Symmetry in Scheduling Problems

James Ostrowski[1], Miguel F. Anjos[2], and Anthony Vannelli[3]

[1]Decision and Information Sciences, Argonne National Lab, Argonne, IL, jostrowski@anl.gov

[2] GERAD & Département de mathématiques et de génie industriel, École Polytechnique de Montréal,, Montreal, QB, anjos@stanfordalumni.org

[3]College of Physical & Engineering Science, University of Guelph, Guelph, ON , vannelli@uoguelph.ca

November 16, 2010

Abstract

The presence of symmetry is common in certain types of scheduling problems. Symmetry can occur when one is scheduling a collection of jobs on multiple identical machines, or if one is determining production schedules for identical machines. General symmetry-breaking methods can be strengthened by taking advantage of the special structure of the symmetry group in scheduling problems. In this paper, we examine the effectiveness of symmetry-breaking methods for scheduling problems. We also present a modified version of orbital branching, a powerful symmetry-breaking procedure, and discuss when it should and should not be used in practice. Using operating room and power generator scheduling problems as sample problems, we will provide computational results comparing different methods of symmetry breaking.

1 Introduction

Symmetry has been an obstacle in mixed integer linear programming (MILP) for more than 40 years. In Jeroslow (1974), Jeroslow presented a class of problems with only one equality constraint on n variables for which branch-and-bound trees contain an exponential number of nodes if symmetry is not removed from the problem. Around the same time period Fulkerson et al. (1973) suggested that a class of highly symmetric covering problems called Steiner Triple Systems (STS) be included in test libraries because they were notoriously difficult, especially considering the small number of variables in the problem. The STS class of problems gained attention when some instances were included in the first MIPLIB library (Bixby et al. 1992). In the past decade, effective symmetry breaking techniques have been developed for MILP problems. Symmetry breaking methods such as isomorphism pruning (Margot 2002) are able to remove all symmetries from the branch-and-bound tree. While we know ways to break symmetry, we do not yet have a clear understanding of the most effective way to break symmetry in particular contexts, and more importantly, how symmetry-breaking methods interact with other MILP features such as branching strategies and cutting plane methods.

To gain some insight on how symmetry breaking affects MILP techniques, we examine symmetry breaking in scheduling problems. Scheduling problems cover a very broad class of problems with important real world applications. The structure of symmetry present in these problems allow for efficient symmetry breaking techniques. It is this structure, also found in bin-packing and graph coloring problems, that makes them ideal candidates for our study.

In this paper we will show that the success of symmetry-breaking techniques in reducing computation time is highly dependent on the branching strategy. If good branching strategies are known a priori, symmetry breaking constraints that augment the branching strategy can be added to the problem with great result. However, if branching strategies are not known, adding symmetry-breaking constraints to the problem formulation may not be the best option. In this case, orbital branching- a symmetry breaking method that removes symmetry *during* the branch-and-bound process, is shown to be most effective.

This paper is divided as follows. Section 2 gives an introduction to symmetry in integer programming. Section 3 uses the operating room scheduling problems to test three different methods of exploiting symmetry in scheduling problems: symmetry-removing constraints, orbitopal fixing, and orbital branching. In Section 4, a version of orbital branching that takes advantage of the special structure of machine-scheduling problems is presented. It is tested against the original version of orbital branching using the unit commitment problem, an important problem in power generation.

2 Problem Symmetry and Fundamental Domains

Typical formulations of scheduling problems contain binary variables $x_{i,j}$, where $x_{i,j}$ equal to one signifies that job i is assigned to machine j, or that machine j is in operation at time i. The x variables can be interpreted as an m by n 0/1 matrix. Symmetry is often present in scheduling problems since there can be many identical machines of a certain type. As a result, given any feasible solution x, equivalent solutions can be generated by permuting the columns of x. For example, suppose one is assigning 5 jobs to 4 identical machines. A possible solution is

$$x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{1}$$

Because all machines are identical, it is possible to create a new schedule by moving to machine 3 all jobs that were assigned to machine 4 and vice versa. This gives a new but equivalent solution

$$x' = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2}$$

Because the machines are identical, any permutation of the columns of x yields a different solution, but all these solutions can have the same real-world interpretation as the original solution.

The presence of symmetry can have a significant negative effect on the performance of branch-and-bound algorithms. In the same way that it allows multiple equivalent solutions, symmetry also allows different subproblems in the branch-and-bound tree to be equivalent. Failure to recognize the equivalence of these subproblems can lead to solving thousands of unnecessary subproblems and making relatively easy problems impossible to solve using branch-and-bound techniques. In order to prevent this from happening, equivalent solutions and subproblems must be identified and removed from the search.

2.1 Preliminary Algebra

Let I^n be the ground set $\{1, \ldots, n\}$ representing the columns of x. Let S^n be the collection of permutations of the ground set. For a permutation $\pi \in S^n$, the matrix $\pi(x)$ is defined by $\pi(x)_{i,j} = x_{i,\pi(j)}$. For any two permutations π and σ in S^n, $\pi(\sigma(x))$ is formed by first permuting the columns of x by σ, then permuting them by π.

The *symmetry group* $\mathcal{G}$ of an MILP instance is the collection of permutations in S^n that map every feasible solution to another feasible solution with the same objective value (Margot 2008). Formally,

$$\mathcal{G} \stackrel{\text{def}}{=} \{\pi \subseteq S^n \mid \forall x \in \mathcal{F},\ \pi(x) \in \mathcal{F} \text{ and } c^\top \pi(x) = c^\top x\}, \tag{3}$$

where $\mathcal{F}$ is the feasible region of the MILP.

As variables become fixed via branching or other considerations, the symmetry present in the columns may disappear. Consider the subproblem $a = (F_1^a, F_0^a)$ formed by fixing $x_{i,j}$ to one for every (i,j) in F_1^a and fixing $x_{k,l}$ to zero for every (k,l) in F_0^a. The symmetry group of subproblem a, $\mathcal{G}^a$, contains all permutations of the columns that map each feasible solution of a to another feasible solution of a with an identical objective value. Suppose $x_{i,j}$ is fixed to one in subproblem a. If permutation π is in $\mathcal{G}^a$ then $x_{i,\pi(j)}$ must be equal to one (because if x is a feasible solution at a then $\pi(x)$ must also be feasible at a for any $\pi \in \mathcal{G}^a$). Unfortunately, for an arbitrary π, there may not be enough information present at node a to easily determine if $x'_{i,\pi(j)}$ can be fixed to one. Rather than using subproblem a's symmetry group $\mathcal{G}^a$, we restrict ourselves to using permutations that map variables fixed to one *only* to other variables fixed to one, and variables fixed to zero to other variables fixed to zero.

The *stabilizer* of a set S is the subset of S^n that maps elements of S only to other elements of S:

$$\text{stab}(S) = \{\pi \in S^n \mid \pi(S) = S\}. \tag{4}$$

The set of permutations that stabilize each set in the collection $\{S_1, S_2, \ldots, S_k\}$ is written as

$$\text{stab}(S_i, S_2, \ldots, S_k) = \bigcap_{j=1}^{k} \text{stab}(S_j). \tag{5}$$

The group $\text{stab}(F_1^a, F_0^a)$ is a subgroup of $\mathcal{G}^a$ and will be used to approximate the symmetry group at node a. A polynomial time algorithm for computing $\bigcap_{j=1}^{k} \text{stab}(\{i_j\})$ is given in Holt (2005), but no polynomial time algorithm is known for computing $\text{stab}(S)$.

Because $\pi(x)$ is generated by permuting the columns of x, it is called an *equivalent* or *symmetric* solution with respect to the symmetry group $\mathcal{G}$. The set of all such equivalent solutions to x with respect to a group $\mathcal{G} \subseteq S^n$ is called an *orbit* of x.

$$\text{orb}(x, \mathcal{G}) \stackrel{\text{def}}{=} \{x' \in \mathbb{R}^n \mid \exists \pi \in \mathcal{G} \text{ such that } x' = \pi(x)\} = \{\pi(x) \mid \pi \in \mathcal{G}\}. \tag{6}$$

In the same way as solutions, orbits can define equivalence in variables.

$$\text{orb}(x_{i,j}, \mathcal{G}) \stackrel{\text{def}}{=} \{x_{i,\pi(j)} \mid \pi \in \mathcal{G}\}. \tag{7}$$

By definition, if $x_{i,j} \in \text{orb}(\mathcal{G}, x_{i,k})$, then $x_{i,k} \in \text{orb}(\mathcal{G}, x_{i,j})$, i.e., the variables $x_{i,j}$ and $x_{i,k}$ share the same orbit. Therefore, the union of the orbits

$$\mathcal{O}(\mathcal{G}) \stackrel{\text{def}}{=} \bigcup_{i=1}^{M} \bigcup_{j=1}^{N} \text{orb}(x_{i,j}, \mathcal{G}) \tag{8}$$

forms a partition of the variables that is referred to as the orbital partition of $\mathcal{G}$, or simply the *orbits* of $\mathcal{G}$.

2.2 Using Symmetry to Reduce the Feasible Region

A popular strategy for solving MILPs is to strengthen the LP relaxation by adding valid inequalities that improve the lower bound. For a general MILP, inequalities are considered valid if and only if they do not remove any integer-feasible solution from the feasible region. However, this condition is more restrictive than necessary. An inequality that removes *some* integer-feasible solutions may be added to the problem formulation so long as one can guarantee that at least one optimal solution satisfies the inequality. In practice, guaranteeing that an optimal solution satisfies an inequality is difficult. Knowledge of the problem's symmetry group allows for the generation of constraints that remove some solutions from the feasible region, while also guaranteeing that at least one optimal solution remains feasible.

Formally, a set F is a fundamental domain of the set $\mathcal{S}$ with respect to the group Γ if and only if for every $x \in \mathcal{F}$, the set $\text{orb}(x, \Gamma) \cap F$ is nonempty. A fundamental domain is *minimal* if it does not contain a smaller fundamental domain.

Fundamental domains are used in the context of MILP to reduce the size of the feasible region. Let F be any fundamental domain of $\mathcal{F}_{MILP}$, the feasible region of the MILP, with respect to the symmetry group $\mathcal{G}$ of the MILP. By definition of the fundamental domain, F contains an optimal solution to the MILP as long as one exists. Thus, constraints that remove integer-feasible solutions from the feasible region may be added to the problem formulation so long as the resulting feasible region still remains a fundamental domain.

Example 2.1 *Consider the problem of assigning 3 unique jobs to 3 identical machines. Let $x_{i,j}$ equal one if job i is assigned to machine j. There are 27 feasible solutions to this problem. They are*

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \tag{9}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{10}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{11}$$

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \quad (12)$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \quad (13)$$

All solutions in each of (9), (10), (11), (12), *and* (13) *are equivalent. Any subset of the feasible region that contains at least one solution from each of these sets of equivalent solutions is a fundamental domain. It is minimal if the subset contains exactly one solution from each collection. For this example, there are over 100 million different fundamental domains, of which 3,888 are minimal.*

As shown in the example, there are many different fundamental domains. Two important issues are choosing a fundamental domain, and generating constraints that restrict the feasible region to the chosen fundamental domain. It is always possible to generate a minimal fundamental domain using lexicographic ordering, ensuring that the columns of x are lexicographically decreasing. For general scheduling problems, the following inequalities, along with the restriction that $x_{i,j}$ is binary, guarantee that the resulting solution has lexicographically decreasing columns:

$$\sum_{i=1}^{n} 2^{n-i} x_{i,j} \geq \sum_{i=1}^{n} 2^{n-1} x_{i,j+1} \ \forall j = 0, \ \ldots, \ n-1. \quad (14)$$

Though adding symmetry-breaking inequalities removes symmetry in the problem, it is not necessarily a good choice to use a fundamental domain based on lexicographic order. The purpose of the symmetry-breaking inequalities is to remove subproblems in the branch-and-bound tree that are equivalent. When using lexicographic order to determine the fundamental domain, branching on variables with large row indices will not lead to the fixing of variables as a result of symmetry considerations. Thus, branching strategies that tend to choose variables with large row indices for branching may not be effective at exploiting symmetry. This is explained in detail in the context of the job scheduling problem, Section 3.5. It will be shown that when using a fundamental domain based on lexicographic ordering, fixing variables with small row indices is necessary to break symmetry. Therefore, from a symmetry-breaking standpoint, branching on variables with small row indices early in the branch-and-bound process is advised. From an MILP standpoint, however, the variables with small row indices may not make good branching candidates, and may produce large branch-and-bound trees. How one resolves these two possibly competing issues can have a significant effect on the efficiency of the solver.

3 Job Scheduling Problems

This section presents methods that exploit symmetry in job scheduling problems that arises when jobs are assigned to similar machines or machines. The solutions of a job scheduling problem can be expressed as a 0/1 matrix x, where $x_{i,j} = 1$ if job i is assigned to machine j. Note that each row of x must contain exactly one 1. In this section, we assume that all machines are identical, meaning that the complete symmetric group acts on the columns of x. As we will briefly explain in Section 3.4, this assumption is not necessary for the methods discussed, but is made for notational convenience. In Sections 3.1, 3.2, and 3.3, we describe different methods from the literature used to exploit this column symmetry.

3.1 Symmetry-Removing Constraints

In (Denton et al. 2009), the authors attack symmetry job scheduling problems, specifically operating room scheduling problems, by adding symmetry-breaking constraints to the original formulation. These constraints restrict the feasible region to a minimal fundamental domain that has lexicographically-decreasing columns. For example, the 0/1 matrix

$$x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{15}$$

does not have lexicographically-decreasing columns, so it is not in the fundamental domain. Permuting columns 3 and 4 gives the matrix

$$x' = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{16}$$

The matrix x' does have lexicographically-decreasing columns, so it is in the fundamental domain.

It is clear that for any x with at least one 1 in the first row and lexicographically decreasing columns $x_{1,1}$ must be one. Similarly, if any $x_{i,j}$ with $i < j$ is equal to one, then x cannot have lexicographically-decreasing columns, so all such x terms can be fixed to zero.

These conditions do not suffice to induce a minimal fundamental domain. Consider again the matrix x from (15). All $x_{i,j}$ with $i < j$ are fixed to zero, but x still does not have lexicographically-decreasing columns. The problem in this case arises because job 4 is assigned to machine 4, but none of jobs 1 through 3 are assigned to machine 3. Because no lower-indexed job is assigned to machine 3, column 3 will be lexicographically smaller than column 4 regardless of where job 5 is assigned. The constraints

$$x_{i,j} \leq \sum_{u=1}^{i-1} x_{u,j-1} \quad \forall (i,j),\ i \geq j \geq 2 \tag{17}$$

guarantee that for any x_{ij} equal to one with i and j greater than one, there is at least one lower-indexed job assigned to machine $j-1$.

While the only 0/1 matrices that satisfy constraints (17) have lexicographically decreasing columns, the constraints can be strengthened to create a tighter LP relaxation. Using the property that each row of x must contain a single one, the constraints in (17) can be written as

$$\sum_{v=j}^{\min\{i,n\}} x_{i,v} \leq \sum_{u=1}^{i-1} x_{u,j-1} \quad \forall (i,j),\ i \geq j. \tag{18}$$

3.2 Orbitopal Fixing

Kaibel and Pfetsch (2008) give a complete linear description of the convex hull of all 0/1 job-scheduling matrices with linearly decreasing columns. It should be noted that the constraints described in Section 3.1 are facets of the polytope, but make up only part of the convex hull. The complete description requires exponentially many inequalities, but in a subsequent paper Kaibel et al. (2007) present a linear-time algorithm that fixes variables based on implications of the constraints in (18).

The first part of the orbitopal fixing algorithm attempts to find variables that can be fixed to zero. As mentioned earlier, it is clear that if any variable $x_{i,j}$ with $i < j$ equals one, then the

resulting solution is not lexicographical maximal. Similarly, if $x_{2,1}$ equals one, then $x_{3,3}$ cannot equal one, because neither $x_{1,2}$ nor $x_{2,2}$ equals one, so fixing $x_{3,3}$ to one would violate a constraint in (18).

In order to recognize which variables can be fixed to zero, they create an m-vector a that provides an upper bound on the largest indexed machine for each of the m jobs. For example, $a[i] = j$ if no job with an index less than or equal to i can be performed by a machine with index strictly greater than j. Trivially, $a[1]$ always equals one. If it is possible to assign job $i-1$ to machine $j-1$ without violating the inequalities in (18), then it will be possible to perform job i in room j, unless of course, $j-1 = n$, in which case there is no room j. It may also happen that $x_{i,j}$ may have been fixed to zero by a previous branching decision. If that were the case, then obviously job i cannot be performed in room j. Thus, element $a[i]$ is equal to $a[i-1]+1$ unless either $a[i-1] = n$ or $x_{i,a[i-1]+1}$ is fixed to zero, in which case $a[i] = a[i-1]$. Since $a[i]$ represents the upper bound on the machine that job i can be assigned to, all variables $x_{i,j}$ can be fixed to zero for $j > a[i]$. A formal description of the algorithm is described in Table 1.

Algorithm 1 Orbitopal Fixing: Zero Setting

Input: Subproblem $a = (F_1^a, F_0^a)$.
Output: Subproblem $a' = (F_1^a, F_0^{a*})$ with $F_0^a \subseteq F_0^{a*}$.

1 Set $a[1] = 1$, $F_0^{a*} = F_0^a$.
2 for $i = 2, \ldots, S$ do:
3 if $a[i-1] = n$ or $(i, a[i-1]+1) \in F_0^a$ then
4 $a[i] = a[i-1]$.
5 else
6 $a[i] = a[i-1] + 1$
7 Set $F_0^{a*} = F_0^{a*} \cup \{(i,j) | j > a[i]\}$.

Orbitopal fixing can also be used to identify variables that must be fixed to one. Consider the partial solution represented by the matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ ? & ? & 0 & 0 \\ ? & ? & ? & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{19}$$

Algorithmically, orbitopal fixing finds variables to fix to one by assuming variables are fixed to zero and using Algorithm 1 to search for contradictions. For instance, suppose $x_{2,2}$ were fixed to zero in the above partial solution. Using Algorithm 1, both $x_{3,3}$ and $x_{4,4}$ would be fixed to zero (as $a[2] = 1$, $a[3] = 2$, and $a[4] = 3$). Because $x_{4,4}$ is already fixed to one, this would cause a contradiction, implying that $x_{2,2}$ must be equal to one. Similarly, $x_{3,3}$ can also be fixed to one. Because there can be only one 1 per column, fixing $x_{2,2}$ to one implies that $x_{2,j}$ can be fixed to zero for all j not equal to 2. The One-Setting portion of orbitopal fixing is described in Algorithm 2.

Algorithm 2 Orbitopal Fixing: One Setting

Input: Subproblem $a = (F_1^a, F_0^a)$.
Output: Subproblem $a' = (F_1^{a'}, F_0^{a'})$ with $F_1^a \subseteq F_0^{a'}$, $F_0^a \subseteq F_0^{a'}$.

1 Set $F_1^{a'} = F_1^a$ and $F_0^{a'} = F_0^a$.
2 for $i = 2, \ldots, S$ with $(i, a[i]) \notin F_1^{a'}$ do:
3 Input $(F_1^{a'}, F_0^{a'} \cup (i, a[i]))$ to Algorithm 1, return $(F_1^{a'*}, F_0^{a'*})$.
4 if $F_1^{a'*} \cap F_0^{a'*} \neq \emptyset$
5 $F_1^{a'} = F_1^{a'} \cup (i, a[i])$, $F_0^{a'} = F_0^{a'} \cup \{(i,j) | j \neq a[i]\}$

3.3 Orbital Branching

Orbital branching (Ostrowski et al. 2007) is an entirely different way of breaking symmetry. Unlike the previous two methods, orbital branching does not choose a fundamental domain a priori. Instead, it uses branching decisions to break symmetry, and by doing so, lets the branching decisions determine the fundamental domain. It should be noted that there is a stronger method to break symmetry called isomorphism pruning (Margot 2002, 2003). For a general problem, orbital branching is not guaranteed to restrict the feasible region to a minimal fundamental domain, while isomorphism pruning is. However, in the case of job scheduling problems, orbital branching does restrict the feasible region to a minimal fundamental domain, making it equivalent to isomorphism pruning. In addition, orbital branching can compute orbits needed for the job scheduling problem (but not for general problems) in linear time.

Let $\mathcal{G}^a$ be the symmetry group of the subproblem represented by node a in the branch-and-bound tree. Let $O = \{(i_1, j),\ (i_2, j),\ \ldots, (i_{|O|}, j)\} \subseteq N^a$ be an orbit of the symmetry group $\mathcal{G}^a$. In the context of symmetry, the traditional branching disjunction can be replaced. Instead of creating one child node with $x_{i_1,j} = 1$ and another with $x_{i_1,j} = 0$, orbital branching uses the disjunction

$$x_{i_h,j} = 1 \vee \sum_{(i_k,j)\in O} x_{i_k,j} = 0. \tag{20}$$

The details of correctness are presented in Ostrowski et al. (2010b), but a rough idea of the method is as follows. The disjunction $\sum_{(i_k,j)\in O} x_{i_k,j} \geq 1 \vee \sum_{(i_k,j)\in O} x_{i_k,j} = 0$ is surely a valid disjunction for any O. In the left child, the one defined by $\sum_{(i_k,j)\in O} x_{i_k,j} \geq 1$, it is known that *some* variable x_i with i in O takes the value of 1. Because all variables are equivalent, it is appropriate to arbitrarily choose one such i and fix that variable to 1. We have the following result:

Theorem 3.1 *For any node a, let l be the left child formed by adding the constraint $x_{i,h} = 1$ for some $(i, h) \in O(\mathcal{G}^a)$, and let r denote the child formed by adding the constraint $\sum_{(i,j)\in O(\mathcal{G}^a)} x_{i,j} = 0$. There is no solution x feasible in l such that $\pi(x)$ is feasible in r for any π in $\mathcal{G}$.*

Proof: Aiming at a contradiction, suppose x is feasible in l and π in $\mathcal{G}$ with $\pi(x)$ feasible in r when orbit O was chosen for branching. Each orbit in $\mathcal{O}(\mathcal{G})$ represents one row in the matrix x. Because each row contain exactly one 1, and because both x and $\pi(x)$ are feasible at node a, $(i, \pi(j)) = (i, j)$ for all (i, j) in F_1^a. This mean that π stabilizes every column j that currently has a job assigned to it, and that $\pi \in \mathcal{G}^a$. By symmetry, $\pi(x)_{i,h}$ must be equal to one, but because $\pi \in \mathcal{G}^a$, $\pi(x)_{i,h}$ was fixed to zero at node r as a result of the branching constraint $\sum_{(i,j)\in O} x_{i,j} = 0$. □

Theorem 3.1 is a stronger version than that described in Ostrowski et al. (2007), which proved no equivalent solutions with respect to $\mathcal{G}^a$, not $\mathcal{G}$, are found in different children of a.

Performing orbital branching requires knowledge of the orbital partition of the variables at every node in the branch-and-bound tree. This in turn requires finding the symmetry group (or at least a subset of the symmetry group). For a general problem, finding the symmetry group requires the use of an algorithm with a worst-case exponential running time. Fortunately finding the symmetry group is easy for job scheduling problems. Permutations of the columns of x are in the symmetry group of a job scheduling subproblem if and only if no job has been assigned to a permuted column. Thus, $x_{i,j}$ and $x_{i,k}$ share the same orbit in a subproblem if and only if no job has been assigned to either machine j or machine k. Thus, finding an orbit to branch on can be done in linear time.

3.4 Comparison Between the Methods

Both the symmetry-reducing constraints and orbitopal fixing restrict the set of feasible solutions to be the matrices x with lexicographically decreasing columns. The difference between the methods is that Denton et al. (2009) restricts the feasible region by explicitly adding inequalities, while the latter uses these inequalities implicitly to fix variables. Explicitly adding these inequalities may

remove fractional solutions that would have been optimal in the LP relaxation, resulting in better LP relaxations and thus, smaller branch-and-bound trees. Another advantage of the inequality method over orbital fixing and orbital branching is that commercial solvers may be able to use these inequalities to generate stronger cutting planes. The cost of the improved relaxations is that the additional inequalities make the LP relaxations more computationally intensive. Also, as many optimal solutions may be made infeasible by the constraints, finding an optimal solution may be more difficult.

Because the minimal fundamental domain is defined a priori, both the constraint version of the formulation and orbitopal fixing can perform more fixing than orbital branching. As an example, consider the subproblem formed by the partial solution

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ & & \vdots & \\ ? & ? & ? & ? \end{pmatrix}. \tag{21}$$

If the columns were forced to be lexicographically decreasing, $x_{5,4}$ must equal zero. This is enforced by both the symmetry-breaking inequalities as well as orbitopal fixing, but *not* by orbital branching.

While there are many applications of job scheduling, we provide computational results for the deterministic operating room (OR) scheduling problem. In this application there is a set of m blocks of surgeries planned for the day and a set of r available operating rooms. It is assumed that the operating rooms are identical and that the time required for each surgery block is known. Using an operating room incurs a one-time fixed cost. If the time required to perform the blocks assigned to a particular operating room is over a predefined limit an overtime cost must be paid for every additional hour. The OR scheduling problem is to find the minimum cost allocation of patients to operating rooms. The problem can be written as follows:

$$Z_D = \min\{\sum_{j=1}^{m}(c^f y_j + c^v o_j)\} \tag{22}$$

$$s.t. \quad x_{ij} \le y_j \quad \forall(i,j) \tag{23}$$

$$\sum_{j=1}^{m} x_{ij} = 1 \quad \forall i \tag{24}$$

$$\sum_{i=1}^{n} d_i x_{ij} \le T y_j + o_j \quad \forall j \tag{25}$$

$$x_{ij},\ y_j \in \{0,1\}\ \forall(i,j), \quad o_j \ge 0\ \forall j, \tag{26}$$

where d_i is the time required to perform surgery block i and T is the total number of hours available for each OR without paying overtime costs. The parameter c^f is the fixed cost associated with opening an OR and c^v is the overtime cost (per minute). The Boolean decision variable y_j indicates if room j is open throughout the day and x_{ij} indicates if surgery block i is performed in room j. The variable o_j records the overtime of room j.

We assume that all operating rooms are identical. If there were L different types of rooms, we can express our solution as a collection of L 0/1 matrices x^l, where $x^l_{i,j} = 1$ means that surgery block i is assigned to the jth room of type l. Note that in this case, rows of x^i are not guaranteed to contain a "1", a fact that is exploited by orbitopal fixing and the symmetry removing constraints.

These methods can still be used by adding a dummy row zero and column row zero to the x^l matrix, where $x^l_{i,0} = 1$ signifies that surgery block i is *not* performed in any room of type l.

Computational results comparing the above methods are done using data based on surgery times given in Gul et al. (2010). All problems have 20 blocks of surgeries and 10 operating rooms. In addition to the surgery time, 20 minutes of prep time for each surgery was added to each block. Results are shown in Table 3. Two different methods of orbitopal fixing were tested, both dealing with how branching variables are determined. Kaibel and Loos (2009) recommend choosing the free variable with smallest row index for branching. This branching strategy is called *minimum indexed branching* (MIB) and selects a variable, with the smallest row index with the additional constraint that its LP solution value is greater than $\frac{1}{n}$. The second method uses CPLEX version 12's preferred branching variable. The results show that orbital branching is significantly better than both orbitopal fixing and constraint generation, solving 21 of the 25 problems faster than either of the other methods. A cutoff of 1,000,000 nodes was used. The best results, by number of nodes and time, for each problem are indicated by bold entries.

Problem	Denton		OF (CPLEX)		OF (MIB)		OB	
	Nodes	Time	Nodes	Time	Nodes	Time	Nodes	Time
1	1000000	589.1	570446	434.4	1000000	741.6	**35514**	**23.2**
2	35502	27.2	193377	146.0	147760	95.8	**2417**	**2.0**
3	**26223**	**26.5**	1000000	572.4	1000000	697.3	158063	102.8
4	11576	9.4	568422	386.4	1000000	636.1	**2326**	**1.5**
5	1000000	553.4	1000000	820.7	1000000	767.9	**1023**	**1.0**
6	15361	15.1	712234	505.3	1000000	669.1	**937**	**1.0**
7	75996	77.1	1000000	790.3	1000000	725.6	**4622**	**3.3**
8	70175	64.5	1000000	825.2	1000000	652.2	**35357**	**25.4**
9	84158	75.7	876960	573.6	1000000	649.5	**2112**	**1.5**
10	10907	6.7	1000000	780.1	1000000	597.4	**9881**	**6.6**
11	41555	45.2	514036	378.8	482927	336.9	**11892**	**8.9**
12	101304	130.1	1000000	746.1	1000000	722.5	**3914**	**2.8**
13	77019	57.6	426468	281.2	1000000	619.9	**5053**	**3.3**
14	129884	138.5	469133	353.7	369447	247.1	**4135**	**3.0**
15	170445	144.0	1000000	818.8	1000000	696.9	**4077**	**3.3**
16	**26645**	**34.0**	1000000	786.46	1000000	879.87	55939	37.9
17	28107	40.3	433824	284.2	986471	583.7	**275**	**0.3**
18	496599	289.4	400112	249.6	1000000	608.5	**14577**	**9.0**
19	108494	109.9	332340	233.9	207160	132.1	**226**	**0.3**
20	74762	61.1	1000000	968.8	1000000	626.0	**67049**	**43.7**
21	25083	30.4	**1488**	**1.0**	22644	13.8	2730	1.7
22	105641	118.6	968257	670.1	1000000	698.7	**1198**	**1.0**
23	105302	87.0	135617	97.2	1000000	651.7	**975**	**1.0**
24	**59870**	**54.3**	1000000	739.6	1000000	645.3	227636	143.7
25	28633	40.2	1000000	746.3	1000000	650.0	**425**	**0.6**

Table 1: Comparison of Methods on OR Scheduling Problems

3.5 Choosing a Fundamental Domain

As mentioned in Section 2.2, there can be more than one minimal fundamental domain. Does it matter which fundamental domain is used? Reducing the feasible region to a fundamental domain can allow for more opportunities to fix variables. It is these fixings that make symmetry-exploiting tools like orbital branching and orbitopal fixing powerful. Ideally, it is preferable to fix variables with respect to symmetry as early in the tree as possible. When the fundamental

domain is generated by choosing only lexicographical minimal solutions, i.e. the fundamental domain described by adding the constraints (18), this additional fixing is done when branching on variables with small row indices. For example, suppose there were 100 surgery blocks and 25 operating rooms. Fixing a variable $x_{100,j}$, for any j, as a result of a branching disjunction does not strengthen any inequality in (18). Branching on the variable $x_{i,j}$ for any $i \leq 25$ and any j, does strengthen inequalities in (18) and leads to additional fixings. For example, fixing $x_{2,2}$ to zero as a result of branching (either by branching on $x_{2,2}$ directly, or by fixing $x_{2,1}$ to one) also allows for the fixing of $x_{3,3}$, $x_{4,4}$, ... $x_{25,25}$ to zero.

Because restriction of the feasible region to the lexicographic fundamental domain strengthens the branching disjunctions associated with variables of small row indices, it is important (for symmetry considerations) to branch on variables with a small row index early in the branch-and-bound tree. However, from an integer programming perspective, the choice of branching variables is very important, and can have a significant effect on the size of the branch-and-bound tree. Good branching candidates from an MILP perspective are those that improve the LP relaxation the most. What if variables that are good branching candidates from a symmetry point of view are bad candidates from an MILP point of view? This can be remedied by a better choice of fundamental domain.

An intuitive and effective branching strategy for bin packing type problems like the OR scheduling problem is to first branch on items with the largest weight, in this case the largest surgery time. Given this branching strategy, a fundamental domain that increases the importance of surgery blocks with larger completion time can be created. This is done by reindexing the variables such that the d_i terms, the completion time for each surgery, form a decreasing sequence. Now, instead of representing a random surgery block, variable $x_{1,j}$ indicates whether the block with the largest completion time is performed in room j.

As orbital branching creates the minimal fundamental domain throughout the branching process, it is not necessary to determine how to best reindex variables. In theory, reindexing should not affect the overall performance of orbital branching, as it uses symmetry to strengthen CPLEX's branching, something that should not be dependent on variable indices. However, in the case of tie-breaks, indices may play a role in determining what variable is chosen for branching.

Computational results for the reindexed problems are given in Table 2. As the results show, reindexing variables leads to significant improvements in computation time for both the constraint method as well as orbitopal fixing. With orbital branching, reindexing the variables does affect the results, but not in a predictable way.

As is shown by the results, choosing an appropriate fundamental domain is important if one wishes to break symmetry by adding constraint or by orbitopal fixing. Doing so seems to lead to the best possible solution times. However, in order to construct an ideal fundamental domain, good branching strategies must be known a priori. If the branching strategy is not know a priori, orbital branching is the more effective choice in solving these problems. One method for determining a branching strategy for general MILP problems is discussed in Margot (2003). The idea is to perform a dive in the branch-and-bound tree using strong branching to choose branching variables. The variables are then reindexed to reflect the order in which they were branched upon during the initial dive. The minimal fundamental domain used is the one generated by the lexicographically minimal solutions in the reindexed problem. Ideally strong branching will branch on variables that influence the LP bound the most first, so these variables should be given smaller indices.

4 Machine Scheduling Problems

This section explores symmetry exploiting methods in machine scheduling problems. Specifically, the problem of determining the on-off status of identical machines over a finite time horizon is explored. Like the job scheduling problem, we express solutions to the machine scheduling problem as an $m \times n$ 0/1 matrix. Unlike the job scheduling problem, there are no restrictions on the number of ones in each row. This fact makes complete symmetry removal much more challenging.

Problem	Denton		OF (CPLEX)		OF (MIB)		OB	
	Nodes	Time	Nodes	Time	Nodes	Time	Nodes	Time
1	**9679**	**19.3**	345317	223.8	101336	62.2	37652	24.5
2	**396**	**0.5**	4471	3.3	6801	4.6	1571	1.5
3	**8209**	**53.2**	1000000	755	163287	113.4	552746	564.8
4	879	1.1	9656	6.6	4218	2.8	**437**	**0.5**
5	**271**	0.6	9218	7.1	409	**0.5**	513	0.6
6	4057	8.2	48475	35.8	4119	3.5	**2341**	**1.8**
7	**6784**	35.3	1000000	843.5	241657	170.1	18929	**23.1**
8	**3332**	20.8	118552	91.3	14626	10.9	9688	**7.4**
9	606	1.1	11169	7.6	**252**	**0.4**	1342	1.2
10	4419	13.5	34802	23.8	22031	14	**1782**	**1.5**
11	**2456**	3.7	11716	8	5562	3.8	2698	**2.1**
12	3311	8.1	241544	177.7	21911	14.9	**2851**	**3.7**
13	**595**	**0.9**	3604	2.5	1457	0.9	1041	1.5
14	**805**	**1.2**	5787	4.2	2510	1.7	8570	6.4
15	**5931**	56.1	1000000	842.5	415775	292.9	73368	**52.9**
16	**6705**	**11.2**	684789	503.2	108536	73.6	12976	16.4
17	**243**	**0.2**	272	0.3	325	0.3	362	0.7
18	**6850**	**5.4**	21455	13.7	11786	7	27595	16.8
19	496	0.6	263	0.4	**186**	**0.3**	2093	2
20	**4537**	**10.3**	804223	553.5	122993	78.4	85444	70.3
21	139	**0.2**	973	0.8	597	0.5	**88**	**0.2**
22	1944	3.4	8244	5.8	2487	1.9	**1036**	**0.9**
23	2897	8.8	136253	100.6	3605	3	**1243**	**1.2**
24	**1381**	3.7	9189	7	2915	**2.5**	6517	4.6
25	1856	4.4	77194	54.6	3551	2.9	**1383**	**1.2**

Table 2: Comparison of OR Scheduling After Reindexing

Unlike Section 3, this section considers the possibility that only subsets of machines are identical. The on/off status of n identical machines of type i over m hours can be represented as an m by n 0/1 matrix, x^i, where a 1 (0) in entry (k, j) means that machine k is (is not) in operation at time j. Because the machines of type i are identical, there is complete symmetry with respect to the columns of x^i.

Recall from Section 2.2 that the following inequalities guarantee that the resulting x^i matrix has lexicographically decreasing columns, and thus define a minimal fundamental domain:

$$\sum_{i=1}^{n} 2^{n-i} x_j^i(t) \geq \sum_{i=1}^{n} 2^{n-1} x_{j+1}^i(t) \ \forall j = 0, \ \ldots, \ n-1. \tag{27}$$

Because of the exponential coefficient, adding constraints like those in (27) are not practical from a computational standpoint. Unlike the job-scheduling problem, the structure of the machine scheduling problem cannot be used to simplify these constraints. Thus, it is not obvious how to set up symmetry removing constraints. Also, orbitopal fixing requires the structure provided by the job scheduling problem and cannot be applied to machine scheduling problems. However, work has been done to study the structure of the set of all lexicographically minimal 0/1 matrices. Kaibel and Loos (2010) present a compact extended formulation for all 0/1 matrices with lexicographically decreasing columns, removing all symmetry resulting from identical machines. Unfortunately, this extended formulation requires $O(TK^3)$ many variables. Even if we were able to define a minimal, or at least a small fundamental domain a priori, we would still have to worry about choosing an appropriate domain. With the OR scheduling case it seemed reasonable to assume that a good branching strategy would be to branch on variables representing surgery block with large completion times first. For a general machine scheduling problem however, it may not be easy to determine a good branching strategy, and as a consequence a good fundamental domain, a priori.

In the case of job scheduling problems, isomorphism pruning was identical to orbital branching. This, however, is not the case in machine scheduling problems, as orbital branching does not guarantee complete symmetry removal. Unlike isomorphism pruning, orbital branching is easily implemented and finding the required orbits can be done in polynomial time for scheduling problems. Therefore, even though isomorphism pruning guarantees complete symmetry removal, only orbital branching will be considered in this section.

Recall from Section 3.3 that at any node a in the branch-and-bound tree, orbital branching chooses an orbit $O^i = \{j_1, \ j_2, \ \ldots, \ j_{|O^i|}\} \subseteq N^a$ of the symmetry group $\mathcal{G}^a$ and branches on the disjunction

$$x_{j_k}^i = 1 \ \vee \ \sum_{j_k \in O^i} x_{j_k}^i = 0. \tag{28}$$

In Section 3.4 we showed that for job scheduling problems, orbital branching is effective at using branching decisions to generate a good fundamental domain. It would make sense then to expect that orbital branching would be well suited for machine scheduling problems where good branching strategies are not known. When using orbital branching on some machine scheduling problems, we were surprised to see that orbital branching was not as effective as we expected it to be. Closer examination of the results showed that much like the constraint method and orbitopal fixing on job scheduling problems, the fundamental domain chosen by orbital branching was not well suited for CPLEX's desired branching behavior. This is an immediate consequence of the fact that more than one "1" can exist in every row of the optimal solution.

Consider the following partial solution

$$x^i = \begin{pmatrix} 1 & ? & ? & ? & ? \\ ? & ? & ? & 1 & ? \\ ? & 1 & ? & ? & ? \\ ? & 1 & 1 & ? & ? \end{pmatrix}.$$

In the subproblem, all of the column permutations have been removed from the symmetry group. What if

$$x^i_{LP} = \begin{pmatrix} 1 & .95 & 1 & ? & ? \\ ? & ? & ? & 1 & ? \\ .97 & 1 & 1 & ? & ? \\ .95 & 1 & 1 & ? & ? \end{pmatrix} \tag{29}$$

is the corresponding LP solution? Technically, there is no symmetry found in this subproblem. However, it is likely that the optimal solution to this problem has each of $x_{1,2}$, $x_{3,1}$, and $x_{4,1}$ equal to one. If each of these variables had been fixed, then all permutations of the first three columns of x would be in the subproblem's symmetry group, and those permutations could be used to strengthen the branching disjunction.

We would like to let CPLEX choose our branching candidate, then augment that branching decision using orbital branching. The problem with this strategy in the machine scheduling case is that if $x_{i,j}$ is chosen to branch on at a node, then it is unlikely that $x_{i,k}$ will be chosen at a child node. This is especially true in cases similar to (29). Variables that take a value near one in the LP solution (and especially if they are equal to one) will likely not be chosen for branching because doing so will not improve the bound. However, from a symmetry point of view, those variables need to be fixed to create larger symmetry groups (to further strengthen the next branch). To exploit symmetry early in the branch-and-bound tree, it is important to branch on variables in the same row as previously fixed variables, but this is not a good branching strategy from an MILP point of view. In contrast to the job scheduling case, where we find a new fundamental domain to better complement a branching strategy, for machine scheduling we adapt our branching decisions to better complement our fundamental domain.

With the aim of creating strong disjunctions that keep as much symmetry in the subproblems as possible, orbital branching is modified as follows.

For any $l \in \mathbb{Z}_+$ and any $O^i = \{j_1,\ j_2,\ \ldots,\ j_{|O^i|}\} \subseteq N^a$, the following disjunction is valid:

$$\sum_{j_k \in O^i} x^i_{j_k} \geq l \ \vee \ \sum_{j_k \in O^i} x^i_{j_k} \leq l-1. \tag{30}$$

If O^i represents a set of variables that are equivalent with respect to the column symmetry in x^i, then Equation (30) can be modified as

$$\sum_{k=1}^{l} x^i_{j_k} = l \ \vee \ \sum_{k=1}^{|O|-l+1} x^i_{j_k} = 0. \tag{31}$$

Theorem 4.1 *The branching disjunction* (31) *is valid for machine scheduling problems.*

Proof: Suppose that a solution feasible at node a was wrongly removed from the feasible region of both child nodes. Let x^* correspond to such a solution that maximizes the function $\sum_{(t,j_k)\in O^i} 2^{|O^i|-k} x^i_{t,j_k}$. It can be assumed that $\sum_{(t,j_k)\in O^i} {x^i_{t,j_k}}^* \geq l$. Because the solution is not feasible in the left child node, there exists $c \leq l$ with $x^* {i_{t,j_c}}^i_{t,j_c} = 0$, and by the pigeonhole property, there exists $d > c$ with ${x^*}^i_{t,k_d} = 1$. Because columns j_c and j_d are equivalent, there exists a permutation in $\mathcal{G}^a$ permuting only these columns. Applying this permutation to the solution either gives a solution feasible in the left child, contradicting that the solution was wrongly removed, or a solution with x'^i with $\sum_{(t,j_k)\in O^i} 2^{|O^i|-k} {x'}^i_{t,j_k} > \sum_{(t,j_k)\in O^i} 2^{|O^i|-k} {x^*}^i_{t,j_k}$, contradicting our choice of ${x^*}^i$. □

Branching in this way has the benefit of keeping symmetry present in the child nodes. As a result, branching decisions are not influenced by ancestors' branching decisions. Also, branching in this way produces more balanced branch-and-bound trees.

4.1 Computational Results

Computational results demonstrating the effectiveness of orbital branching and modified orbital branching are presented in this section. We have chosen to use instances of the Unit Commitment problem to make up the testbed for this section. The unit commitment problem is an example of a production scheduling problem in the power industry. It can be formulated as

$$\text{Minimize} \sum_{t \in T} \sum_{j \in J} c_j(p_j^i(t)) \tag{32}$$

$$\text{subject to} \sum_{j \in J} p_j^i(t) \geq D(t), \quad \forall\, t \in T \tag{33}$$

$$p_j^i \in \Pi^i, \quad \forall j \in J^i. \tag{34}$$

where Π^i represents the set of feasible production schedules for generators of type i. The function $c_j(p_j^i(t))$ gives the cost of generator j producing $p_j(t)$ units of electricity at time t. It is generally assumed to be a quadratic function, but for this paper it is approximated by a piecewise linear function. Strong inequalities for the linear approximation, called *perspective cuts*, are given in Frangioni and Gentile (2006) and Frangioni et al. (2009).

The operational constraints defining Π^i ensure that generators of type i operate within their physical limits. These limits typically include limitations of how much the power output from a generator can change over time, and minimum amounts of time a generator must be turned on/off. An explanation of different formulations of $\Pi_j(t)$ can be found in Ostrowski et al. (2010a).

The solution of the unit commitment problem can be represented by the collection of matrices v^i and p^i, v^i representing the on/off status and p^i representing the power generated. For each generator type i, and for every solution, one can apply the same permutation to columns of v^i and p^i to create an equivalent solution. Because these matrices are interdependent, attention will focus on exploiting symmetry in the v^i matrices.

Random instances on 21 to 42 generators were created using eight different types of generators. These generators are described in Ostrowski et al. (2010a). Both versions of orbital branching were implemented in CPLEX 12.1 using the branch callback feature. Unfortunately, using callback functions disables other CPLEX features, notably "dynamic search". In addition to comparing classical orbital branching with the version discussed in the section, we also give results based on CPLEX's default (dynamic search plus additional features) and CPLEX with the disabled features (using traditional branch-and-cut). Computational results are given in Table 3. Problems not solved within two hours are denoted by "-". As the table indicates, the modified orbital branching performs significantly better than the original method. CPLEX with dynamic search seems to perform better than traditional orbital branching, while the traditional branch-and-cut version of CPLEX solves only one problem. The difference between CPLEX using the traditional branch-and-cut and dynamic search is remarkable. One wonders how effective modified orbital branching would be on these problems if it were incorporated into dynamic search.

5 Conclusions

If symmetry is present in an MILP problem, it must be dealt with in an effective manner. There are many strategies one can use to break symmetry, but as we have shown in this paper, some methods may interact poorly with branching rules. By studying the impact of each symmetry breaking technique on branching disjunctions, one can modify the symmetry breaking to better fit the problem.

We have shown that given specific knowledge of the job scheduling problem, a fundamental domain can be chosen a priori to increase the strength of branching disjunctions. Constraints added to the problem can be modified to enhance the branching decisions and this approach seems

# of Generators	CPLEX Only				OB		Modified OB	
	Dynamic Search		B&B					
	Nodes	Time	Nodes	Time	Nodes	Time	Nodes	Time
21	923	544.0	-	-	5498	482.8	190	57.9
23	499	386.5	82539	6415.5	3190	342.2	390	78.9
23	878	1227.6	-	-	-	-	715	308.6
24	3259	1169.3	-	-	6259	691.2	517	155.7
26	972	978.4	-	-	37150	5461.1	206	138.4
26	516	529.5	-	-	2574	366.2	180	68.4
26	558	558.4	-	-	3830	628.6	219	107.4
26	500	425.6	-	-	13790	1552.5	158	74.4
26	515	465.3	-	-	27890	2015.2	218	111.4
26	4369	1320.9	-	-	16805	1758.5	341	104.0
27	579	535.3	-	-	3323	494.5	187	105.4
27	545	594.3	-	-	-	-	7222	1339.8
28	522	679.5	-	-	-	-	720	307.7
28	532	444.0	-	-	5162	578.0	358	107.8
29	1182	975.6	-	-	-	-	-	-
30	1793	1514.6	-	-	-	-	2523	631.0
30	541	862.7	-	-	-	-	1252	381.8
31	1268	1210.3	-	-	4010	6553.9	6521	1197.4
31	538	783.5	-	-	13475	3660.6	113	107.3
31	537	712.5	-	-	-	-	842	296.1
31	1172	1360.8	-	-	19929	4599.6	570	220.7
34	544	739.0	-	-	-	-	4201	1401.9
35	600	1204.9	-	-	21612	4697.3	1190	404.5
37	4029	2808.2	-	-	-	-	946	447.0
42	994	1540.1	-	-	-	-	495	396.8

Table 3: Comparison of Methods on UC Problems

to be the most effective. If one does not know effective branching strategies, however, adding constraints a priori can be a very weak strategy. In this case, creating the fundamental domain *while* branching using orbital branching can be very effective.

The effectiveness of orbital branching to generate fundamental domains that enhance branching does not extend to machine scheduling problems. However, by slightly modifying the branching rule to better adapt to the fundamental domain we see a sizable decrease in the computational time needed.

While we used scheduling problems as an example, we expect relationships between symmetry breaking and branching to exist for general integer programming problems. Examining the relationship between symmetry-breaking techniques and MILP techniques is a fruitful area for future research.

References

Bixby, R., E. Boyd, R. Indovina. 1992. MIPLIB a test set of mixed integer programming problems. Tech. Rep. 25:2, SIAM News.

Denton, B., A. Miller, H. Balasubramanian, T. Huschka. 2009. Optimal allocation of surgery blocks to operating rooms under uncertainty. *Operations Research* **accepted**.

Frangioni, A., C. Gentile. 2006. Perspective cuts for a class of convex 0-1 mixed integer programs. *Math Programming, Series A* **106**.

Frangioni, A., C. Gentile, F. Lacalandra. 2009. Tighter approximated milp formulations for unit commitment problems. *IEEE Transactions on Power Systems* **24**(1).

Fulkerson, D. R., G. L. Nemhauser, L. E. Trotter. 1973. Two computationally difficult set covering problems that arise in computing the 1-width of incidence matrices of Steiner triples. *Mathematical Programming Study* **2** 72–81.

Gul, S., B. Denton, J. Fowler, T. Huschka. 2010. Bi-criteria scheduling of surgical services for an outpatient procedure center. *Working Paper* .

Holt, Derek F. 2005. *Handbook of Computational Group Theory (Discrete Mathematics and Its Applications)*. Chapman & Hall/CRC.

Jeroslow, R. 1974. Trtivial integer programs unsolvable by branch-and-bound. *Mathematical Programming* **6** 105–109.

Kaibel, V., A. Loos. 2009. Branched polyhedral systems. *working paper* .

Kaibel, V., A. Loos. 2010. Branched polyhedral systems. *IPCO 2010: The Fourteenth Conference on Integer Programming and Combinatorial Optimization*, *Lecture Notes in Computer Science*, vol. 6080. Springer, 177–190.

Kaibel, V., M. Peinhardt, M.E. Pfetsch. 2007. Orbitopal fixing. *IPCO 2007: The Twelfth Conference on Integer Programming and Combinatorial Optimization*. Springer, 74–88.

Kaibel, V., M.E. Pfetsch. 2008. Packing and partitioning orbitopes. *Mathemathical Programming* **114** 1–36.

Margot, F. 2002. Pruning by isomorphism in branch-and-cut. *Mathematical Programming* **94** 71–90.

Margot, F. 2003. Exploiting orbits in symmetric ILP. *Mathematical Programming, Series B* **98** 3–21.

Margot, F. 2008. Symmetry in integer linear programming. *Tepper Working Paper* **E37**.

Ostrowski, J., M.F. Anjos, A. Vannelli. 2010a. Tight mixed integer linear programming formulations for generator self-scheduling and unit commitment. *Technical Report* .

Ostrowski, J., J. Linderoth, F. Rossi, S. Smriglio. 2007. Orbital branching. *IPCO 2007: The Twelfth Conference on Integer Programming and Combinatorial Optimization*, *Lecture Notes in Computer Science*, vol. 4517. Springer, 104–118.

Ostrowski, J., J. Linderoth, F. Rossi, S. Smriglio. 2010b. Orbital branching. *Mathematical Programming* To appear.