{Y}} p_t^{g,[a,b)}, \ \forall t \in \mathcal{T}$

$t = 0$). As long as $\mathcal{P}^{[a,b)}$ is a polytope, we can always do the disaggregation step for power on lines 8 and 9. Hence $\mathcal{P}^{[a,b)}$ begin as polytope is a sufficient condition for exact aggregation of the extended formulation. Ancillary services can be handled in a similar fashion; see Appendix F for more details.

5.4.2 3-Bin for Fast-Ramping

Let $U = \sum_{g\in\mathcal{K}} u^g$, $V = \sum_{g\in\mathcal{K}} v^g$, and $W = \sum_{g\in\mathcal{K}} w^g$. We will first consider the aggregated 3-bin model for commitment status:

$$U_t - U_{t-1} = V_t - W_t \qquad \forall t \in \mathcal{T} \tag{5.8a}$$

$$\sum_{i=t-\mathbf{UT}+1}^{t} V_i \leq U_t \qquad \forall t \in [\mathbf{UT}^{\mathcal{K}}, \mathbf{T}] \tag{5.8b}$$

$$\sum_{i=t-\mathbf{DT}+1}^{t} W_i \leq |\mathcal{K}| - U_t \qquad \forall t \in [\mathbf{DT}^{\mathcal{K}}, \mathbf{T}] \tag{5.8c}$$

$$U_t, V_t, W_t \in \{0, \ldots, |\mathcal{K}|\} \qquad \forall t \in \mathcal{T} \tag{5.8d}$$

Algorithm 5.2 demonstrates how to disaggregate a solution (U^*, V^*, W^*) to (5.8) by constructing a feasible 3-bin schedule for a generator g and leaving a feasible solution $(\hat{U}, \hat{V}, \hat{W})$ to (5.8) after g is removed from $\mathcal{K}$. Similar to Algorithm 5.1, it essentially takes shutdowns whenever possible when on (lines 7 – 11) and startups whenever possible when off (lines 12 – 14). Thus, the schedule

is clearly feasible for g, and taking startups/shutdowns whenever possible ensures (5.8) remains feasible for $\mathcal{K} \setminus \{g\}$ (and so all the integer bounds decrease by 1), and as before, Theorems 5.2 and 5.3 establish optimality.

Algorithm 5.2 (PEEL OFF 3-BIN) Constructs feasible generator schedules from a solution of (5.8).

Initialize all u^g, v^g, w^g to 0.
Initialize all $\hat{U}$, $\hat{V}$, $\hat{W}$ to U^*, V^*, W^* respectively.
if $U_1^* \geq 1$ **then**
  $u_1^g \leftarrow 1$; $\hat{U}_1 \leftarrow U_1^* - 1$;
$t \leftarrow 2$
while $t \leq \mathbf{T}$ **do**
  if $u_{t-1}^g = 1$ **then**
    if $\sum_{i=t-\mathbf{UT}^{\mathcal{K}}+1}^{t-1} v_i = 0$ **and** $W_t^* \geq 1$ **then**
      $w_t^g \leftarrow 1$; $\hat{W}_t \leftarrow W_t^* - 1$;
    else
      $u_t \leftarrow 1$; $\hat{U}_t \leftarrow U_t^* - 1$;
  else ▷ $u_{t-1} = 0$
    if $\sum_{i=t-\mathbf{DT}^{\mathcal{K}}+1}^{t-1} w_i = 0$ **and** $V_t^* \geq 1$ **then**
      $u_t, v_t \leftarrow 1$; $\hat{U}_t \leftarrow U_t^* - 1$; $\hat{V}_t \leftarrow V_t^* - 1$;
  $t \leftarrow t + 1$;

Note that historical data can be leveraged after the first time period. If the generator is on at $t = 1$, then we can arbitrarily assign it a historical startup v_t. Similarly, if the generator is off at $t = 1$, we can assign a historical shutdown w_t. Because ramping constraints are not active, initial conditions can be arbitrarily assigned based on whether the generator is on or not. We expand this result and provide a proof of correctness – including the case with time-dependent startup costs – in Appendix F.

Consider the power output for an aggregated set of identical fast-ramping generators. Along with (5.8), we have the aggregated power $P = \sum_{g \in \mathcal{K}} p^g$ with the constraints for $\mathbf{UT}^{\mathcal{K}} \geq 2$:

$$\underline{\mathbf{P}}^{\mathcal{K}} U_t \leq P_t \qquad \forall t \in \mathcal{T} \qquad (5.9a)$$

$$P_t \leq \overline{\mathbf{P}}^{\mathcal{K}} U_t + (\mathbf{SU}^{\mathcal{K}} - \overline{\mathbf{P}}^{\mathcal{K}}) V_t + (\mathbf{SD}^{\mathcal{K}} - \overline{\mathbf{P}}^{\mathcal{K}}) W_{t+1}, \qquad \forall t \in \mathcal{T}. \qquad (5.9b)$$

Because (5.9) is a sum of constraints, it is clearly valid. Further, using the result from Algorithm 5.2 along with Theorem 5.1, we can construct a feasible and optimal disaggregation for power output. For simplicity suppose $\mathbf{SU}^{\mathcal{K}} = \mathbf{SD}^{\mathcal{K}}$; we handle the remaining two cases in Appendix F. If $u_t^g = 1$,

$v_t^g = 0$, and $w_{t+1}^g = 0$ then

$$p_t^g = \frac{P_t^* - \min\{\mathbf{SU}^{\mathcal{K}}, P_t^*/U_t^*\} \cdot V_t^* - \min\{\mathbf{SD}^{\mathcal{K}}, P_t^*/U_t^*\} \cdot W_{t+1}^*}{U_t^* - V_t^* - W_{t+1}^*}. \tag{5.10}$$

If the generator is just starting up such that $v_t^g = 1$, then $p_t^g = \min\{\mathbf{SU}^{\mathcal{K}}, P_t^*/U_t^*\}$, and similarly if shutting down ($w_{t+1}^g = 1$), then $p_t^g = \min\{\mathbf{SD}^{\mathcal{K}}, P_t^*/U_t^*\}$. If $u_t^g = 0$ then $p_t^g = 0$. Spinning reserves and piecewise linear costs can be disaggregated in a similar fashion, as can generators with $\mathbf{UT}^{\mathcal{K}} = 1$; see Appendix F for details.

Taken together then, we see that for a generators whose dispatch can be described using (5.6) (or the modified version thereof when $\mathbf{UT} = 1$), for exact aggregation it is sufficient to just consider this formulation with the u, v, and w variables being allowed to take on general integers, as described by equations (5.8) and (5.9). We also note that ancillary services, like reserves, can be handled in a similar fashion, so long as total power available for generation plus other services is describable by (5.6).

5.5 The Potential Impact and Benefit of Aggregation

Aggregating identical generators provides a way of efficiently exploiting the presence of symmetry in UC, leading to several computational advantages. First, UC instances with large numbers of identical generators may have alternative optimal solutions. Consider the case where two identical generators in a UC instance have different schedules. Permuting these schedules will lead to different solutions in a disaggregated UC model. If there are k generators of the same type, then there may be as many as $k!$ different optimal solutions. While a variety of methods can be used to combat the effects of symmetry in MILP models, these generally rely on symmetry-breaking cuts or clever branching, and experience has shown that explicitly aggregating symmetry away is the most successful way of exploiting symmetry – when possible.

We additionally observe that aggregate UC solutions may encode more than just symmetric solutions found by permuting generator schedules. For example, consider the U variables defined in formulation (5.3), and assume generators are not ramp-constrained. Assuming $\mathbf{UT}^{\mathcal{K}} \leq 3$, there are two ways to feasibly disaggregate the on/off solution. The first is

$$u^{g_1} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, u^{g_2} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} \tag{5.11}$$

and the other is:

$$u^{g_1} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}, u^{g_2} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}. \tag{5.12}$$

These solutions are not symmetric, but they do have identical objective function values. As is the case with symmetric solutions, such mutually non-dominating solutions may lead to more tree exploration in the branch-and-cut process. Aggregation allows us to consider these solutions simultaneously.

Now, we further illustrate the disaggregation process. When $\mathbf{UT}^{\mathcal{K}} \leq 3$, Algorithm 5.2 decomposes the aggregate solution to formulation (5.3) – again assuming redundant ramping limits – to (5.11). Then, by (5.10) the power output associated with solution (5.11) is given as

$$p^{g_1} = \begin{pmatrix} 200 \\ 200 \\ 200 \\ 200 \\ 200 \end{pmatrix}, p^{g_2} = \begin{pmatrix} 0 \\ 100 \\ 200 \\ 100 \\ 0 \end{pmatrix}. \tag{5.13}$$

On the other hand, if $\mathbf{UT}^{\mathcal{K}} > 3$, then Algorithm 5.2 with equation (5.10) yields

$$p^{g_1} = \begin{pmatrix} 200 \\ 200 \\ 200 \\ 100 \\ 0 \end{pmatrix}, p^{g_2} = \begin{pmatrix} 0 \\ 100 \\ 200 \\ 200 \\ 200 \end{pmatrix}. \tag{5.14}$$

Finally, we discuss the impact of aggregation on market fairness. In practice, generation capacity is often not utility-owned. Algorithms 5.1 and 5.2 greedily schedule each generator, such that a generator disaggregated first may have a very different schedule than the last generator disaggregated. One possible way to address this problem is to randomly permute the order generators are disaggregated in, so that in the long run (over many days and hence many UC solves), no generator is preferred over another.

Further, the main purpose of Algorithms 5.1 and 5.2 is to demonstrate that under certain conditions, a feasible disaggregation always exists, and can be found in polynomial time. A practitioner could use other methods to disaggregate identical generators. For example, an auxiliary MILP could be formulated to solve the disaggregation problem with whatever objective function is desirable. One such objective function is to minimize the difference in run-time between the symmetric generators; another possibility is to minimize the difference in economic profit between the symmetric generators. In this way, aggregation could be exploited to make electricity markets fairer than the current practice of allowing the MILP solver to break these ties arbitrarily.

5.6 Computational Experiments

To test the effectiveness of our aggregation approach, we selected two unit commitment test sets from the literature. The first, described further in Chapter 4, is based on real-world data gathered from the California Independent System Operator (CAISO). Each of the twenty instances share a set of 610 thermal generators. We consider five 48-hour demand scenarios, crossed with static spinning reserve requirements that varied between 0%, 1%, 3%, and 5% of system demand. The scenarios labeled with dates correspond to real-world load profiles from the corresponding calendar date. In contrast, "Scenario400" is a hypothetical scenario where wind supply is on average 40% of demand (we use a "net-load" formulation for wind supply). We approximated the quadratic cost

Table 5.1: Ostrowski Instances: Generator Performance Data

Gen	$\overline{\mathbf{P}}$ (MW)	$\underline{\mathbf{P}}$ (MW)	**UT(DT)** (h)	**SU(SD)** (MW/h)	**RU(RD)** (MW/h)	**TC** (h)
1	455	150	8	150	225	14
2	455	150	8	150	225	14
3	130	20	5	20	50	10
4	130	20	5	20	50	10
5	162	25	6	25	60	11
6	80	20	3	20	60	8
7	85	25	3	25	60	6
8	55	10	1	20	135	2

curves provided with two piecewise segments. Each thermal generator has two startup categories. We refer to these instances as "CAISO" instances.

The other UC test set is taken from [119]. The instances in this test case – which we refer to as the "Ostrowski" instances – are constructed by replicating the thermal generators in UC instances originally introduced in [79] and [17]. These generators have been used as a baseline to create large UC instances through replication in much of the UC literature [12, 13, 45, 50, 47, 51, 70, 110, 162, 161, 40]. The parameters and cost curves used for the eight base generators are provided in Tables 5.1 and 5.2. In Table 5.2, a, b, and c denote the coefficients of the quadratic cost function such that $\mathbf{c}^p(p_t) = a^2 p_t + b p_t + c$; $\mathbf{c}^H$ and $\mathbf{c}^C$ respectively denote the hot- and cold-startup costs. The number of copies of each generator type in each of the twenty instances is specified in Table 5.3. The number of thermal generators in the Ostrowski instances ranges from 28 to 187. The demand curve for each instance is given as a percentage of total system capacity, as reported in Table 5.4, and for each instance the reserve level was fixed to 3% of system demand. We use a two segment piecewise approximation for production costs. In contrast to the CAISO instances, the Ostrowski instances consider only a 24 hour scheduling horizon.

We consider the base UC MILP formulation described in Chapter 4, which represents the state-of-the-art. The performance of our aggregation approach is analyzed relative to this baseline.

All computational experiments were conducted on a Dell PowerEdge T620 server with two Intel Xeon E5-2670 processors, for a total of 16 cores, 32 threads, and 256GB of RAM, running the Ubuntu 14.04.5 Linux operating system. The Gurobi 7.0.1 MILP solver [66] was used in all experiments, and the solver was allowed to use all 32 threads in each experimental trial.

Table 5.2: Ostrowski Instances: Generator Cost Data

Gen	c ($)	b ($/MW)	a ($/MW^2)	$\mathbf{c}^H$ ($)	$\mathbf{c}^C$ ($)
1	1000	16.19	0.00048	4500	9000
2	970	17.26	0.00031	5000	10000
3	700	16.60	0.00200	550	1100
4	680	16.50	0.00211	560	1120
5	450	19.70	0.00398	900	1800
6	370	22.26	0.00712	170	340
7	480	27.74	0.00079	260	520
8	660	25.92	0.00413	30	60

Table 5.3: Ostrowski Instances: Number of Generators of Each Type

	Generator								Total
Problem	1	2	3	4	5	6	7	8	Gens
1	12	11	0	0	1	4	0	0	28
2	13	15	2	0	4	0	0	1	35
3	15	13	2	6	3	1	1	3	44
4	15	11	0	1	4	5	6	3	45
5	15	13	3	7	5	3	2	1	49
6	10	10	2	5	7	5	6	5	50
7	17	16	1	3	1	7	2	4	51
8	17	10	6	5	2	1	3	7	51
9	12	17	4	7	5	2	0	5	52
10	13	12	5	7	2	5	4	6	54
11	46	45	8	0	5	0	12	16	132
12	40	54	14	8	3	15	9	13	156
13	50	41	19	11	4	4	12	15	156
14	51	58	17	19	16	1	2	1	165
15	43	46	17	15	13	15	6	12	167
16	50	59	8	15	1	18	4	17	172
17	53	50	17	15	16	5	14	12	182
18	45	57	19	7	19	19	5	11	182
19	58	50	15	7	16	18	7	12	183
20	55	48	18	5	18	17	15	11	187

5.6.1 CAISO Instances

Of the total 610 thermal generators in the CAISO instances, there is no symmetry among the 36 slow-ramping generators, but the 574 fast-ramping generators do have some non-trivial symmetry. Aggregation allows us to reduce these 574 fast-ramping generators to 429 aggregated generators,

Table 5.4: Ostrowski Instances: Demand (% of Total Capacity)

Time	1	2	3	4	5	6	7	8	9	10	11	12
Demand	71%	65%	62%	60%	58%	58%	60%	64%	73%	80%	82%	83%
Time	13	14	15	16	17	18	19	20	21	22	23	24
Demand	82%	80%	79%	79%	83%	91%	90%	88%	85%	84%	79%	74%

Table 5.5: CAISO: Selected Generator Performance Characteristics

Number Identical	$\overline{\mathbf{P}}$ (MW)	$\underline{\mathbf{P}}$ (MW)	c \$*10^3	b (\$/MW)*10^3	a (\$/MW^2)*10^3	$\mathbf{c}^C$ \$*10^3	$\mathbf{c}^H$ \$*10^3
8	106.3	47.835	2.19187	0.02498	0.0000581	3.189	4.252
7	3.1	0.93	0.00662	0.03977	0	0.093	0.124
6	1.4	0.35	0.00662	0.04452	0	0.07	0.098
5	100	45	2.06698	0.02498	0.0000593	3.0	4.0
5	3.5	1.05	0.00415	0.04206	0	0.0105	0.14
4	180	81	3.59086	0.02402	0.0000297	5.4	7.2
4	75	74.25	0.13711	0.02726	0	2.25	3.00
4	70	53.9	0.21868	0.02493	0.0000148	2.1	2.8
4	49.9	22.455	1.09041	0.02568	0.0000636	1.497	1.996
4	49.5	22.275	1.05231	0.02497	0.0000611	1.485	1.98
4	12.2	3.05	0.04126	0.04326	0	0.61	0.854
4	11.2	5.04	0.24371	0.0295	0.0000496	0.336	0.448
4	11	4.95	0.23929	0.02495	0.0000533	0.33	0.44
4	10.75	4.8375	0.23392	0.02495	0.0000517	0.3225	0.43
3	100	45	1.7298	0.02381	0.0000672	3.0	4.0
3	78	77.22	0.06097	0.02831	0	2.34	3.12
3	21.69	5.4225	0.07335	0.04326	0	1.0845	1.5183
3	21.6	21.384	-0.76024	0.06329	0	0.648	0.864
3	16.27	4.0675	0.05502	0.04326	0	0.8135	1.1389
3	3.4	1.02	0.00381	0.03977	0	0.102	0.136
3	1.3	0.6	-0.00728	0.04197	0	0.039	0.052

with the largest aggregation representing 8 physical generators. Overall, we obtained 36 aggregated generators. In Table 5.5 we report on a subset of the aggregated generators, specifically excluding those for which we can only aggregate two generators. The CAISO test set is dominated by flexible, fast-ramping generators, with $\mathbf{UT} = \mathbf{DT} = 1$, $\mathbf{TC} = 2$, $\mathbf{SU}, \mathbf{SD} \geq \overline{\mathbf{P}}$, and $\mathbf{RU}, \mathbf{RD} \geq (\overline{\mathbf{P}} - \underline{\mathbf{P}})$. Consequently, we omit this information for purposes of brevity. Similar to Table 5.2, a, b, and c denote coefficients for the quadratic cost function, while $\mathbf{c}^H$ and $\mathbf{c}^C$ respectively denote the hot- and cold-startup costs. Because we are not using the EF UC formulation, our aggregation approach allows us to reduce the size relative to the standard 3-bin formulation by 24%.

Table 5.6: Computational Results for CAISO UC Instances

	Time (s)		Nodes	
Instance	**3-bin**	**3-bin+A**	**3-bin**	**3-bin+A**
2014-09-01 0%	31.35	**14.25**	0	0
2014-12-01 0%	25.77	**12.38**	0	0
2015-03-01 0%	24.08	**14.27**	0	0
2015-06-01 0%	13.11	**8.50**	0	0
Scenario400 0%	27.29	**23.63**	0	0
2014-09-01 1%	20.52	**16.44**	0	0
2014-12-01 1%	38.48	**24.69**	95	**0**
2015-03-01 1%	21.75	**19.11**	0	0
2015-06-01 1%	39.87	**15.59**	47	**0**
Scenario400 1%	47.54	**44.63**	**0**	1438
2014-09-01 3%	81.47	**38.27**	**7**	122
2014-12-01 3%	65.01	**36.53**	1292	**125**
2015-03-01 3%	50.79	**25.04**	0	0
2015-06-01 3%	87.25	**41.23**	**0**	115
Scenario400 3%	131.28	**69.45**	2055	**880**
2014-09-01 5%	47.07	**30.95**	95	**7**
2014-12-01 5%	83.87	**66.90**	**1203**	3978
2015-03-01 5%	80.57	**21.65**	923	**0**
2015-06-01 5%	**26.99**	43.79	**0**	402
Scenario400 5%	**115.53**	118.51	**3867**	4225
Geometric Mean:	43.85	**27.55**		

In Table 5.6 we report the wall-clock time and number of branch-and-cut nodes explored before termination for the respective formulations, where "3-bin" denotes the UC formulation proposed in Chapter 4 and "3-bin+A" denotes the aggregation formulation for fast-ramping generators introduced in this chapter. We left all Gurobi parameter settings at their default value, such that the solver terminated when the optimality gap was less than or equal to 0.01%. We did not impose a time limit for these experiments. Despite their size, we observe that these instances are not difficult given the current state-of-the-art UC formulation and a modern commercial MILP solver. Yet, we do observe a geometric mean improvement of 37% in wall clock time with aggregation, across the twenty CAISO instances. Aggregation is only slower in only two of the twenty instances, and in both cases the difference is minimal.

Next, considering the number of nodes explored during the branch-and-cut search process, we see that neither formulation has an advantage. We conjecture this is likely due to role of Gurobi's incumbent-finding heuristics. Specifically, because the typical CAISO instance has a root gap of

Table 5.7: Computational Results for Ostrowski UC Instances

	Time (s)		Nodes	
Instance	**3-bin**	**EF/3-bin+A**	**3-bin**	**EF/3-bin+A**
1	**8.44**	14.02	1509	**68**
2	154.75	**21.07**	48129	**157**
3	703.94	**100.33**	316704	**4464**
4	**14.84**	17.28	8532	**60**
5	143.18	**57.22**	131320	**4350**
6	95.41	**28.00**	62394	**72**
7	(0.0238%)	**119.22**	535361*	**4854**
8	(0.0107%)	**71.00**	1378310*	**9267**
9	(0.0169%)	**125.63**	819798*	**12217**
10	(0.0327%)	**82.89**	751319*	**11549**
11	(0.0186%)	**18.76**	73976*	**1155**
12	(0.0240%)	**22.91**	42729*	**460**
13	(0.0266%)	**74.43**	41325*	**6464**
14	(0.0144%)	**19.75**	41469*	**15**
15	780.76	**39.63**	120599	**3091**
16	(0.0162%)	**90.31**	42102*	**2597**
17	154.37	**27.88**	2114	**1059**
18	(0.0121%)	**22.36**	60651*	**151**
19	(0.0195%)	**21.30**	42683*	**2436**
20	106.44	**18.46**	527	**0**
Geometric Mean:	>349.77	**38.03**		

<0.01% (see Chapter 4), Gurobi only needs to find a high-quality solution before terminating, and does not need to expend significant effort proving that a solution is optimal within optimality gap tolerance.

5.6.2 Ostrowski Instances

For the Ostrowski instances, generators 1 – 5 have non-redundant ramping constraints, so we use the extended formulation in our aggregation, and for generators 6 – 8 we use the 3-bin fast-ramping aggregation. In all cases there are no more than 8 generators in the aggregated model. Due to the difference in difficulty relative to the CAISO instances, we impose a time limit of 900 seconds for these experiments.

We report the results of experiments on the Ostrowski instances in Table 5.7, recording the terminating optimality gap in parentheses for cases when the time limit was reached. Even with a state-of-the-art UC MILP formulation (i.e., that of Chapter 4) and 900 seconds of wall-clock time,

Gurobi fails to establish optimality within tolerance for over half of the 20 instances. Given that the Ostrowski instances have half the number of time periods and far fewer thermal generators than the CAISO instances, one might expect these instances to be easier. However, as demonstrated in [120, 87] and Table 5.7, even a modern MILP solver with sophisticated, general symmetry detection routines cannot handle UC instances with large numbers of identical generators. In comparison, our aggregation approach significantly reduces the difficulty of these instances, to the point where they can be solved in at most two minutes of wall clock time. Further, our aggregation approach requires far fewer nodes during the branch-and-cut process, often by an order of magnitude or more. As reported in Appendix G, aggregation also outperforms the static symmetry breaking inequalities proposed in [87] for the 3-bin formulation.

5.7 Conclusion

We have shown that symmetry due to the presence of identical generators is present in both real-world and academic UC instances, and we posited an aggregation method that can mitigate computational issues induced by this symmetry. While modern MILP solvers possess sophisticated symmetry-detection technology, they are unable to address this form of UC symmetry. Our aggregation approach requires a fairly straightforward reformulation of the UC MILP, with an associated disaggregation method. Thus, our approach is viable in practice for addressing symmetry in UC.

Chapter 6

Conclusions

In this chapter we draw some conclusions and suggest directions for further research.

6.1 Detecting Almost Symmetries of Graphs

Chapter 2 introduced a branch-and-bound framework for almost symmetry detection on simple graphs. Graphs with a few hundred vertices exhibited much more almost symmetry than the minimum amount required by the Erdős-Rényi bound. While the computational results presented in Chapter 2 are not promising as a general method for almost symmetry detection, there are several possible avenues of investigation.

First, it may be possible to specialize the Hungarian algorithm for the specific matching problem solved to increase its efficiency. The bounding process could be improved by considering heuristics to find a lower bound on the clique number for the compliment of the permutations graph. Incumbent heuristics could be added to the branch-and-bound framework to improve performance. Additionally, almost symmetries do not need to be maximal to be of practical use, and in most contexts a heuristic would suffice.

The question that spurred the investigation of almost symmetries is still open. That is, the presence of symmetry in real-world integer programming problems is a major hindrance if not properly dealt with. Can almost symmetries cause a similar problem, and if so, what methods can be used to mitigate their presence?

6.2 The Unit Commitment Problem

Chapters 3, 4, and 5 investigated the unit commitment problem. Chapter 3 introduced an extended formulation for a generic generator in UC. This formulation is integral, but has the disadvantage of being large. Because the generator formulation is too large to usually be used directly in a UC formulation, a cutting plane procedure is proposed.

However, the cut-generation LP is still a large linear program, and solving it for more than a 24-hour time horizon may be problematic. Specialized methods, such a column generation, may make it possible to solve this large-scale LP faster, especially since most columns will be non-basic for an optimal cut.

Chapter 4 proposed a new formulation for time-dependent start-up costs. This formulation has more variables than is necessary to specify the problem, while being more compact than the full extended formulation. Chapter 4 also shows that large-scale unit commitment instances, especially with the improved formulation, often have very small relative integrality gaps – usually less than 1%. This is despite not employing the tight formulation for ramping-constrained generators investigated in Chapter 3. With cuts and presolve reductions a modern MILP solver can usually tighten the gap another order of magnitude.

Chapter 5 explored an aggregation approach for exact symmetry handling in unit commitment, using the formulations proposed in Chapters 3 and 4. Exploiting symmetry in this fashion can often result in large computational speedups. Further, the large extended formulation from Chapter 3 can be made practical in a UC instance if it represents several generators instead of one. One unexplored application of this exact aggregation approach is in capacity expansion problems, where often it is assumed there are at most 10s of different generators to choose from, and more than one type of each may be built.

Another consequence of Chapter 5 is a proof that the start-up cost formulation proposed in Chapter 4 is integer for objective vectors corresponding to increasing start-up costs. This is by Remark 1.1 and Theorem F.4. Taken together then, Chapters 4 and 5 suggest that deterministic UC is not a difficult problem computationally, at least as presented in this thesis. However, the addition of ACOPF constraints to represent the distribution network complicates things considerably. That being said, solving stochastic and robust versions of UC, which often use deterministic UC as a subproblem, are definitely within reach. Employing stochastic methods will become more of a necessity as more renewable generation comes on-line.

Table 6.1: Almost Symmetry in the CAISO Generators

opt% / feas%	Generators
Full Model	410
Aggregated	328
1% / 0%	295
1% / 1%	294
2% / 0%	288
2% / 1%	287
2% / 2%	286
5% / 0%	259
5% / 1%	256
5% / 2%	254
5% / 5%	251
10% / 0%	249
10% / 1%	245
10% / 2%	242
10% / 5%	239
10% / 10%	237

6.3 Future Work

Future research will entail bringing almost symmetries to bear on the unit commitment problem. Some work has already been completed in this line of investigation. Because almost symmetries of the unit commitment problem are simply almost symmetries of the power generators' parameters, it is easy to design heuristic methods for almost symmetry detection in unit commitment.

As an example, consider the CAISO generators used in Chapters 4 and 5. Recall this test set has 410 schedulable generators. In Table 6.1 we report the reduction in generators that exploiting almost symmetry allows. The column "opt% / feas%" reports the percentage we allow optimality and feasibility to be relaxed, respectively, and in the "Generators" column we report the number of aggregated generators in the reduced model. The "Aggregated" entry reports the dimension reduction allowed by the work in Chapter 5.

As we can see, even a simple heuristic approach to almost symmetry allows us to reduce the dimension of the unit commitment problem by another 30% over exact symmetry reductions, albeit at the cost of relaxing optimality and feasibility (by a controlled amount). However, as reported in this thesis, solving large-scale unit commitment instances exactly is usually not difficult, at least for deterministic problems. Using the formulation developed in Chapter 4 with the symmetry reduction techniques from Chapter 5, none of the 48-hour CAISO instances take more than two minutes to

solve on commodity hardware with commercial solver. Thus it is unlikely that further reductions would have much impact on the solve time. Additionally, there is no reason with a problem this easy to settle for an approximation.

Research along this line must then move on to consider more difficult versions of the unit commitment problem. Exploiting almost symmetries may be of value in unit commitment with the addition of ACOPF constraints and stochasticity caused by uncertainty in renewable output. Finally, there may be other structured problems, like unit commitment, for which it is easy to detect and exploit almost symmetry.

Bibliography

[1] Arlazarov, V., Zuev, I., Uskov, A., and Faradzhev, I. (1974). An algorithm for the reduction of finite non-oriented graphs to canonical form. {*USSR*} *Computational Mathematics and Mathematical Physics*, 14(3):195 – 201. 12, 14

[2] Arvind, V., Köbler, J., Kuhnert, S., and Vasudev, Y. (2012). Approximate graph isomorphism. In *Mathematical Foundations of Computer Science 2012*, pages 100–111. Springer. 20, 22

[3] Babai, L. (2016). Graph isomorphism in quasipolynomial time. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2016, pages 684–697, New York, NY, USA. ACM. 10, 20

[4] Babai, L., Kantor, W. M., and Luks, E. M. (1983). Computational complexity and the classification of finite simple groups. In *Foundations of Computer Science, 1983., 24th Annual Symposium on*, pages 162–171. IEEE. 10

[5] Babai, L. and Kucera, L. (1979). Canonical labelling of graphs in linear average time. In *Foundations of Computer Science, 1979., 20th Annual Symposium on*, pages 39–46. IEEE. 11

[6] Balas, E. (1979). Disjunctive programming. *Annals of Discrete Mathematics*, 5:3–51. 49, 70, 152, 153

[7] Balas, E. (1998). Disjunctive programming: Properties of the convex hull of feasible points. *Discrete Applied Mathematics*, 89(1):3–44. 49, 70, 152, 153

[8] Balas, E., Ceria, S., and Cornuéjols, G. (1993). A lift-and-project cutting plane algorithm for mixed 0–1 programs. *Mathematical Programming*, 58(1-3):295–324. 57, 61, 62

[9] Baum, S. and Trotter Jr, L. E. (1978). Integer rounding and polyhedral decomposition for totally unimodular systems. In *Optimization and Operations Research*, pages 15–23. Springer. 99, 100, 185, 187

[10] Bertsimas, D. and Tsitsiklis, J. N. (1997). *Introduction to linear optimization*, volume 6. Athena Scientific Belmont, MA. 2

[11] Bonami, P., Lodi, A., Tramontani, A., and Wiese, S. (2015). On mathematical programming with indicator constraints. *Mathematical Programming*, 151(1):191–223. 152

[12] Borghetti, A., Frangioni, A., Lacalandra, F., Lodi, A., Martello, S., Nucci, C., and Trebbi, A. (2001). Lagrangian relaxation and tabu search approaches for the unit commitment problem. In *Power Tech Proceedings, 2001 IEEE Porto*, volume 3. 108

[13] Borghetti, A., Frangioni, A., Lacalandra, F., and Nucci, C. A. (2003). Lagrangian heuristics based on disaggregated bundle methods for hydrothermal unit commitment. *IEEE Transactions on Power Systems*, 18(1):313–323. 108

[14] Brandenberg, R., Huber, M., and Silbernagl, M. (2015). The summed start-up costs in a unit commitment problem. *EURO Journal on Computational Optimization*, pages 1–36. 74

[15] Buchheim, C. and Jünger, M. (2003). An integer programming approach to fuzzy symmetry detection. In *International Symposium on Graph Drawing*, pages 166–177. Springer. 23

[16] Carlson, B., Chen, Y., Hong, M., Jones, R., Larson, K., Ma, X., Nieuwesteeg, P., Song, H., Sperry, K., Tackett, M., et al. (2012). MISO unlocks billions in savings through the application of operations research for energy and ancillary services markets. *Interfaces*, 42(1):58–73. 49

[17] Carrion, M. and Arroyo, J. M. (2006). A computationally efficient mixed-integer linear formulation for the thermal unit commitment problem. *IEEE Transactions on Power Systems*, 21(3):1371–1378. 50, 74, 76, 91, 97, 108, 162

[18] Chen, Y. (2016). Personal correspondence. Principal Advisor at Midcontinent Independent System Operator. 73

[19] Cheung, K., Gade, D., Silva-Monroy, C., Ryan, S., Watson, J., Wets, R., and Woodruff, D. (2015a). Toward scalable stochastic unit commitment - part 2: Solver configuration and performance assessment. *Energy Systems*, 6(3):417–438. 95

[20] Cheung, K., Gade, D., Silva-Monroy, C., Ryan, S. M., Watson, J.-P., Wets, R. J.-B., and Woodruff, D. L. (2015b). Toward scalable stochastic unit commitment. *Energy Systems*, 6(3):417–438. 91

[21] Conforti, M., Cornuéjols, G., and Zambelli, G. (2014). *Integer Programming*, volume 271. Springer. 2, 4, 154

[22] Cook, W., Dash, S., Fukasawa, R., and Goycoolea, M. (2009). Numerically safe gomory mixed-integer cuts. *INFORMS Journal on Computing*, 21(4):641–649. 4

[23] Corneil, D. G. and Gotlieb, C. C. (1970). An efficient algorithm for graph isomorphism. *Journal of the ACM (JACM)*, 17(1):51–64. 11, 12, 13

[24] Cornuéjols, G., Margot, F., and Nannicini, G. (2013). On the safety of gomory cut generators. *Mathematical Programming Computation*, 5(4):345–395. 4

[25] Crawford, J., Ginsberg, M., Luks, E., and Roy, A. (1996). Symmetry-breaking predicates for search problems. *KR*, 96:148–159. 13

[26] Culberson, J., Johnson, D., Lewandowski, G., and Trick, M. (2015). Graph coloring instances. http://mat.gsia.cmu.edu/COLOR/instances.html. 35

[27] Damcı-Kurt, P., Küçükyavuz, S., Rajan, D., and Atamtürk, A. (2015). A polyhedral study of production ramping. *Mathematical Programming*, pages 1–31. 51, 54, 69, 74

[28] Darga, P. T., Liffiton, M. H., Sakallah, K. A., and Markov, I. L. (2004). Exploiting structure in symmetry detection for CNF. In *Proceedings of the 41st annual Design Automation Conference*, pages 530–534. ACM. 14

[29] Darga, P. T., Sakallah, K. A., and Markov, I. L. (2008). Faster symmetry discovery using sparsity of symmetries. In *Proceedings of the 45th Annual Design Automation Conference*, DAC '08, pages 149–154, New York, NY, USA. ACM. 7, 14, 24

[30] Denton, B. T., Miller, A. J., Balasubramanian, H. J., and Huschka, T. R. (2010). Optimal allocation of surgery blocks to operating rooms under uncertainty. *Operations research*, 58(4-part-1):802–816. 5

[31] Desrochers, M. and Soumis, F. (1989). A column generation approach to the urban transit crew scheduling problem. *Transportation Science*, 23(1):1–13. 5

[32] Dumas, Y., Desrosiers, J., and Soumis, F. (1991). The pickup and delivery problem with time windows. *European journal of operational research*, 54(1):7–22. 5

[33] Dummit, D. and Foote, R. (2004). *Abstract Algebra*. Wiley. 6

[34] Eckstein, J., Hart, W. E., and Phillips, C. A. (2015). PEBBL: an object-oriented framework for scalable parallel branch and bound. *Mathematical Programming Computation*, 7(4):429–469. 34

[35] Erdős, P. and Rényi, A. (1963). Asymmetric graphs. *Acta Mathematica Hungarica*, 14(3):295–315. 21, 35

[36] Faenza, Y., Oriolo, G., and Stauffer, G. (2010). The hidden matching structure of the composition of strips: a polyhedral perspective. In *14th Aussois Workshop on Combinatorial Optimization, Aussois (January 2010)*. 152, 156

[37] Fair Isaac Corporation (2017). FICO Xpress Optimization. http://www.fico.com/en/products/fico-xpress-optimization. 4

[38] Feige, U. (2002). Relations between average case complexity and approximation complexity. In *Proceedings of the thiry-fourth annual ACM symposium on theory of computing*, pages 534–543. ACM. 22

[39] Feige, U. and Kilian, J. (1996). Zero knowledge and the chromatic number. In *Computational Complexity, 1996. Proceedings., Eleventh Annual IEEE Conference on*, pages 278–287. IEEE. 32

[40] Feizollahi, M. J., Costley, M., Ahmed, S., and Grijalva, S. (2015). Large-scale decentralized unit commitment. *International Journal of Electrical Power & Energy Systems*, 73:97–106. 74, 108

[41] Fischetti, M. and Liberti, L. (2012). Orbital shrinking. In *International Symposium on Combinatorial Optimization*, pages 48–58. Springer. 6, 96

[42] Fischetti, M., Liberti, L., Salvagnin, D., and Walsh, T. (2017). Orbital shrinking: Theory and applications. *Discrete Applied Mathematics*, 222:109–123. 6

[43] Forrest, J., Ralphs, T., Fylstra, D., Hafer, L., Hart, B., Kristjannson, B., Phillips, C., Saltzman, M., Straver, E., Watson, J.-P., and Santos, H. G. (2017). COIN-OR Branch-and-Cut MIP Solver. https://projects.coin-or.org/Cbc. 4

[44] Fox, M., Long, D., and Porteous, J. (2007). Discovering near symmetry in graphs. In *Proceedings of the 22nd national conference on Artificial intelligence-Volume 1*, pages 415–420. AAAI Press. 24

[45] Frangioni, A. and Gentile, C. (2006a). Perspective cuts for a class of convex 0–1 mixed integer programs. *Mathematical Programming*, 106(2):225–236. 108

[46] Frangioni, A. and Gentile, C. (2006b). Solving nonlinear single-unit commitment problems with ramping constraints. *Operations Research*, 54(4):767–775. 49, 51, 55

[47] Frangioni, A. and Gentile, C. (2009). A computational comparison of reformulations of the perspective relaxation: SOCP vs. cutting planes. *Operations Research Letters*, 37(3):206–210. 108

[48] Frangioni, A. and Gentile, C. (2015a). An extended MIP formulation for the single-unit commitment problem with ramping constraints. In *17th British-French-German conference on Optimization*, London. 49, 57

[49] Frangioni, A. and Gentile, C. (2015b). New MIP formulations for the single-unit commitment problems with ramping constraints. IASI Research Report 15-06. 49, 57

[50] Frangioni, A., Gentile, C., and Lacalandra, F. (2008). Solving unit commitment problems with general ramp constraints. *International Journal of Electrical Power & Energy Systems*, 30(5):316–326. 108

[51] Frangioni, A., Gentile, C., and Lacalandra, F. (2009). Tighter approximated milp formulations for unit commitment problems. *IEEE Trans. Power Syst.*, 24(1):105–113. 108

[52] Fürstenberg, C. (2015). A drawing of a graph. http://en.wikipedia.org/wiki/Graph_theory#mediaviewer/File:6n-graf.svg. 21

[53] Gade, D., Hackebeil, G., Ryan, S. M., Watson, J.-P., Wets, R. J.-B., and Woodruff, D. L. (2016). Obtaining lower bounds from the progressive hedging algorithm for stochastic mixed-integer programs. *Mathematical Programming*, 157(1):47–67. 91

[54] Garcia-Gonzalez, J., de la Muela, R. M. R., Santos, L. M., and Gonzalez, A. M. (2008). Stochastic joint optimization of wind generation and pumped-storage units in an electricity market. *IEEE Transactions on Power Systems*, 23(2):460–468. 96

[55] Garey, M. R., Johnson, D. S., and Stockmeyer, L. (1974). Some simplified NP-complete problems. In *Proceedings of the sixth annual ACM symposium on theory of computing*, pages 47–63. ACM. 32

[56] Garver, L. L. (1962). Power generation scheduling by integer programming-development of theory. *Power Apparatus and Systems, Part III. Transactions of the American Institute of Electrical Engineers*, 81(3):730–734. 50, 52, 73

[57] GE Energy (2014). PJM renewable integration study. PJM Interconnection. 66, 83

[58] Gentile, C., Morales-Espana, G., and Ramos, A. (2016). A tight MIP formulation of the unit commitment problem with start-up and shut-down constraints. *EURO Journal on Computational Optimization*, pages 1–25. 51, 74, 95, 96, 187, 198

[59] Gilmore, P. C. and Gomory, R. E. (1961). A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849–859. 5

[60] Godsil, C. and Royle, G. (2001). *Algebraic Graph Theory*. Springer-Verlag. 6

[61] Goldreich, O., Micali, S., and Wigderson, A. (1991). Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM (JACM)*, 38(3):690–728. 10, 20

[62] Golumbic, M. C. (2004). *Algorithmic graph theory and perfect graphs*, volume 57. Elsevier. 99

[63] Gomory, R. E. (1958). Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278. 3

[64] Gomory, R. E. (1960). An algorithm for the mixed integer problem. Research memorandum rm-2597, The Rand Corporation. 3

[65] Grünbaum, B. (2003). *Convex polytopes, volume 221 of Graduate Texts in Mathematics*. Springer-Verlag. 153

[66] Gurobi Optimization, Inc. (2016). Gurobi optimizer reference manual. `http://www.gurobi.com`. 4, 95, 96, 108

[67] Hartke, S. G. and Radcliffe, A. (2009). Mckay's canonical graph labeling algorithm. *Communicating mathematics*, 479:99–111. 14

[68] Herstein, I. N. (2006). *Topics in algebra*. John Wiley & Sons. 6

[69] International Business Machines Corporation (2017). IBM CPLEX Optimizer. `https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`. 4, 95

[70] Jabr, R. (2012). Tight polyhedral approximation for mixed-integer linear programming unit commitment formulations. *IET Generation, Transmission & Distribution*, 6(11):1104–1111. 108

[71] Jeroslow, R. G. (1987). Representability in mixed integer programming, I: characterization results. *Discrete Applied Mathematics*, 17(3):223–243. 153

[72] Jünger, M., Liebling, T. M., Naddef, D., Nemhauser, G. L., Pulleyblank, W. R., Reinelt, G., Rinaldi, G., and Wolsey, L. A. (2009). *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-art*. Springer Science & Business Media. 2, 4

[73] Junttila, T. A. and Kaski, P. (2007). Engineering an efficient canonical labeling tool for large and sparse graphs. In *ALENEX*, volume 7, pages 135–149. SIAM. 11, 15

[74] Kaibel, V., Peinhardt, M., and Pfetsch, M. E. (2011). Orbitopal fixing. *Discrete Optimization*, 8(4):595–610. 6

[75] Kaibel, V. and Pfetsch, M. (2008). Packing and partitioning orbitopes. *Mathematical Programming*, 114(1):1–36. 5

[76] Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM. 2

[77] Katebi, H., Sakallah, K. A., and Markov, I. L. (2012a). Conflict anticipation in the search for graph automorphisms. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 243–257. Springer. 11

[78] Katebi, H., Sakallah, K. A., and Markov, I. L. (2012b). Graph symmetry detection and canonical labeling: Differences and synergies. *arXiv preprint arXiv:1208.6271*. 15

[79] Kazarlis, S. A., Bakirtzis, A., and Petridis, V. (1996). A genetic algorithm solution to the unit commitment problem. *IEEE Trans. Power Syst.*, 11(1):83–92. 91, 108

[80] Khachiyan, L. G. (1979). A polynomial algorithm in linear programming. In *Soviet Mathematics Doklady*, volume 20, pages 191–194. 2

[81] Kim, K., Botterud, A., and Qiu, F. (2017). Temporal decomposition for improved unit commitment in power system production cost modeling. ANL Technical Report. ANL/MCS-P7073-0717. 74

[82] Knueven, B., Ostrowski, J., and Watson, J.-P. (2017). Online companion for a novel matching formulation for startup costs in unit commitment. 202

[83] Knuth, D. E. (1993). *The Stanford GraphBase: a platform for combinatorial computing*, volume 37. Addison-Wesley Reading. 34

[84] Krall, E., Higgins, M., and O'Neill, R. P. (2012). RTO unit commitment test system. Federal Energy Regulatory Commission. 61, 82, 83, 168

[85] Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97. 29

[86] Lee, J., Leung, J., and Margot, F. (2004). *Discrete Optimization*, 1(1):77–85. 50, 74, 96

[87] Lima, R. M. and Novais, A. Q. (2016). Symmetry breaking in MILP formulations for unit commitment problems. *Computers & Chemical Engineering*, 85:162–176. 5, 95, 96, 113, 201, 207, 208, 210

[88] Lin, C.-L. (1994). Hardness of approximating graph transformation problem. In *Algorithms and Computation*, pages 74–82. Springer. 20

[89] López-Presa, J. L., Anta, A. F., and Chiroque, L. N. (2011). Conauto-2.0: Fast isomorphism testing and automorphism group computation. *arXiv preprint arXiv:1108.1060*. 11, 15

[90] López-Presa, J. L. and Fernández, A. (2004). Graph isomorphism testing without full automorphism group computation. 15

[91] Lynch, M. (1968). Storage and retrieval of information on chemical structures by computer. *Endeavour*, 27(101):68. 11

[92] Magnanti, T. L. and Wolsey, L. A. (1995). Optimal trees. *Handbooks in Operations Research and Management Science*, 7:503–615. 152

[93] Maher, S. J., Fischer, T., Gally, T., Gamrath, G., Gleixner, A., Gottwald, R. L., Hendel, G., Koch, T., Lübbecke, M. E., Miltenberger, M., Müller, B., Pfetsch, M. E., Puchert, C., Rehfeldt, D., Schenker, S., Schwarz, R., Serrano, F., Shinano, Y., Weninger, D., Witt, J. T., and Witzig, J. (2017). The SCIP Optimization Suite 4.0. Technical Report 17-12, ZIB, Takustr.7, 14195 Berlin. 4

[94] Malkin, P. (2003). Minimum runtime and stoptime polyhedra. CORE Report, Université catholique de Louvain. 50, 96, 100, 187

[95] Margot, F. (2002). Pruning by isomorphism in branch-and-cut. *Mathematical Programming*, 94(1):71–90. 7, 23

[96] Margot, F. (2003). Exploiting orbits in symmetric ILP. *Mathematical Programming*, 98(1-3):3–21. 7, 23

[97] Margot, F. (2010). Symmetry in integer linear programming. In *50 Years of Integer Programming 1958-2008*, pages 647–686. Springer. 5

[98] Markov, I. (2007). Almost-symmetries of graphs. In *Proc. International Symmetry Conference (ISC)*, pages 60–70. 23

[99] Mathon, R. (1978). Sample graphs for graph isomorphism testing. In *Proc. 9th S.E. Conf. Combinatorics, Graph Theory and Computing*, pages 499–517. 12

[100] Mathon, R. (1979). A note on the graph isomorphism counting problem. *Information Processing Letters*, 8(3):131–136. 10, 20

[101] McKay, B. D. (1981). *Practical graph isomorphism*. Department of Computer Science, Vanderbilt University. 13, 21

[102] McKay, B. D. and Piperno, A. (2013). nauty and traces user's guide (version 2.5). *Computer Science Department, Australian National University, Canberra, Australia*. 14

[103] McKay, B. D. and Piperno, A. (2014). Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60:94–112. 7, 11, 14, 15, 24, 34

[104] McKay, B. D. and Piperno, A. (2015). Nauty traces – graphs. `http://pallini.di.uniroma1.it/Graphs.html`. 35

[105] Mehrotra, A. and Trick, M. A. (1996). A column generation approach for graph coloring. *Informs Journal on Computing*, 8(4):344–354. 5

[106] Meller, R. D., Narayanan, V., and Vance, P. H. (1998). Optimal facility layout design. *Operations Research Letters*, 23(3):117–127. 5

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

[108] Miyazaki, T. (1997). The complexity of McKay's canonical labeling algorithm. In *Groups and Computation II*, volume 28, pages 239–256. Aer. Math. Soc.: Providence, RI. 14

[109] Morales-España, G., Gentile, C., and Ramos, A. (2015). Tight MIP formulations of the power-based unit commitment problem. *OR Spectrum*, pages 1–22. 51, 95

[110] Morales-España, G., Latorre, J. M., and Ramos, A. (2013a). Tight and compact MILP formulation for the thermal unit commitment problem. *IEEE Transactions on Power Systems*, 28(4):4897–4908. 74, 75, 76, 77, 91, 95, 100, 108, 187, 198, 201

[111] Morales-España, G., Latorre, J. M., and Ramos, A. (2013b). Tight and compact MILP formulation of start-up and shut-down ramping in unit commitment. *IEEE Transactions on Power Systems*, 28(2):1288–1296. 53, 74

[112] Morgan, H. (1965). The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service. *Journal of Chemical Documentation*, 5(2):107–113. 11

[113] Muckstadt, J. A. and Wilson, R. C. (1968). An application of mixed-integer programming duality to scheduling thermal generating systems. *IEEE Transactions on Power Apparatus and Systems*, (12). 74

[114] Munkres, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial & Applied Mathematics*, 5(1):32–38. 29

[115] Nowak, M. P. and Römisch, W. (2000). Stochastic lagrangian relaxation applied to power scheduling in a hydro-thermal system under uncertainty. *Annals of Operations Research*, 100(1-4):251–272. 74, 76, 162

[116] O'Donnell, R., Wright, J., Wu, C., and Zhou, Y. (2014). Hardness of robust graph isomorphism, lasserre gaps, and asymmetry of random graphs. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1659–1677. SIAM. 20, 21, 22

[117] O'Neill, R. P. (2007). It's getting better all the time (with mixed integer programming). HEPG Forty-Ninth Plenary Session. 16, 49

[118] Ostrowski, J. (2008). *Symmetry in Integer Programming*. PhD thesis, Lehigh University. 5

[119] Ostrowski, J., Anjos, M. F., and Vannelli, A. (2012). Tight mixed integer linear programming formulations for the unit commitment problem. *IEEE Transactions on Power Systems*, 27(1):39. 51, 74, 91, 92, 95, 97, 108

[120] Ostrowski, J., Anjos, M. F., and Vannelli, A. (2015). Modified orbital branching for structured symmetry with an application to unit commitment. *Mathematical Programming*, 150(1):99–129. 70, 92, 95, 113

[121] Ostrowski, J., Linderoth, J., and Margot, F. (2017). Flexible isomorphism pruning. Working paper. 7

[122] Ostrowski, J., Linderoth, J., Rossi, F., and Smriglio, S. (2008). Constraint orbital branching. *Integer Programming and Combinatorial Optimization*, pages 225–239. 8

[123] Ostrowski, J., Linderoth, J., Rossi, F., and Smriglio, S. (2011). Orbital branching. *Mathematical Programming*, 126(1):147–178. 7, 8, 23, 95

[124] Palmintier, B. and Webster, M. (2011). Impact of unit commitment constraints on generation expansion planning with renewables. In *Power and Energy Society General Meeting, 2011 IEEE*, pages 1–7. IEEE. 96

[125] Pan, K. and Guan, Y. (2016). A polyhedral study of the integrated minimum-up/-down time and ramping polytope. arXiv:1604.02184. 51, 54, 69, 74

[126] Parris, R. and Read, R. (1969). A coding procedure for graphs. *Scientific Report. UWI/CC*, 10. 11

[127] Piperno, A. (2008). Search space contraction in canonical labeling of graphs. *arXiv preprint arXiv:0804.4881*. 15

[128] PJM (2016a). PJM - ancillary services. `http://pjm.com/markets-and-operations/ancillary-services.aspx`. Accessed: 2016-01-07. 63, 82, 169

[129] PJM (2016b). PJM - system operations. http://www.pjm.com/markets-and-operations/ops-analysis.aspx. Accessed: 2016-01-07. 63, 82, 169

[130] Pochet, Y. and Wolsey, L. A. (1993). Lot-sizing with constant batches: Formulation and valid inequalities. *Mathematics of Operations Research*, 18(4):767–785. 152

[131] Pochet, Y. and Wolsey, L. A. (2006). *Production planning by mixed integer programming*. Springer Science & Business Media. 77, 162, 184, 189

[132] Puget, J.-F. (1993). On the satisfiability of symmetrical constrained satisfaction problems. In *Methodologies for Intelligent Systems*, pages 350–361. Springer. 5

[133] Rajan, D. and Takriti, S. (2005). Minimum up/down polytopes of the unit commitment problem with start-up costs. IBM Research Report. RC23628 (W0506-050). 50, 51, 52, 74, 96

[134] Ramanan, P., Yildirim, M., Chow, E., and Gebraeel, N. (2017). Asynchronous decentralized framework for unit commitment in power systems. *Procedia Computer Science*, 108:665–674. 74

[135] Read, R. C. and Corneil, D. G. (1977). The graph isomorphism disease. *Journal of Graph Theory*, 1(4):339–363. 11, 20

[136] Rockafellar, R. (1970). *Convex Analysis*. Princeton University Press. 153

[137] Rothvoß, T. (2017). The matching polytope has exponential extension complexity. *Journal of the ACM (JACM)*, 64(6):41. 4, 51

[138] Rudich, S. and Wigderson, A. (2004). *Computational complexity theory*. American Mathematical Soc. 10

[139] Salvagnin, D. (2005). A dominance procedure for integer programming. Master's thesis, University of Padua. 7

[140] Schrijver, A. (1986). *Theory of linear and integer programming*. John Wiley & Sons. 18

[141] Sen, S. and Kothari, D. (2002). An equivalencing technique for solving the large-scale thermal unit commitment problem. In *The Next Generation of Electric Power Unit Commitment Models*, pages 211–225. Springer. 96

[142] Sherali, H. D. and Smith, J. C. (2001). Improving discrete model representations via symmetry considerations. *Management Science*, 47(10):1396–1407. 5

[143] Shortt, A. and O'Malley, M. (2010). Impact of variable generation in generation resource planning models. In *Power and Energy Society General Meeting, 2010 IEEE*, pages 1–6. IEEE. 96

[144] Silbernagl, M., Huber, M., and Brandenberg, R. (2016). Improving accuracy and efficiency of start-up cost formulations in MIP unit commitment by modeling power plant temperatures. *IEEE Trans. Power Syst.*, 31(4):2578–2586. 74, 77, 80, 162

[145] Simoglou, C. K., Biskas, P. N., and Bakirtzis, A. G. (2010). Optimal self-scheduling of a thermal producer in short-term electricity markets by MILP. *IEEE Trans. Power Syst.*, 25(4):1965–1977. 74

[146] Stachniss, C. (2015). C implementation of the hungarian method. `http://www2.informatik.uni-freiburg.de/~stachnis/misc.html`. 34

[147] Stoer, J. and Witzgall, C. (1970). *Convexity and Optimization in Finite Dimensions*. Springer. 153

[148] Sussenguth, E. H. (1965). A graph-theoretic algorithm for matching chemical structures. *Journal of Chemical Documentation*, 5(1):36–43. 11

[149] Takriti, S., Birge, J., and Long, E. (1996). A stochastic model for the unit commitment problem. *IEEE Trans. Power Syst.*, 11(3):1497–1508. 90

[150] Trespalacios, F. and Grossmann, I. E. (2014). Review of mixed-integer nonlinear and generalized disjunctive programming methods. *Chemie Ingenieur Technik*, 86(7):991–1012. 2

[151] Tseng, C.-L. (1996). *On Power System Generation Unit Commitment Problems*. PhD thesis, University of California at Berkeley. 16

[152] Unger, S. H. (1964). Git–a heuristic program for testing pairs of directed line graphs for isomorphism. *Communications of the ACM*, 7(1):26–34. 11

[153] Vance, P. H. (1993). *Crew scheduling, cutting stock, and column generation: Solving huge integer programs*. PhD thesis, Georgia Institute of Technology. 5

[154] Vance, P. H., Barnhart, C., Johnson, E. L., and Nemhauser, G. L. (1994). Solving binary cutting stock problems by column generation and branch-and-bound. *Computational optimization and applications*, 3(2):111–130. 5

[155] Vance, P. H., Barnhart, C., Johnson, E. L., and Nemhauser, G. L. (1997). Airline crew scheduling: A new formulation and decomposition algorithm. *Operations Research*, 45(2):188–200. 5

[156] West, D. B. (2001). *Introduction to Graph Theory*. Prentice-Hall. 6, 9

[157] Weyl, H. (1950). The elementary theory of convex polyhedra. *Contributions to the Theory of Games*, 1(24):3–18. 153

[158] Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83. 90, 176

[159] Wolsey, L. A. (1998). *Integer Programming*, volume 42. Wiley New York. 78

[160] Wood, A. J., Wollenberg, B. F., and Sheblé, G. B. (2013). *Power Generation, Operation and Control*. John Wiley & Sons. 73, 95

[161] Yang, L., Jian, J., Wang, Y., and Dong, Z. (2015a). Projected mixed integer programming formulations for unit commitment problem. *International Journal of Electrical Power & Energy Systems*, 68:195–202. 108

[162] Yang, L., Jian, J., Zhu, Y., and Dong, Z. (2015b). Tight relaxation method for unit commitment problem using reformulation and lift-and-project. *IEEE Transactions on Power Systems*, 30(1):13–23. 108

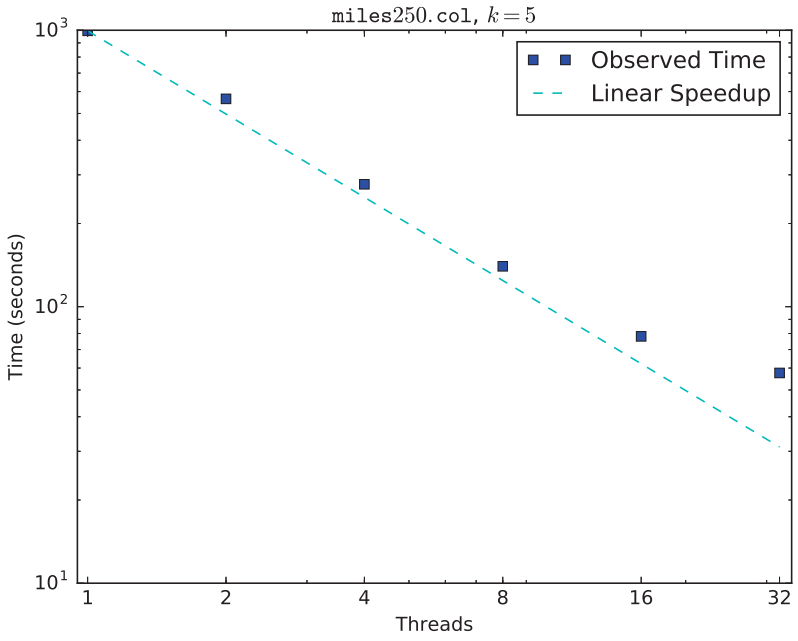
Appendices

Appendix A

Detailed Computational Results for Chapter 2

In this appendix we break-out the scaling graph in Figure 2.4 into scaling graphs for the individual cases tested. These are presented in Figure A.1. We also present the rest of the results on branching choice robustness, similar to that in Table 2.4, in Table A.1. Also, we provide more test-cases for the branching strategy similar to those given in Figure 2.5. These results are in Figure A.2. Finally, in Table A.2 we present some computational results on the test suite from Table 2.2 with a modified branching strategy. In particular, at each node we create three children, two of which are deletion nodes for the two edges highest-ranked by *edgeUse* and the third fixing both these edges. The hope is this makes the fixing child stronger, making for a more balanced tree while also not loosing much due to the robustness of the branching selection demonstrated in Section 2.4.1. As we can see, this branching strategy is better for the random graphs, `miles1500.col`, and `le450_5b.col`, but is worse for the remaining graphs. Overall this strategy apparently does not effectively exploit robustness of *edgeUse* branching.

Figure A.1: Detailed scaling on various problems



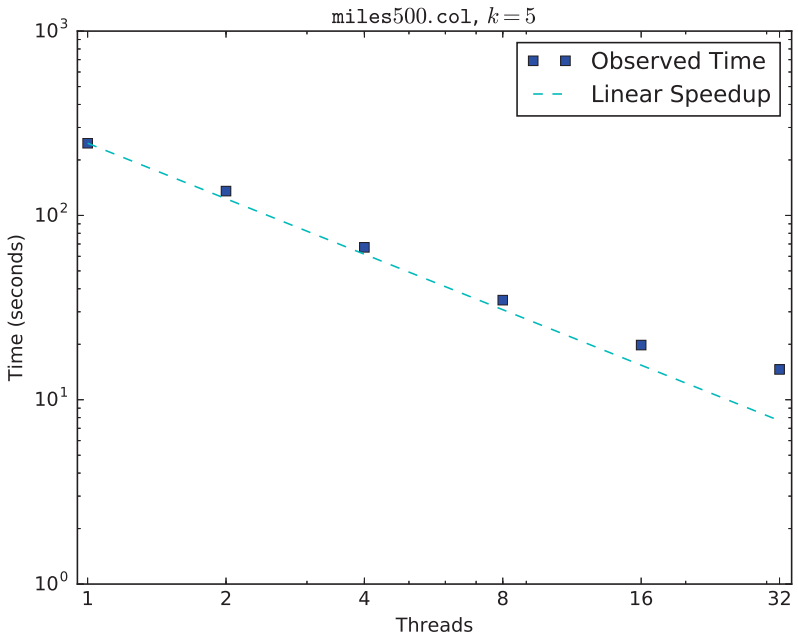
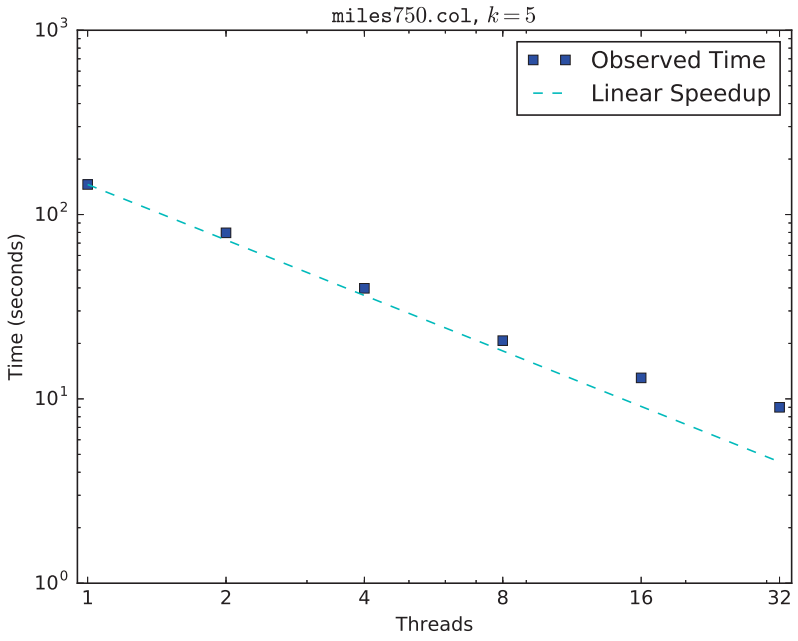


Figure A.1: (continued)



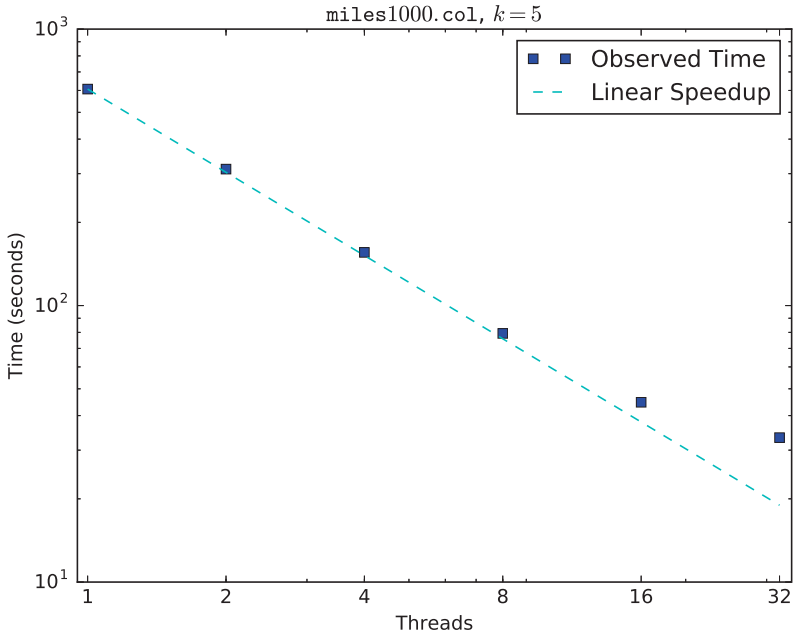
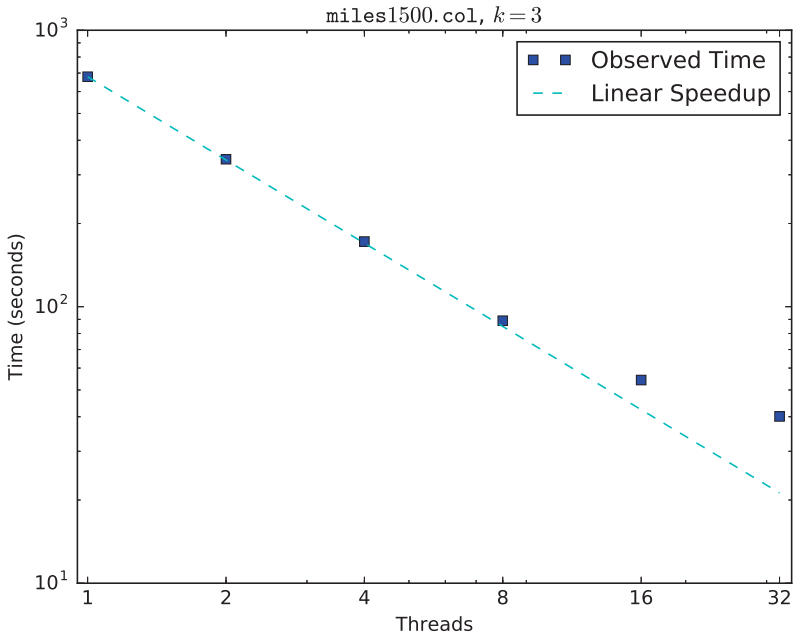


Figure A.1: (continued)



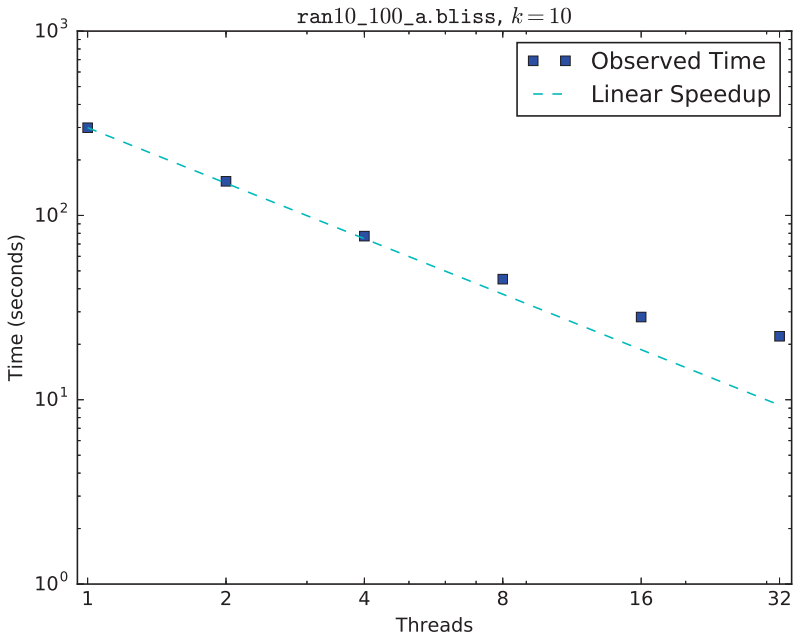
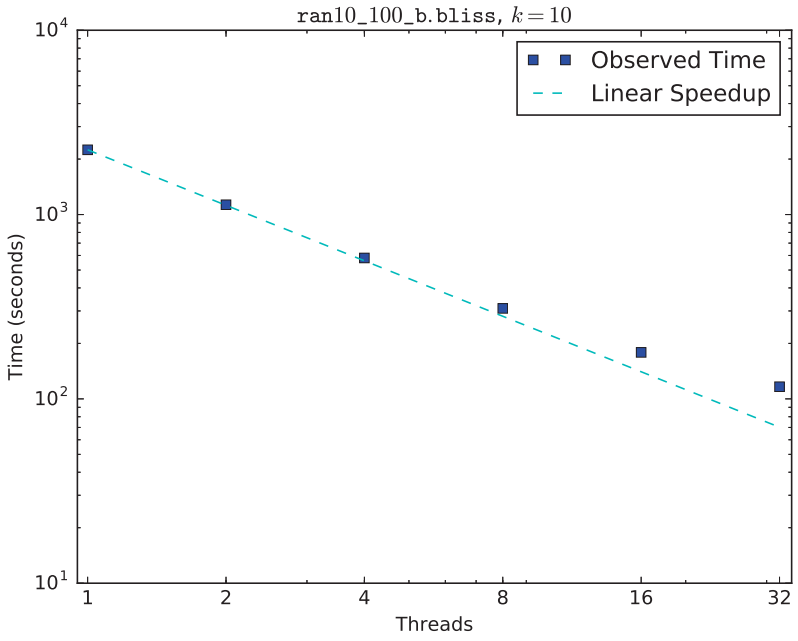


Figure A.1: (continued)



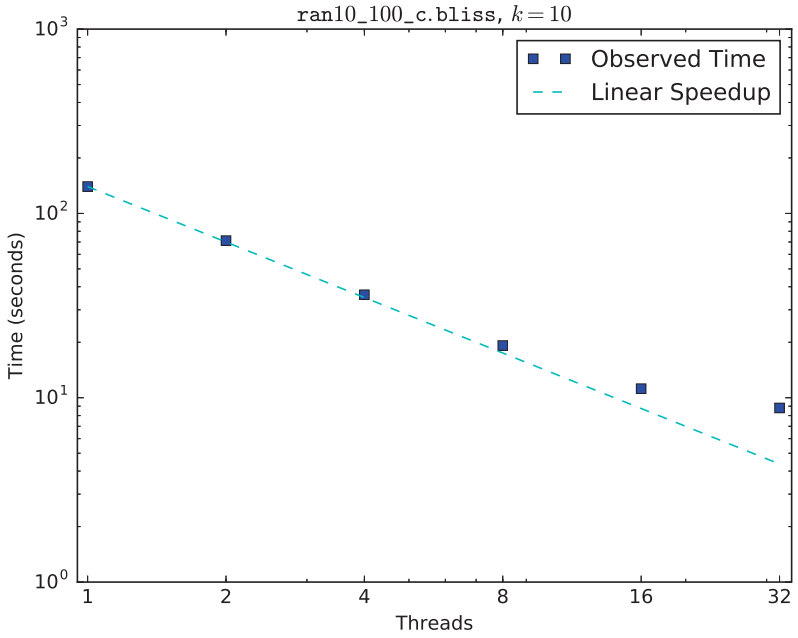


Figure A.1: (continued)

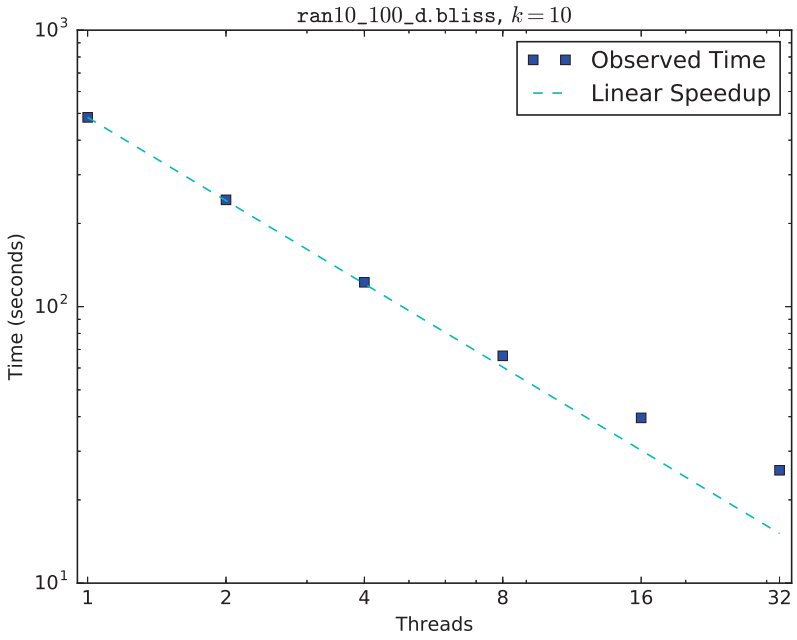


Figure A.1: (continued)

Table A.1: *edgeUse* branching robustness, additional instances

`miles250.col`, $k = 5$

Rank	% of Nodes
1	76.85%
2	8.34%
3	0.73%
4	0.47%
5	0.28%
6	0.24%
7	0.19%
8	0.17%
9	0.15%
10	0.15%
11+	12.43%

`miles500.col`, $k = 5$

Rank	% of Nodes
1	67.44%
2	1.32%
3	0.20%
4	0.28%
5	0.13%
6	0.19%
7	0.19%
8	0.28%
9	0.39%
10	0.36%
11+	29.21%

`miles1000.col`, $k = 5$

Rank	% of Nodes
1	80.33%
2	0.44%
3	0.13%
4	0.23%
5	2.44%
6	3.64%
7	1.74%
8	0.76%
9	0.34%
10	0.54%
11+	9.40%

Table A.1: (continued)

`miles1500.col`,
$k = 3$

Rank	% of Nodes
1	96.95%
2	0.58%
3	0.37%
4	0.12%
5	0.09%
6	0.11%
7	0.06%
8	0.05%
9	0.10%
10	0.15%
11+	1.42%

`le450_5b.col`,
$k = 15$

Rank	% of Nodes
1	84.48%
2	5.17%
3	3.45%
4	1.72%
5	3.45%
6	1.72%
7+	0.00%

`le450_15b.col`,
$k = 15$

Rank	% of Nodes
1	70.05%
2	0.32%
3	0.22%
4	0.39%
5	0.12%
6	0.05%
7	0.05%
8	0.06%
9	0.06%
10	0.06%
11+	28.62%

Table A.1: (continued)

`ran10_100_b.bliss`, $k = 10$

Rank	% of Nodes
1	61.30%
2	6.98%
3	2.19%
4	0.93%
5	0.79%
6	0.46%
7	0.95%
8	2.07%
9	2.68%
10	1.83%
11+	19.82%

`ran10_100_c.bliss`, $k = 10$

Rank	% of Nodes
1	60.92%
2	2.76%
3	1.32%
4	2.21%
5	0.60%
6	0.39%
7	0.14%
8	0.15%
9	0.17%
10	0.16%
11+	31.19%

`ran10_100_d.bliss`, $k = 10$

Rank	% of Nodes
1	79.29%
2	1.33%
3	0.52%
4	0.15%
5	0.60%
6	4.66%
7	0.60%
8	1.69%
9	2.92%
10	0.09%
11+	8.15%

Figure A.2: Random branching (box plot) vs. *edgeUse* branching (■, number of nodes on right) vs. local branching (×, number of nodes on left), additional instances

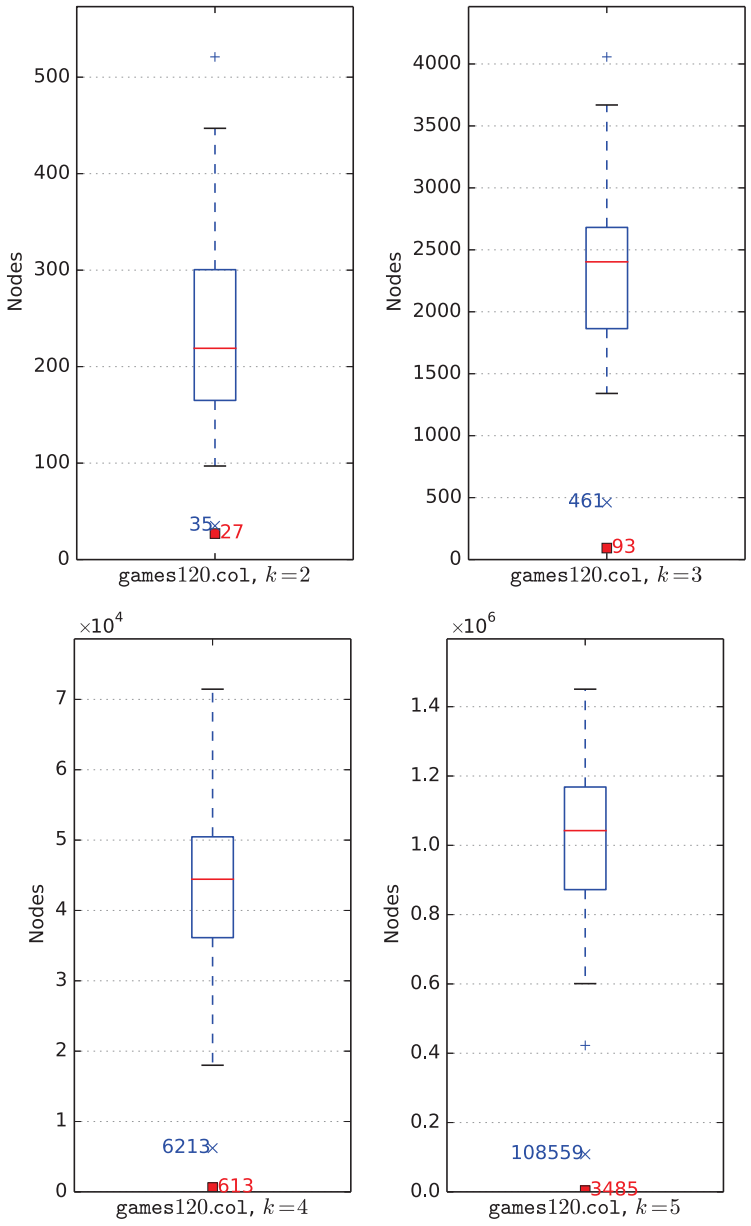


Figure A.2: (continued)

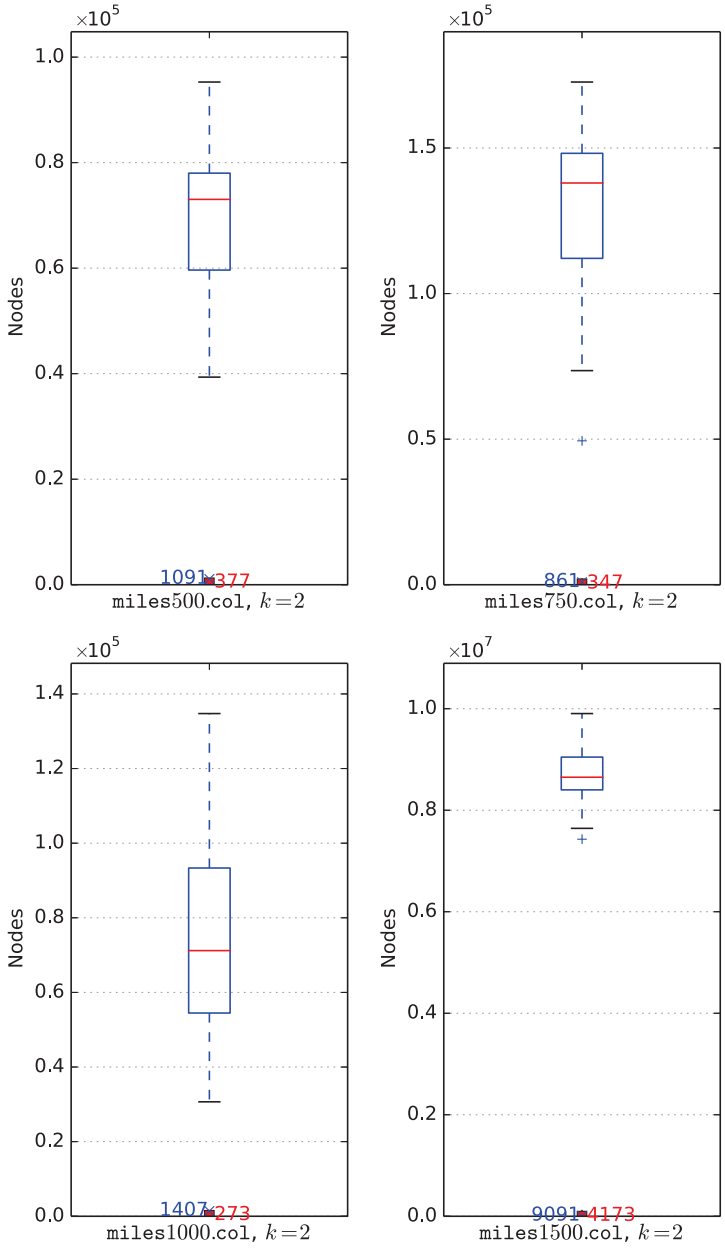


Figure A.2: (continued)

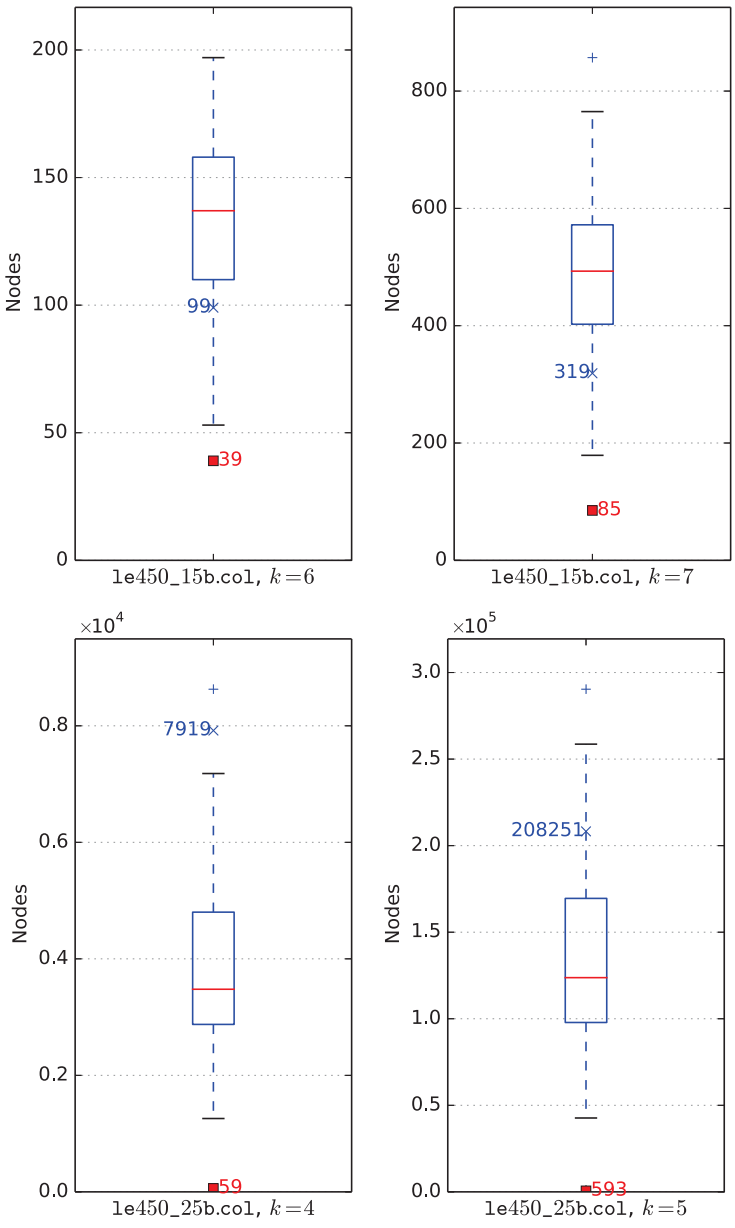


Figure A.2: (continued)

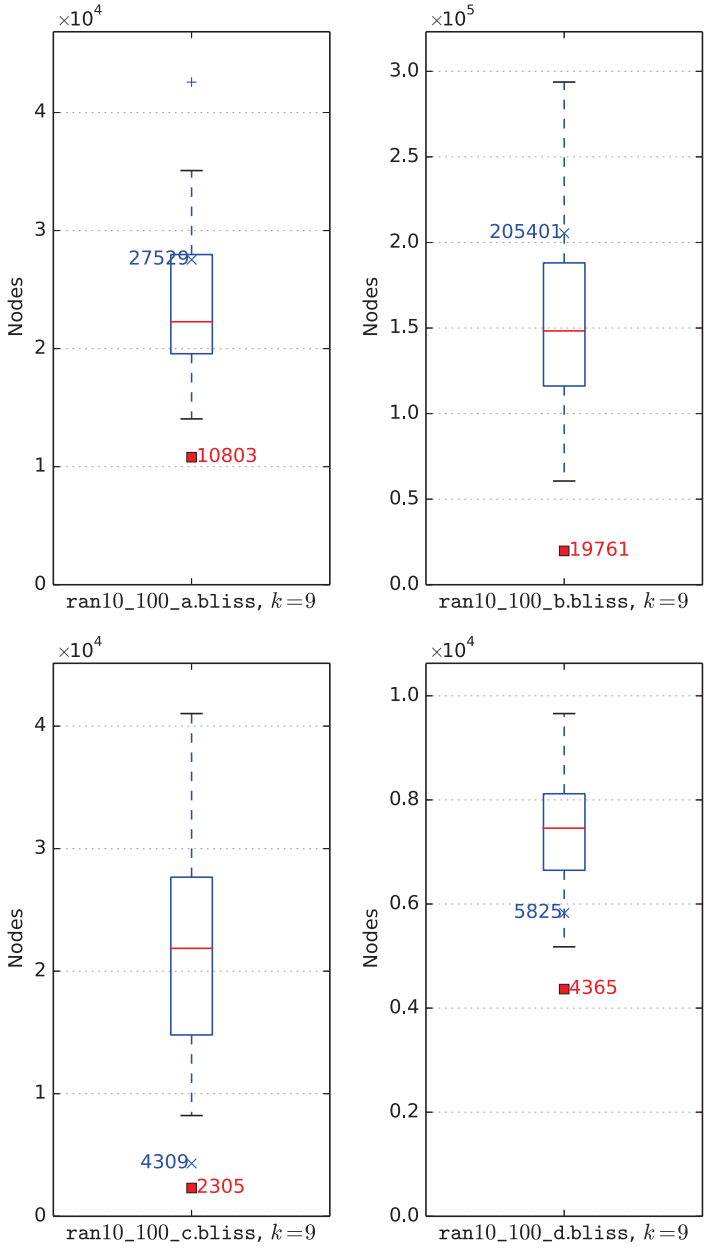


Figure A.2: (continued)

Table A.2: Computational Results – three children per node

games120.col											$n = 120$ $e = 638$
k	0	1	2	3	4	5	6	7	8	9	
γ_k^G	119	118	117	114	113	112	112	111	111	112*	
seconds	0.0	0.0	0.1	0.2	0.3	0.4	2.4	28.5	938.7	†	

miles250.col									$n = 128$ $e = 387$
k	0	1	2	3	4	5	6	7	
γ_k^G	108	106	104	102	100	99	97	98*	
seconds	0.0	0.1	0.2	0.7	3.1	55.8	852.7	†	

miles500.col										$n = 128$ $e = 1170$
k	0	1	2	3	4	5	6	7	8	
γ_k^G	114	113	111	110	109	108	107	108*	107*	
seconds	0.0	0.1	0.1	0.3	1.6	16.3	212.0	†	†	

miles750.col										$n = 128$ $e = 2113$
k	0	1	2	3	4	5	6	7	8	
γ_k^G	122	121	120	119	118	117	116	115	115*	
seconds	0.0	0.1	0.2	0.4	1.5	9.3	99.4	872.3	†	

miles1000.col									$n = 128$ $e = 3216$
k	0	1	2	3	4	5	6	7	
γ_k^G	123	122	121	120	119	118	118	117*	
seconds	0.0	0.1	0.2	0.5	2.0	29.9	422.6	†	

miles1500.col							$n = 128$ $e = 5198$
k	0	1	2	3	4	5	
γ_k^G	102	101	100	99	98	97*	
seconds	0.0	0.7	1.7	24.5	586.6	†	

le450_5b.col									$n = 450$ $e = 5734$
k	0	13	14	15	16	17	18	19	
γ_k^G	450	450	450	450	450	450	450	449	
seconds	0.0	1.1	1.3	1.6	1.8	5.1	387.6	2847	

le450_15b.col													$n = 450$ $e = 8169$
k	0	1	2	4	5	7	8	10	11	14	15	16	17
γ_k^G	450	450	449	449	448	448	447	447	446	446	445	446*	447*
seconds	0.0	0.1	0.2	0.4	0.6	0.7	0.8	1.1	1.3	47.2	176.3	†	†

le450_25b.col												$n = 450$ $e = 8263$
k	0	1	2	3	4	5	6	7	8	9	10	
γ_k^G	450	450	449	449	449	448	448	447	447	447	448*	
seconds	0.0	0.1	0.2	0.4	0.6	0.6	1.1	2.0	13.9	129.4	†	

ran10_100_a.bliss									$n = 100$ $e = 502$
k	0	6	7	8	9	10	11	12	
γ_k^G	100	100	99	99	99	99	99	100*	
seconds	0.0	0.0	0.1	0.4	1.9	14.0	342.9	†	

ran10_100_b.bliss										$n = 100$ $e = 464$
k	0	3	4	7	8	9	10	11	12	
γ_k^G	100	100	99	99	99	99	98	98	100*	
seconds	0.0	0.0	0.1	0.2	2.0	6.8	96.4	1644	†	

ran10_100_c.bliss											$n = 100$ $e = 525$
k	0	5	6	7	8	9	10	11	12	13	
γ_k^G	100	100	99	99	99	99	99	98	98	100*	
seconds	0.0	0.0	0.1	0.1	0.4	1.9	7.2	101.6	1561	†	

ran10_100_d.bliss									$n = 100$ $e = 514$
k	0	1	7	8	9	10	11	12	
γ_k^G	100	100	100	99	99	99	99	100*	
seconds	0.0	0.0	0.1	0.6	4.5	22.1	466.0	†	

Appendix B

Code Structure for `FindAlmostSymmetry`

This appendix provides a brief description for the code structure for `FindAlmostSymmetry`. The full source code is available at http://dx.doi.org/10.5281/zenodo.840558.

The files `NautyGraph.hpp` and `NautyGraph.cpp` provide a C++ wrapper for `nauty`'s dense graphs with dynamic allocation, allowing for the easy addition and removal of edges. The two main classes are `DenseGraph` and `NautyGraph`. `DenseGraph` is a class for managing graph objects packed in a bit vector format which can be used in `nauty`. `NautyGraph` is derived from `DenseGraph` and provides methods to call `nauty`'s automorphism routine and accessing the results. For each node A, E_A^F (the graph of fixed edges) and P_A (the graph of permutations) are stored as `DenseGraphs`. A `NautyGraph` is used for storing the graph G. `NautyGraph.hpp` and `NautyGraph.cpp` also define the class `EdgeList`, which is used for storing E_A^D for each node A. These three classes all have `utilib::PackObject` (from `PEBBL`) as a parent class, allow them to be passed around using MPI by `PEBBL` (or `utilib`'s MPI routines).

The files `findAlmost.hpp` and `parFindAlmost.hpp` provide the header information for the functions and classes used for the solver, and `findAlmost.cpp` implements these. The classes defined in `findAlmost.hpp` provide hooks for `PEBBL`'s serial interface. Class `findAlmost` is a derived class of `PEBBL`'s `branching` class, and its methods and attributes set up the problem and are common to every search node: the original graph G, the starting budget k, and solver options. Class `findAlmostSub` is derived from `PEBBL`'s `branchSub` class and provides attributes which are specific to a node, and methods which do the node processing. Most of the main **while** loop in Algorithm 6 is contained in `findAlmostSub::boundComputation()`, which `PEBBL` repeatedly calls until either this node is pruned by bound or no more refining can be done for this node,

in which case a branching edge is selected using `findAlmostSub::findBranchEdge()`. Finally, class `findAlmostSol` is derived from `PEBBL`'s solution class, and is used for storing, passing, and writing solutions. The remaining functions have names corresponding to those in the paper, e.g., BuildCostMatrix is implemented in `buildCostMatrix()`.

Similarly, the classes in `parFindAlmost.hpp` provide the hooks for `PEBBL`'s parallel interface and also has the header information for function using MPI. The function `parRefineByMatching()` is similar to `refineByMatching()` but parallelizes the repeated calls to `hungarian_solve()`. Class `parFindAlmost` is derived from `findAlmost` and `PEBBL`'s `parallelBranching` class, and has methods that allow instances of it to be passed around using MPI messaging. Similarly, class `parFindAlmostSub` is derived from `findAlmostSub` and `PEBBL`'s `parallelBranchSub`. In addition to facilitating subproblem passing through MPI, `parFindAlmostSub` also replaces a few of the methods of `findAlmostSub` for parallel subproblem management – in particular ensuring `parRefineByMatching()` is called while the solver is ramping up.

Appendix C

Constrained Minkowski Sums of Polyhedra

We will prove Theorem 3.1 by extending the classical result of Balas on disjunctive programs. The success of disjunctive programming as initially laid out by Balas [6, 7] toward the practical solvability of problems involving indicator constraints is clear; see [11] for a recent overview. We consider an extension of Balas's classical result (Theorem C.1), a weaker version of which is given as a lemma by Faenza et al. [36], and show it can be used to model constrained Minkowski sums of polyhedra. Generalizing the convexity constraint is an approach that has been taken before, namely for the shortest path polytope [130] and the tree packing polytope [92], which would be sufficient to prove Theorem 3.1. However, we show that any integer polytope could be used in place of the convexity constraint, allowing for a great deal of modeling flexibility. Note that this result is stronger than we need, but we provide it here for completeness.

The goal of this section is to arrive at a polyhedral representation of constrained Minkowski sums of polyhedra using indicator variables. First we must dispense with some definitions. Scalar multiples and Minkowski sums for sets in $\mathbb{R}^n$ are defined in their usual way as

$$\lambda C := \{\lambda x \mid x \in C\}, \tag{C.1}$$

$$C_1 + C_2 := \{x_1 + x_2 \mid x_1 \in C_1,\, x_2 \in C_2\}. \tag{C.2}$$

For a set $S \subset \mathbb{R}^n$, $\text{conv}(S)$ is the convex hull of S and $\text{cone}(S)$ is the conic hull of S. The orthogonal projection of $S \subset \mathbb{R}^n \times \mathbb{R}^p$ onto $\mathbb{R}^n$ is denoted $\text{proj}_x(S) := \{x \in \mathbb{R}^n \mid \exists y \in \mathbb{R}^p \text{ s.t. } (x, y) \in S\}$.

A system of linear inequalities $Ax \leq b$ is said to be a *perfect formulation* of a set $S \subset \mathbb{R}^n$ if $\text{conv}(S) = \{x \in \mathbb{R}^n \mid Ax \leq b\}$. For a polyhedron $P \subset \mathbb{R}^n$ we say that a polyhedron $Q \subset \mathbb{R}^n \times \mathbb{R}^p$ is an *extended formulation* of P if $\text{proj}_x(Q) = P$. Such an extended formulation is said to be *compact* when only a polynomial number of variables and constraints in the size of the input are needed to describe Q. For convenience we (again) use the notation $[m] = \{1, \ldots, m\}$ and subscripts to indicate the components of a vector.

Naturally our tools are those of convex analysis [136, 147, 65], with the Minkowski-Weyl theorem for polyhedra [157] playing a lead role. To motivate the framework developed in this section, consider convex combinations of polyhedra. Suppose we have a collection $P^1, \ldots, P^m$ of nonempty polyhedra, and notice $\text{conv}\left(\cup_{i\in[m]} P^i\right) = \bigcup\{\sum_{i=1}^m \gamma_i P^i \mid \sum_{i=1}^m \gamma_i = 1, \gamma \geq 0\}$. An interesting question is when is such a set closed and polyhedral. Indeed Theorem 9.8 and subsequent corollaries in Rockafellar [136] give sufficient conditions for closedness. Balas [6, 7] provides sufficient conditions for polyhedreality along with an extended formulation for such a set. We restate Balas's result.

Theorem C.1. *Consider m polyhedra $P^i = \{x \in \mathbb{R}^n \mid A^i x \leq b^i\}$ and their polyhedral recession cones $R^i = \{x \in \mathbb{R}^n \mid A^i x \leq 0\}$ and let Q^i be a (bounded) polytope such that $P^i = Q^i + R^i$. Define the set $S = conv\left(\cup_{i\in[m]} P^i\right)$ and polyhedron $P = conv(\cup_{i\in[m]} Q^i) + cone(\cup_{i\in[m]} R^i)$. Then the polyhedron*

$$Y = \begin{cases} A^i x^i \leq \gamma_i b^i, \ i \in [m] \\ \sum_{i\in[m]} x^i = x \\ \sum_{i\in[m]} \gamma_i = 1 \\ \gamma_i \geq 0, \ i \in [m] \end{cases} \tag{C.3}$$

provides an extended formulation of P. If each P^i, $i \in [m]$, is nonempty then $cl(S) = P$. Additionally, the vertices of Y have binary γ_i.

In the context of Theorem C.1 we also have the following result from Jeroslow [71] and Corollary 9.8.1 in Rockafellar [136]:

Theorem C.2. *If $P^1, \ldots, P^m$ are all nonempty and have identical recession cones then $S = P$ and so Y provides a polyhedral extended formulation for S.*

We would like to generalize the above theorems to allow for different combinations of polyhedra. To be precise, suppose Γ is a polyhedron in $\mathbb{R}^m$, and consider the set $\bigcup\{\sum_{i=1}^m \gamma_i P^i \mid \gamma \in \Gamma\}$. A

natural question is this: can we derive results similar in spirit to those of the preceding theorems? We answer this question in the affirmative, with a few restrictions on Γ.

To see what some of these restrictions must be, consider the challenges of using indicator variables as in (C.3). Suppose we have a polyhedron P with a representation $Ax \leq b$. Clearly $\gamma P = \{x \mid Ax \leq \gamma b\}$ for all $\gamma > 0$. The first issue is for $\gamma < 0$, $\gamma P = \{x \mid Ax \geq \gamma b\}$. This shows that allowing the sign to switch on γ will not allow the easy modeling of inequalities, and therefore we will, without loss of generality, only consider nonnegative indicator variables. Another issue dealing with the discontinuity of γP when γ is near 0 is that by definition $0P = \{0\}$ whereas $\{x \mid Ax \leq 0b\} = \{x \mid Ax \leq 0\}$, which is the polyhedral recession cone of P. This demonstrates that in a formulation like (C.3), while the indicator variables γ allow for "control" over the finite part of P, the recession directions of P are always included. Similarly, if P is empty, the polyhedral recession cone $\{x \mid Ax \leq 0\}$ is not, and will be included in a formulation like (C.3). For ease of exposition we will restrict ourselves to the case when each polyhedron is nonempty, but note that with some extra notation we could extend the results of Section A to include possibly empty polyhedra.

A The Extended Formulation

Now consider the set $S := \bigcup_{\gamma \in \Gamma} \left(\sum_{i=1}^{m} \gamma_i P^i \right)$, where P^i, $i \in [m]$, are nonempty polyhedra in $\mathbb{R}^n$ and $\Gamma \subseteq \mathbb{R}^m_+$ is a nonempty, nonnegative polyhedron. The goal is to arrive at a polyhedral representation for S. The exposition here follows that found in 21, Section 4.9.

Theorem C.3. *Consider m nonempty polyhedra $P^i = \{x \in \mathbb{R}^n \mid A^i x \leq b^i\}$, $i \in [m]$, and for each $i \in [m]$ let Q^i be a (bounded) polytope in $\mathbb{R}^n$ and R^i be a (closed convex) cone in $\mathbb{R}^n$ such that $P^i = Q^i + R^i$. Let $\Gamma \subseteq \mathbb{R}^m_+$ be a nonempty polyhedron. Consider the set $P := \bigcup_{\gamma \in \Gamma} \left(\sum_{i=1}^{m} \gamma_i Q^i + \sum_{i=1}^{m} R^i \right)$ and consider the polyhedron $Y \subseteq \mathbb{R}^{n+nm+m}$ defined by*

$$Y := \begin{cases} A^i x^i \leq \gamma_i b^i, \ i \in [m] \\ \sum_{i=1}^{m} x^i = x \\ (\gamma_1, \ldots, \gamma_m) = \gamma \in \Gamma. \end{cases} \tag{C.4}$$

Then $P = proj_x(Y) := \{x \in \mathbb{R}^n \mid \exists (x^1, \ldots, x^m, \gamma) \in \mathbb{R}^{nm+m} \text{ s.t. } (x, x^1, \ldots, x^m, \gamma) \in Y\}$. In particular, P is a polyhedron.

Proof. Let $x \in P$ (P is nonempty as the union of the sum of nonempty sets). There exists points $q^i \in Q^i$, $r^i \in R^i$ and $\gamma \in \Gamma$ such that $x = \sum_{i=1}^m \gamma_i q^i + \sum_{i=1}^m r^i$. Define $x^i = \gamma_i q^i + r^i$ for $i \in [m]$. Now by construction $x = \sum_{i=1}^m x^i$ and $A^i x^i = A^i(\gamma_i q^i + r^i) = \gamma_i A^i q^i + A^i r^i \leq \gamma_i b^i + 0$ for all $i \in [m]$. Hence $(x, x^1, \ldots, x^m, \gamma) \in Y$, so $P \subseteq \text{proj}_x(Y)$.

Conversely, let $(x, x^1, \ldots, x^m, \gamma) \in Y$. Consider $I^+ := \{i \mid \gamma_i > 0\}$ and $I^0 := \{i \mid \gamma_i = 0\}$. For $i \in I^+$, $A^i x^i \leq \gamma_i b^i$ and so $x^i \in \gamma_i Q^i + R^i$. For $i \in I^0$, $A^i x^i \leq 0$ and so $x^i \in R^i = \gamma_i Q^i + R^i$. Since $x = \sum_{i=1}^m x^i \in \sum_{i=1}^m (\gamma_i Q^i + R^i)$ and $\gamma \in \Gamma$, this shows $x \in P$, and hence $\text{proj}_x(Y) \subseteq P$.

As the projection of a polyhedron, P is itself a polyhedron. □

Remark C.1. For all $\Gamma \subseteq \mathbb{R}^m_+$, Y provides a polynomial-size (in $\dim(P^i)$ and $\dim(\Gamma)$) polyhedral representation of P. Further, if for all $i \in [m]$, P^i is bounded (i.e., $R^i = \{0\}$), then $P = S$ and Y provides a compact formulation for S.

Remark C.2. If $\Gamma \subseteq \mathbb{R}^m_{++}$ (the open, strictly positive orthant), then $\gamma_i P^i = \gamma_i Q^i + R^i$ $\forall (\gamma_1, \ldots, \gamma_m) \in \Gamma$. Therefore $P = S$ and so Y provides a compact formulation for S.

The next theorem demonstrates that $\text{cl}(S) = P$ with a restriction on Γ.

Theorem C.4. *Let $\Gamma \subseteq \mathbb{R}^m_+$ and $P^1, \ldots, P^m \subseteq \mathbb{R}^n$ be nonempty polyhedra. Suppose there exists $\hat{\gamma} \in \Gamma$ such that $\hat{\gamma}_i > 0$ $\forall i \in [m]$. Then for P and S defined as above, cl$(S) = P$.*

Proof. First consider $\text{cl}(S) \subseteq P$. Since P as a polyhedron is closed, it suffices to show $S \subseteq P$. Hence let $x \in S$. Then $\exists \gamma \in \Gamma$, $p^i \in P^i$ for $i \in [m]$ such that $x = \sum_{i=1}^m \gamma_i p^i$. As above for each $i \in [m]$, consider $P^i = Q^i + R^i$, so for each $i \in [m]$ we have $p^i = q^i + r^i$ for $q^i \in Q^i$ and $r^i \in R^i$. Thus $x = \sum_{i=1}^m \gamma_i q^i + \sum_{i=1}^m \gamma_i r^i$, and since $\gamma_i q^i \in \gamma_i Q^i$ and $\gamma_i r^i \in R^i$ (as R^i is a closed convex cone, $\gamma_i \geq 0$), we have $x \in P$.

Conversely, let $x \in P$. Then there exists $\gamma \in \Gamma$, $q^i \in Q^i$, and $r^i \in R^i$ such that $x = \sum_{i=1}^m \gamma_i q^i + \sum_{i=1}^m r^i$. By assumption $\exists \hat{\gamma} \in \Gamma$ that is strictly positive. By convexity, $(1-\epsilon)\gamma + \epsilon\hat{\gamma} \in \Gamma$ $\forall \epsilon \in (0,1)$; further $(1-\epsilon)\gamma + \epsilon\hat{\gamma} > 0$ $\forall \epsilon \in (0,1)$. Define $x^\epsilon := \sum_{i=1}^m [(1-\epsilon)\gamma_i + \epsilon\hat{\gamma}_i] q^i + \sum_{i=1}^m r^i$. Clearly $\lim_{\epsilon \to 0^+} x^\epsilon = x$, and we see that $x^\epsilon = \sum_{i=1}^m [(1-\epsilon)\gamma_i + \epsilon\hat{\gamma}_i](q^i + r^i/[(1-\epsilon)\gamma_i + \epsilon\hat{\gamma}_i])$. Since $q^i + r^i/[(1-\epsilon)\gamma_i + \epsilon\hat{\gamma}_i] \in P^i$ for $i \in [m]$, $\epsilon \in (0,1)$ and $(1-\epsilon)\gamma + \epsilon\hat{\gamma} \in \Gamma$ $\forall \epsilon \in (0,1)$, we have that $x^\epsilon \in S$ $\forall \epsilon \in (0,1)$. Hence $x \in \text{cl}(S)$. □

The requirement that Γ have a strictly positive element should not be seen as overly restrictive. If for some i, $\gamma_i = 0$ $\forall \gamma \in \Gamma$, then we should probably discard this particular P^i since it never contributes to the sum.

Remark C.3. If there exists $\hat{\gamma} \in \Gamma$ such that $\hat{\gamma} > 0$ and $P^1, \ldots, P^m$ are all nonempty, then Theorems C.3 and C.4 together imply that $\text{cl}(S) = \text{proj}_x(Y)$.

Theorem C.5. *Suppose $P^1, \ldots, P^m$ are nonempty polyhedra with identical recession cones, and $\Gamma \subset \mathbb{R}^m_+$ is a polyhedron such that $0 \notin \Gamma$. Then $S = \bigcup_{\gamma \in \Gamma} \left(\sum_{i=1}^m \gamma_i P^i \right)$ is a polyhedron and $S = proj_x(Y)$.*

Proof. Let $x \in P$. Then there exists $q^i \in Q^i$, $r^i \in R^i$ and $\gamma \in \Gamma$ such that $x = \sum_{i=1}^m \gamma_i q^i + \sum_{i=1}^m r^i$. By assumption there exist $j \in [m]$ such that $\gamma_j > 0$. As the P^i's have identical recession cones, we have $\sum_{i=1}^m r^i \in \gamma_j P^j$. Define $p^j = q^j + \sum_{i=1}^m r^i / \gamma_j$ and $p^i = q^i$ for $i \neq j$, and it follows that $x = \sum_{i=1}^m \gamma_i p^i$. Hence $x \in S$. The result then follows from Theorem C.3. □

Remark C.4. To see the necessity of $0 \notin \Gamma$, consider the sets S and P when $\gamma = 0$. If $P^1, \ldots, P^m$ have the same recession cone R, we see that $P\big|_{\gamma=0} = \sum_{i=1}^m 0Q^i + \sum_{i=1}^m R^i = R$, whereas $S\big|_{\gamma=0} = \sum_{i=1}^m 0P^i = \{0\}$, and $R = \{0\}$ if and only if all the P^i's are bounded. Hence we can do away with the assumption $0 \notin \Gamma$ in Theorem C.5 if all the P^i's are bounded.

It may be that Γ is the continuous relaxation of some integer set which determines the polyhedra P^i simultaneously allowed in the sum. The next theorem shows that vertices and extreme rays of Y have γ components which are vertices and extreme rays of Γ, hence if Γ is a perfect formulation for some integer set, vertices of Y will have integer γ. Further, even if Γ is not a perfect formulation, this shows that to find solutions with integer γ one need only consider cuts on Γ and not the entire polyhedron Y. For ease of notation, for $y \in Y$ define y_Γ to be the components of y in Γ. Finally, we note that a version of Theorem C.6 appears as Lemma 5 in [36], although it is restricted to the pure integer case, and the proof is merely sketched. We provide a complete proof and drop any assumption of integrality.

Theorem C.6. *$Y = conv(V) + cone(R)$, for finite sets V and R, where for each vertex $v \in V$, v_Γ is a vertex of Γ and for each extreme ray $r \in R$, r_Γ is an extreme ray of Γ. That is, $proj_\gamma(Y) = \Gamma$.*

Proof. Let $y \in Y$ such that $y = (x, x_1, \ldots, x_m, \gamma_1, \ldots, \gamma_m)$ and define $\gamma := y_\Gamma$. Since Γ is a polyhedron, by the Minkowski-Weyl theorem there exist vectors $v^1, \ldots, v^p, r^1, \ldots, r^q \in \mathbb{R}^m$ and $\lambda \in \mathbb{R}^p_+$, $\mu \in \mathbb{R}^q_+$ such that $\gamma = \sum_{k=1}^p \lambda_k v^k + \sum_{l=1}^q \mu_l r^l$ and $\sum_{k=1}^p \lambda_k = 1$. In particular, we have

$$\gamma_i = \sum_{k=1}^p \lambda_k v_i^k + \sum_{l=1}^q \mu_l r_i^l, \text{ with } \sum_{k=1}^p \lambda_k = 1, \ \forall i \in [m] \tag{C.5}$$

Let $I_+ = \{i \mid \gamma_i > 0\}$ and $I_0 = \{i \mid \gamma_i = 0\}$. Define

$$x_i^k := \begin{cases} x_i v_i^k/\gamma_i \text{ if } i \in I_+ \\ x_i \text{ if } i \in I_0 \text{ and } \lambda_k > 0 \\ \hat{x}_i^k \in v_i^k P^i \text{ if } i \in I_0 \text{ and } \lambda_k = 0 \end{cases} \qquad \forall k \in [p], \tag{C.6}$$

$$x_i^l := \begin{cases} x_i r_i^l/\gamma_i \text{ if } i \in I_+ \\ 0 \text{ if } i \in I_0 \text{ and } \mu_l > 0 \\ \hat{x}_i^l \in r_i^l P^i \text{ if } i \in I_0 \text{ and } \mu_l = 0 \end{cases} \qquad \forall l \in [q], \tag{C.7}$$

and $x^k = \sum_{i=1}^m x_i^k$ for $k \in [p]$ and $x^l = \sum_{i=1}^m x_i^l$ for $l \in [q]$. For $k \in [p]$ define $y^k := (x^k, x_1^k, \ldots, x_m^k, v_1^k, \ldots, v_m^k)$ and for $l \in [q]$ define $y^l := (x^l, x_1^l, \ldots, x_m^l, r_1^l, \ldots, r_m^l)$.

We first check the feasibility of the points constructed above. So for each $k \in [p]$, consider y^k. By construction $y_\Gamma^k \in \Gamma$ and $x^k = \sum_{i=1}^m x_i^k$, so for feasibility we need verify that $A^i x_i^k \leq v_i^k b^i$. Suppose $i \in I_+$, then $A^i x_i \leq \gamma_i b^i$, and multiplying both sides by v_i^k and dividing by γ_i shows x_i^k is feasible. Now suppose $i \in I_0$ and so $A^i x_i \leq 0$. If $\lambda_k > 0$, then we must have $v_i^k = 0$, so x_i^k is feasible. If $\lambda_k = 0$, x_i^k is feasible by construction (since each P^i is nonempty we can always find such a point $\hat{x}_i^k$). The feasibility of y^l for each $l \in [q]$ is similar.

Now we need show $y = \sum_{k=1}^p \lambda_k y^k + \sum_{l=1}^q \mu_l y^l$ to complete the proof. So first suppose $i \in I_+$, then $\sum_{k=1}^p \lambda_k x_i^k + \sum_{l=1}^q \mu_l x_i^l = \sum_{k=1}^p \lambda_k x_i v_i^k/\gamma_i + \sum_{l=1}^q \mu_l x_i r_i^l/\gamma_i = \frac{x_i}{\gamma_i}(\sum_{k=1}^p \lambda_k v_i^k + \sum_{l=1}^q \mu_l r_i^l) = x_i$. Conversely, suppose $i \in I_0$, then $\sum_{k=1}^p \lambda_k x_i^k + \sum_{l=1}^q \mu_l x_i^l = \sum_{k:\lambda_k>0} \lambda_k x_i + \sum_{k:\lambda_k=0} \lambda_k \hat{x}_i^k + \sum_{l:\mu_l>0} \mu_l 0 + \sum_{l:\mu_l=0} \mu_l \hat{x}_i^l = \sum_{k:\lambda_k>0} \lambda_k x_i + 0 + 0 + 0 = x_i \sum_{k:\lambda_k>0} \lambda_k = x_i$. It then follows, $\sum_{k=1}^p \lambda_k x^k + \sum_{l=1}^q \mu_l x^l = \sum_{k=1}^p \lambda_k \sum_{i=1}^m x_i^k + \sum_{l=1}^q \mu_l \sum_{i=1}^m x_i^l = \sum_{i=1}^m (\sum_{k=1}^p \lambda_k x_i^k + \sum_{l=1}^q \mu_l x_i^l) = \sum_{i=1}^m x_i = x$. Hence, we have shown $y = \sum_{k=1}^p \lambda_k y^k + \sum_{l=1}^q \mu_l y^l$ with $\lambda, \mu \geq 0$ and $\sum_{k=1}^p \lambda_k = 1$, proving the theorem. □

As mentioned, Theorem C.6 demonstrates that if Γ is a perfect formulation of some integer set and the variables x^i are continuous, then Y (under the given assumptions) provides a perfect formulation for $S\big|_{\mathbb{Z}_+} = \bigcup_{\gamma \in \Gamma \cap \mathbb{Z}_+} \{\sum_{i=1}^m \gamma_i P^i\}$. Noting that the polyhedron of Theorem 3.1 is exactly of this form, we see that the vertices of the polytope D must have integer γ.

Appendix D

Specification of UC formulations

In this appendix we specify the unit commitment formulations tested in Section 4.6. We use the same nomenclature as Chapter 4. All six formulations share a common set of constraints and variables on generator operation and system balance.

$$\sum_{g\in\mathcal{G}} \big(p_g(t) + \underline{P}_g u_g(t)\big) + p_W(t) = D(t) \qquad \forall t \in \mathcal{T} \qquad \text{(D.1a)}$$

$$\sum_{g\in\mathcal{G}} r_g(t) \geq R(t) \qquad \forall t \in \mathcal{T} \qquad \text{(D.1b)}$$

$$p_g(t) + r_g(t) \leq (\overline{P}_g - \underline{P}_g)u_g(t) - (\overline{P}_g - SU_g)v_g(t) \qquad \forall t \in \mathcal{T},\ \forall g \in \mathcal{G}^1 \qquad \text{(D.1c)}$$

$$p_g(t) + r_g(t) \leq (\overline{P}_g - \underline{P}_g)u_g(t) - (\overline{P}_g - SD_g)w_g(t{+}1) \qquad \forall t \in \mathcal{T},\ \forall g \in \mathcal{G}^1 \qquad \text{(D.1d)}$$

$$p_g(t) + r_g(t) \leq (\overline{P}_g - \underline{P}_g)u_g(t) - (\overline{P}_g - SU_g)v_g(t) - (\overline{P}_g - SD_g)w_g(t{+}1) \qquad \forall t \in \mathcal{T},\ \forall g \in \mathcal{G}^{>1} \qquad \text{(D.1e)}$$

$$p_g(t) + r_g(t) - p_g(t{-}1) \leq RU_g \qquad \forall t \in \mathcal{T},\ \forall g \in \mathcal{G} \qquad \text{(D.1f)}$$

$$p_g(t{-}1) - p_g(t) \leq RD_g \qquad \forall t \in \mathcal{T},\ \forall g \in \mathcal{G} \qquad \text{(D.1g)}$$

$$p_g(t) = \sum_{l\in\mathcal{L}_g} p_g^l(t) \qquad \forall t \in \mathcal{T},\ \forall g \in \mathcal{G} \qquad \text{(D.1h)}$$

$$p_g^l(t) \leq (\overline{P}_g^l - \overline{P}_g^{l-1}) \qquad \forall t \in \mathcal{T},\ \forall l \in \mathcal{L}_g,\ \forall g \in \mathcal{G} \qquad \text{(D.1i)}$$

$$u_g(t) - u_g(t{-}1) = v_g(t) - w_g(t) \qquad \forall t \in \mathcal{T},\ \forall g \in \mathcal{G} \qquad \text{(D.1j)}$$

$$\sum_{i=t-UT_g+1}^{t} v_g(i) \leq u_g(t) \qquad \forall t \in [UT_g, T],\ \forall g \in \mathcal{G} \qquad \text{(D.1k)}$$

$$\sum_{i=t-DT_g+1}^{t} w_g(i) \leq 1 - u_g(t) \qquad \forall t \in [DT_g, T],\ \forall g \in \mathcal{G} \qquad \text{(D.1l)}$$

$$p_W(t) \leq W(t) \qquad \forall t \in \mathcal{T} \qquad \text{(D.1m)}$$

$$p_g^l(t) \in \mathbb{R}_+ \qquad \forall t \in \mathcal{T},\ \forall l \in \mathcal{L}_g,\ \forall g \in \mathcal{G} \qquad \text{(D.1n)}$$

$$p_g(t), r_g(t) \in \mathbb{R}_+ \qquad \forall t \in \mathcal{T},\ \forall g \in \mathcal{G} \qquad \text{(D.1o)}$$

$$p_W(t) \in \mathbb{R}_+ \qquad \forall t \in \mathcal{T} \qquad \text{(D.1p)}$$

$$u_g(t), v_g(t), w_g(t) \in \{0,1\} \qquad \forall t \in \mathcal{T},\ \forall g \in \mathcal{G}. \qquad \text{(D.1q)}$$

A One Binary Formulation (1-bin)

$$\min \sum_{g \in \mathcal{G}} \sum_{t \in \mathcal{T}} \left(\sum_{l \in \mathcal{L}_g} (c_g^l\, p_g^l(t)) + c_g^R\, u_g(t) + c_g^{SU}(t) \right) \qquad \text{(D.2a)}$$

subject to:

Constraints (D.1a) – (D.1q)

$$c_g^{SU}(t) \geq c_g^s \left(u_g(t) - \sum_{i=1}^{\underline{T}_g^s} u_g(t-i) \right) \qquad \forall s \in \mathcal{S}_g,\ \forall g \in \mathcal{G},\ \forall t \in \mathcal{T} \qquad \text{(D.2b)}$$

$$c_g^{SU}(t) \geq 0 \qquad \forall g \in \mathcal{G},\ \forall t \in \mathcal{T} \qquad \text{(D.2c)}$$

B Strengthened One Binary Formulation (1-Bin*)

$$\min \sum_{g \in \mathcal{G}} \sum_{t \in \mathcal{T}} \left(\sum_{l \in \mathcal{L}_g} (c_g^l\, p_g^l(t)) + c_g^R\, u_g(t) + c_g^{SU}(t) \right) \qquad \text{(D.3a)}$$

subject to:

Constraints (D.1a) – (D.1q)

$$c_g^{SU}(t) \geq c_g^s \left(u_g(t) - \sum_{i=1}^{DT_g} u_g(t-i) \right) - \sum_{k=1}^{s-1} \left((c_g^s - c_g^k) \sum_{i=\underline{T}_g^k+1}^{\overline{T}_g^k} u_g(t-i) \right) \quad \forall s \in \mathcal{S}_g,\ \forall g \in \mathcal{G},\ \forall t \in \mathcal{T} \qquad \text{(D.3b)}$$

$$c_g^{SU}(t) \geq 0 \qquad \forall g \in \mathcal{G},\ \forall t \in \mathcal{T} \tag{D.3c}$$

C Three Binary Formulation (3-bin)

$$\min \sum_{g\in\mathcal{G}} \sum_{t\in\mathcal{T}} \left(\sum_{l\in\mathcal{L}_g} (c_g^l\, p_g^l(t)) + c_g^R\, u_g(t) + c_g^{SU}(t) \right) \tag{D.4a}$$

subject to:

Constraints (D.1a) – (D.1q)

$$c_g^{SU}(t) \geq c_g^s v_g(t) - \sum_{k=1}^{s-1} \left((c_g^s - c_g^k) \sum_{i=\underline{T}_g^k}^{\overline{T}_g^k - 1} w_g(t-i) \right) \qquad \forall s \in \mathcal{S}_g,\ \forall g \in \mathcal{G},\ \forall t \in \mathcal{T} \tag{D.4b}$$

$$c_g^{SU}(t) \geq 0 \qquad \forall g \in \mathcal{G},\ \forall t \in \mathcal{T} \tag{D.4c}$$

D Startup Type Indicator Formulation (STI)

$$\min \sum_{g\in\mathcal{G}} \sum_{t\in\mathcal{T}} \left(\sum_{l\in\mathcal{L}_g} (c_g^l\, p_g^l(t)) + c_g^R\, u_g(t) + \sum_{s=1}^{S} c^s \delta^s(t) \right) \tag{D.5a}$$

subject to:

Constraints (D.1a) – (D.1q)

$$\delta_g^s(t) \leq \sum_{i=\underline{T}_g^s}^{\overline{T}_g^s - 1} w_g(t-i) \qquad \forall s \in \mathcal{S}_g \setminus S_g,\ \forall g \in \mathcal{G},\ \forall t \in \mathcal{T} \tag{D.5b}$$

$$v_g(t) = \sum_{s=1}^{S_g} \delta_g^s(t) \qquad \forall g \in \mathcal{G},\ \forall t \in \mathcal{T}. \tag{D.5c}$$

E Matching Formulation (Match)

$$\min \sum_{g\in\mathcal{G}} \sum_{t\in\mathcal{T}} \left(\sum_{l\in\mathcal{L}_g} (c_g^l\, p_g^l(t)) + c_g^R\, u_g(t) + c_g^S v(t) + \sum_{s=1}^{S_g - 1} (c_g^s - c_g^S) \left(\sum_{t'=t-\underline{T}_g^s}^{t-\overline{T}_g^s + 1} x_g(t', t) \right) \right) \tag{D.6a}$$

subject to:

Constraints (D.1a) – (D.1q)

$$\sum_{t'=t-TC_g+1}^{t-DT_g} x_g(t',t) \leq v_g(t) \qquad \forall g \in \mathcal{G},\ \forall t \in \mathcal{T} \tag{D.6b}$$

$$\sum_{t'=t+DT_g}^{t+TC_g-1} x_g(t,t') \leq w_g(t) \qquad \forall g \in \mathcal{G},\ \forall t \in \mathcal{T}, \tag{D.6c}$$

F Extended Formulation (EF)

$$\min \sum_{g\in\mathcal{G}}\sum_{t\in\mathcal{T}}\left(\sum_{l\in\mathcal{L}_g}(c_g^l\, p_g^l(t)) + c_g^R\, u_g(t) + \sum_{s=1}^{S_g} c_g^s \left(\sum_{t'=t-\overline{T}^s+1}^{t-\underline{T}^s} x_g(t',t)\right)\right) \tag{D.7a}$$

subject to:

Constraints (D.1a) – (D.1q)

$$\sum_{\{t'|t'>t\}} y_g(t,t') = v_g(t) \qquad \forall g \in \mathcal{G},\ \forall t \in \mathcal{T} \tag{D.7b}$$

$$\sum_{\{t'|t'<t\}} y_g(t',t) = w_g(t) \qquad \forall g \in \mathcal{G},\ \forall t \in \mathcal{T} \tag{D.7c}$$

$$\sum_{\{t'|t'<t\}} x_g(t',t) = v_g(t) \qquad \forall g \in \mathcal{G},\ \forall t \in \mathcal{T} \tag{D.7d}$$

$$\sum_{\{t'|t'>t\}} x_g(t,t') = w_g(t) \qquad \forall g \in \mathcal{G},\ \forall t \in \mathcal{T} \tag{D.7e}$$

$$\sum_{\{\tau,\tau'|\tau\leq t<\tau'\}} y_g(\tau,\tau') = u_g(t) \qquad \forall g \in \mathcal{G},\ \forall t \in \mathcal{T}. \tag{D.7f}$$

Appendix E

Detailed Computational Results for Chapter 4

This is an appendix detailing the computational results in Chapter 4. Here we present complete tables of computational results used to make the summary tables in Chapter 4 as well as a statical analysis of the computational performance of the various formulations.

A Computational Results

In this section we present full tables for the computational results reported in Chapter 4. The computational platform used for all experiments is a Dell PowerEdge T620 with two Intel Xeon E5-2670 processors for a total of 16 cores and 32 threads, 256GB of RAM, running the Ubuntu 14.04.5 operating system. The latest major versions of Gurobi (7.0.1) and CPLEX (12.7.1.0) were used when the experiments were conducted.

When referring to a startup cost formulation, we use the same notation as in Chapter 4. That is, "EF" is the extended formulation from [131], "Match" is the matching formulation introduced in Chapter 4, "STI" is the startup type indicator formulation introduced in Chapter 4, "3-bin" is the three-binary formulation also introduced in Chapter 4, "1-bin*" is the strengthened one-binary formulation introduced in [144], and "1-bin" is the typical formulation in the generator's status variables from [115, 17].

We use the same base unit commitment model to benchmark the different startup cost formulations, the full specification of which can be found in Appendix D.

Table E.1: Gurobi Computational Results for CAISO Instances: Wall Clock Time. When instances are solved to optimality, reported quantities are seconds to solution. Otherwise, reported quantities in parentheses are the optimality gap after 600 seconds.

Instance	EF	Match	STI	3-bin	1-bin*	1-bin
2014-09-01 0%	357.21	**30.24**	46.80	53.55	(0.028%)	(0.041%)
2014-12-01 0%	169.30	23.89	**23.06**	65.81	(0.073%)	(0.068%)
2015-03-01 0%	166.04	24.68	41.76	**16.46**	(0.042%)	(0.053%)
2015-06-01 0%	163.38	**12.64**	18.76	24.74	(0.017%)	(0.020%)
Scenario400 0%	335.67	**26.60**	65.02	173.37	(0.403%)	(0.383%)
2014-09-01 1%	462.78	**20.39**	22.44	31.83	(0.055%)	(0.045%)
2014-12-01 1%	381.80	36.43	**28.90**	85.78	(0.072%)	(0.069%)
2015-03-01 1%	178.40	**20.41**	35.08	67.20	(0.079%)	(0.090%)
2015-06-01 1%	274.25	41.60	**39.03**	70.83	(0.020%)	(0.028%)
Scenario400 1%	(0.012%)	**46.08**	83.29	182.19	(0.376%)	(0.446%)
2014-09-01 3%	598.73	75.69	**63.26**	87.48	(0.043%)	(0.036%)
2014-12-01 3%	(0.011%)	63.64	**54.88**	93.39	(0.083%)	(0.087%)
2015-03-01 3%	217.10	**48.91**	73.06	99.57	(0.112%)	(0.110%)
2015-06-01 3%	329.79	84.66	**38.13**	83.26	(0.024%)	(0.022%)
Scenario400 3%	(0.013%)	**129.50**	243.01	356.10	(0.495%)	(0.538%)
2014-09-01 5%	412.24	46.80	**44.92**	119.49	(0.037%)	(0.037%)
2014-12-01 5%	(0.012%)	**86.41**	107.14	113.69	(0.104%)	(0.082%)
2015-03-01 5%	(0.010%)	**83.49**	87.22	94.95	(0.115%)	(0.105%)
2015-06-01 5%	(0.010%)	**28.28**	66.97	151.47	(0.031%)	(0.031%)
Scenario400 5%	(0.014%)	115.02	**107.02**	(0.014%)	(0.514%)	(0.570%)
Geometric Mean:	>370.3	**43.12**	52.84	>91.43	>600	>600

A.1 CAISO Instances

We report the computational experiments based on the "CAISO" generators, which are based on real-world market data from the California Independent System Operator. This test set has 610 generators. Four 48-hour demand scenrios are based on historical data corresponding to the date listed (2014-09-01, 2014-12-01, 2015-03-01, 2015-06-01), and one hypothetical high-wind scenario where wind supply is on average 40% of energy demanded (Scenario400).

For each scenario we considered four reserve levels: 0%, 1%, 3%, and 5%. In Tables E.1 - E.5 for each instance we report the demand/wind scenario followed by the reserve level. A 600 second time limit was imposed for these instances for both solvers.

Gurobi 7.0.1

All Gurobi settings besides the time limit were left at defaults. In Table E.1 we report the wall-clock time reported by Gurobi at termination, or if Gurobi hit the 600 second time-limit, we report in parentheses the terminating optimality gap. In the last row we report the geometric mean solve time across the 20 instances for each formulation, inserting 600 seconds into the calculation in the event the solver times out.

As we can see, the EF, 1-bin* and 1-bin variants are uncompetitive. The EF variant is large in comparison to the others, which significantly slows down the initial LP solve as well as root node processing (i.e., heuristics and cut-generation). Conversely, the 1-bin and 1-bin* variants are more compact than Match or STI, but the overall weakness of the formulations (see Table E.5) prevents Gurobi from finding and certifying an optimal solution (with $< 0.01\%$ optimality gap) within the time limit. The 3-bin variant is as compact as the 1-bin variants, and while it is more competitive than the latter, for nearly all of these instances it comes in 3rd place behind Match and STI, and fails to solve in one case. The Match and STI variants have broadly similar performance, with Match pulling ahead given its advantage in the hypothetical Scenario400. Based on computational time these instances are "easy" for both Match and STI, in the sense that all 20 instances solve to optimality in under 5 minutes.

In Table E.2 we report the number of branch-and-cut nodes explored by Gurobi at termination, indicating with a * when the solver terminated because of the 600 second time limit. In the last row we report the shifted geometric mean node count across all twenty instances (this value is calculated by adding 1 to all node counts and then computing the geometric mean, so as to avoid multiplication by 0).

As remarked above, because the EF variant is so large, Gurobi only leaves the root node for the EF variant in one instance, and in all other cases Gurobi either finds an optimal solution at the root note or hits the wall-clock limit before beginning to branch. On average Gurobi uses slightly fewer nodes for the Match variant over the STI, and similarly 3-bin, when it solves, only uses a few more nodes on average than STI. Turning to the 1-bin variants, we can see Gurobi processed several thousand nodes in each instance before hitting the time limit, which was not enough to overcome the weakness of these formulations.

Table E.2: Gurobi Computational Results for CAISO Instances: Nodes Explored. Cells report the number of tree nodes explored during branch-and-cut search. Entries with a terminating "*" report the number of tree nodes explored when the 600 second time limit is hit. Otherwise, the entries represent the number of tree nodes required to identify an optimal solution.

Instance	EF	Match	STI	3-bin	1-bin*	1-bin
2014-09-01 0%	**0**	**0**	110	**0**	11895*	9285*
2014-12-01 0%	**0**	**0**	**0**	3	5904*	4067*
2015-03-01 0%	**0**	**0**	**0**	**0**	2565*	4419*
2015-06-01 0%	**0**	**0**	**0**	**0**	7247*	15893*
Scenario400 0%	**0**	**0**	**0**	2373	5604*	5801*
2014-09-01 1%	**0**	**0**	**0**	**0**	7164*	10934*
2014-12-01 1%	**0**	95	**0**	160	4072*	7144*
2015-03-01 1%	**0**	**0**	**0**	**0**	2415*	2684*
2015-06-01 1%	**0**	47	47	**0**	11853*	5499*
Scenario400 1%	0*	**0**	47	1854	7210*	6542*
2014-09-01 3%	**0**	7	31	31	14071*	12852*
2014-12-01 3%	0*	1292	366	**144**	3900*	6024*
2015-03-01 3%	**0**	**0**	58	1	2476*	2124*
2015-06-01 3%	**0**	**0**	**0**	**0**	8362*	5153*
Scenario400 3%	0*	**2055**	2874	2309	6518*	6657*
2014-09-01 5%	**0**	95	147	2497	7763*	9316*
2014-12-01 5%	0*	1203	**40**	138	4497*	3681*
2015-03-01 5%	3783*	923	2971	**758**	2264*	2263*
2015-06-01 5%	0*	**0**	100	1125	3894*	5783*
Scenario400 5%	0*	3867	**140**	3848*	6684*	6907*
Shifted Geo. Mean:	>1.510	**13.50**	20.22	>39.09	>5914	>5827

CPLEX 12.7.1.0

In Table E.3 we report the wall-clock time required by CPLEX to reach an optimal solution, or if the solver hit the time limit, we report the optimality gap at termination in parentheses.

Overall CPLEX performs better on these instances than Gurobi, but we still see the EF and 1-bin variants are uncompetitive. CPLEX is able to solve all the instances using Match, STI, and 3-bin within the time limit, but it is obvious that 3-bin is the inferior of these three: in the worst case (Scenario400 5%) it needs 423 seconds, whereas STI in the worst case needs 223 seconds and Match only needs 110 seconds in the worst case. Though STI outperforms Match in mean solve time, this highlights that Match has flatter performance profile than STI on this instances. In a similar fashion, we see that 3-bin is sometimes the fastest, but for all the high-wind Scenario400 instances it performs significantly worse than Match or STI. 3-bin underperformed on these instances for Gurobi as well. This is in spite of the fact that STI and 3-bin exhibit the same optimality gap

Table E.3: CPLEX Computational Results for CAISO Instances: Wall Clock Time. When instances are solved to optimality, reported quantities are seconds to solution. Otherwise, reported quantities in parentheses are the optimality gap after 600 seconds.

Instance	EF	Match	STI	3-bin	1-bin*	1-bin
2014-09-01 0%	277.51	47.36	**32.54**	38.42	(0.060%)	(0.080%)
2014-12-01 0%	176.86	**25.03**	30.41	45.46	(0.103%)	(0.118%)
2015-03-01 0%	167.34	22.82	**19.78**	20.76	(0.082%)	(0.111%)
2015-06-01 0%	141.00	19.68	19.97	**19.01**	(0.023%)	(0.056%)
Scenario400 0%	189.98	**39.40**	52.17	218.87	(0.801%)	(0.956%)
2014-09-01 1%	245.80	46.41	**19.75**	24.74	(0.068%)	(0.097%)
2014-12-01 1%	189.27	41.73	**36.97**	66.30	(0.111%)	(0.146%)
2015-03-01 1%	172.37	37.36	36.69	41.24	(0.059%)	(0.102%)
2015-06-01 1%	188.68	34.94	30.25	**19.89**	(0.026%)	(0.064%)
Scenario400 1%	288.39	63.93	**53.29**	319.76	(0.818%)	(0.874%)
2014-09-01 3%	415.29	62.67	40.45	**37.27**	(0.067%)	(0.108%)
2014-12-01 3%	292.17	72.06	**52.07**	101.31	(0.120%)	(0.135%)
2015-03-01 3%	346.39	58.97	**43.02**	55.51	(0.129%)	(0.102%)
2015-06-01 3%	180.42	38.56	20.41	**19.97**	(0.066%)	(0.073%)
Scenario400 3%	(0.012%)	**110.20**	140.43	420.13	(0.678%)	(0.774%)
2014-09-01 5%	272.24	60.57	**29.44**	53.30	(0.065%)	(0.091%)
2014-12-01 5%	381.81	75.47	**62.67**	102.32	(0.134%)	(0.164%)
2015-03-01 5%	273.29	**35.96**	44.08	58.95	(0.135%)	(0.161%)
2015-06-01 5%	291.17	71.24	**38.51**	59.93	(0.039%)	(0.095%)
Scenario400 5%	(0.012%)	**94.47**	222.68	422.80	(0.766%)	(1.384%)
Geometric Mean:	>261.1	48.02	**40.75**	62.87	>600	>600

on all these instances (see Table E.5). One possible explanation of this phenomenon is the extra indicator variables $\delta_g^s(t)$ make it easier for both Gurobi and CPLEX to generate strong cutting planes. Another possibility is that branching on these indicator variables is often advantageous.

In Table E.4 we report the number of branch-and-cut nodes CPLEX explored during search, with a * indicating that the solver terminated because it reached the 600 second wall-clock limit. In the last row we report the shifted geometric mean across the 20 instances, which is calculated the same way it was in Table E.2.

We see that for the tighter formulations (EF, Match, STI, and 3-bin), CPLEX often finds and proves an optimal solution at the root node or only a few nodes into the tree. For the 1-bin variants, CPLEX often explores more than 10000 nodes before hitting the wall-clock time limit. Additionally, considering instances Scenario400 3% and Scenario400 5%, we observe that Match was able to out-perform STI on these instances because it required less enumeration. Similarly,

Table E.4: CPLEX Computational Results for CAISO Instances: Nodes Explored. Cells report the number of tree nodes explored during branch-and-cut search. Entries with a terminating "*" report the number of tree nodes explored when the 600 second time limit is hit. Otherwise, the entries represent the number of tree nodes required to identify an optimal solution.

Instance	EF	Match	STI	3-bin	1-bin*	1-bin
2014-09-01 0%	**0**	**0**	**0**	**0**	33839*	31955*
2014-12-01 0%	**0**	**0**	**0**	**0**	15956*	17895*
2015-03-01 0%	**0**	**0**	**0**	**0**	18359*	17284*
2015-06-01 0%	**0**	**0**	**0**	**0**	26114*	23025*
Scenario400 0%	**0**	**0**	**0**	5701	8586*	6955*
2014-09-01 1%	**0**	**0**	**0**	**0**	35138*	28739*
2014-12-01 1%	**0**	**0**	**0**	263	16691*	15893*
2015-03-01 1%	**0**	**0**	**0**	**0**	25618*	23716*
2015-06-01 1%	**0**	**0**	**0**	**0**	24161*	20693*
Scenario400 1%	**0**	**0**	**0**	5748	8766*	7822*
2014-09-01 3%	45	41	126	**0**	24013*	26919*
2014-12-01 3%	2	**0**	21	1531	13099*	14593*
2015-03-01 3%	5	**0**	2	137	16259*	20094*
2015-06-01 3%	**0**	**0**	**0**	**0**	10014*	18983*
Scenario400 3%	803*	**508**	5662	5833	6445*	6008*
2014-09-01 5%	**0**	3	**0**	38	23902*	23423*
2014-12-01 5%	43	34	**2**	2700	7752*	10689*
2015-03-01 5%	**0**	**0**	36	40	14412*	10718*
2015-06-01 5%	3	4	**0**	6	16081*	11000*
Scenario400 5%	1166*	**62**	6378	5815	5922*	5944*
Shifted Geo. Mean:	>3.60	**2.83**	4.75	>32.6	>15442	>15210

3-bin requires more than 5000 nodes on each of the Scenario400 instances, explaining its relative weakness on these high-wind instances.

Relative Integrality Gap

In Table E.5 we report the relative integrality gap for each instance and formulation. This is calculated by solving the LP relaxation for each problem and instance, which has value z^*_{LP}, and comparing that to the best integer solution found across all twelve runs for each instance, z^*_{IP}. The corresponding integrality gap can be then calculated by appealing to the formula

$$\text{relative integrality gap} = \frac{z^*_{IP} - z^*_{LP}}{z^*_{IP}}. \tag{E.1}$$

The values in Table E.5 report this ratio as a percentage.

Table E.5: Computational Results for CAISO Instances: Relative Integrality Gap (%).

Instance	EF	Match	STI	3-bin	1-bin*	1-bin
2014-09-01 0%	0.0097	0.0097	0.0229	0.0229	0.9878	1.0506
2014-12-01 0%	0.0058	0.0058	0.0190	0.0190	1.0813	1.1370
2015-03-01 0%	0.0020	0.0020	0.0270	0.0270	1.5774	1.5774
2015-06-01 0%	0.0012	0.0012	0.0102	0.0102	0.8885	0.8915
Scenario400 0%	0.0113	0.0113	0.1288	0.1288	4.6156	4.6972
2014-09-01 1%	0.0106	0.0106	0.0239	0.0239	1.0058	1.0682
2014-12-01 1%	0.0059	0.0059	0.0198	0.0198	1.0906	1.1509
2015-03-01 1%	0.0037	0.0037	0.0326	0.0326	1.6411	1.6411
2015-06-01 1%	0.0044	0.0044	0.0134	0.0134	0.9105	0.9105
Scenario400 1%	0.0128	0.0128	0.1302	0.1302	4.6721	4.7553
2014-09-01 3%	0.0149	0.0149	0.0283	0.0283	1.0452	1.1093
2014-12-01 3%	0.0089	0.0089	0.0245	0.0245	1.1165	1.1803
2015-03-01 3%	0.0119	0.0119	0.0428	0.0428	1.7416	1.7446
2015-06-01 3%	0.0087	0.0087	0.0180	0.0180	0.9373	0.9451
Scenario400 3%	0.0201	0.0201	0.1372	0.1372	4.7249	4.8072
2014-09-01 5%	0.0081	0.0081	0.0217	0.0217	1.0657	1.1348
2014-12-01 5%	0.0107	0.0107	0.0265	0.0265	1.1415	1.2094
2015-03-01 5%	0.0091	0.0091	0.0459	0.0459	1.7700	1.7798
2015-06-01 5%	0.0084	0.0084	0.0181	0.0181	0.9474	0.9559
Scenario400 5%	0.0237	0.0237	0.1400	0.1400	4.7721	4.8568
Geometric Mean:	0.0079	0.0079	0.0328	0.0328	1.5251	1.5688

Examining Table E.5, we see that EF and Match, as well as STI and 3-bin, always have identical gaps. One way of viewing the observed equivalence of EF and Match is that although Match is not a perfect formulation for startup costs like EF is, the only vertices that are fractional in Match are sub-optimal – at least for reasonable (i.e., concave increasing) startup costs. We suspect a similar situation is playing itself out in the comparison between STI and 3-bin. Turning to the 1-bin variants, we see the optimality gap for these is quite large relative to the other formulations tested, which helps to explain their weak computational performance. Additionally, across these instances Match is able to close 40-90% of the integrality gap over STI, which helps explain its performance despite requiring more integer variables.

A.2 FERC Instances

We report the computational experiments based on the "FERC" generators, which are drawn from the RTO Unit Commitment Test System provided by the Federal Energy Regulatory Commission [84], which itself is based on market data gathered from the PJM Interconnection.

The FERC set of generators consists of a "Winter" set and a "Summer" set, and each test set has approximately 900 generators. Demand, reserve, and wind scenarios for 2015 were constructed based on market data available on the PJM website [128, 129]. Twelve days were selected from 2015, one from each month, to create a variety of scenarios. We used the Summer generators for the months April – September and the Winter generators for the remaining months.

Using the data collected, we determined wind power was 2% of load, on average, in 2015. We created then for each day selected two scenarios, one with the actual wind data from 2015 (2% Wind Penetration), and another where the wind data from 2015 was multiplied by a constant factor of 15 (30% Wind Penetration). Hence the 2% wind scenarios correspond to the problem facing system operators today, whereas the 30% wind scenarios correspond to problems that system operators may face in the future under high renewables penetration.

For all solvers a time limit of 900 seconds was imposed for these computational experiments.

Gurobi 7.0.1

Because this test set is larger than CAISO, Gurobi often selects the deterministic concurrent optimizer to solve the root LP (this solves the root node using one core for primal simplex, one core for dual simplex, and the remaining cores for parallel barrier). Preliminary experiments showed that this choice resulted in a random, and often large (i.e. greater than 30 seconds), "concurrent spin time," which is the time spend ensuring this concurrent LP solver is deterministic. Gurobi recommended setting the `Method` parameter to 3 to eliminate this lag, which selects the non-deterministic concurrent optimizer. This is the same LP solver without the logic to ensure determinism. Hence we set the `Method` parameter to 3 for the FERC experiments on Gurobi. As the solver almost always solved the root LPs in this case using parallel barrier, a practitioner wanting to ensure determinism could set the `Method` parameter to 2 without loosing performance.

In Table E.6 we report the wall-clock time for the FERC instances, inserting in parentheses the terminating optimality gap when the solver hits the time limit of 900 seconds. (In the 2% wind penetration case, for the 1-bin formulation, instance 2015-07-01, Gurobi found an optimal solution before the solver terminated, so we report the time.)

For the 2% wind instances we observe the 1-bin variants perform better than the CASIO instances, but they are still uncompetitive with Match, STI, and 3-bin variants. The EF is similarly uncompetitive. We see that 3-bin is significantly worse than Match or STI, and for one instance (2015-09-01) takes over 800 seconds to find an optimal solution, whereas STI in the worst case needs

Table E.6: Gurobi Computational Results for FERC Instances: Wall Clock Time. When instances are solved to optimality, reported quantities are seconds to solution. Otherwise, reported quantities in parentheses are the optimality gap after 900 seconds.

(a) 2% Wind Penetration

Instance	EF	Match	STI	3-bin	1-bin*	1-bin
2015-01-01	511.91	**111.34**	193.07	241.63	(0.017%)	(0.046%)
2015-02-01	586.95	**85.12**	314.07	463.66	(0.143%)	(0.172%)
2015-03-01	807.3	**152.24**	177.44	245.77	649.54	596.24
2015-04-01	(0.012%)	190.62	321.6	**177.27**	675.45	660.92
2015-05-01	512.55	**177.51**	191.29	186.68	334.17	416.03
2015-06-01	619.8	142.57	**139.16**	211.92	406.68	575.42
2015-07-01	(0.017%)	411.00	491.22	**260.41**	(0.014%)	901.87
2015-08-01	808.34	**113.13**	350.52	449.67	(0.11%)	(0.165%)
2015-09-01	(0.016%)	313.79	**284.31**	840.5	(0.101%)	(0.113%)
2015-10-01	605.11	132.95	**113.69**	133.48	582.63	582.58
2015-11-02	573.13	**109.88**	200.83	209.22	(0.073%)	(0.136%)
2015-12-01	(0.013%)	116.25	**114.15**	242.18	(0.055%)	(0.105%)
Geometric Mean:	>701.4	**153.60**	218.36	266.53	>710.7	>738.6

(b) 30% Wind Penetration

Instance	EF	Match	STI	3-bin	1-bin*	1-bin
2015-01-01	712.53	**127.22**	(0.902%)	(1.334%)	(4.083%)	(3.808%)
2015-02-01	612.38	**114.78**	(0.043%)	(0.158%)	(0.952%)	(0.959%)
2015-03-01	895.97	647.78	**480.77**	496.35	(0.386%)	(0.460%)
2015-04-01	(0.024%)	**140.82**	236.23	425.71	(0.276%)	(1.054%)
2015-05-01	(0.016%)	**104.62**	119.06	110.24	312.33	337.55
2015-06-01	698.12	222.54	141.18	**110.06**	(0.408%)	(0.101%)
2015-07-01	(0.015%)	**126.98**	346.18	230.15	(0.222%)	(0.105%)
2015-08-01	(0.019%)	395.87	379.42	**227.92**	(0.768%)	(0.870%)
2015-09-01	(0.012%)	**245.73**	780.9	(0.035%)	(0.254%)	(0.256%)
2015-10-01	(0.036%)	439.03	**352.54**	533.72	617.14	607.19
2015-11-02	789.73	**182.40**	618.73	782.18	(0.803%)	(1.065%)
2015-12-01	674.84	**312.67**	361.35	421.08	(0.035%)	(0.035%)
Geometric Mean:	>807.9	**214.70**	>390.3	>400.9	>798.5	>802.6

only 491 seconds (2015-07-01), and Match in the worst case needs only 411 seconds (2015-07-01). Overall Match outperforms the other variants on these instances using Gurobi.

Turning to the 30% wind instances, we first note that Match is the only variant that solves all 12 instances, and also dominants the other variants in geometric mean solve time. It is interesting to note, turning to a moment to Table E.10, that for the 2015-01-01 and 2015-02-01 instances, Match is able to close 95% and 67% of the integrality gap, respectively, over STI. This explains why only

Table E.7: Gurobi Computational Results for FERC Instances: Nodes Explored. Cells report the number of tree nodes explored during branch-and-cut search. Entries with a terminating "*" report the number of tree nodes explored when the 900 second time limit is hit.

(a) 2% Wind Penetration

Instance	EF	Match	STI	3-bin	1-bin*	1-bin
2015-01-01	**0**	**0**	**0**	**0**	1443*	101*
2015-02-01	**0**	**0**	47	976	3864*	3964*
2015-03-01	**0**	**0**	**0**	**0**	1537	1487
2015-04-01	0*	**0**	**0**	**0**	**0**	**0**
2015-05-01	**0**	**0**	**0**	**0**	**0**	**0**
2015-06-01	**0**	**0**	**0**	**0**	**0**	**0**
2015-07-01	0*	**0**	123	47	281*	505
2015-08-01	**0**	**0**	720	1756	1194*	420*
2015-09-01	0*	**46**	425	3543	4087*	4613*
2015-10-01	**0**	**0**	**0**	**0**	**0**	**0**
2015-11-02	**0**	**0**	**0**	**0**	15*	15*
2015-12-01	0*	**0**	**0**	**0**	46*	30*
Shifted Geo. Mean:	>1.00	**1.38**	5.91	9.03	>67.5	>50.8

(b) 30% Wind Penetration

Instance	EF	Match	STI	3-bin	1-bin*	1-bin
2015-01-01	**0**	**0**	4440*	7761*	147*	1592*
2015-02-01	**0**	**0**	2067*	2079*	3791*	3387*
2015-03-01	**0**	47	550	47	31*	47*
2015-04-01	0*	**0**	**0**	47	1858*	79*
2015-05-01	0*	**0**	**0**	**0**	**0**	**0**
2015-06-01	**0**	**0**	**0**	**0**	31*	15*
2015-07-01	0*	**0**	**0**	**0**	15*	47*
2015-08-01	0*	879	60	**0**	15*	31*
2015-09-01	0*	**0**	2573	4180*	5759*	4241*
2015-10-01	0*	2501	1775	2199	2100	**1161**
2015-11-02	**0**	**0**	256	390	31*	15*
2015-12-01	**0**	**0**	**0**	387	655*	603*
Shifted Geo. Mean:	>1.00	**4.66**	>51.7	>78.2	>142	>130

Match and EF were able to solve these instances within the time limit. Here again we find again that the EF and 1-bin variants are uncompetitive, and while 3-bin is sometimes the fastest to a solution (e.g. 2015-08-01), it exhibits more performance variability than either STI or Match.

In Table E.7 we report the number of branch-and-cut nodes explored at termination; instances when Gurobi terminated because the time limit of 900 seconds was reached are denoted with a *.

In the last row we report the shifted geometric mean across the twelve runs of each wind type, which is calculated the same way it was for Table E.5.

Across both wind levels it is interesting to note that Gurobi often spends the majority of the time at the root node, first solving the LP and then in cut generation and root-node heuristics. For the largest formulation, EF, Gurobi either finds the optimal at the root node or terminates without having branched. Additionally, the low node count observed in most of the instances for the 1-bin variants reflect this fact as well; Gurobi spends most of the time at the root node attempting to tighten this formulations with cuts. Using the Match variant Gurobi solves nearly all the 2% wind instances at the root node, and only has to explore a significant portion of the tree for a few of the 30% wind instances.

CPLEX 12.7.1.0

For this experiment all CPLEX settings were preserved at default, save setting the 900 second wall-clock time limit.

In Table E.8 we report the wall-clock time using CPLEX for the FERC instances, replacing the time with the terminating optimality gap in parentheses when the solver reaches the 900 second time limit without certifying an optimal solution.

Similar to the experience with the CAISO instances, CPLEX overall performs better on this test set than Gurobi. Examining the solver output suggests that one potential reason for this is CPLEX's dual simplex method was usually successful at finding the optimal LP solution in a reasonable amount of time, at least when compared to Gurobi.

Considering the 2% wind instances, we see that Match, STI, and 3-bin variants solve every instance, with STI exhibiting the best performance overall. Similar to before, the EF, 1-bin*, and 1-bin variants are not competitive. Looking at just Match, STI, and 3-bin, the 3-bin variant exhibits severe performance variability: it solves five of the twelve instances the fastest, but it has the worst-case longest run time of these three – 738 seconds vs. 222 seconds for Match and 150 seconds for STI.

Turning to the 30% wind instances, we note that Match is the only variant able to solve all twelve instances in the time limit required, and is the fastest in geometric mean. Interestingly CPLEX was able to solve the instance 2015-02-01 using STI in a reasonable time. Comparing the terminating optimality gaps, we see that for instance 2015-01-01, Gurobi terminated with a gap of 0.902% for STI, whereas CPLEX terminated with a gap of only 0.078%. This suggests CPLEX

Table E.8: CPLEX Computational Results for FERC Instances: Wall Clock Time. When instances are solved to optimality, reported quantities are seconds to solution. Otherwise, reported quantities in parentheses are the optimality gap after 900 seconds.

(a) 2% Wind Penetration

Instance	EF	Match	STI	3-bin	1-bin*	1-bin
2015-01-01	340.32	127.12	116.39	**103.33**	(0.017%)	(0.016%)
2015-02-01	382.86	**123.08**	144.91	737.72	(0.264%)	(0.261%)
2015-03-01	624.50	127.39	**100.59**	172.34	498.82	578.84
2015-04-01	602.51	**115.71**	144.15	134.50	292.09	297.05
2015-05-01	323.48	83.25	**73.91**	108.68	206.31	153.57
2015-06-01	353.67	119.69	94.44	**79.89**	256.10	339.02
2015-07-01	868.58	221.59	93.79	**87.97**	352.34	516.29
2015-08-01	344.70	131.51	**92.35**	193.95	(0.135%)	(0.130%)
2015-09-01	(0.011%)	143.91	**127.92**	527.99	(0.144%)	(0.148%)
2015-10-01	445.44	164.97	150.02	**143.53**	355.31	393.93
2015-11-02	475.37	175.06	134.81	**129.18**	867.88	(0.017%)
2015-12-01	440.35	138.84	**122.01**	131.57	427.34	523.87
Geometric Mean:	>477.6	135.60	**113.70**	162.42	>498.3	>536.1

(b) 30% Wind Penetration

Instance	EF	Match	STI	3-bin	1-bin*	1-bin
2015-01-01	455.13	**155.13**	(0.078%)	(1.252%)	(4.206%)	(4.163%)
2015-02-01	489.58	**165.69**	310.78	(0.152%)	(1.975%)	(1.740%)
2015-03-01	618.62	**179.55**	214.57	242.20	(0.112%)	(0.114%)
2015-04-01	857.72	247.31	258.42	**210.11**	(0.753%)	(0.774%)
2015-05-01	414.45	128.01	101.18	**82.64**	262.59	240.91
2015-06-01	460.15	166.51	109.42	**100.83**	(0.035%)	(0.040%)
2015-07-01	506.11	182.80	131.57	**121.17**	(0.037%)	(0.041%)
2015-08-01	(0.019%)	196.02	161.15	**140.66**	(0.198%)	(0.162%)
2015-09-01	896.64	178.61	**173.09**	736.56	(0.949%)	(0.607%)
2015-10-01	(0.012%)	**269.18**	277.81	559.67	514.99	738.49
2015-11-02	447.73	**189.60**	248.55	(0.022%)	(0.480%)	(0.260%)
2015-12-01	636.79	202.57	**176.64**	215.01	(0.104%)	(0.096%)
Geometric Mean:	>604.1	**185.02**	>210.8	>296.8	>775.3	>793.2

may be better than Gurobi at tightening the STI formulation either through cuts or presolve, which may explain the difference in performance between the two solvers. For the other variants these instances are largely similar to those preceding: 3-bin exhibits performance variability and is inferior to both Match and STI, and the EF, 1-bin*, and 1-bin variants are uncompetitive.

In Table E.9 we report the number of nodes explored at termination, denoting with a * when the solver terminated because it reached the 900 second wall-clock time limit.

Table E.9: CPLEX Computational Results for FERC Instances: Nodes Explored. Cells report the number of tree nodes explored during branch-and-cut search. Entries with a terminating "*" report the number of tree nodes explored when the 900 second time limit is hit.

(a) 2% Wind Penetration

Instance	EF	Match	STI	3-bin	1-bin*	1-bin
2015-01-01	**0**	**0**	**0**	**0**	5704*	5696*
2015-02-01	**0**	**0**	**0**	5688	3434*	3825*
2015-03-01	11	**0**	**0**	**0**	2274	3868
2015-04-01	**0**	**0**	**0**	**0**	84	88
2015-05-01	**0**	**0**	**0**	**0**	73	**0**
2015-06-01	**0**	**0**	**0**	**0**	**0**	**0**
2015-07-01	**0**	**0**	**0**	**0**	722	1943
2015-08-01	**0**	**0**	**0**	1045	5672*	5694*
2015-09-01	0*	**0**	**0**	5507	5658*	5826*
2015-10-01	**0**	**0**	**0**	**0**	**0**	27
2015-11-02	**0**	**0**	**0**	**0**	3768	2871*
2015-12-01	**0**	**0**	**0**	**0**	165	647
Shifted Geo. Mean:	>1.23	**1.00**	**1.00**	7.52	>355.5	>413.8

(b) 30% Wind Penetration

Instance	EF	Match	STI	3-bin	1-bin*	1-bin
2015-01-01	**0**	**0**	2576*	3790*	1094*	1384*
2015-02-01	**0**	**0**	259	5530*	2145*	2370*
2015-03-01	**0**	**0**	**0**	**0**	3005*	2748*
2015-04-01	**0**	**0**	**0**	**0**	1856*	1762*
2015-05-01	**0**	**0**	**0**	**0**	**0**	**0**
2015-06-01	**0**	**0**	**0**	**0**	2335*	2366*
2015-07-01	**0**	**0**	**0**	**0**	3207*	2901*
2015-08-01	0*	**0**	**0**	**0**	1570*	1284*
2015-09-01	**0**	**0**	**0**	5593	3468*	2497*
2015-10-01	3196*	2849	2723	**144**	4438	5830
2015-11-02	**0**	**0**	**0**	3990*	1288*	1608*
2015-12-01	**0**	**0**	**0**	**0**	2164*	1669*
Shifted Geo. Mean:	>1.95	**1.94**	>5.91	>25.3	>1171	>1153

Taking both wind levels together, observe for the Match variant CPLEX solves all but one instance at the root node, and for the STI variant it solves all but three of the 24 instances at the root note. In a similar fashion, when the EF variant solves it is often at the root node. The 3-bin variant also solves most of the instances at the root node as well. For the 1-bin variants, CPLEX often explores more nodes than Gurobi, but only explores a few thousand before the wall-clock time limit is reached.

Table E.10: Computational Results for FERC Instances: Relative Integrality Gap (%)

(a) 2% Wind Penetration

Instance	EF	Match	STI	3-bin	1-bin*	1-bin
2015-01-01	0.0284	0.0284	0.0362	0.0362	0.5500	0.5500
2015-02-01	0.0423	0.0423	0.0717	0.0717	0.9187	0.9187
2015-03-01	0.0327	0.0327	0.0334	0.0334	0.3727	0.3727
2015-04-01	0.0540	0.0540	0.0540	0.0540	0.5788	0.5788
2015-05-01	0.0456	0.0456	0.0456	0.0456	0.4131	0.4131
2015-06-01	0.0375	0.0375	0.0375	0.0375	1.0321	1.0321
2015-07-01	0.0796	0.0796	0.0796	0.0796	1.3827	1.3827
2015-08-01	0.1233	0.1233	0.1422	0.1422	1.5661	1.5661
2015-09-01	0.5283	0.5283	0.5542	0.5542	1.9062	1.9063
2015-10-01	0.1140	0.1140	0.1141	0.1141	1.0522	1.0522
2015-11-02	0.0760	0.0760	0.0797	0.0797	1.4377	1.4377
2015-12-01	0.0629	0.0629	0.0654	0.0654	1.1305	1.1305
Geometric Mean:	0.0683	0.0683	0.0746	0.0746	0.9113	0.9113

(b) 30% Wind Penetration

Instance	EF	Match	STI	3-bin	1-bin*	1-bin
2015-01-01	0.0924	0.0924	1.7525	1.7525	7.4916	7.4916
2015-02-01	0.1703	0.1703	0.5119	0.5119	3.7359	3.7359
2015-03-01	0.0995	0.0995	0.1140	0.1140	1.9207	1.9207
2015-04-01	0.8124	0.8124	0.8476	0.8476	6.8377	6.8377
2015-05-01	0.0729	0.0729	0.0729	0.0729	1.5792	1.5792
2015-06-01	0.0807	0.0807	0.0859	0.0859	2.5388	2.5388
2015-07-01	0.1383	0.1383	0.1412	0.1412	2.5278	2.5278
2015-08-01	0.3701	0.3701	0.3904	0.3904	4.1100	4.1100
2015-09-01	0.2884	0.2884	0.3747	0.3747	3.2275	3.2275
2015-10-01	1.1342	1.1342	1.1446	1.1446	2.9962	2.9962
2015-11-02	0.1825	0.1825	0.2626	0.2626	2.8615	2.8615
2015-12-01	0.2558	0.2558	0.2738	0.2738	1.2704	1.2704
Geometric Mean:	0.2060	0.2060	0.3141	0.3141	3.0031	3.0031

Relative Integrality Gap

In Table E.10 we report the relative integrality gap for the FERC instances, calculated in the exact same fashion as the CAISO relative integrality gap results reported in Table E.5.

First, we observe the same pattern as we did for CAISO: the integrality gaps for EF and Match are always the same, as are those for STI and 3-bin. Otherwise, the results here are significantly different than those for CAISO. We note 1-bin* is no tighter than 1-bin for the FERC instances. Turning to the 2% wind instances, we see that EF and Match often are not tighter than STI and

3-bin, or are only marginally so. This explains STI's performance dominance on the 2% instances in CPLEX – the extra variables from Match are not, in these instances, buying much (or any) additional tightness over STI. Match and EF only close 8% of the optimally gap in geometric mean over STI, which is significantly less than the 75% geometric mean gap closure observed for CAISO.

Considering now the 30% wind instances, we see in particular that Match closes a large portion of the optimality gap over STI in the 2015-01-01 and 2015-02-01 instances, with modest reductions in every instance except 2015-05-01. We also observe that in general, the high-wind instances, both here and in Table E.5, have larger integrality gaps than low-wind instances across all formulations. This should be expected as large amounts of renewables generation imply large net-load swings, which should result in more generator switching and generator ramping.

B Statistical Analysis

In this section we report the results of a statical analysis of the computational results above, using the Wilcoxon signed-rank test [158]. To separate out the potential contributions to performance variability, we considered five sets of instances for each solver: (i) "All" ($n = 44$) – which consists of the entire test suite, (ii) "CAISO" ($n = 20$) – which is the CAISO set of instances, (iii) "FERC" ($n = 24$) – which is the FERC set of instances, (iv) "High Wind" ($n = 16$) – which consists of the Scenario400 instances from CAISO and the 30% Wind Penetration instances from FERC, and (v) "Low Wind" ($n = 28$) – which is all the other instances not in High Wind. We note that for $n \lessapprox 20$ this statistical test starts to become underpowered.

B.1 Gurobi 7.0.1

In Table E.11 we report the mean differences in solve times and the results of the Wilcoxon signed-rank test across the five sets described above on the Gurobi computational experiments. In each cell we report the column mean solve time minus the row mean solve time; hence a negative number implies the column was faster than the row, whereas a positive number implies the row was faster than the column. Because the Wilcoxon test is for difference in arithmetic mean, the results in these tables report the difference in arithmetic mean solve time, whereas the summary results in Section A report the geometric mean solve time. Looking at the entire test set we can see that the Match formulation outperforms the others at the $\alpha = 0.01$ using Gurobi. Match also outperforms STI in the breakdowns at the $\alpha = 0.05$ level, except for the CAISO test set. STI in turn outperforms

Table E.11: Results of the Wilcoxon signed-rank test for Gurobi computational experiments. Each cell reports the column mean solve time minus the row mean solve time. A "*" indicates the difference is significant at the $\alpha = 0.05$ level; a "**" indicates the difference is significant at the $\alpha = 0.01$ level.

(a) All ($n = 44$)

Formulation	EF	Match	STI	3bin	1bin*	1bin
EF		-466.0**	-383.3**	-327.1**	96.5**	101.3**
Match	466.0**		82.7**	138.9**	562.5**	567.3**
STI	383.3**	-82.7**		56.2**	479.8**	484.6**
3bin	327.1**	-138.9**	-56.2**		423.6**	428.4**
1bin*	-96.5**	-562.5**	-479.8**	-423.6**		4.8
1bin	-101.3**	-567.3**	-484.6**	-428.4**	-4.8	

(b) CAISO ($n = 20$)

Formulation	EF	Match	STI	3bin	1bin*	1bin
EF		-360.3**	-348.1**	-284.0**	188.0**	188.5**
Match	360.3**		12.2	76.3**	548.3**	548.8**
STI	348.1**	-12.2		64.1**	536.0**	536.5**
3bin	284.0**	-76.3**	-64.1**		472.0**	472.5**
1bin*	-188.0**	-548.3**	-536.0**	-472.0**		0.5
1bin	-188.5**	-548.8**	-536.5**	-472.5**	-0.5	

(c) FERC ($n = 24$)

Formulation	EF	Match	STI	3bin	1bin*	1bin
EF		-554.1**	-412.7**	-363.0**	20.3	28.6
Match	554.1**		141.5*	191.1*	574.4**	582.8**
STI	412.7**	-141.5*		49.7	432.9**	441.3**
3bin	363.0**	-191.1*	-49.7		383.3**	391.7**
1bin*	-20.3	-574.4**	-432.9**	-383.3**		8.4
1bin	-28.6	-582.8**	-441.3**	-391.7**	-8.4	

Table E.11: (continued)

(d) High Wind ($n = 16$)

Formulation	EF	Match	STI	3bin	1bin*	1bin
EF		-534.2**	-362.9**	-285.8**	26.0	27.4
Match	534.2**		171.3*	248.5*	560.2**	561.6**
STI	362.9**	-171.3*		77.1	388.9**	390.3**
3bin	285.8**	-248.5*	-77.1		311.7**	313.1**
1bin*	-26.0	-560.2**	-388.9**	-311.7**		1.4
1bin	-27.4	-561.6**	-390.3**	-313.1**	-1.4	

(e) Low Wind ($n = 28$)

Formulation	EF	Match	STI	3bin	1bin*	1bin
EF		-427.1**	-395.0**	-350.7**	136.8**	143.5**
Match	427.1**		32.1*	76.3**	563.8**	570.6**
STI	395.0**	-32.1*		44.3**	531.8**	538.5**
3bin	350.7**	-76.3**	-44.3**		487.5**	494.2**
1bin*	-136.8**	-563.8**	-531.8**	-487.5**		6.7
1bin	-143.5**	-570.6**	-538.5**	-494.2**	-6.7	

3-bin overall and in several of the breakout sets. These statistics also bear out the larger observation that the EF, 1-bin, and 1-bin* variants are uncompetitive with any of Match, STI, and 3-bin.

B.2 CPLEX 12.7.1.0

In Table E.12 we report the mean differences in solve time and the results of the Wilcoxon signed-rank test for the CPLEX computational experiments. As with Table E.11, In each cell we report the column mean solve time minus the row mean solve time; so a negative number implies the column was faster than the row, and a positive number implies the row was faster than the column. While Match still has the best mean overall, the Wilcoxon test is not able to differentiate it from STI and 3-bin. Interestingly STI is better than 3-bin at the $\alpha = 0.01$ level. We also note that on the Low Wind instances STI is able to out-perform Match using CPLEX at the $\alpha = 0.01$, which bears out the observations from the computational results above. The magnitude of the difference is not large, however. Turning to the High Wind instances, we see Match is able to out-perform the other formulations save STI at the $\alpha = 0.05$ level; it is likely the low power of the test at $n = 16$ makes it difficult to distinguish Match and STI statistically. Finally we observe that overall the EF, 1-bin, and 1-bin* variants are significantly worse than the Match, STI, and 3-bin variants.

Table E.12: Results of the Wilcoxon signed-rank test for CPLEX computational experiments. Each cell reports the column mean solve time minus the row mean solve time. A "*" indicates the difference is significant at the $\alpha = 0.05$ level; a "**" indicates the difference is significant at the $\alpha = 0.01$ level.

(a) All ($n = 44$)

Formulation	EF	Match	STI	3bin	1bin*	1bin
EF		-328.3**	-317.1**	-218.3**	212.8**	227.1**
Match	328.3**		11.3	110.0	541.2**	555.4**
STI	317.1**	-11.3		98.8**	529.9**	544.1**
3bin	218.3**	-110.0	-98.8**		431.1**	445.4**
1bin*	-212.8**	-541.2**	-529.9**	-431.1**		14.2
1bin	-227.1**	-555.4**	-544.1**	-445.4**	-14.2	

(b) CAISO ($n = 20$)

Formulation	EF	Match	STI	3bin	1bin*	1bin
EF		-234.5**	-236.2**	-180.1**	314.7**	314.7**
Match	234.5**		-1.7	54.4	549.2**	549.2**
STI	236.2**	1.7		56.0**	550.8**	550.8**
3bin	180.1**	-54.4	-56.0**		494.8**	494.8**
1bin*	-314.7**	-549.2**	-550.8**	-494.8**		0.0
1bin	-314.7**	-549.2**	-550.8**	-494.8**	0.0	

(c) FERC ($n = 24$)

Formulation	EF	Match	STI	3bin	1bin*	1bin
EF		-406.5**	-384.5**	-250.1*	128.0	154.1*
Match	406.5**		22.0	156.4	534.5**	560.6**
STI	384.5**	-22.0		134.4	512.5**	538.6**
3bin	250.1*	-156.4	-134.4		378.1**	404.2**
1bin*	-128.0	-534.5**	-512.5**	-378.1**		26.1
1bin	-154.1*	-560.6**	-538.6**	-404.2**	-26.1	

Table E.12: (continued)

(d) High Wind ($n = 16$)

Formulation	EF	Match	STI	3bin	1bin*	1bin
EF		-422.2**	-361.8**	-175.4	186.1*	195.3*
Match	422.2**		60.4	246.8*	608.3**	617.5**
STI	361.8**	-60.4		186.4*	547.9**	557.1**
3bin	175.4	-246.8*	-186.4*		361.5**	370.7**
1bin*	-186.1*	-608.3**	-547.9**	-361.5**		9.2
1bin	-195.3*	-617.5**	-557.1**	-370.7**	-9.2	

(e) Low Wind ($n = 28$)

Formulation	EF	Match	STI	3bin	1bin*	1bin
EF		-274.7**	-291.5**	-242.8**	228.1**	245.2**
Match	274.7**		-16.8**	31.9	502.8**	519.9**
STI	291.5**	16.8**		48.7**	519.6**	536.7**
3bin	242.8**	-31.9	-48.7**		470.9**	488.0**
1bin*	-228.1**	-502.8**	-519.6**	-470.9**		17.1*
1bin	-245.2**	-519.9**	-536.7**	-488.0**	-17.1*	

C Summary

Considering Tables E.1 and E.6 together, it is unambiguous that Match performs better than the other variants on Gurobi, followed by STI and then 3-bin. This is born out in the statical analysis of these results in Table E.11. Given that Match is as tight as EF in all instances while needing many fewer variables, but not too many additional variables over STI, this result is not surprising.

Conversely, the computational results using CPLEX reported in Tables E.3 and E.8 are a bit more ambiguous, and this is reflected in Table E.12. While using CPLEX Match is often slower in the average case than STI, using the Match formulation CPLEX solved every of the 44 instances considered in under 5 minutes, and hence it exhibited the better worst-case performance.

Appendix F

Aggregation/Disaggregation Details

In this appendix we detail how reserves, piecewise linear operating costs, and time-dependent startup costs can be exactly aggregated. Additionally we specify how to disaggregate solutions for fast-ramping generators with different startup and shutdown ramp rates. For ease of presentation this appendix (mostly) is self-contained.

A Nomenclature

A.1 Indices and Sets

$g \in \mathcal{G}$	Thermal generators
$l \in \mathcal{L}^g$	Piecewise production cost intervals for generator g: $1, \ldots, \mathbf{L}_g$.
$s \in \mathcal{S}^g$	Startup categories for generator g, from hottest (1) to coldest ($\mathbf{S}_g$).
$t \in \mathcal{T}$	Hourly time steps: $1, \ldots, \mathbf{T}$.
$[t, t') \in \mathcal{X}^g$	Feasible intervals of non-operation for generator g with respect to its minimum downtime, that is, $[t, t') \in \mathcal{T} \times \mathcal{T}$ such that $t' \geq t + \mathbf{DT}^g$, including times (as necessary) before and after the planning period $\mathcal{T}$.
$[t, t') \in \mathcal{Y}^g$	Feasible intervals of operation for generator g with respect to its minimum uptime, that is, $[t, t') \in \mathcal{T} \times \mathcal{T}$ such that $t' \geq t + \mathbf{UT}^g$, including times (as necessary) before and after the planning period $\mathcal{T}$.

A.2 Parameters

$\mathbf{c}^{l,g}$	Cost coefficient for piecewise segment l for generator g (\$/MWh).

$\mathbf{c}^{R,g}$ Cost of generator g running and operating at minimum production $\underline{\mathbf{P}}_g$ (\$/h).

$\mathbf{c}^{s,g}$ Startup cost of category s for generator g (\$).

$\mathbf{D}_t$ Load (demand) at time t (MW).

$\mathbf{DT}^g$ Minimum down time for generator g (h).

$\overline{\mathbf{P}}^g$ Maximum power output for generator g (MW).

$\overline{\mathbf{P}}^{l,g}$ Maximum power available for piecewise segment l for generator g (MW) ($\overline{\mathbf{P}}^{0,g} = \underline{\mathbf{P}}^g$).

$\underline{\mathbf{P}}^g$ Minimum power output for generator g (MW).

$\mathbf{R}_t$ Spinning reserve at time t (MW).

$\mathbf{RD}^g$ Ramp-down rate for generator g (MW/h).

$\mathbf{RU}^g$ Ramp-up rate for generator g (MW/h).

$\mathbf{SD}^g$ Shutdown rate for generator g (MW/h).

$\mathbf{SU}^g$ Startup rate for generator g (MW/h).

$\mathbf{TC}^g$ Time down after which generator g goes cold, i.e., enters state S^g.

$\underline{\mathbf{T}}^{s,g}$ Time offline after which the startup category s is available ($\underline{\mathbf{T}}^{1,g} = \mathbf{DT}^g$, $\underline{\mathbf{T}}^{S^g,g} = \mathbf{TC}^g$)

$\overline{\mathbf{T}}^{s,g}$ Time offline after which the startup category s is no longer available ($= \underline{\mathbf{T}}^{s+1,g}$, $\overline{\mathbf{T}}^{S_g,g} = +\infty$)

$\mathbf{UT}^g$ Minimum run time for generator g (h).

A.3 Variables

p_t^g Power output for generator g at time t (MW).

$p_t^{l,g}$ Power from piecewise interval l for generator g at time t (MW).

r_t^g Spinning reserves provided by generator g at time t (MW), ≥ 0.

u_t^g Commitment status of generator g at time t, $\in \{0,1\}$.

v_t^g Startup status of generator g at time t, $\in \{0,1\}$.

w_t^g Shutdown status of generator g at time t, $\in \{0,1\}$.

$c_t^{SU,g}$ Startup cost for generator g at time t (\$), ≥ 0.

$x_{[t,t')}^g$ Indicator arc for shutdown at time t, startup at time t', uncommitted for $i \in [t,t')$, for generator g, $\in \{0,1\}$, $[t,t') \in \mathcal{X}^g$.

$y_{[t,t')}^g$ Indicator arc for startup at time t, shutdown at time t', committed for $i \in [t,t')$, for generator g, $\in \{0,1\}$, $[t,t') \in \mathcal{Y}^g$.

B Unit Commitment Formulation

This section lays out the basic unit commitment formulation we consider for the computational tests above.

$$\min \sum_{g \in G} \mathbf{c}^g(p^g) \tag{F.1a}$$

subject to:

$$\sum_{g \in \mathcal{G}} p_t^g = \mathbf{D}_t \qquad \forall t \in \mathcal{T} \tag{F.1b}$$

$$\sum_{g \in \mathcal{G}} r_t^g \geq \mathbf{R}_t \qquad \forall t \in \mathcal{T} \tag{F.1c}$$

$$(p^g, r^g) \in \Pi^g \qquad \forall g \in \mathcal{G}, \tag{F.1d}$$

where $\mathbf{c}^g(p^g)$ is the cost function of generator g producing an output of p^g over the time horizon $\mathcal{T}$, and Π^g represents the set of feasible schedules for generator g. Here $\mathbf{c}^g$ includes possibly piecewise linear convex production costs, as well as time-dependent startup costs. We will see how we can modify the formulations in the main body to allow for these additional modeling features while still maintaining the property that we can disaggregate solutions to aggregated generators.

For completeness we restate the theorems from the main text.

Theorem 5.1. *Consider identical generators $g_1, g_2 \in \mathcal{G}$, and assume their production costs are increasing and convex. Then there exists an optimal solution with $p_t^{g_1} = p_t^{g_2}$ or (inclusive) one of the nominal or startup/shutdown ramping constraints is binding for generator g_1 or g_2 for all times t for which they are both on.*

Theorem 5.2. *Suppose generator g_1 is turned off at time t. If identical generator g_2 can also be turned off at time t, there exists an optimal solution where the generator that has been on for the least amount of time is turned off.*

Theorem 5.3. *Suppose generator g_1 is turned on at time t. If an identical generator g_2 can also be turned on at time t, and there are no time-dependent startup costs for g_1 and g_2, then there exists an optimal solution where g_2 is turned on at t.*

While Theorem 5.3 is useful for when start-up costs are not time-dependent, this is often not a realistic assumption. Therefore, we have the following two theorems which will be of use when start-up costs are time-dependent.

Theorem F.1. *Suppose generator g_1 is turned on at time t, and has been off for at least* $\mathbf{TC}(= \mathbf{TC}^{g_1} = \mathbf{TC}^{g_2})$ *time periods. If identical generator g_2 can also be turned on at time t and has been off for at least* $\mathbf{TC}$ *time periods, there exists an optimal solution where where g_2 is turned on at time t.*

Proof. Similar to the proof of Theorem 5.3. □

Theorem F.2. *If identical generators $g_1, g_2 \in \mathcal{G}$ both shut down at time t and g_1 starts up at time $t_1 \geq t + \mathbf{DT}$ $(\mathbf{DT} = \mathbf{DT}^{g_1} = \mathbf{DT}^{g_2})$ and g_2 starts up at time $t_2 \geq t + \mathbf{DT}$, then an equally good solution exists where g_1 starts up at time t_2 and g_2 starts up at time t_1.*

Proof. Like in the proof of Theorem 5.3, we may permute the remainder of each generator's schedule without affecting feasibility or the objective value. □

C Disaggregating the Extended Formulation

First, we consider the extended formulation, which is more straightforward than disaggregating the 3-bin formulation. We can add reserves and piecewise linear production costs by adding new variables $r_t^{[a,b),g}$ $\forall [a,b) \in \mathcal{Y}^g,\ \forall t \in \mathcal{T}$ and $p^{[a,b),l,g}$ $\forall [a,b) \in \mathcal{Y}^g,\ \forall l \in \mathcal{L}^g,\ \forall t \in \mathcal{T}$. We add time-dependent startup costs by replacing the packing polytope we used above with the shortest path polytope [131]. The resulting formulation is

$$\mathbf{A}^{[a,b)} p^{[a,b)} + \mathbf{A}'^{[a,b)} r^{[a,b)} + \sum_{l \in \mathcal{L}} \mathbf{A}^{[a,b),l} p^{[a,b),l} \leq \mathbf{b}^{[a,b)} y_{[a,b)} \qquad \forall [a,b) \in \mathcal{Y} \qquad \text{(F.2a)}$$

$$\sum_{[a,b) \in \mathcal{Y}} p_t^{[a,b)} = p_t \qquad \forall t \in \mathcal{T} \qquad \text{(F.2b)}$$

$$\sum_{[a,b) \in \mathcal{Y}} r_t^{[a,b)} = r_t \qquad \forall t \in \mathcal{T} \qquad \text{(F.2c)}$$

$$\sum_{\{[c,d) \in \mathcal{X} \mid t=d\}} x_{[c,d)} = \sum_{\{[a,b) \in \mathcal{Y} \mid t=a\}} y_{[a,b)} \qquad \forall t \in \mathcal{T} \qquad \text{(F.2d)}$$

$$\sum_{\{[a,b) \in \mathcal{Y} \mid t=b\}} y_{[a,b)} = \sum_{\{[c,d) \in \mathcal{X} \mid t=c\}} x_{[c,d)} \qquad \forall t \in \mathcal{T} \qquad \text{(F.2e)}$$

$$\sum_{\{[a,b)\in\mathcal{Y} \mid a\leq 0\}} y_{[a,b)} + \sum_{\{[c,d)\in\mathcal{X} \mid c\leq 0\}} x_{[c,d)} = 1 \tag{F.2f}$$

$$\sum_{\{[a,b)\in\mathcal{Y} \mid b>\mathbf{T}\}} y_{[a,b)} + \sum_{\{[c,d)\in\mathcal{X} \mid d>\mathbf{T}\}} x_{[c,d)} = 1, \tag{F.2g}$$

where the polytope

$$\{p^{[a,b)}, r^{[a,b)}, p^{[a,b),1}, \ldots, p^{[a,b),\mathbf{L}} \in \mathbb{R}_+ \mid \mathbf{A}^{[a,b)}p^{[a,b)} + \mathbf{A}'^{[a,b)}r^{[a,b)} + \sum_{l\in\mathcal{L}} \mathbf{A}^{[a,b),l}p^{[a,b),l} \leq \mathbf{b}^{[a,b)}\} \tag{F.3}$$

represents feasible production given that the generator is turned on at time a and turned off at time b. Piecewise production costs can then be handled by placing the appropriate objective coefficient on the $p^{[a,b),l}$ variables and time-dependent startup costs are accounted for by placing the appropriate objective coefficient on the $x_{[c,d)}$ variables.

Similar to the EF presented in the main text, we see that the underlying shortest path polytope (F.2e, F.2e, F.2f, F.2g) with nonnegativity, has a totally unimodular constraint matrix, and thus has the integer decomposition property [9]. Hence if we have k generators have identical parameters, we can replace (F.2f) and (F.2g) with

$$\sum_{\{[a,b)\in\mathcal{Y} \mid a\leq 0\}} Y_{[a,b)} + \sum_{\{[c,d)\in\mathcal{X} \mid c\leq 0\}} X_{[c,d)} = k \tag{F.4a}$$

$$\sum_{\{[a,b)\in\mathcal{Y} \mid b>\mathbf{T}\}} Y_{[a,b)} + \sum_{\{[c,d)\in\mathcal{X} \mid d>\mathbf{T}\}} X_{[c,d)} = k, \tag{F.4b}$$

thus pushing k units of flow through the graph. Changing these right-hand-sides doesn't affect the integrality of the full polytope (F.2a – F.2e), (F.4) (see Chapter 3).

As before, allowing capital variables to represent aggregated variables for identical generators, now $Y_{[a,b)}$ represents *how many* of the generators are on during the interval $[a, b)$ and $X_{[a,b)}$ represents how many generators are off during the interval $[a, b)$. Since there are separate power variables for each on interval $[a, b)$, like before with the EF in the main text, Theorem 5.1 enables us to disaggregate power easily once the status variables are disaggregated.

Letting $\mathcal{K} \subset \mathcal{G}$ be some set of identical generators, consider the extended formulation for these aggregated generators

$$\mathbf{A}^{[a,b)}P^{[a,b)} + \mathbf{A}'^{[a,b)}R^{[a,b)} + \sum_{l\in\mathcal{L}} \mathbf{A}^{[a,b),l}P^{[a,b),l} \leq \mathbf{b}^{[a,b)}Y_{[a,b)} \qquad \forall [a,b) \in \mathcal{Y} \tag{F.5a}$$

$$\sum_{[a,b)\in\mathcal{Y}} P_t^{[a,b)} = P_t \qquad \forall t \in \mathcal{T} \tag{F.5b}$$

$$\sum_{[a,b)\in\mathcal{Y}} R_t^{[a,b)} = R_t \qquad \forall t \in \mathcal{T} \tag{F.5c}$$

$$\sum_{\{[c,d)\in\mathcal{X} \mid t=d\}} X_{[c,d)} = \sum_{\{[a,b)\in\mathcal{Y} \mid t=a\}} Y_{[a,b)} \qquad \forall t \in \mathcal{T} \tag{F.5d}$$

$$\sum_{\{[a,b)\in\mathcal{Y} \mid t=b\}} Y_{[a,b)} = \sum_{\{[c,d)\in\mathcal{X} \mid t=c\}} X_{[c,d)} \qquad \forall t \in \mathcal{T} \tag{F.5e}$$

$$\sum_{\{[a,b)\in\mathcal{Y} \mid a\leq 0\}} Y_{[a,b)} + \sum_{\{[c,d)\in\mathcal{X} \mid c\leq 0\}} X_{[c,d)} = |\mathcal{K}| \tag{F.5f}$$

$$\sum_{\{[a,b)\in\mathcal{Y} \mid b>\mathbf{T}\}} Y_{[a,b)} + \sum_{\{[c,d)\in\mathcal{X} \mid d>\mathbf{T}\}} X_{[c,d)} = |\mathcal{K}|. \tag{F.5g}$$

We can then write down an easy algorithm to decompose solutions to (F.5).

Algorithm F.1 (Peel Off EF) Constructs feasible generator schedules from a solution of (F.5).

Initialize all $\hat{P}^{[a,b)}$ to $P^{*[a,b)}$ and all $p^{g,[a,b)}$ to 0.
Find a feasible s,t path based on (F.5d – F.5g) and store in $x^g,\ y^g$.
$\hat{X} \leftarrow X^* - x^g$, $\hat{Y} \leftarrow Y^* - y^g$
for $[a,b) \in \mathcal{Y}$ with $y^g_{[a,b)} = 1$ **do**
  for $t \in [a,b) \cap \mathcal{T}$ **do**
    $p_t^{g,[a,b)} \leftarrow P_t^{*[a,b)}/Y^*_{[a,b)}$; $\hat{P}_t^{[a,b)} \leftarrow P_t^{*[a,b)} - p_t^{g,[a,b)}$
    $r_t^{g,[a,b)} \leftarrow R_t^{*[a,b)}/Y^*_{[a,b)}$; $\hat{R}_t^{[a,b)} \leftarrow R_t^{*[a,b)} - r_t^{g,[a,b)}$
    for $l \in \mathcal{L}$ **do**
      $p_t^{g,[a,b),l} \leftarrow P_t^{*[a,b),l}/Y^*_{[a,b)}$; $\hat{P}_t^{[a,b),l} \leftarrow P_t^{*[a,b),l} - p_t^{g,[a,b),l}$
for $t \in \mathcal{T}$ **do**
  $\hat{P}_t \leftarrow \sum_{[a,b)\in\mathcal{Y}} \hat{P}_t^{[a,b)}$
  $\hat{R}_t \leftarrow \sum_{[a,b)\in\mathcal{Y}} \hat{R}_t^{[a,b)}$
  $p_t^g \leftarrow \sum_{[a,b)\in\mathcal{Y}} p_t^{g,[a,b)}$
  $r_t^g \leftarrow \sum_{[a,b)\in\mathcal{Y}} r_t^{g,[a,b)}$

After running Algorithm F.1 $|\mathcal{K}|-1$ times we are left with $|\mathcal{K}|$ feasible (and by Theorem 5.1 optimal) schedules, one for each generator in $\mathcal{K}$. We formalize this in Theorem F.3.

First we need a simple lemma regarding the decomposability of polytopes.

Lemma F.1. *Let P a polytope such that $P := \{x \in \mathbb{R}^n_+ \mid Ax \leq b\}$ and for $k \geq 1$ define $kP := \{x \mid \frac{1}{k}x \in P\} = \{x \in \mathbb{R}^n_+ \mid Ax \leq kb\}$. If $y \in kP$, then $\frac{1}{k}y \in P$ and $\frac{(k-1)}{k}y \in (k-1)P$.*

Proof. It suffices to notice that $\frac{1}{k}y$ is feasible for the system $\{x \in \mathbb{R}^n_+ \mid Ay - (k-1)b \leq Ax \leq b\}$. □

Now we turn to the main result.

Theorem F.3. *Algorithm F.1 returns a feasible solution for (F.2) and a feasible solution for (F.5) for the remaining $\mathcal{K} \setminus \{g\}$ generators. That is, after applying Algorithm F.1 $|\mathcal{K}| - 1$ times we have a feasible and optimal solution for every $g \in \mathcal{K}$.*

Proof. Notice the feasible s, t path leaves the equalities (F.2d – F.2g) and (F.5d – F.5g) feasible for x^g, y^g and $\hat{X}$, $\hat{Y}$ respectively. Further, the solutions constructed for the $p^{g,[a,b)}$, $r^{g,[a,b)}$, and $p^{g,[a,b),l}$ variables (lines 6 – 9) are exactly of the type prescribed by Lemma F.1, and so both these and the $\hat{P}^{[a,b)}$, $\hat{R}^{[a,b)}$, and $\hat{P}^{[a,b),l}$ variables are feasible for (F.2a) and (F.5a) respectively. Theorem 5.1 ensures this assignment is optimal as well. The equalities (F.2b, F.2c) and (F.5b, F.5c) follow from lines 11 – 14.

The last statement follows from inducting on the size of $\mathcal{K}$. □

D Disaggregating the 3-bin polytope

Recalling the traditional 3-bin formulation for fast ramping generators (when $\mathbf{UT}^g \geq 2$) [110, 58]:

$$\underline{\mathbf{P}}^g u_t^g \leq p_t^g, \qquad \forall t \in \mathcal{T}, \tag{F.6a}$$

$$p_t^g + r_t^g \leq \overline{\mathbf{P}}^g u_t^g + (\mathbf{SU}^g - \overline{\mathbf{P}}^g) v_t^g + (\mathbf{SD}^g - \overline{\mathbf{P}}^g) w_{t+1}^g, \qquad \forall t \in \mathcal{T}, \tag{F.6b}$$

$$u_t^g - u_{t-1}^g = v_t^g - w_t^g, \qquad \forall t \in \mathcal{T}, \tag{F.6c}$$

$$\sum_{i=t-\mathbf{UT}+1}^{t} v_i^g \leq u_t^g, \qquad \forall t \in [\mathbf{UT}^g, \mathbf{T}], \tag{F.6d}$$

$$\sum_{i=t-\mathbf{DT}+1}^{t} w_i^g \leq 1 - u_t^g, \qquad \forall t \in [\mathbf{DT}^g, \mathbf{T}], \tag{F.6e}$$

$$p_t^g,\ r_t^g \in \mathbb{R}_+, \qquad \forall t \in \mathcal{T}, \tag{F.6f}$$

$$u_t^g,\ v_t^g,\ w_t^g \in \{0, 1\}, \qquad \forall t \in \mathcal{T}. \tag{F.6g}$$

It has the property that the constraint matrix defined by (F.6c, F.6d, F.6e) is totally unimodular [94], and so it too has the integer decomposition property [9]. When $\mathbf{UT}^g = 1$, (F.6b) is replaced with the following [110, 58]:

$$p_t^g + r_t^g \leq \overline{\mathbf{P}}^g u_t^g + (\mathbf{SU}^g - \overline{\mathbf{P}}^g) v_t^g, \qquad \forall t \in \mathcal{T}, \tag{F.7a}$$

$$p_t^g + r_t^g \leq \overline{\mathbf{P}}^g u_t^g + (\mathbf{SD}^g - \overline{\mathbf{P}}^g) w_{t+1}^g, \qquad \forall t \in \mathcal{T}. \tag{F.7b}$$

D.1 Generator Production Cost Function

The convex production costs are typically approximated by piecewise linear costs. This is done by partitioning the interval $[\underline{\mathbf{P}}, \overline{\mathbf{P}}]$ into L subintervals with breakpoints $\overline{\mathbf{P}}^l$, with $\overline{\mathbf{P}}^0 = \underline{\mathbf{P}}$, $\overline{\mathbf{P}}^L = \overline{\mathbf{P}}$, and $\overline{\mathbf{P}}^l < \overline{\mathbf{P}}^{l+1}$. Let $\mathbf{c}^l$ be the marginal cost for the segment $[\overline{\mathbf{P}}^{l-1}, \overline{\mathbf{P}}^l]$. The variable c_t, then, represents the production cost for time t given the following constraints:

$$c_t = \sum_{l=1}^{k} \mathbf{c}^l p_t^l \qquad \forall t \in \mathcal{T} \tag{F.8a}$$

$$p_t = \underline{\mathbf{P}} u_t + \sum_{l=1}^{k} p_t^l \qquad \forall t \in \mathcal{T} \tag{F.8b}$$

$$0 \leq p_t^l \leq (\overline{\mathbf{P}}^l - \overline{\mathbf{P}}^{l-1}) u_t \qquad \forall l \in \mathcal{L},\ \forall t \in \mathcal{T} \tag{F.8c}$$

The cost functions of the generators need to be modified to account for the aggregation. Fortunately, using the intuition behind Theorem 5.1, the lack of ramping constraints ensure if two identical generators are on in a given time period, they must have the same production, meaning that allowing u to be a general integer is also sufficient. The only exception to this rule is when there is a startup/shutdown rate. Without loss of generality assume $\mathbf{SU} = \mathbf{SD} = \overline{\mathbf{P}}^{l'}$. We substitute (F.8c) by

$$p_t^l \leq (\overline{\mathbf{P}}^l - \overline{\mathbf{P}}^{l-1}) u_t \qquad \forall l \in [l'],\ \forall t \in \mathcal{T} \tag{F.9a}$$

$$p_t^l \leq (\overline{\mathbf{P}}^l - \overline{\mathbf{P}}^{l-1})(u_t - v_t - w_{t+1}) \qquad \forall l > l',\ \forall t \in \mathcal{T} \tag{F.9b}$$

$$p_t^l \geq 0 \qquad \forall l \in \mathcal{L},\ \forall t \in \mathcal{T} \tag{F.9c}$$

when $\mathbf{UT} \geq 2$ and

$$p_t^l \leq (\overline{\mathbf{P}}^l - \overline{\mathbf{P}}^{l-1}) u_t \qquad \forall l \in [l'],\ \forall t \in \mathcal{T} \tag{F.10a}$$

$$p_t^l \leq (\overline{\mathbf{P}}^l - \overline{\mathbf{P}}^{l-1})(u_t - v_t) \qquad \forall l > l',\ \forall t \in \mathcal{T} \tag{F.10b}$$

$$p_t^l \leq (\overline{\mathbf{P}}^l - \overline{\mathbf{P}}^{l-1})(u_t - w_{t+1}) \qquad \forall l > l',\ \forall t \in \mathcal{T} \tag{F.10c}$$

$$p_t^l \geq 0 \qquad \forall l \in \mathcal{L},\ \forall t \in \mathcal{T} \tag{F.10d}$$

when $\mathbf{UT} = 1$. When the generator has just turned on (about to turn off), then its power output cannot be above **SU** (**SD**). Hence constraints of the form (F.9b) and (F.10b, F.10c) cut off these solutions in the p^l variables just as (F.6b) and (F.7a, F.7b) do for the p variables.

D.2 Generator Startup Costs

There are many different proposed formulations for time-dependent startup costs. One of which, [131], provides a perfect (and totally unimodular) formulation. It is:

$$\sum_{\{t'|[t,t')\in\mathcal{Y}\}} y_{[t,t')} = v_t \qquad \forall t \in \mathcal{T} \tag{F.11a}$$

$$\sum_{\{t'|[t',t)\in\mathcal{Y}\}} y_{[t',t)} = w_t \qquad \forall t \in \mathcal{T} \tag{F.11b}$$

$$\sum_{\{t'|[t',t)\in\mathcal{X}\}} x_{[t',t)} = v_t \qquad \forall t \in \mathcal{T} \tag{F.11c}$$

$$\sum_{\{t'|[t,t')\in\mathcal{X}\}} x_{[t,t')} = w_t \qquad \forall t \in \mathcal{T} \tag{F.11d}$$

$$\sum_{\{[\tau,\tau')\in\mathcal{Y}|t\in[\tau,\tau')\}} y_{[\tau,\tau')} = u_t \qquad \forall t \in \mathcal{T}. \tag{F.11e}$$

Like the extended formulation in the generator model, this formulation can be quite large, containing $O(T^2)$ many variables. However, after adding these constraints to 3-bin model for fast-ramping generators, the resulting formulation still satisfies the integer decomposition property, and we can show this aggregation is valid for the generator schedule.

Chapter 4 suggests a more compact formulation of startup costs:

$$\sum_{t'=t-\mathbf{TC}+1}^{t-\mathbf{DT}} x_{[t',t)} \le v_t \qquad \forall t \in \mathcal{T}, \tag{F.12a}$$

$$\sum_{t'=t+\mathbf{DT}}^{t+\mathbf{TC}-1} x_{[t,t')} \le w_t \qquad \forall t \in \mathcal{T}, \tag{F.12b}$$

(where the sums are understood to be taken over valid t') and the objective function is

$$c_t^{SU} = \mathbf{c}^S v_t + \sum_{s=1}^{S-1} (\mathbf{c}^s - \mathbf{c}^S) \left(\sum_{t'=t-\overline{\mathbf{T}}^s+1}^{t-\underline{\mathbf{T}}^s} x_{[t',t)} \right) \qquad \forall t \in \mathcal{T}, \tag{F.12c}$$

and dropping the remaining $x_{[t,t')}$ for which $t' \geq t + \mathbf{TC}$. The resulting computational experiments suggest that this formulation dominates (F.11) computationally, in addition to the other formulations examined. The reason is that while (F.12) is not an integer polytope, it is integer "in the right direction," that is, any fractional vertex for 3-bin models using (F.12) will be dominated by an integer solution for all reasonable objective coefficients (the fractionally enters the formulation by allowing cooler startups to be assigned even when the generator is still hot).

D.3 Disaggregating schedules

In this section we will show how to decompose solutions to the aggregated formulation with various common features, such as time-dependent startup costs, reserves, and piecewise linear production costs. Suppose $\mathcal{K} \subset \mathcal{G}$ such that all generators in $\mathcal{K}$ have identical properties (save initial status). Let $U = \sum_{g \in \mathcal{K}} u^g$, $V = \sum_{g \in \mathcal{K}} v^g$, $W = \sum_{g \in \mathcal{K}} w^g$, and $X = \sum_{g \in \mathcal{K}} x^g$. Let $\mathbf{UT} = \mathbf{UT}^g$, $\mathbf{DT} = \mathbf{DT}^g$, and $\mathbf{TC} = \mathbf{TC}^g$ for some (every) $g \in \mathcal{K}$.

First consider the aggregated 3-bin model for commitment status with startup costs:

$$U_t - U_{t-1} = V_t - W_t \qquad \forall t \in \mathcal{T} \tag{F.13a}$$

$$\sum_{i=t-\mathbf{UT}+1}^{t} V_i \leq U_t \qquad \forall t \in [\mathbf{UT}, \mathbf{T}] \tag{F.13b}$$

$$\sum_{i=t-\mathbf{DT}+1}^{t} W_i \leq |\mathcal{K}| - U_t \qquad \forall t \in [\mathbf{DT}, \mathbf{T}] \tag{F.13c}$$

$$\sum_{t'=t-\mathbf{TC}+1}^{t-\mathbf{DT}} X_{[t',t)} \leq V_t \qquad \forall t \in \mathcal{T},\ \forall g \in \mathcal{G} \tag{F.13d}$$

$$\sum_{t'=t+\mathbf{DT}}^{t+\mathbf{TC}-1} X_{[t,t')} \leq W_t \qquad \forall t \in \mathcal{T},\ \forall g \in \mathcal{G} \tag{F.13e}$$

$$U_t, V_t, W_t \in \{0, \ldots, |\mathcal{K}|\} \qquad \forall t \in \mathcal{T} \tag{F.13f}$$

$$X_{[t,t')} \in \{0, \ldots, |\mathcal{K}|\} \qquad \forall\ t, t' \in \mathcal{T}^2 \text{ with } \mathbf{DT} \leq t' - t < \mathbf{TC}. \tag{F.13g}$$

Algorithm F.2 demonstrates how to disaggregate a solution to (F.13). We establish the correctness of Algorithm F.2 with the next theorem.

Algorithm F.2 (PeelOff) Constructs feasible generator schedules from a solution of (F.13).

Initialize all u^g, v^g, w^g, x^g to 0.
Initialize all $\hat{U}$, $\hat{V}$, $\hat{W}$, $\hat{X}$ to U^*, V^*, W^*, X^*, respectively.
if $U_1^* \geq 1$ **then**
  $u_1^g \leftarrow 1$; $\hat{U}_1 \leftarrow U_1^* - 1$;
  Assign historical startup v_t, for $-\mathbf{UT} < t \leq 0$.
else
  Assign historical shutdown w_t, for $-\mathbf{DT} < t \leq 0$.
$t \leftarrow 2$
while $t \leq \mathbf{T}$ **do**
  if $u_{t-1}^g = 1$ **then** ▷ If on in the previous period
    if $\sum_{i=t-\mathbf{UT}+1}^{t-1} v_i = 0$ **and** $W_t^* \geq 1$ **then** ▷ If we can turn off and a turn off is available
      $w_t^g \leftarrow 1$; $\hat{W}_t \leftarrow W_t^* - 1$; ▷ Turn off
      if $\exists t'$ s.t. $t + TD \leq t' < t + \mathbf{TC}$ **and** $X_{[t,t')}^* \geq 1$ **then** ▷ If there is a hot-start available
        $u_{t'}^g, v_{t'}^g \leftarrow 1$; $x_{[t,t')}^g \leftarrow 1$; ▷ Take it
        $\hat{U}_{t'} \leftarrow U_{t'}^* - 1$; $\hat{V}_{t'} \leftarrow V_{t'}^* - 1$;
        $\hat{X}_{[t,t')} \leftarrow X_{[t,t')}^* - 1$;
        $t \leftarrow t' + 1$;
      else if $\exists t' \geq t + \mathbf{TC}$ s.t. $\sum_{t'=t+\mathbf{DT}}^{t+\mathbf{TC}-1} X_{[t,t')}^* < V_{t'}^*$ **then** ▷ Else take some cold start, if possible
        $u_{t'}^g, v_{t'}^g \leftarrow 1$; $\hat{U}_{t'} \leftarrow U_{t'}^* - 1$; $\hat{V}_{t'} \leftarrow V_{t'}^* - 1$;
        $t \leftarrow t' + 1$;
      else ▷ If not, stay off for the rest of the time horizon
        $t \leftarrow \mathbf{T} + 1$;
    else ▷ If we could not turn off or a turn off was not available
      $u_t \leftarrow 1$; $\hat{U}_t \leftarrow U_t^* - 1$; ▷ Stay on
      $t \leftarrow t + 1$;
  else ▷ $u_{t-1} = 0$, i.e., off previously
    if $\sum_{i=t-\mathbf{DT}+1}^{t-1} w_i = 0$ **and** $V_t^* \geq 1$ **then** ▷ If we can turn on and a turn on is available
      $u_t, v_t \leftarrow 1$; $\hat{U}_t \leftarrow U_t^* - 1$; $\hat{V}_t \leftarrow V_t^* - 1$; ▷ Turn on
      if $\exists t' \in (t - \mathbf{TC}, t - \mathbf{DT}] \cap \mathbb{Z}$ s.t. $X_{[t',t)}^* \geq 1$ **then** ▷ If there is a historical hot-start
        $x_{[t',t)}^g \leftarrow 1$; $\hat{X}_{[t',t)} \leftarrow X_{[t',t)}^* - 1$; ▷ Take it
        $w_{t'}^g \leftarrow 1$; $\hat{W}_{t'} \leftarrow W_{t'}^* - 1$; ▷ Assign historical data for $t \leq -\mathbf{DT}$
      $t \leftarrow t + 1$;
    else ▷ If no turn on feasible or available
      $t \leftarrow t + 1$; ▷ Stay off

Theorem F.4. *Suppose* (U^*, V^*, W^*, X^*) *is a feasible solution for* (F.13)*. Then for every* $g \in \mathcal{K}$ *there exist* $(u^{g*}, v^{g*}, w^{g*}, x^{g*})$ *feasible for the minimum up-time/down-time system for* $\mathcal{K}$*:*

$$
\begin{aligned}
& u_t^g - u_{t-1}^g = v_t^g - w_t^g && \forall t \in \mathcal{T},\ \forall g \in \mathcal{K} \\
& \sum_{i=t-\mathbf{UT}+1}^{t} v_i^g \le u_t^g && \forall t \in [\mathbf{UT}, \mathbf{T}],\ \forall g \in \mathcal{K} \\
& \sum_{i=t-\mathbf{DT}+1}^{t} w_i^g \le 1 - u_t^g && \forall t \in [\mathbf{DT}, \mathbf{T}],\ \forall g \in \mathcal{K} \\
& \sum_{t'=t-\mathbf{TC}+1}^{t-\mathbf{DT}} x_{[t',t)} \le v_t^g && \forall t \in \mathcal{T},\ \forall g \in \mathcal{K} \\
& \sum_{t'=t+\mathbf{DT}}^{t+\mathbf{TC}-1} x_{[t,t')} \le w_t^g && \forall t \in \mathcal{T},\ \forall g \in \mathcal{K} \\
& u_t^g, v_t^g, w_t^g \in \{0,1\} && \forall t \in \mathcal{T},\ \forall g \in \mathcal{K} \\
& x_{[t,t')} \in \{0, \dots, |\mathcal{K}|\} && \forall\ t, t' \in \mathcal{T}^2 \text{ with } \mathbf{DT} \le t' - t < \mathbf{TC}.
\end{aligned}
$$

Proof. Clearly this is true when $|\mathcal{K}| = 1$. We will proceed by induction on the size of $\mathcal{K}$, "peeling off" feasible binary vectors and leaving behind a still feasible crushed system. Suppose (U^*, V^*, W^*, X^*) is feasible (F.13), and $|\mathcal{K}| = I$. We wish to find a feasible solution to the following system:

$$
\begin{aligned}
& U_t^* = \hat{U}_t + u_t^g,\ V_t^* = \hat{V}_t + v_t^g,\ W_t^* = \hat{W}_t + w_t^g && \forall t \in \mathcal{T} && \text{(F.14a)} \\
& X_{[t,t')}^* = \hat{X}_{[t,t')} + x_{[t,t')} && \forall\ t, t' \in \mathcal{T}^2 \text{ with } \mathbf{DT} \le t' - t < \mathbf{TC} && \text{(F.14b)}
\end{aligned}
$$

$$
\begin{aligned}
& u_t^g - u_{t-1}^g = v_t^g - w_t^g && \forall t \in \mathcal{T} && \text{(F.15a)} \\
& \sum_{i=t-\mathbf{UT}+1}^{t} v_i^g \le u_t^g && \forall t \in [\mathbf{UT}, \mathbf{T}] && \text{(F.15b)} \\
& \sum_{i=t-\mathbf{DT}+1}^{t} w_i^g \le 1 - u_t^g && \forall t \in [\mathbf{DT}, \mathbf{T}] && \text{(F.15c)} \\
& \sum_{t'=t-\mathbf{TC}+1}^{t-\mathbf{DT}} x_{[t',t)} \le v_t^g && \forall t \in \mathcal{T} && \text{(F.15d)} \\
& \sum_{t'=t+\mathbf{DT}}^{t+\mathbf{TC}-1} x_{[t,t')} \le w_t^g && \forall t \in \mathcal{T} && \text{(F.15e)}
\end{aligned}
$$

$$u_t^g, v_t^g, w_t^g \in \{0,1\} \qquad \forall t \in \mathcal{T} \tag{F.15f}$$

$$x^g_{[t,t')} \in \{0,1\} \qquad \forall\ t,t' \in \mathcal{T}^2 \text{ with } \mathbf{DT} \le t' - t < \mathbf{TC} \tag{F.15g}$$

$$\hat{U}_t - \hat{U}_{t-1} = \hat{V}_t - \hat{W}_t \qquad \forall t \in \mathcal{T} \tag{F.16a}$$

$$\sum_{i=t-\mathbf{UT}+1}^{t} \hat{V}_i \le \hat{U}_t \qquad \forall t \in [\mathbf{UT}, \mathbf{T}] \tag{F.16b}$$

$$\sum_{i=t-\mathbf{DT}+1}^{t} \hat{W}_i \le (I-1) - \hat{U}_t \qquad \forall t \in [\mathbf{DT}, \mathbf{T}] \tag{F.16c}$$

$$\sum_{t'=t-\mathbf{TC}+1}^{t-\mathbf{DT}} \hat{X}_{[t',t)} \le \hat{V}_t \qquad \forall t \in \mathcal{T} \tag{F.16d}$$

$$\sum_{t'=t+\mathbf{DT}}^{t+\mathbf{TC}-1} \hat{X}_{[t,t')} \le \hat{W}_t \qquad \forall t \in \mathcal{T} \tag{F.16e}$$

$$\hat{U}_t, \hat{V}_t, \hat{W}_t \in \{0,\dots,I-1\} \qquad \forall t \in \mathcal{T} \tag{F.16f}$$

$$\hat{X}_{[t,t')} \in \{0,\dots,I-1\} \qquad \forall\ t,t' \in \mathcal{T}^2 \text{ with } \mathbf{DT} \le t' - t < \mathbf{TC}. \tag{F.16g}$$

Algorithm F.2 constructs a feasible solution to (F.14, F.15, F.16) from a solution of (F.13). To see this, first notice that the solution returned by Algorithm F.2 always has (F.14). Similarly, Algorithm F.2 constructs a feasible solution for (u^g, v^g, w^g, x^g), so (F.15) holds. Further, (F.13a) and (F.15a) together imply (F.16a). Notice that given the bounds on $\hat{U}$ and (F.16b - F.16e), we get the bounds on $\hat{V}$, $\hat{W}$, and $\hat{X}$, and the proof is finished. Therefore we check the bounds on $\hat{U}$ and (F.16b–F.16e), proceeding by contraction each time.

$\hat{U}_t \le I-1$: Let t be the first time period such that $\hat{U}_t > I-1$ (notice $t > 1$ by line 8 in Algorithm F.2). Then $\hat{U}_t = I$ and further, $u_t = 0$, $U_t^* = I$. Now by (F.13c), $\sum_{i=t-\mathbf{DT}+1}^{t} W_i^* \le I - I = 0$. Therefore $W_i^* = 0,\ \forall i \in [t-\mathbf{DT}+1, t]$, yielding $w_i^g = 0,\ \forall i \in [t-\mathbf{DT}+1, t]$. Since $U_t^* = I$ and $W_t^* = 0$, (F.13a) gives $I = U_{t-1}^* + V_t^*$. If $V_t^* > 0$, since g is eligible for a turn-on, this contracts line 27 in Algorithm F.2. If $V_t^* = 0$, then $U_{t-1}^* > I-1$, contracting the minimality of t.

(F.16b): Suppose there is t with $\sum_{i=t-\mathbf{UT}+1} \hat{V}_i > \hat{U}_i$. We have the following relation:

$$U_t^* \overset{v_i^g \geq 0}{\geq} U_t^* - \sum_{i=t-\mathbf{UT}+1}^{t} v_i^g \overset{\text{(F.13b)}}{\geq} \sum_{i=t-\mathbf{UT}+1}^{t} (V_i^* - v_i^g)$$
$$\overset{\text{(F.14a)}}{=} \sum_{i=t-\mathbf{UT}+1}^{t} \hat{V}_t > \hat{U}_t \overset{\text{(F.14a)}}{=} U_t^* - u_t^g \overset{u_t^g \leq 1}{\geq} U_t^* - 1 \tag{F.17}$$

Since the far left and far right differ by only 1, and all quantities are integer, in order for the strict inequality to hold, all weak inequalities in (F.17) must be equalities. Therefore we have (a) $u_t^g = 1$, (b) $\sum_{i=t-\mathbf{UT}+1}^{t} v_i = 0$, and (c) $\sum_{i=t-\mathbf{UT}+1}^{t} V_i^* = U_t^*$. Together (a) and (b) imply $u_i^g = 1$ and $U_i^* \geq 1$ for every $i \in \{t - \mathbf{UT}, \ldots, t\}$. Therefore generator g started-up most recently at some time $\hat{i}$ such that $\hat{i} \leq t - \mathbf{UT}$; i.e $v_{\hat{i}} = 1$. Line 11 of Algorithm F.2 implies then that $W_i^* = 0$ for every $i \in \{\hat{i} + \mathbf{UT}, \ldots, t\}$, and hence (d) $U_i^* - U_{i-1}^* = V_i^*$ for every $i \in \{\hat{i} + \mathbf{UT}, \ldots, t\}$ by equation (F.13a). There are but two cases then.

Case 1. Suppose $\hat{i} + \mathbf{UT} \leq t - \mathbf{UT} + 1$. Then $\sum_{i=t-\mathbf{UT}+1}^{t} V_i^* = U_t^* - U_{t-\mathbf{UT}}^*$ by (d). By (c) then $U_{t-\mathbf{UT}}^* = 0$ but (a) and (b) give $U_{t-\mathbf{UT}}^* \geq 1$.

Case 2. Suppose $t - \mathbf{UT} + 1 < \hat{i} + \mathbf{UT}$. We have:

$$U_t^* \overset{\text{(c)}}{=} \sum_{i=t-\mathbf{UT}+1}^{t} V_i^* = \sum_{i=t-\mathbf{UT}+1}^{\hat{i}+\mathbf{UT}-1} V_i^* + \sum_{i=\hat{i}+\mathbf{UT}}^{t} V_i^* \overset{\text{(d)}}{=} \sum_{i=t-\mathbf{UT}+1}^{\hat{i}+\mathbf{UT}-1} V_i^* + U_t^* - U_{\hat{i}+\mathbf{UT}-1}^* \tag{F.18}$$

Subtracting U_t^* from both sides we get the relation $0 = \sum_{i=t-\mathbf{UT}+1}^{\hat{i}+\mathbf{UT}-1} V_i^* - U_{\hat{i}+\mathbf{UT}-1}^*$. Finally then, we see

$$0 = \sum_{i=t-\mathbf{UT}+1}^{\hat{i}+\mathbf{UT}-1} V_i^* - U_{\hat{i}+\mathbf{UT}-1}^* \overset{\text{(F.13b)}}{\leq} \sum_{i=t-\mathbf{UT}+1}^{\hat{i}+\mathbf{UT}-1} V_i^* - \sum_{i=\hat{i}}^{\hat{i}+\mathbf{UT}-1} V_i^* = -\sum_{i=\hat{i}}^{t-\mathbf{UT}} V_i^* \overset{v_{\hat{i}}=1}{\leq} -1, \tag{F.19}$$

yielding the contraction desired.

(F.16c): Very similar to the proof for (F.16b).

(F.16d): Supposing there is t with $\sum_{t'=t-\mathbf{DT}+1}^{t-\mathbf{DT}} \hat{X}_{[t',t)} > \hat{V}_t$ we can use the same technique from the proof of (F.16b) to get the following string of inequalities:

$$V_t^* \geq V_t^* - \sum_{t'=t-\mathbf{TC}+1}^{t-\mathbf{DT}} x_{[t',t)}^g \overset{\text{(F.13d)}}{\geq} \sum_{t'=t-\mathbf{TC}+1}^{t-\mathbf{DT}} X_{[t',t)}^* - \sum_{t'=t-\mathbf{TC}+1}^{t-\mathbf{DT}} x_{[t',t)}^g$$

$$\overset{\text{(F.14b)}}{=} \sum_{t'=t-\mathbf{TC}+1}^{t-\mathbf{DT}} \hat{X}_{[t',t)} > \hat{V}_t \overset{\text{(F.14a)}}{=} V_t^* - v_t^g \geq V_t^* - 1 \tag{F.20}$$

Hence we may conclude (a) $v_t^g = 1$, (b) $\sum_{t'=t-\mathbf{TC}+1}^{t-\mathbf{DT}} x_{[t',t)}^g = 0$, and (c) $\sum_{t'=t-\mathbf{TC}+1}^{t-\mathbf{DT}} X_{[t',t)}^* = V_t^*$. We have (a) implies that $V_t^* \geq 1$, so $\sum_{t'=t-\mathbf{TC}+1}^{t-\mathbf{DT}} X_{[t',t)}^* \geq 1$, so $\exists t' \in \{t-\mathbf{DT}+1, \ldots, t-\mathbf{TC}\}$ such that $X_{[t',t)}^* \geq 1$. But line 14 in Algorithm F.2 then sets $x_{[t',t)}^g = 1$, a contraction.

(F.16e): This is the same as (F.16d).

Hence Algorithm F.2 constructs a feasible solution for (F.14, F.15, F.16). We can then proceed in this manner until $I = 1$, proving the theorem. □

Notice that Algorithm F.2 constructs solutions exactly of the type in Theorems 5.2 and F.1. Theorem F.2 justifies the arbitrary choice in lines 13 – 20 for which shutdown/startup path the generator takes. Hence we do not loose anything in optimality or feasibility by aggregating a fast-ramping generator's status variables, even in the presence of time-dependent startup costs. Next we will see how to disaggregate the power and reserve variables.

D.4 Disaggregating Power and Reserves

Disaggregating Power when UT ≥ 2

Consider the power output for an aggregated set of identical fast-ramping generators. First we will consider the case when $\mathbf{UT} \geq 2$. Along with (F.13), we the aggregated power $P = \sum_{g \in \mathcal{K}} p^g$ and reserves $R = \sum_{g \in \mathcal{K}} r^g$ with the constraints:

$$\underline{\mathbf{P}} U_t \leq P_t \qquad \forall t \in \mathcal{T}, \tag{F.21a}$$

$$P_t + R_t \leq \overline{\mathbf{P}} U_t + (\mathbf{SU} - \overline{\mathbf{P}}) V_t + (\mathbf{SD} - \overline{\mathbf{P}}) W_{t+1}, \qquad \forall t \in \mathcal{T}. \tag{F.21b}$$

Since (F.21) is just a sum of constraints, it is clearly valid.

If $\mathbf{SU} \geq \mathbf{SD}$, then Algorithm F.3a demonstrates how to disaggregate power. On the other hand, when $\mathbf{SD} \geq \mathbf{SU}$ disaggregation can be done in an analogous fashion, as shown in Algorithm F.3b. The essential logic of Algorithms F.3a and F.3b is that of Theorem 5.1. That is, either the power outputs of all generators are equal, or either the startup- and/or shutdown-ramping constraints are active. When $\mathbf{SU} = \mathbf{SD}$ then Algorithms F.3a and F.3b give the same result.

Algorithm F.3a (Peel Off Power) Constructs feasible generator schedule from a solution of (F.21) when $\mathbf{SU} \geq \mathbf{SD}$.

for $g \in \mathcal{K},\ t \in \mathcal{T}$ **do**
  if $P_t^*/U_t^* \leq \mathbf{SD}$ **then** ▷ If the average power is less than **SD**
    if $u_t^g = 1$ **then**
      $p_t^g \leftarrow P_t^*/U_t^*$ ▷ Give all generators on average power
    else
      $p_t^g \leftarrow 0$
  else if $(P_t^* - \mathbf{SD} \cdot W_{t+1}^*)/U_t^* \leq \mathbf{SU}$ **then** ▷ If not, check if remaining average power $\leq$ **SU**
    if $w_{t+1}^g = 1$ **then**
      $p_t^g \leftarrow \mathbf{SD}$ ▷ Give generators shutting down **SD**
    else if $u_t^g = 1$ **then**
      $p_t^g \leftarrow (P_t^* - \mathbf{SD} \cdot W_{t+1}^*)/U_t^*$ ▷ Give all others on remaining average power
    else
      $p_t^g \leftarrow 0$
  else ▷ $(P_t^* - \mathbf{SD} \cdot W_{t+1}^*)/U_t^* > \mathbf{SU}$, so we need separate out generators starting
    if $w_{t+1}^g = 1$ **then**
      $p_t^g \leftarrow \mathbf{SD}$ ▷ Give generators shutting down **SD**
    else if $v_t^g = 1$ **then**
      $p_t^g \leftarrow \mathbf{SU}$ ▷ Give (remaining) generators starting up **SU**
    else if $u_t^g = 1$ **then**
      $p_t^g \leftarrow (P_t^* - \mathbf{SU} \cdot V_t^* - \mathbf{SD} \cdot W_{t+1}^*)/U_t^*$ ▷ all others on get remaining average power
    else
      $p_t^g \leftarrow 0$

Algorithm F.3b (Peel Off Power) Constructs feasible generator schedule from a solution of (F.21) when $\mathbf{SD} \geq \mathbf{SU}$.

for $g \in \mathcal{K},\ t \in \mathcal{T}$ **do**
  if $P_t^*/U_t^* \leq \mathbf{SU}$ **then** ▷ If the average power is less than **SU**
    if $u_t^g = 1$ **then**
      $p_t^g \leftarrow P_t^*/U_t^*$ ▷ Give all generators on average power
    else
      $p_t^g \leftarrow 0$
  else if $(P_t^* - \mathbf{SU} \cdot V_t^*)/U_t^* \leq \mathbf{SD}$ **then** ▷ If not, check if remaining average power $\leq$ **SD**
    if $v_t^g = 1$ **then**
      $p_t^g \leftarrow \mathbf{SU}$ ▷ Give generators starting up **SU**
    else if $u_t^g = 1$ **then**
      $p_t^g \leftarrow (P_t^* - \mathbf{SU} \cdot V_t^*)/U_t^*$ ▷ Give all others on remaining average power
    else
      $p_t^g \leftarrow 0$
  else ▷ $(P_t^* - \mathbf{SU} \cdot V_t^*)/U_t^* > \mathbf{SD}$, so we need separate out generators stopping
    if $v_t^g = 1$ **then**
      $p_t^g \leftarrow \mathbf{SU}$ ▷ Give generators starting up **SU**
    else if $w_{t+1}^g = 1$ **then**
      $p_t^g \leftarrow \mathbf{SD}$ ▷ Give (remaining) generators shutting down **SD**
    else if $u_t^g = 1$ **then**
      $p_t^g \leftarrow (P_t^* - \mathbf{SU} \cdot V_t^* - \mathbf{SD} \cdot W_{t+1}^*)/U_t^*$ ▷ all others on get remaining average power
    else
      $p_t^g \leftarrow 0$

Disaggregating Power when UT $= 1$

When $\mathbf{UT} = 1$, we need consider a modified version of the aggregated generator's production constraint, as is the case for a single generator [110, 58]. Again using the aggregated variables from before, consider the aggregated production constraints

$$\underline{\mathbf{P}} U_t \leq P_t \qquad \forall t \in \mathcal{T} \tag{F.22a}$$

$$P_t + R_t \leq \overline{\mathbf{P}} U_t + (\mathbf{SU} - \overline{\mathbf{P}}) V_t, \qquad \forall t \in \mathcal{T} \tag{F.22b}$$

$$P_t + R_t \leq \overline{\mathbf{P}} U_t + (\mathbf{SD} - \overline{\mathbf{P}}) W_{t+1}, \qquad \forall t \in \mathcal{T}. \tag{F.22c}$$

When $\mathbf{SU} \geq \mathbf{SD}$, we can again use Algorithm F.3a, except we need modify line 20 to

$$p_t^g \leftarrow (P_t^* - \mathbf{SU} \cdot \min\{V_t^* - W_{t+1}^*,\ 0\} - \mathbf{SD} \cdot W_{t+1}^*)/U_t^*. \tag{F.23}$$

The correctness of Algorithm F.3a the modification above to line 20 follows from Theorem 5.1 and Algorithm F.2. That is, $\min\{V_t^*,\ W_{t+1}^*\}$ generators turn on at time t and turn off at time $t+1$, and hence $\min\{V_t^* - W_{t+1}^*,\ 0\}$ will get $\mathbf{SU}$ power at line 18.

Similarly when $\mathbf{SD} \geq \mathbf{SU}$, we can use Algorithm F.3b, modifying line 20 to

$$p_t^g \leftarrow (P_t^* - \mathbf{SU} \cdot V_t^* - \mathbf{SD} \cdot \min\{W_{t+1}^* - V_t^*,\ 0\})/U_t^*. \tag{F.24}$$

The correctness of Algorithm F.3b with the modification to line 20 is exactly analogous to that in the $\mathbf{SU} \geq \mathbf{SD}$ case above.

Disaggregating Reserves

Having disaggregated power, reserves r^g can be disaggregated by considering $(p_t^g + r_t^g)$ and $(P_t^* + R_t^*)$ in Algorithm F.3a or F.3b (or their modified analogs when $\mathbf{UT} = 1$) above in place of p_t^g and P_t^* respectively.

D.5 Disaggregating Piecewise Linear Production Costs

Disaggregating Piecewise Linear Production Costs when UT ≥ 2

In the case of piecewise production costs we can modify (F.9) by considering the aggregated piecewise production variables $P^l = \sum_{g \in \mathcal{K}} p^{l,g}$. We consider the sum of constraints of the

form (F.9), recalling l' is such that $\mathbf{SU} = \mathbf{SD} = \overline{\mathbf{P}}^{l'}$

$$P_t^l \leq (\overline{\mathbf{P}}^l - \overline{\mathbf{P}}^{l-1})U_t \qquad \forall l \in [l'],\ \forall t \in \mathcal{T} \tag{F.25a}$$

$$P_t^l \leq (\overline{\mathbf{P}}^l - \overline{\mathbf{P}}^{l-1})(U_t - V_t - W_{t+1}) \qquad \forall l > l',\ \forall t \in \mathcal{T} \tag{F.25b}$$

$$P_t^l \geq 0 \qquad \forall l \in \mathcal{L},\ \forall t \in \mathcal{T} \tag{F.25c}$$

If the generator is on and did not just turn on nor is about to turn off ($u_t^g = 1,\ v_t^g = 0,\ w_{t+1}^g = 0$), then $p_t^{l,g} = P_t^{*l}/U_t^*$. If the generator just turned on ($u_t^g = 1,\ v_t^g = 1,\ w_{t+1}^g = 0$), then

$$p_t^{l,g} = P_t^{*l}/U_t^* \qquad \forall l \in [l'],\ \forall t \in \mathcal{T} \tag{F.26a}$$

$$p_t^{l,g} = 0 \qquad \forall l > l',\ \forall t \in \mathcal{T}, \tag{F.26b}$$

and similarly if the generator is just about to turn off ($u_t^g = 1,\ v_t^g = 0,\ w_{t+1}^g = 1$) Notice that with the aggregated linking constraints for piecewise production

$$P_t = \underline{\mathbf{P}} U_t + \sum_{l \in \mathcal{L}} P_t^l \qquad \forall t \in \mathcal{T},$$

the assignment given by (F.26) is compatible with Algorithms F.3a and F.3b. $p_t^{l,g}$ is 0 for all l, t when the generator is off ($u_t^g = 0$).

Consider the case when $\mathbf{SU} > \mathbf{SD}$. Letting $\overline{\mathbf{P}}^{l^{\mathbf{SU}}} = \mathbf{SU}$ and $\overline{\mathbf{P}}^{l^{\mathbf{SD}}} = \mathbf{SD}$, ($l^{\mathbf{SU}} > l^{\mathbf{SD}}$) we may use a modified version of (F.25)

$$P_t^l \leq (\overline{\mathbf{P}}^l - \overline{\mathbf{P}}^{l-1})U_t \qquad \forall l \in [l^{\mathbf{SD}}],\ \forall t \in \mathcal{T} \tag{F.27a}$$

$$P_t^l \leq (\overline{\mathbf{P}}^l - \overline{\mathbf{P}}^{l-1})(U_t - W_{t+1}) \qquad \forall l \in (l^{\mathbf{SD}}, l^{\mathbf{SU}}],\ \forall t \in \mathcal{T} \tag{F.27b}$$

$$P_t^l \leq (\overline{\mathbf{P}}^l - \overline{\mathbf{P}}^{l-1})(U_t - V_t - W_{t+1}) \qquad \forall l > l^{\mathbf{SU}},\ \forall t \in \mathcal{T} \tag{F.27c}$$

$$P_t^l \geq 0 \qquad \forall l \in \mathcal{L},\ \forall t \in \mathcal{T}. \tag{F.27d}$$

Like before, if the generator did not just turn on nor is about to turn off ($u_t^g = 1,\ v_t^g = 0,\ w_{t+1}^g = 0$), then $p^{l,g} = P_t^{*l}/U_t^*$. If the generator just turned on ($u_t^g = 1,\ v_t^g = 1,\ w_{t+1}^g = 0$), then

$$p_t^{l,g} = P_t^{*l}/U_t^* \qquad \forall l \in [l^{\mathbf{SU}}],\ \forall t \in \mathcal{T} \tag{F.28a}$$

$$p_t^{l,g} = 0 \qquad \forall l > l^{\mathbf{SU}},\ \forall t \in \mathcal{T}. \tag{F.28b}$$

If the generator is just about to turn off ($u_t^g = 1,\ v_t^g = 0,\ w_{t+1}^g = 1$), then

$$p_t^{l,g} = P_t^{*l}/U_t^* \qquad \forall l \in [l^{\mathbf{SD}}],\ \forall t \in \mathcal{T} \tag{F.29a}$$

$$p_t^{l,g} = 0 \qquad \forall l > l^{\mathbf{SD}},\ \forall t \in \mathcal{T}. \tag{F.29b}$$

Finally, $p_t^{l,g} = 0$ for all l, t when the generator g is off ($u_t^g = 0$).

The case when $\mathbf{SD} > \mathbf{SU}$ can be handled similarly.

Disaggregating Piecewise Linear Production Costs when UT = 1

When $\mathbf{UT} = 1$, the aggregated piecewise power constraints need to be modified as well. Under the assumption $\overline{\mathbf{P}}^{l'} = \mathbf{SU} = \mathbf{SD}$, consider the aggregated version of (F.10)

$$P_t^l \le (\overline{\mathbf{P}}^l - \overline{\mathbf{P}}^{l-1})U_t \qquad \forall l \in [l'],\ \forall t \in \mathcal{T} \tag{F.30a}$$

$$P_t^l \le (\overline{\mathbf{P}}^l - \overline{\mathbf{P}}^{l-1})(U_t - V_t) \qquad \forall l > l',\ \forall t \in \mathcal{T} \tag{F.30b}$$

$$P_t^l \le (\overline{\mathbf{P}}^l - \overline{\mathbf{P}}^{l-1})(U_t - W_{t+1}) \qquad \forall l > l',\ \forall t \in \mathcal{T} \tag{F.30c}$$

$$P_t^l \ge 0 \qquad \forall l \in \mathcal{L},\ \forall t \in \mathcal{T}. \tag{F.30d}$$

Without loss of generality, we can assign the piecewise power making the same modifications that were necessary for the power production. In particular, we have that $p_t^{l,g} = P_t^{*l}/U_t^*$ if $u_t^g = 1,\ v_t^g = w_{t+1}^g = 0$. If is a startup or shutdown (or both), ($u_t^g = 1,\ v_t^g$ and/or $w_{t+1}^g = 1$), then we can use (F.26).

Assuming $\mathbf{SU} \neq \mathbf{SD}$, we can introduce $l^{\mathbf{SU}}$ and $l^{\mathbf{SD}}$ as before for (F.27)

$$P_t^l \le (\overline{\mathbf{P}}^l - \overline{\mathbf{P}}^{l-1})U_t \qquad \forall l \in [\max\{l^{\mathbf{SU}}, l^{\mathbf{SD}}\}],\ \forall t \in \mathcal{T} \tag{F.31a}$$

$$P_t^l \le (\overline{\mathbf{P}}^l - \overline{\mathbf{P}}^{l-1})(U_t - V_t) \qquad \forall l > l^{\mathbf{SU}},\ \forall t \in \mathcal{T} \tag{F.31b}$$

$$P_t^l \le (\overline{\mathbf{P}}^l - \overline{\mathbf{P}}^{l-1})(U_t - W_{t+1}) \qquad \forall l > l^{\mathbf{SD}},\ \forall t \in \mathcal{T} \tag{F.31c}$$

$$P_t^l \ge 0 \qquad \forall l \in \mathcal{L},\ \forall t \in \mathcal{T}. \tag{F.31d}$$

If we have $u_t^g = 1,\ v_t^g = 0,\ w_{t+1}^g = 0$, then $p_t^{l,g} = P_t^{*l}/U_t^{*l}$. Then there are three other cases to consider. Suppose $\mathbf{SU} > \mathbf{SD}$. If the generator is just starting and does not shutdown ($u_t^g = 1, v_t^g = 1, w_{t+1}^g = 0$), then (F.28) applies, and if the generator is shutting down ($u_t^g = 1,\ v_t^g = 0$ or $1,\ w_{t+1}^g = 1$), then (F.29) applies.

The case when $\mathbf{SD} > \mathbf{SU}$ is handled analogously. That is, if the generator is shutting down and did just startup ($u_t^g = 1,\ v_t^g = 0,\ w_{t+1}^g = 1$), then (F.29) is used, and if the generator is starting up ($u_t^g = 1,\ v_t^g = 1, w_t^g = 0$ or 1) then (F.28) applies.

E Full Formulations

In this section we specify the unit commitment formulations tested in Section 5.6.

E.1 Three Binary Formulation

Suppose that generator operating cost is piecewise linear and convex in p_t^g, and let $\mathcal{G}^{>1} = \{g \in \mathcal{G} \mid \mathbf{UT}^g > 1\}$ and $\mathcal{G}^1 = \{g \in \mathcal{G} \mid \mathbf{UT}^g = 1\}$. The following is the "3-bin" formulation from [110] and Chapter 4, which is used as a basis for comparison in Section 5.6. In Appendix G we consider this formation with the addition of static symmetry-breaking inequalities from [87].

$$\min \sum_{g\in\mathcal{G}}\sum_{t\in\mathcal{T}}\left(\sum_{l\in\mathcal{L}^g}(\mathbf{c}^{l,g}p_t^{l,g})+\mathbf{c}^{R,g}u_t^g+\mathbf{c}^{S,g}v_t^g+\sum_{s=1}^{S^g-1}(\mathbf{c}^{s,g}-\mathbf{c}^{S,g})\left(\sum_{t'=t-\overline{\mathbf{T}}^{s,g}+1}^{t-\underline{\mathbf{T}}^{s,g}} x_{[t',t)}^g\right)\right) \tag{F.32a}$$

subject to:

$$\sum_{g\in\mathcal{G}}(p_t^g+\underline{\mathbf{P}}^g u_t^g)=\mathbf{D}_t \qquad \forall t\in\mathcal{T} \tag{F.32b}$$

$$\sum_{g\in\mathcal{G}} r_t^g \geq \mathbf{R}_t \qquad \forall t\in\mathcal{T} \tag{F.32c}$$

$$p_t^g+r_t^g \leq (\overline{\mathbf{P}}^g-\underline{\mathbf{P}}^g)u_t^g-(\overline{\mathbf{P}}^g-\mathbf{SU}^g)v_t^g \qquad \forall t\in\mathcal{T},\forall g\in\mathcal{G}^1 \tag{F.32d}$$

$$p_t^g+r_t^g \leq (\overline{\mathbf{P}}^g-\underline{\mathbf{P}}^g)u_t^g-(\overline{\mathbf{P}}^g-\mathbf{SD}^g)w_{t+1}^g \qquad \forall t\in\mathcal{T},\forall g\in\mathcal{G}^1 \tag{F.32e}$$

$$p_t^g+r_t^g \leq (\overline{\mathbf{P}}^g-\underline{\mathbf{P}}^g)u_t^g-(\overline{\mathbf{P}}^g-\mathbf{SU}^g)v_t^g-(\overline{\mathbf{P}}^g-\mathbf{SD}^g)w_{t+1}^g \qquad \forall t\in\mathcal{T},\forall g\in\mathcal{G}^{>1} \tag{F.32f}$$

$$p_t^g+r_t^g-p_{t-1}^g \leq \mathbf{RU}^g \qquad \forall t\in\mathcal{T},\forall g\in\mathcal{G} \tag{F.32g}$$

$$p_{t-1}^g-p_t^g \leq \mathbf{RD}^g \qquad \forall t\in\mathcal{T},\forall g\in\mathcal{G} \tag{F.32h}$$

$$p_t^g=\sum_{l\in\mathcal{L}^g} p_t^{l,g} \qquad \forall t\in\mathcal{T},\forall g\in\mathcal{G} \tag{F.32i}$$

$$p_t^{l,g} \leq (\overline{\mathbf{P}}^{l,g}-\overline{\mathbf{P}}^{l-1,g}) \qquad \forall t\in\mathcal{T},\forall l\in\mathcal{L}^g,\forall g\in\mathcal{G} \tag{F.32j}$$

$$u_t^g-u_{t-1}^g=v_t^g-w_t^g \qquad \forall t\in\mathcal{T},\forall g\in\mathcal{G} \tag{F.32k}$$

$$\sum_{i=t-\mathbf{UT}^g+1}^{t} v_i^g \leq u_t^g \qquad \forall t \in [\mathbf{UT}^g, \mathbf{T}], \forall g \in \mathcal{G} \qquad \text{(F.32l)}$$

$$\sum_{i=t-\mathbf{DT}^g+1}^{t} w_i^g \leq 1 - u_t^g \qquad \forall t \in [\mathbf{DT}^g, \mathbf{T}], \forall g \in \mathcal{G} \qquad \text{(F.32m)}$$

$$\sum_{t'=t-\mathbf{TC}^g+1}^{t-\mathbf{DT}^g} x_{[t',t)}^g \leq v_t^g \qquad \forall t \in \mathcal{T}, \forall g \in \mathcal{G} \qquad \text{(F.32n)}$$

$$\sum_{t'=t+\mathbf{DT}^g}^{t+\mathbf{TC}^g-1} x_{[t,t')}^g \leq w_t^g \qquad \forall t \in \mathcal{T}, \forall g \in \mathcal{G} \qquad \text{(F.32o)}$$

$$p_t^{l,g} \in \mathbb{R}_+ \qquad \forall t \in \mathcal{T}, \forall l \in \mathcal{L}^g, \forall g \in \mathcal{G} \qquad \text{(F.32p)}$$

$$p_t^g, r_t^g \in \mathbb{R}_+ \qquad \forall t \in \mathcal{T}, \forall g \in \mathcal{G} \qquad \text{(F.32q)}$$

$$u_t^g, v_t^g, w_t^g \in \{0,1\} \qquad \forall t \in \mathcal{T}, \forall g \in \mathcal{G} \qquad \text{(F.32r)}$$

$$x_{[t,t')}^g \in \{0,1\} \qquad \forall [t,t') \in \mathcal{X}^g, \forall g \in \mathcal{G}. \qquad \text{(F.32s)}$$

E.2 Aggregation Formulation

We now lay out the aggregation formulation introduced in this paper, denoted "3-bin+A" and "EF/3-bin+A" in Section 5.6. For $g_1, g_2 \in \mathcal{G}$, define g_1 to be equivalent to g_2 if and only if g_1 and g_2 have identical parameters, and denote this relationship $g_1 \sim g_2$ (note that $\sim$ is an equivalence relation). Consider the set of generators $\mathcal{G}_{\not\sim} = \{g \in \mathcal{G} \mid g \not\sim g' \ \forall g' \in \mathcal{G} \setminus \{g\}\}$, that is, $\mathcal{G}_{\not\sim}$ is the subset of generators in their own equivalence classes. Let $\mathcal{G}_{\sim} = \mathcal{G} \setminus \mathcal{G}_{\not\sim}$, and partition it into two subsets, $\mathcal{G}_{\sim F} = \{g \in \mathcal{G}_{\sim} \mid \mathbf{RU}^g, \mathbf{RD}^g \geq (\overline{\mathbf{P}}^g - \underline{\mathbf{P}}^g)\}$ and $\mathcal{G}_{\sim S} = \mathcal{G}_{\sim} \setminus \mathcal{G}_{\sim F}$. Hence $\mathcal{G}_{\sim S}$ is the set of slow-ramping generators where each $g \in \mathcal{G}_{\sim S}$ has some other $g' \in \mathcal{G}_{\sim S}$ with $g \neq g'$ and $g \sim g'$. In a similar way $\mathcal{G}_{\sim F}$ is the set of fast-ramping generators with this property. So both $\mathcal{G}_{\sim F}$ and $\mathcal{G}_{\sim S}$ can be partitioned into equivalence classes under $\sim$, with each class having more than one member. Denote these $\mathscr{G}_F$ and $\mathscr{G}_S$, respectively. Note then that each $\mathcal{K} \in \mathscr{G}_F$ ($\mathscr{G}_S$) is a set of identical generators, which can be represented in a UC model using the aggregation techniques described in this paper. Further, notice that $\{\mathcal{G}_{\not\sim}\} \cup \mathscr{G}_F \cup \mathscr{G}_S$ is a partition of the set of generators $\mathcal{G}$. We represent each $g \in \mathcal{G}_{\not\sim}$ using the traditional 3-bin formulation, each $\mathcal{K} \in \mathscr{G}_F$ using the aggregated 3-bin formulation, and each $\mathcal{K} \in \mathscr{G}_S$ using the aggregated extended formulation.

For all the data used in this paper, $\mathbf{SD}^g = \mathbf{SU}^g$, so without loss of generality assume there exists $\hat{l}^g \in \mathcal{L}^g$ such that $\overline{\mathbf{P}}^{\hat{l}^g,g} = \mathbf{SU}^g = \mathbf{SD}^g$ for each $g \in \mathcal{G}$ (the formulation for the other cases is found in [82]). As above, let $\mathcal{G}_{\not\sim}^1 = \{g \in \mathcal{G}_{\not\sim} \mid \mathbf{UT}^g = 1\}$ and $\mathcal{G}_{\not\sim}^{>1} = \{g \in \mathcal{G}_{\not\sim} \mid \mathbf{UT}^g > 1\}$.

Similarly let $\mathscr{G}_F^1 = \{\mathcal{K} \in \mathscr{G}_F \mid \mathbf{UT}^{\mathcal{K}} = 1\}$ and $\mathscr{G}_F^{>1} = \{\mathcal{K} \in \mathscr{G}_F \mid \mathbf{UT}^{\mathcal{K}} > 1\}$. We use capital letters to represent aggregated variables. The resulting formulation is as follows.

$$\begin{aligned}\min \sum_{g\in\mathcal{G}}\sum_{t\in\mathcal{T}}\Bigg(&\sum_{l\in\mathcal{L}^g}(\mathbf{c}^{l,g}p_t^{l,g})+\mathbf{c}^{R,g}u_t^g+\mathbf{c}^{S,g}v_t^g+\sum_{s=1}^{S^g-1}(\mathbf{c}^{s,g}-\mathbf{c}^{S,g})\left(\sum_{t'=t-\overline{\mathbf{T}}^{s,g}+1}^{t-\underline{\mathbf{T}}^{s,g}}x_{[t',t)}^g\right)\Bigg)\\
+\sum_{\mathcal{K}\in\mathscr{G}_F}\sum_{t\in\mathcal{T}}\Bigg(&\sum_{l\in\mathcal{L}^{\mathcal{K}}}(\mathbf{c}^{l,\mathcal{K}}P_t^{l,\mathcal{K}})+\mathbf{c}^{R,\mathcal{K}}U_t^{\mathcal{K}}+\mathbf{c}^{S,\mathcal{K}}V_t^{\mathcal{K}}+\sum_{s=1}^{S^{\mathcal{K}}-1}(\mathbf{c}^{s,\mathcal{K}}-\mathbf{c}^{S,\mathcal{K}})\left(\sum_{t'=t-\overline{\mathbf{T}}^{s,\mathcal{K}}+1}^{t-\underline{\mathbf{T}}^{s,\mathcal{K}}}X_{[t',t)}^{\mathcal{K}}\right)\Bigg)\\
+\sum_{\mathcal{K}\in\mathscr{G}_S}\sum_{t\in\mathcal{T}}\Bigg(&\sum_{l\in\mathcal{L}^{\mathcal{K}}}(\mathbf{c}^{l,\mathcal{K}}P_t^{l,\mathcal{K}})+\mathbf{c}^{R,\mathcal{K}}U_t^{\mathcal{K}}+\sum_{s=1}^{S^{\mathcal{K}}}\mathbf{c}^{s,\mathcal{K}}\left(\sum_{t'=t-\overline{\mathbf{T}}^{s,\mathcal{K}}+1}^{t-\underline{\mathbf{T}}^{s,\mathcal{K}}}X_{[t',t)}^{\mathcal{K}}\right)\Bigg)\end{aligned}\tag{F.33}$$

subject to:

$$\sum_{g\in\mathcal{G}}(p_t^g+\underline{\mathbf{P}}^g u_t^g)+\sum_{\mathcal{K}\in\mathscr{G}_F}\left(P_t^{\mathcal{K}}+\underline{\mathbf{P}}^{\mathcal{K}}U_t^{\mathcal{K}}\right)+\sum_{\mathcal{K}\in\mathscr{G}_S}\left(P_t^{\mathcal{K}}+\underline{\mathbf{P}}^{\mathcal{K}}U_t^{\mathcal{K}}\right)=\mathbf{D}_t \qquad \forall t\in\mathcal{T} \tag{F.34a}$$

$$\sum_{g\in\mathcal{G}}r_t^g+\sum_{\mathcal{K}\in\mathscr{G}_F}R_t^{\mathcal{K}}+\sum_{\mathcal{K}\in\mathscr{G}_S}R_t^{\mathcal{K}}\geq\mathbf{R}_t \qquad \forall t\in\mathcal{T} \tag{F.34b}$$

$$p_t^g+r_t^g\leq(\overline{\mathbf{P}}^g-\underline{\mathbf{P}}^g)u_t^g-(\overline{\mathbf{P}}^g-\mathbf{SU}^g)v_t^g \qquad \forall t\in\mathcal{T},\forall g\in\mathcal{G}_\gamma^1 \tag{F.35a}$$

$$p_t^g+r_t^g\leq(\overline{\mathbf{P}}^g-\underline{\mathbf{P}}^g)u_t^g-(\overline{\mathbf{P}}^g-\mathbf{SD}^g)w_{t+1}^g \qquad \forall t\in\mathcal{T},\forall g\in\mathcal{G}_\gamma^1 \tag{F.35b}$$

$$p_t^g+r_t^g\leq(\overline{\mathbf{P}}^g-\underline{\mathbf{P}}^g)u_t^g-(\overline{\mathbf{P}}^g-\mathbf{SU}^g)v_t^g-(\overline{\mathbf{P}}^g-\mathbf{SD}^g)w_{t+1}^g \qquad \forall t\in\mathcal{T},\forall g\in\mathcal{G}_\gamma^{>1} \tag{F.35c}$$

$$p_t^g+r_t^g-p_{t-1}^g\leq\mathbf{RU}^g \qquad \forall t\in\mathcal{T},\forall g\in\mathcal{G}_\gamma \tag{F.35d}$$

$$p_{t-1}^g-p_t^g\leq\mathbf{RD}^g \qquad \forall t\in\mathcal{T},\forall g\in\mathcal{G}_\gamma \tag{F.35e}$$

$$p_t^g=\sum_{l\in\mathcal{L}^g}p_t^{l,g} \qquad \forall t\in\mathcal{T},\forall g\in\mathcal{G}_\gamma \tag{F.35f}$$

$$p_t^{l,g}\leq(\overline{\mathbf{P}}^{l,g}-\overline{\mathbf{P}}^{l-1,g}) \qquad \forall t\in\mathcal{T},\forall l\in\mathcal{L}^g,\forall g\in\mathcal{G}_\gamma \tag{F.35g}$$

$$u_t^g-u_{t-1}^g=v_t^g-w_t^g \qquad \forall t\in\mathcal{T},\forall g\in\mathcal{G}_\gamma \tag{F.35h}$$

$$\sum_{i=t-\mathbf{UT}^g+1}^{t}v_i^g\leq u_t^g \qquad \forall t\in[\mathbf{UT}^g,\mathbf{T}],\forall g\in\mathcal{G}_\gamma \tag{F.35i}$$

$$\sum_{i=t-\mathbf{DT}^g+1}^{t}w_i^g\leq 1-u_t^g \qquad \forall t\in[\mathbf{DT}^g,\mathbf{T}],\forall g\in\mathcal{G}_\gamma \tag{F.35j}$$

$$\sum_{t'=t-\mathbf{TC}^g+1}^{t-\mathbf{DT}^g}x_{[t',t)}^g\leq v_t^g \qquad \forall t\in\mathcal{T},\forall g\in\mathcal{G}_\gamma \tag{F.35k}$$

$$\sum_{t'=t+\mathbf{DT}^g}^{t+\mathbf{TC}^g-1} x^g_{[t,t')} \leq w^g_t \qquad \forall t\in\mathcal{T},\forall g\in\mathcal{G}_{\not\sim} \qquad \text{(F.35l)}$$

$$p^{l,g}_t \in \mathbb{R}_+ \qquad \forall t\in\mathcal{T},\forall l\in\mathcal{L}^g,\forall g\in\mathcal{G}_{\not\sim} \qquad \text{(F.35m)}$$

$$p^g_t, r^g_t \in \mathbb{R}_+ \qquad \forall t\in\mathcal{T},\forall g\in\mathcal{G}_{\not\sim} \qquad \text{(F.35n)}$$

$$u^g_t, v^g_t, w^g_t \in \{0,1\} \qquad \forall t\in\mathcal{T},\forall g\in\mathcal{G}_{\not\sim} \qquad \text{(F.35o)}$$

$$x^g_{[t,t')} \in \{0,1\} \qquad \forall [t,t')\in\mathcal{X}^g,\forall g\in\mathcal{G}_{\not\sim} \qquad \text{(F.35p)}$$

$$P^{\mathcal{K}}_t + R^{\mathcal{K}}_t \leq (\overline{\mathbf{P}}^{\mathcal{K}} - \underline{\mathbf{P}}^{\mathcal{K}})U^{\mathcal{K}}_t - (\overline{\mathbf{P}}^{\mathcal{K}} - \mathbf{SU}^{\mathcal{K}})V^{\mathcal{K}}_t \qquad \forall t\in\mathcal{T},\forall g\in\mathscr{G}^1_F \qquad \text{(F.36a)}$$

$$P^{\mathcal{K}}_t + R^{\mathcal{K}}_t \leq (\overline{\mathbf{P}}^{\mathcal{K}} - \underline{\mathbf{P}}^{\mathcal{K}})U^{\mathcal{K}}_t - (\overline{\mathbf{P}}^{\mathcal{K}} - \mathbf{SD}^{\mathcal{K}})W^{\mathcal{K}}_{t+1} \qquad \forall t\in\mathcal{T},\forall \mathcal{K}\in\mathscr{G}^1_F \qquad \text{(F.36b)}$$

$$P^{\mathcal{K}}_t + R^{\mathcal{K}}_t \leq (\overline{\mathbf{P}}^{\mathcal{K}} - \underline{\mathbf{P}}^{\mathcal{K}})U^{\mathcal{K}}_t - (\overline{\mathbf{P}}^{\mathcal{K}} - \mathbf{SU}^{\mathcal{K}})V^{\mathcal{K}}_t - (\overline{\mathbf{P}}^{\mathcal{K}} - \mathbf{SD}^{\mathcal{K}})W^{\mathcal{K}}_{t+1} \qquad \forall t\in\mathcal{T},\forall \mathcal{K}\in\mathscr{G}^{\geq 1}_F \qquad \text{(F.36c)}$$

$$P^{\mathcal{K}}_t = \sum_{l\in\mathcal{L}^{\mathcal{K}}} P^{l,\mathcal{K}}_t \qquad \forall t\in\mathcal{T},\forall \mathcal{K}\in\mathscr{G}_F \qquad \text{(F.36d)}$$

$$P^{l,\mathcal{K}}_t \leq (\overline{\mathbf{P}}^{l,\mathcal{K}} - \overline{\mathbf{P}}^{l-1,\mathcal{K}})U^{\mathcal{K}}_t \qquad \forall t\in\mathcal{T},\forall l\leq \hat{l}^{\mathcal{K}},\forall \mathcal{K}\in\mathscr{G}_F \qquad \text{(F.36e)}$$

$$P^{l,\mathcal{K}}_t \leq (\overline{\mathbf{P}}^{l,\mathcal{K}} - \overline{\mathbf{P}}^{l-1,\mathcal{K}}) \cdot (U^{\mathcal{K}}_t - V^{\mathcal{K}}_t) \qquad \forall t\in\mathcal{T},\forall l> \hat{l}^{\mathcal{K}},\forall \mathcal{K}\in\mathscr{G}^1_F \qquad \text{(F.36f)}$$

$$P^{l,\mathcal{K}}_t \leq (\overline{\mathbf{P}}^{l,\mathcal{K}} - \overline{\mathbf{P}}^{l-1,\mathcal{K}}) \cdot (U^{\mathcal{K}}_t - W^{\mathcal{K}}_{t+1}) \qquad \forall t\in\mathcal{T},\forall l> \hat{l}^{\mathcal{K}},\forall \mathcal{K}\in\mathscr{G}^1_F \qquad \text{(F.36g)}$$

$$P^{l,\mathcal{K}}_t \leq (\overline{\mathbf{P}}^{l,\mathcal{K}} - \overline{\mathbf{P}}^{l-1,\mathcal{K}}) \cdot (U^{\mathcal{K}}_t - V^{\mathcal{K}}_t - W^{\mathcal{K}}_{t+1}) \qquad \forall t\in\mathcal{T},\forall l> \hat{l}^{\mathcal{K}},\forall \mathcal{K}\in\mathscr{G}^{\geq 1}_F \qquad \text{(F.36h)}$$

$$U^{\mathcal{K}}_t - U^{\mathcal{K}}_{t-1} = V^{\mathcal{K}}_t - W^{\mathcal{K}}_t \qquad \forall t\in\mathcal{T},\forall \mathcal{K}\in\mathscr{G}_F \qquad \text{(F.36i)}$$

$$\sum_{i=t-\mathbf{UT}^{\mathcal{K}}+1}^{t} V^{\mathcal{K}}_i \leq U^{\mathcal{K}}_t \qquad \forall t\in[\mathbf{UT}^{\mathcal{K}},\mathbf{T}],\forall \mathcal{K}\in\mathscr{G}_F \qquad \text{(F.36j)}$$

$$\sum_{i=t-\mathbf{DT}^{\mathcal{K}}+1}^{t} W^{\mathcal{K}}_i \leq 1 - U^{\mathcal{K}}_t \qquad \forall t\in[\mathbf{DT}^{\mathcal{K}},\mathbf{T}],\forall \mathcal{K}\in\mathscr{G}_F \qquad \text{(F.36k)}$$

$$\sum_{t'=t-\mathbf{TC}^{\mathcal{K}}+1}^{t-\mathbf{DT}^{\mathcal{K}}} X^{\mathcal{K}}_{[t',t)} \leq V^{\mathcal{K}}_t \qquad \forall t\in\mathcal{T},\forall \mathcal{K}\in\mathscr{G}_F \qquad \text{(F.36l)}$$

$$\sum_{t'=t+\mathbf{DT}^{\mathcal{K}}}^{t+\mathbf{TC}^{\mathcal{K}}-1} X^{\mathcal{K}}_{[t,t')} \leq W^{\mathcal{K}}_t \qquad \forall t\in\mathcal{T},\forall \mathcal{K}\in\mathscr{G}_F \qquad \text{(F.36m)}$$

$$P^{l,\mathcal{K}}_t \in \mathbb{R}_+ \qquad \forall t\in\mathcal{T},\forall l\in\mathcal{L}^{\mathcal{K}},\forall \mathcal{K}\in\mathscr{G}_F \qquad \text{(F.36n)}$$

$$P_t^{\mathcal{K}},R_t^{\mathcal{K}}\in\mathbb{R}_+ \qquad \forall t\in\mathcal{T},\forall\mathcal{K}\in\mathscr{G}_F \qquad \text{(F.36o)}$$

$$U_t^{\mathcal{K}},V_t^{\mathcal{K}},W_t^{\mathcal{K}}\in\{0,...,|\mathcal{K}|\} \qquad \forall t\in\mathcal{T},\forall\mathcal{K}\in\mathscr{G}_F \qquad \text{(F.36p)}$$

$$X_{[t,t')}^{\mathcal{K}}\in\{0,...,|\mathcal{K}|\} \qquad \forall[t,t')\in\mathcal{X}^{\mathcal{K}},\forall\mathcal{K}\in\mathscr{G}_F \qquad \text{(F.36q)}$$

$$P_t^{[a,b),\mathcal{K}}+R_t^{[a,b),\mathcal{K}}\leq(\overline{\mathbf{P}}^{\mathcal{K}}-\underline{\mathbf{P}}^{\mathcal{K}})Y_{[a,b)}^{\mathcal{K}} \qquad \forall t\in[a,b),\forall[a,b)\in\mathcal{Y}^{\mathcal{K}},\forall\mathcal{K}\in\mathscr{G}_S \qquad \text{(F.37a)}$$

$$P_a^{[a,b),\mathcal{K}}+R_a^{[a,b),\mathcal{K}}\leq(\mathbf{SU}^{\mathcal{K}}-\underline{\mathbf{P}}^{\mathcal{K}})Y_{[a,b)}^{\mathcal{K}} \qquad \forall[a,b)\in\mathcal{Y}^{\mathcal{K}},\forall\mathcal{K}\in\mathscr{G}_S \qquad \text{(F.37b)}$$

$$P_{b-1}^{[a,b),\mathcal{K}}+R_{b-1}^{[a,b),\mathcal{K}}\leq(\mathbf{SD}^{\mathcal{K}}-\underline{\mathbf{P}}^{\mathcal{K}})Y_{[a,b)}^{\mathcal{K}} \qquad \forall[a,b)\in\mathcal{Y}^{\mathcal{K}},\forall\mathcal{K}\in\mathscr{G}_S \qquad \text{(F.37c)}$$

$$P_t^{[a,b),\mathcal{K}}+R_t^{[a,b),\mathcal{K}}-P_{t-1}^{[a,b),\mathcal{K}}\leq\mathbf{RU}^{\mathcal{K}}Y_{[a,b)}^{\mathcal{K}} \qquad \forall t\in(a,b),\forall[a,b)\in\mathcal{Y}^{\mathcal{K}},\forall\mathcal{K}\in\mathscr{G}_S \qquad \text{(F.37d)}$$

$$P_{t-1}^{[a,b),\mathcal{K}}-P_t^{[a,b),\mathcal{K}}\leq\mathbf{RD}^{\mathcal{K}}Y_{[a,b)}^{\mathcal{K}} \qquad \forall t\in(a,b),\forall[a,b)\in\mathcal{Y}^{\mathcal{K}},\forall\mathcal{K}\in\mathscr{G}_S \qquad \text{(F.37e)}$$

$$P_t^{[a,b),l,\mathcal{K}}\leq(\overline{\mathbf{P}}^{l,g}-\overline{\mathbf{P}}^{l-1,g})Y_{[a,b)}^{\mathcal{K}} \qquad \forall t\in[a,b),\forall[a,b)\in\mathcal{Y}^{\mathcal{K}},\forall l\in\mathcal{L}^{\mathcal{K}},\forall\mathcal{K}\in\mathscr{G}_S \qquad \text{(F.37f)}$$

$$\sum_{\{[a,b)\in\mathcal{Y}^{\mathcal{K}}|t\in[a,b)\}}Y_t^{[a,b),\mathcal{K}}=U_t^{\mathcal{K}} \qquad \forall t\in\mathcal{T},\forall\mathcal{K}\in\mathscr{G}_S \qquad \text{(F.37g)}$$

$$\sum_{\{[a,b)\in\mathcal{Y}^{\mathcal{K}}|t\in[a,b)\}}P_t^{[a,b),\mathcal{K}}=P_t^{\mathcal{K}} \qquad \forall t\in\mathcal{T},\forall\mathcal{K}\in\mathscr{G}_S \qquad \text{(F.37h)}$$

$$\sum_{\{[a,b)\in\mathcal{Y}^{\mathcal{K}}|t\in[a,b)\}}R_t^{[a,b),\mathcal{K}}=R_t^{\mathcal{K}} \qquad \forall t\in\mathcal{T},\forall\mathcal{K}\in\mathscr{G}_S \qquad \text{(F.37i)}$$

$$\sum_{\{[a,b)\in\mathcal{Y}^{\mathcal{K}}|t\in[a,b)\}}P_t^{[a,b),l,\mathcal{K}}=P_t^{l,\mathcal{K}} \qquad \forall t\in\mathcal{T},\forall l\in\mathcal{L}^{\mathcal{K}},\forall\mathcal{K}\in\mathscr{G}_S \qquad \text{(F.37j)}$$

$$\sum_{\{[c,d)\in\mathcal{X}^{\mathcal{K}}|t=d\}}X_{[c,d)}^{\mathcal{K}}=\sum_{\{[a,b)\in\mathcal{Y}^{\mathcal{K}}|t=a\}}Y_{[a,b)}^{\mathcal{K}} \qquad \forall t\in\mathcal{T},\forall\mathcal{K}\in\mathscr{G}_S \qquad \text{(F.37k)}$$

$$\sum_{\{[a,b)\in\mathcal{Y}^{\mathcal{K}}|t=b\}}Y_{[a,b)}^{\mathcal{K}}=\sum_{\{[c,d)\in\mathcal{X}^{\mathcal{K}}|t=c\}}X_{[c,d)}^{\mathcal{K}} \qquad \forall t\in\mathcal{T},\forall\mathcal{K}\in\mathscr{G}_S \qquad \text{(F.37l)}$$

$$\sum_{\{[a,b)\in\mathcal{Y}^{\mathcal{K}}|a\leq0\}}Y_{[a,b)}^{\mathcal{K}}+\sum_{\{[c,d)\in\mathcal{X}^{\mathcal{K}}|c\leq0\}}X_{[c,d)}^{\mathcal{K}}=|\mathcal{K}| \qquad \forall\mathcal{K}\in\mathscr{G}_S \qquad \text{(F.37m)}$$

$$\sum_{\{[a,b)\in\mathcal{Y}^{\mathcal{K}}|b>\mathbf{T}\}}Y_{[a,b)}^{\mathcal{K}}+\sum_{\{[c,d)\in\mathcal{X}^{\mathcal{K}}|d>\mathbf{T}\}}X_{[c,d)}^{\mathcal{K}}=|\mathcal{K}| \qquad \forall\mathcal{K}\in\mathscr{G}_S \qquad \text{(F.37n)}$$

$$P_t^{[a,b),l,\mathcal{K}}\in\mathbb{R}_+ \qquad \forall t\in[a,b),\forall[a,b)\in\mathcal{Y}^{\mathcal{K}},\forall l\in\mathcal{L}^{\mathcal{K}},\forall\mathcal{K}\in\mathscr{G}_S \qquad \text{(F.37o)}$$

$$P_t^{[a,b),\mathcal{K}},R_t^{[a,b),\mathcal{K}}\in\mathbb{R}_+ \qquad \forall t\in[a,b),\forall[a,b)\in\mathcal{Y}^{\mathcal{K}},\forall\mathcal{K}\in\mathscr{G}_S \qquad \text{(F.37p)}$$

$$Y_{[a,b)}^{\mathcal{K}}\in\{0,...,|\mathcal{K}|\} \qquad \forall[a,b)\in\mathcal{Y}^{\mathcal{K}},\forall\mathcal{K}\in\mathscr{G}_S \qquad \text{(F.37q)}$$

$$X_{[c,d)}^{\mathcal{K}}\in\{0,...,|\mathcal{K}|\} \qquad \forall[c,d)\in\mathcal{X}^{\mathcal{K}},\forall\mathcal{K}\in\mathscr{G}_S. \qquad \text{(F.37r)}$$

In total, we have that (F.33) is the objective function, Constraints (F.34) describe the demand and reserve requirement for the system, Constraints (F.35) describe the technical constraints for the generators that could not be aggregated, Constraints (F.36) describe the technical constraints for the fast-ramping generators that were aggregated, and Constraints (F.37) describe the technical constraints for the slow-ramping generators that could be aggregated. Variables not appearing in the above model are assumed to be 0.

Appendix G

Additional Computational Tests for Chapter 5

In this appendix we present some additional computational results to complement those in Chapter 5. The notation is that of Appendix F.

A Symmetry Breaking Inequalities

In addition to the formulations considered before, we consider the addition of the "S3" variables and inequalities from [87] to the base 3-bin UC formulation. Lima and Novais [87] propose introducing new variables yon^g which indicate if generator g ever turned on during the time horizon, along with inequalities to enforce this

$$\sum_{t \in \mathcal{T}} u_t^g \geq yon^g \qquad \forall g \in \mathcal{G} \tag{G.1}$$

$$u_t^g \leq yon^g \qquad \forall t \in \mathcal{T},\ \forall g \in \mathcal{G}. \tag{G.2}$$

They then propose the following two symmetry-breaking inequalities for each set of identical generators $\mathcal{K}$

$$yon^g \geq yon^{g+1} \qquad \forall g, g{+}1 \in \mathcal{K} \tag{G.3}$$

$$\sum_{t \in \mathcal{T}} u_t^g \geq \sum_{t \in \mathcal{T}} u_t^{g+1} \qquad \forall g, g{+}1 \in \mathcal{K}, \tag{G.4}$$

Table G.1: Additional Computational Results for CAISO UC Instances

	Time (s)			Nodes		
Instance	**3-bin**	**3-bin+SBC**	**3-bin+A**	**3-bin**	**3-bin+SBC**	**3-bin+A**
2014-09-01 0%	31.35	34.07	**14.25**	0	0	0
2014-12-01 0%	25.77	29.36	**12.38**	0	0	0
2015-03-01 0%	24.08	34.47	**14.27**	0	0	0
2015-06-01 0%	13.11	14.25	**8.50**	0	0	0
Scenario400 0%	27.29	35.99	**23.63**	0	0	0
2014-09-01 1%	20.52	29.38	**16.44**	0	0	0
2014-12-01 1%	38.48	86.41	**24.69**	95	566	**0**
2015-03-01 1%	21.75	35.76	**19.11**	0	0	0
2015-06-01 1%	39.87	30.42	**15.59**	47	**0**	**0**
Scenario400 1%	47.54	57.85	**44.63**	**0**	154	1438
2014-09-01 3%	81.47	92.60	**38.27**	**7**	696	122
2014-12-01 3%	65.01	120.03	**36.53**	1292	**95**	125
2015-03-01 3%	50.79	77.96	**25.04**	**0**	3	**0**
2015-06-01 3%	87.25	147.05	**41.23**	**0**	79	115
Scenario400 3%	131.28	147.61	**69.45**	2055	**140**	880
2014-09-01 5%	47.07	69.51	**30.95**	95	176	**7**
2014-12-01 5%	83.87	132.97	**66.90**	1203	**79**	3978
2015-03-01 5%	80.57	72.70	**21.65**	923	**0**	**0**
2015-06-01 5%	**26.99**	96.56	43.79	**0**	162	402
Scenario400 5%	115.53	**95.50**	118.51	3867	**31**	4225
Geometric Mean:	43.85	59.69	**27.55**			

where $g, g+1 \in \mathcal{K}$ is understood to be two consecutive generators in $\mathcal{K}$. (G.3) enforces that if generator $g+1$ ever turns on then generator g does as well, and (G.4) enforces that generator g is scheduled in at least as many time periods as generator $g+1$. As pointed out in [87], this eliminates many, but not every, source of symmetry in UC.

B Computational Results

Here we present computational results for the instances tested in the main text with the addition of the "S3" variables and inequalities from [87], which we label as "3-bin+SBC". The computational platform is as described in the main text.

Table G.2: Additional Computational Results for Ostrowski UC Instances

	Time (s)			Nodes		
Instance	**3-bin**	**3-bin +SBC**	**EF/3-bin +A**	**3-bin**	**3-bin +SBC**	**EF/3-bin +A**
1	**8.44**	54.82	14.02	1509	13700	**68**
2	154.75	44.73	**21.07**	48129	10370	**157**
3	703.94	127.23	**100.33**	316704	25802	**4464**
4	**14.84**	109.15	17.28	8532	26297	**60**
5	143.18	101.73	**57.22**	131320	17784	**4350**
6	95.41	45.03	**28.00**	62394	7160	**72**
7	(0.0238%)	270.34	**119.22**	535361*	75097	**4854**
8	(0.0107%)	**57.36**	71.00	1378310*	29362	**9267**
9	(0.0169%)	167.56	**125.63**	819798*	49318	**12217**
10	(0.0327%)	287.18	**82.89**	751319*	133909	**11549**
11	(0.0186%)	(0.0220%)	**18.76**	73976*	20838*	**1155**
12	(0.0240%)	(0.0265%)	**22.91**	42729*	7505*	**460**
13	(0.0266%)	(0.0264%)	**74.43**	41325*	13420*	**6464**
14	(0.0144%)	(0.0203%)	**19.75**	41469*	4127*	**15**
15	780.76	(0.0105%)	**39.63**	120599	35111*	**3091**
16	(0.0162%)	(0.0223%)	**90.31**	42102*	8770*	**2597**
17	154.37	237.90	**27.88**	2114	2185	**1059**
18	(0.0121%)	(0.0214%)	**22.36**	60651*	3972*	**151**
19	(0.0195%)	(0.0250%)	**21.30**	42683*	6025*	**2436**
20	106.44	628.20	**18.46**	527	4288	**0**
Geometric Mean:	>349.77	>277.82	**38.03**			

B.1 CAISO Instances

In Table G.1 we report the computational results for the CAISO instances described above. As we can see, in almost all cases the symmetry-breaking constraints are unhelpful. This is to be expected since these instances have a relatively tight root optimality gap, and the extra constraints serve to slow effective cut generation and heuristic search at the root node. Overall they serve to slow the solver down over 3-bin, though in one instance the 3-bin+SBC variant finds a high-quality solution fastest and with fewest nodes.

B.2 Ostrowski Instances

As in the main text, we set a time limit of 900 seconds for the Ostrowski instances and report the terminating MIP gap in parentheses when the solver terminates at the time limit. In Table G.2 we

report the computational results for the Ostrowski instances from the main text. As reported in [87], the symmetry-breaking constraints are helpful overall for the smaller Ostrowski instances (1–10), but they perform worse than 3-bin on the larger instances (11–20). In comparison, EF/3-bin+A has a relatively flat performance profile across all 20 instances, suggesting that, when handled properly, identical generators can be leveraged to significantly reduce the computational burden of UC.

Vita

Bernard (Ben) Albert Knueven was born on June 17th, 1988, to Leo and Mary Jo Knueven. He attended Colerain High School in Cincinnati, Ohio. Upon graduation, he enrolled at Northern Kentucky University, and in the spring of 2010 he completed undergraduate degrees in music and mathematics. Ben then accepted a graduate teaching assistantship at Miami University in Oxford, Ohio, where he studied mathematics. He earned a master's degree in mathematics in the summer of 2012. Ben remained at Miami University the following school year as an instructor, teaching courses in pre-calculus and calculus.

In the fall of 2013 Ben 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. In the summer of 2015 Ben was selected to receive an Office of Science Graduate Student Research award from the U.S. Department of Energy, Office of Science. As a result, he spent seven months spanning 2015 and 2016 in Albuquerque, New Mexico, where he worked in the Discrete Math & Optimization Department at Sandia National Laboratories under the tutelage of Jean-Paul Watson. He continued his work at Sandia as student intern remotely from Knoxville. Ben completed his PhD in Industrial Engineering in December 2017.