Orbital Branching

James Ostrowski[1], Jeff Linderoth[1], Fabrizio Rossi[2], and Stefano Smriglio[2]

[1] Department of Industrial and Systems Engineering, Lehigh University, 200 W. Packer Ave., Bethlehem, PA 18015, USA
{jao204,jtl3}@lehigh.edu

[2] Dipartimento di Informatica, Università di L'Aquila, Via Vetoio I-67010 Coppito (AQ), Italy
{rossi,smriglio}@di.univaq.it

Abstract. We introduce *orbital branching*, an effective branching method for integer programs containing a great deal of symmetry. The method is based on computing groups of variables that are equivalent with respect to the symmetry remaining in the problem after branching, including symmetry which is not present at the root node. These groups of equivalent variables, called orbits, are used to create a valid partitioning of the feasible region which significantly reduces the effects of symmetry while still allowing a flexible branching rule. We also show how to exploit the symmetries present in the problem to fix variables throughout the branch-and-bound tree. Orbital branching can easily be incorporated into standard IP software. Through an empirical study on a test suite of symmetric integer programs, the question as to the most effective orbit on which to base the branching decision is investigated. The resulting method is shown to be quite competitive with a similar method known as *isomorphism pruning* and significantly better than a state-of-the-art commercial solver on symmetric integer programs.

1 Introduction

In this work, we focus on packing and covering integer programs (IP)s of the form

$$\max_{x\in\{0,1\}^n} \{e^T x \mid Ax \leq e\} \text{ and} \tag{PIP}$$

$$\min_{x\in\{0,1\}^n} \{e^T x \mid Ax \geq e\} \ , \tag{CIP}$$

where $A \in \{0,1\}^{m\times n}$, and e is a vector of ones of conformal size. Our particular focus is on cases when (CIP) or (PIP) is highly-symmetric, a concept we formalize as follows. Let Π^n be the set of all permutations of $I^n = \{1,\ldots,n\}$. Given a permutation $\pi \in \Pi^n$ and a permutation $\sigma \in \Pi^m$, let $A(\pi,\sigma)$ be the matrix obtained by permuting the columns of A by π and the rows of A by σ, i.e. $A(\pi,\sigma) = P_\sigma A P_\pi$, where P_σ and P_π are permutation matrices. The *symmetry group* $\mathcal{G}$ of the matrix A is the set of permutations

$$\mathcal{G}(A) \stackrel{\text{def}}{=} \{\pi \in \Pi^n \mid \exists \sigma \in \Pi^m \text{ such that } A(\pi,\sigma) = A\} \ .$$

M. Fischetti and D.P. Williamson (Eds.): IPCO 2007, LNCS 4513, pp. 104–118, 2007.

So, for any $\pi \in \mathcal{G}(A)$, if $\hat{x}$ is feasible for (CIP) or (PIP) (or the LP relaxations of (CIP) or (PIP)), then if the permutation π is applied to the coordinates of $\hat{x}$, the resulting solution, which we denote as $\pi(\hat{x})$, is also feasible. Moreover, the solutions $\hat{x}$ and $\pi(\hat{x})$ have equal objective value.

This equivalence of solutions induced by symmetry is a major factor that might confound the branch-and-bound process. For example, suppose $\hat{x}$ is a (non-integral) solution to an LP relaxation of PIP or CIP, with $0 < \hat{x}_j < 1$, and the decision is made to branch down on variable x_j by fixing $x_j = 0$. If $\exists \pi \in \mathcal{G}(A)$ such that $[\pi(\hat{x})]_j = 0$, then $\pi(\hat{x})$ is a feasible solution for this child node, and $e^T\hat{x} = e^T(\pi(\hat{x}))$, so the relaxation value for the child node will not change. If the cardinality of $\mathcal{G}(A)$ is large, then there are many permutations through which the parent solution of the relaxation can be preserved in this manner, resulting in many branches that do not change the bound on the parent node. Symmetry has long been recognized as a curse for solving integer programs, and auxiliary (often extended) formulations are often sought that reduce the amount of symmetry in an IP formulation [1,2,3]. In addition, there is a body of research on valid inequalities that can help exclude symmetric feasible solutions [4,5,6]. Kaibel and Pfetsch [7] formalize many of these arguments by defining and studying the properties of a polyhedron known as an orbitope, the convex hull of lexicographically maximal solutions with respect to a symmetry group. Kaibel et al. [8] then use the properties of orbitopes to remove symmetry in partitioning problems.

A different idea, *isomorphism pruning*, introduced by Margot [9,10] in the context of IP and dating back to Bazaraa and Kirca [11], examines the symmetry group of the problem in order to prune isomorphic subproblems of the enumeration tree. The branching method introduced in this work, *orbital branching*, also uses the symmetry group of the problem. However, instead of examining this group to ensure that an isomorphic node will never be evaluated, the group is used to guide the branching decision. At the cost of potentially evaluating isomorphic subproblems, orbital branching allows for considerably more flexibility in the choice of branching entity than isomorphism pruning. Furthermore, orbital branching can be easily incorporated within a standard MIP solver and even exploit problem symmetry that may only be locally present at a nodal subproblem.

The remainder of the paper is divided into five sections. In Sect. 2 we give some mathematical preliminaries. Orbital branching is introduced and formalized in Sect. 3, and a mechanism to fix additional variables based on symmetry considerations called *orbital fixing* is described there. A more complete comparison to isomorphism pruning is also presented in Sect. 3. Implementation details are provided in Sect. 4, and computational results are presented in Sect. 5. Conclusions about the impact of orbital branching and future research directions are given in Sect. 6.

2 Preliminaries

Orbital branching is based on elementary concepts from algebra that we recall in this section to make the presentation self-contained. Some definitions are made in terms of an arbitrary permutation group Γ, but for concreteness, the reader may consider the group Γ to be the symmetry group of the matrix $\mathcal{G}(A)$.

For a set $S \subseteq I^n$, the *orbit* of S under the action of Γ is the set of all subsets of I^n to which S can be sent by permutations in Γ, i.e.,

$$\mathrm{orb}(S,\Gamma) \stackrel{\mathrm{def}}{=} \{S' \subseteq I^n \mid \exists \pi \in \Gamma \text{ such that } S' = \pi(S)\} \ .$$

In the orbital branching we are concerned with the orbits of sets of cardinality one, corresponding to decision variables x_j in PIP or CIP. By definition, if $j \in \mathrm{orb}(\{k\},\Gamma)$, then $k \in \mathrm{orb}(\{j\},\Gamma)$, i.e. the variable x_j and x_k share the same orbit. Therefore, the union of the orbits

$$\mathcal{O}(\Gamma) \stackrel{\mathrm{def}}{=} \bigcup_{j=1}^{n} \mathrm{orb}(\{j\},\Gamma)$$

forms a partition of $I^n = \{1, 2, \ldots, n\}$, which we refer to as the orbital partition of Γ, or simply the *orbits* of Γ. The orbits encode which variables are "equivalent" with respect to the symmetry Γ.

The stabilizer of a set $S \subseteq I^n$ in Γ is the set of permutations in Γ that send S to itself.

$$\mathrm{stab}(S,\Gamma) = \{\pi \in \Gamma \mid \pi(S) = S\} \ .$$

The stabilizer of S is a subgroup of Γ.

We characterize a node $a = (F_1^a, F_0^a)$ of the branch-and-bound enumeration tree by the indices of variables fixed to one F_1^a and fixed to zero F_0^a at node a. The set of free variables at node a is denoted by $N^a = I^n \setminus F_0^a \setminus F_1^a$. At node a, the set of feasible solutions to (CIP) or (PIP) is denoted by $\mathcal{F}(a)$, and the value of an optimal solution for the subtree rooted at node a is denoted as $z^*(a)$.

3 Orbital Branching

In this section we introduce orbital branching, an intuitive way to exploit the orbits of the symmetry group $\mathcal{G}(A)$ when making branching decisions. The classical 0-1 branching variable dichotomy does not take advantage of the problem information encoded in the symmetry group. To take advantage of this information in orbital branching, instead of branching on individual variables, orbits of variables are used to create the branching dichotomy. Informally, suppose that at the current subproblem there is an orbit of cardinality k in the orbital partitioning. In orbital branching, the current subproblem is divided into $k+1$ subproblems: the first k subproblems are obtained by fixing to one in turn each variable in the orbit while the $(k+1)^{\mathrm{st}}$ subproblem is obtained by fixing all variables in the orbit to zero. For any pair of variables x_i and x_j in the same orbit, the subproblem created when x_i is fixed to one is essentially equivalent to the subproblem created when x_j is fixed to one. Therefore, we can keep in the subproblem list only *one* representative subproblem, pruning the $(k-1)$ equivalent subproblems. This is formalized below.

Let $A(F_1^a, F_0^a)$ be the matrix obtained by removing from the constraint matrix A all columns in $F_0^a \cup F_1^a$ and either all rows intersecting columns in F_1^a (CIP case) or all columns nonorthogonal to columns in F_1^a (PIP case). Note that if $x \in \mathcal{F}(a)$ and x is feasible with respect to the matrix A, then x is feasible with respect to the matrix $A(F_1^a, F_0^a)$.

Let $O = \{i_1, i_2, \ldots, i_{|O|}\} \subseteq N^a$ be an orbit of the symmetry group $\mathcal{G}(A(F_1^a, F_0^a))$. Given a subproblem a, the disjunction

$$x_{i_1} = 1 \vee x_{i_2} = 1 \vee \ldots x_{i_O} = 1 \vee \sum_{i \in O} x_i = 0 \tag{1}$$

induces a feasible division of the search space. In what follows, we show that for any two variables $x_j, x_k \in O$, the two children $a(j)$ and $a(k)$ of a, obtained by fixing respectively x_j and x_k to 1 have the same optimal solution value. As a consequence, disjunction (1) can be replaced by the binary disjunction

$$x_h = 1 \vee \sum_{i \in O} x_i = 0 \ , \tag{2}$$

where h is a variable in O. Formally, we have Theorem 1.

Theorem 1. *Let O be an orbit in the orbital partitioning $\mathcal{O}(\mathcal{G}(A(F_1^a, F_0^a)))$, and let j, k be two variable indices in O. If $a(j) = (F_1^a \cup \{j\}, F_0^a)$ and $a(k) = (F_1^a \cup \{k\}, F_0^a)$ are the child nodes created when branching on variables x_j and x_k, then $z^*(a(j)) = z^*(a(k))$.*

Proof. Let x^* be an optimal solution of $a(j)$ with value $z^*(a(j))$. Obviously x^* is also feasible for a. Since j and k are in the same orbit O, there exists a permutation $\pi \in \mathcal{G}(A(F_1^a, F_0^a))$ such that $\pi(j) = k$. By definition, $\pi(x^*)$ is a feasible solution of a with value $z^*(a(j))$ such that $x_k = 1$. Therefore, $\pi(x^*)$ is feasible for $a(k)$, and $z^*(a(k)) = z^*(a(j))$. □

The basic *orbital branching* method is formalized in Algorithm 1.

Algorithm 1. Orbital Branching

Input:	Subproblem $a = (F_1^a, F_0^a)$, non-integral solution $\hat{x}$.
Output:	Two child subproblems b and c.
Step 1.	Compute orbital partition $\mathcal{O}(\mathcal{G}(A(F_1^a, F_0^a))) = \{O_1, O_2, \ldots, O_p\}$.
Step 2.	Select orbit O_{j^*}, $j^* \in \{1, 2, \ldots, p\}$.
Step 3.	Choose arbitrary $k \in O_{j^*}$. Return subproblems $b = (F_1^a \cup \{k\}, F_0^a)$ and $c = (F_1^a, F_0^a \cup O_{j^*})$.

The consequence of Theorem 1 is that the search space is limited, but orbital branching has also the relevant effect of reducing the likelihood of encountering symmetric solutions. Namely, no solutions in the left and right child nodes of the current node will be symmetric with respect to the local symmetry. This is formalized in Theorem 2.

Theorem 2. *Let b and c be any two subproblems in the enumeration tree. Let a be the first common ancestor of b and c. For any $x \in \mathcal{F}(b)$ and $\pi \in \mathcal{G}(A(F_0^a, F_1^a))$, $\pi(x)$ does not belong $\mathcal{F}(c)$.*

Proof. Suppose not, i.e., that there $\exists x \in \mathcal{F}(b)$ and a permutation $\pi \in \mathcal{G}(A(F_0^a, F_1^a))$ such that $\pi(x) \in \mathcal{F}(c)$. Let $O_i \in \mathcal{O}(\mathcal{G}(A(F_1^a, F_0^a)))$ be the orbit chosen to branch on at subproblem a. W.l.o.g. we can assume $x_k = 1$ for some $k \in O_i$. We have that $x_k = [\pi(x)]_{\pi(k)} = 1$, but $\pi(k) \in O_i$. Therefore, by the orbital branching dichotomy, $\pi(k) \in F_0^c$, so $\pi(x) \notin \mathcal{F}(c)$. □

Note that by using the matrix $A(F_1^a, F_0^a)$, orbital branching attempts to use symmetry found at all nodes in the enumeration tree, not just the symmetry found at the root node. This makes it possible to prune nodes whose corresponding solutions are not symmetric in the original IP.

3.1 Orbital Fixing

In orbital branching, all variables fixed to zero and one are removed from the constraint matrix at every node in the enumeration tree. As Theorem 2 demonstrates, using orbital branching in this way ensures that any two nodes are not equivalent with respect to the symmetry found at their first common ancestor. It is possible however, for two child subproblems to be equivalent with respect to a symmetry group found elsewhere in the tree. In order to combat this type of symmetry we perform *orbital fixing*, which works as follows.

Consider the symmetry group $\mathcal{G}(A(F_1^a, \emptyset))$ at node a. If there exists an orbit O in the orbital partition $\mathcal{O}(\mathcal{G}(A(F_1^a, \emptyset)))$ that contains variables such that $O \cap F_0^a \neq \emptyset$ and $O \cap N^a \neq \emptyset$, then all variables in O can be fixed to zero. In the following theorem, we show that such variable setting (orbital fixing) excludes feasible solutions only if there exists a feasible solution of the same objective value to the left of the current node in the branch and bound tree. (We assume that the enumeration tree is oriented so that the branch with an additional variable fixed at one is the left branch).

To aid in our development,we introduce the concept of a *focus node*. For $x \in \mathcal{F}(a)$, we call node $b(a, x)$ a focus node of a with respect to x if $\exists y \in \mathcal{F}(b)$ such that $e^T x = e^T y$ and b is found to the left of a in the tree.

Theorem 3. *Let $\{O_1, O_2, \dots O_q\}$ be an orbital partitioning of $\mathcal{G}(A(F_1^a, \emptyset))$ at node a, and let the set*

$$S \stackrel{\text{def}}{=} \{j \in N^a \mid \exists k \in F_0^a \text{ and } j, k \in O_\ell \text{ for some } \ell \in \{1, 2, \dots q\}\}$$

be the set of free variables that share an orbit with a variable fixed to zero at a. If $x \in \mathcal{F}(a)$ with $x_i = 1$ for some $i \in S$, then there exists a focus node for a with respect to x.

Proof. Suppose that a is the first node in any enumeration tree where S is nonempty. Then, there exist $j \in F_0^a$ and $i \in S$ such that $i \in \text{orb}(\{j\}, \mathcal{G}(A(F_1^a, \emptyset)))$, i.e., there exists a $\pi \in \mathcal{G}(A(F_1^a, \emptyset))$ with $\pi(i) = j$. W.l.o.g., suppose that j is any of the first such variables fixed to zero on the path from the root node to a and let c be the subproblem in which such a fixing occurs. Let $\rho(c)$ be the parent node of c. By our choice of j as the first fixed variable, for all $i \in F_0^a$, we have $x_{\pi(i)} = 0$. Then, there exists $x \in \mathcal{F}(a)$ with $x_i = 1$ such that $\pi(x)$ is not feasible in a (since it does not satisfy the bounds) but it is feasible in $\rho(c)$ and has the same objective value of x. Since j was fixed by orbital branching then the left child of $\rho(c)$ has $x_h = 1$ for some $h \in \text{orb}(\{j\}, \mathcal{G}(A(F_1^{\rho(c)}, F_0^{\rho(c)})))$. Let $\pi' \in \mathcal{G}(A(F_1^{\rho(c)}, F_0^{\rho(c)}))$ have $\pi'(j) = h$. Then $\pi'(\pi(x))$ is feasible in the left node with the same objective value of x. The left child node of $\rho(c)$ is then the focus node of a with respect to x.

If a is not a first node in the enumeration tree one can apply the same argument to the first ancestor b of a such that $S \neq \emptyset$. The focus node of $c = (b, x)$ is then a focus node of (a, x).

$\square$

An immediate consequence of Theorem 3 is that for all $i \in F_0^a$ and for all $j \in \text{orb}(\{i\}, \mathcal{G}(A(F_1^a, \emptyset)))$ one can set $x_j = 0$. We update orbital branching to include orbital fixing in Algorithm 2.

Algorithm 2. Orbital Branching with Orbital Fixing

Input: Subproblem $a = (F_1^a, F_0^a)$ (with free variables $N^a = I^n \setminus F_1^a \setminus F_0^a$), fractional solution $\hat{x}$.

Output: Two child nodes b and c.

Step 1. Compute orbital partition $\mathcal{O}(\mathcal{G}(A(F_1^a, \emptyset))) = \{\hat{O}_1, \hat{O}_2, \ldots, \hat{O}_q\}$. Let $S \stackrel{\text{def}}{=} \{j \in N^a \mid \exists k \in F_0^a \text{ and } (j \cap k) \in \hat{O}_\ell \text{ for some } \ell \in \{1, 2, \ldots q\}\}$.

Step 2. Compute orbital partition $\mathcal{O}(\mathcal{G}(A(F_1^a, F_0^a))) = \{O_1, O_2, \ldots, O_p\}$.

Step 3. Select orbit O_{j^*}, $j^* \in \{1, 2, \ldots, p\}$.

Step 4. Choose arbitrary $k \in O_{j^*}$. Return child subproblems $b = (F_1^a \cup \{k\}, F_0^a \cup S)$ and $c = (F_1^a, F_0^a \cup O_{j^*} \cup S)$.

In orbital fixing, the set S of additional variables set to zero is a function of F_0^a. Variables may appear in F_0^a due to a branching decision or due to traditional methods for variable fixing in integer programming, e.g. reduced cost fixing or implication-based fixing. Orbital fixing, then, gives a way to *enhance* traditional variable-fixing methods by including the symmetry present at a node of the branch and bound tree.

3.2 Comparison to Isomorphism Pruning

The fundamental idea behind isomorphism pruning is that for each node $a = (F_1^a, F_0^a)$, the orbits $\text{orb}(F_1^a, \mathcal{G}(A))$ of the "equivalent" sets of variables to F_1^a are computed. If there is a node $b = (F_1^b, F_0^b)$ elsewhere in the enumeration tree such that $F_1^b \in \text{orb}(F_1^a, \mathcal{G}(A))$, then the node a need not be evaluated—the node a is pruned by isomorphism. A very distinct and powerful advantage of this method is that *no* nodes whose sets of fixed variables are isomorphic will be evaluated. One disadvantage of this method is that computing $\text{orb}(F_1^a, \mathcal{G}(A))$ can require computational effort on the order of $O(n|F_1^a|!)$. A more significant disadvantage of isomorphism pruning is that $\text{orb}(F_1^a, \mathcal{G}(A))$ may contain many equivalent subsets to F_1^a, and the entire enumeration tree must be compared against this list to ensure that a is not isomorphic to any other node b. In a series of papers, Margot offers a way around this second disadvantage [9,10]. The key idea introduced is to declare one *unique representative* among the members of $\text{orb}(F_1^a, \mathcal{G}(A))$, and if F_1^a is not the unique representative, then the node a may safely be pruned. The advantage of this extension is that it is trivial to check whether or not node a may be pruned once the orbits $\text{orb}(F_1^a, \mathcal{G}(A))$ are computed. The disadvantage of the method is ensuring that the unique representative occurs *somewhere* in the branch and bound tree requires a relatively inflexible branching rule. Namely, *all* child nodes at a fixed depth must be created by branching on the *same* variable.

Orbital branching does not suffer from this inflexibility. By not focusing on pruning *all* isomorphic nodes, but rather eliminating the symmetry through branching, orbital branching offers a great deal more flexibility in the choice of branching entity. Another advantage of orbital branching is that by using the symmetry group $\mathcal{G}(A(F_1^a, F_0^a))$, symmetry *introduced* as a result of the branching process is also exploited.

Both methods allow for the use of traditional integer programming methodologies such as cutting planes and fixing variables based on considerations such as reduced costs and implications derived from preprocessing. In isomorphism pruning, for a variable fixing to be valid, it must be that *all* non-isomorphic optimal solutions are in agreement with the fixing. Orbital branching does not suffer from this limitation. A powerful idea in both methods is to combine the variable fixing with symmetry considerations in order to fix many additional variables. This idea is called *orbit setting* in [10] and *orbital fixing* in this work (see Sect. 3.1).

4 Implementation

The orbital branching method has been implemented using the user application functions of MINTO v3.1 [12]. The branching dichotomy of Algorithm 1 or 2 is implemented in the `appl_divide()` method, and reduced cost fixing is implemented in `appl_bounds()`. The entire implementation, including code for all the branching rules subsequently introduced in Sect. 4.2 consists of slightly over 1000 lines of code. All advanced IP features of MINTO were used, including *clique inequalities*, which can be useful for instances of (PIP).

4.1 Computing $\mathcal{G}(\cdot)$

Computation of the symmetry groups required for orbital branching and orbital fixing is done by computing the automorphism group of a related graph. Recall that the automorphism group $\mathrm{Aut}(G(V,E))$ of a graph $G = (V,E)$, is the set of permutations of V that leave the incidence matrix of G unchanged, i.e.

$$\mathrm{Aut}(G(V,E)) = \{\pi \in \Pi^{|V|} \mid (i,j) \in E \Leftrightarrow (\pi(i), \pi(j)) \in E\} \ .$$

The matrix A whose symmetry group is to be computed is transformed into a bipartite graph $G(A) = (N, M, E)$ where vertex set $N = \{1, 2, \ldots, n\}$ represents the variables, and vertex set $M = \{1, 2, \ldots, m\}$ represents the constraints. The edge $(i,j) \in E$ if and only if $a_{ij} = 1$. Under this construction, feasible solutions to (PIP) are subsets of the vertices $S \subseteq N$ such that each vertex $i \in M$ is adjacent to *at most* one vertex $j \in S$. In this case, we say that S *packs* M. Feasible solutions to (CIP) correspond to subsets of vertices $S \subseteq N$ such that each vertex $i \in M$ is adjacent to *at least* one vertex $j \in S$, or S *covers* M. Since applying members of the automorphism group preserves the incidence structure of a graph, if S packs (covers) M, and $\pi \in \mathrm{stab}(M, \mathrm{Aut}(G(A)))$, then there exists a $\sigma \in \Pi^m$ such that $\sigma(M) = M$ and $\pi(S)$ packs (covers) $\sigma(M)$. This implies that if $\pi \in \mathrm{stab}(M, \mathrm{Aut}(G(A)))$, then the restriction of π to N must be an element of $\mathcal{G}(A)$, i.e. using the graph $G(A)$, one can find elements of symmetry group $\mathcal{G}(A)$. In particular, we compute the orbital partition of the stabilizer of the constraint vertices M in the automorphism group of $G(A)$, i.e.

$$\mathcal{O}(\mathrm{stab}(M, \mathrm{Aut}(G(A)))) = \{O_1, O_2, \ldots, O_p\} \ .$$

The orbits $O_1, O_2, \ldots, O_p$ in the orbital partition are such that if $i \in M$ and $j \in N$, then i and j are not in the same orbit. We can then refer to these orbits as *variable* orbits and *constraint* orbits. In orbital branching, we are concerned only with the variable orbits.

There are several software packages that can compute the automorphism groups required to perform orbital branching. The program nauty [13], by McKay, has been shown to be quite effective [14], and we use nauty in our orbital branching implementation.

The complexity of computing the automorphism group of a graph is not known to be polynomial time. However, nauty was able to compute the symmetry groups of our problems very quickly, generally faster than solving an LP at a given node. One explanation for this phenomenon is that the running time of nauty's backtracking algorithm is correlated to the size of the symmetry group being computed. For example, computing the automorphism group of the clique on 2000 nodes takes 85 seconds, while graphs of comparable size with little or no symmetry require fractions of a second. The orbital branching procedure quickly reduces the symmetry group of the child subproblems, so explicitly recomputing the group by calling nauty is computational very feasible. In the table of results presented in the Appendix, we state explicitly the time required in computing automorphism groups by nauty.

4.2 Branching Rules

The orbital branching rule introduced in Sect. 3 leaves significant freedom in choosing the orbit on which to base the partitioning. In this section, we discuss mechanisms for deciding on which orbit to branch. As input to the branching decision, we are given a fractional solution $\hat{x}$ and orbits $O_1, O_2, \ldots O_p$ (consisting of all currently free variables) of the orbital partitioning $\mathcal{O}(\mathcal{G}(A(F_0^a, F_1^a)))$ for the subproblem at node a. Output of the branching decision is an index j^* of an orbit on which to base the orbital branching. We tested six different branching rules.

Rule 1: Branch Largest: The first rule chooses to branch on the largest orbit O_{j^*}:

$$j^* \in \arg \max_{j \in \{1, \ldots p\}} |O_j| \ .$$

Rule 2: Branch Largest LP Solution: The second rule branches on the orbit O_{j^*} whose variables have the largest total solution value in the fractional solution $\hat{x}$:

$$j^* \in \arg \max_{j \in \{1, \ldots p\}} \hat{x}(O_j) \ .$$

Rule 3: Strong Branching: The third rule is a strong branching rule. For each orbit j, two tentative child nodes are created and their bounds z_j^+ and z_j^- are computed by solving the resulting linear programs. The orbit j^* for which the product of the change in linear program bounds is largest is used for branching:

$$j^* \in \arg \max_{j \in \{1, \ldots p\}} (|e^T \hat{x} - z_j^+|)(|e^T \hat{x} - z_j^-|) \ .$$

Note that if one of the potential child nodes in the strong branching procedure would be pruned, either by bound or by infeasibility, then the bounds on the variables may be fixed to their values on the alternate child node. We refer to this as *strong branching fixing*, and in the computational results in the Appendix, we report the number of variables fixed in this manner. As discussed at the end of Sect. 3.1, variables fixed by strong branching fixing may result in additional variables being fixed by orbital fixing.

Rule 4: Break Symmetry Left: This rule is similar to *strong branching*, but instead of fixing a variable and computing the change in objective value bounds, we fix a variable and compute the change in the size of the symmetry group. Specifically, for each orbit j, we compute the size of the symmetry group in the resulting left branch if orbit j (including variable index i_j) was chosen for branching, and we branch on the orbit that reduces the symmetry by as much as possible:

$$j^* \in \arg \min_{j \in \{1, \ldots p\}} (|\mathcal{G}(A(F_1^a \cup \{i_j\}, F_0^a))|) \ .$$

Rule 5: Keep Symmetry Left: This branching rule is the same as **Rule 4**, except that we branch on the orbit for which the size of the child's symmetry group would remain the largest:

$$j^* \in \arg \max_{j \in \{1, \ldots p\}} (|\mathcal{G}(A(F_1^a \cup \{i_j\}, F_0^a))|) \ .$$

Rule 6: Branch Max Product Left: This rule attempts to combine the fact that we would like to branch on a large orbit at the current level and also keep a large orbit at the second level on which to base the branching dichotomy. For each orbit $O_1, O_2, \ldots, O_p$, the orbits $P_1^j, P_2^j, \ldots, P_q^j$ of the symmetry group $\mathcal{G}(A(F_1^a \cup \{i_j\}, F_0^a))$ of the left child node are computed for some variable index $i_j \in O_j$. We then choose to branch on the orbit j^* for which the product of the orbit size and the largest orbit of the child subproblem is largest:

$$j^* \in \arg \max_{j \in \{1,\ldots p\}} \left(|O_j| (\max_{k \in \{1,\ldots q\}} |P_k^j|) \right) \ .$$

5 Computational Experiments

In this section, we give empirical evidence of the effectiveness of orbital branching, we investigate the impact of choosing the orbit on which branching is based, and we demonstrate the positive effect of orbital fixing. The computations are based on the instances whose characteristics are given in Table 1. The instances beginning with `cod` are used to compute maximum cardinality binary error correcting codes [15], the instances whose names begin with `cov` are covering designs [16], the instance `f5` is the "football pool problem" on five matches [17], and the instances `sts` are the well-known Steiner-triple systems [18]. The `cov` formulations have been strengthened with a number of Schöenheim inequalities, as derived by Margot [19]. All instances, save for `f5`, are available from Margot's web site: `http://wpweb2.tepper.cmu.edu/fmargot/lpsym.html`.

The computations were run on machines with AMD Opteron processors clocked at 1.8GHz and having 2GB of RAM. The COIN-OR software `Clp` was used to solve the linear programs at nodes of the branch and bound tree. All code was compiled with the GNU family of compilers using the flags `-O3 -m32`. For each instance, the (known) optimal solution value was set to aid pruning and reduce the "random" impact of finding a feasible solution in the search. Nodes were searched in a best-first fashion. When the size of the maximum orbit in the orbital partitioning is less than or equal to two, nearly all of the symmetry in the problem has been eliminated by the branching procedure, and there is little use to perform orbital branching. In this case, we use MINTO's default branching strategy. The CPU time was limited in all cases to four hours.

Table 1. Symmetric Integer Programs

Name	Variables
cod83	256
cod93	512
cod105	1024
cov1053	252
cov1054	2252
cov1075	120
cov1076	120
cov954	126
f5	243
sts27	27
sts45	45

In order to succinctly present the results, we use performance profiles of Dolan and Moré [20]. A performance profile is a relative measure of the effectiveness of one solution method in relation to a group of solution methods on a fixed set of problem instances. A performance profile for a solution method m is essentially a plot of the probability that the performance of m (measured in this case with CPU time) on a given instance in the test suite is within a factor of β of the *best* method for that instance.

Figure 1 shows the results of an experiment designed to compare the performance of the six different orbital branching rules introduced in Sect. 4.2. In this experiment, both reduced cost fixing and orbital fixing were used. A complete table showing the number of nodes, CPU time, CPU time computing automorphism groups, the number of variables fixed by reduced cost fixing, orbital fixing, and strong branching fixing, and the deepest tree level at which orbital branching was performed is shown in the Appendix.

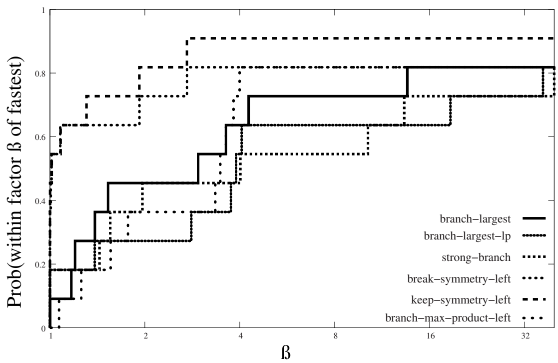

Fig. 1. Performance Profile of Branching Rules

A somewhat surprising result from the results depicted in Fig. 1 is that the most effective branching method was Rule 5, the method that keeps the symmetry group size large on the left branch. (This method gives the "highest" line in Fig. 1). The second most effective branching rule appears to be the rule that tries to *reduce* the group size by as much as possible. While these methods may not prove to be the most robust on a richer suite of difficult instances, one conclusion that we feel safe in making from this experiment is that considering *the impact* on the symmetry of the child node of the current branching decision is important. Another important observation is that for specific instances, the choice of orbit on which to branch can have a huge impact on performance. For example, for the instance cov1054, branching rules 4 and 5 both reduce the number of child nodes to 11, while other mechanisms that do not consider the impact of the branching decision on the symmetry of the child nodes cannot solve the problem in four hours of computing time.

The second experiment was aimed at measuring the impact of performing orbital fixing, as introduced in Sect. 3.1. Using branching rule 5, each instance in Table 1 was run both with and without orbital fixing. Figure 2 shows a performance profile comparing the results in the two cases. The results shows that orbital fixing has a significant positive impact.

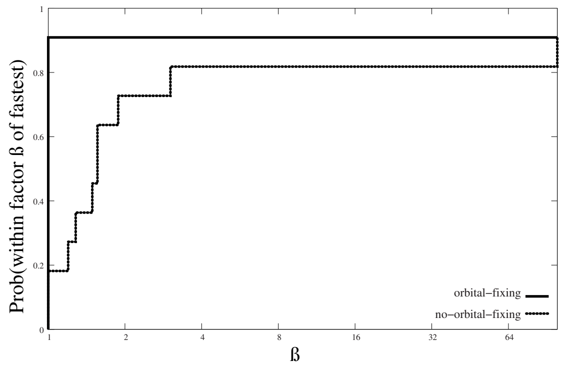

Fig. 2. Performance Profile of Impact of Orbital Fixing

The final comparison we make here is between orbital branching (with keep-symmetry-left branching), the isomorphism pruning algorithm of Margot, and the commercial solver CPLEX version 10.1, which has features for symmetry detection and handling. Table 2 summarizes the results of the comparison. The results for isomorphism pruning are taken directly from the paper of Margot using the most sophisticated of his branching rules "BC4" [10]. The paper [10] does not report results on `sts27` or `f5`. The CPLEX results were obtained on an Intel Pentium 4 CPU clocked at 2.40GHz. Since the results were obtained on three different computer architectures and each used a different LP solver for the child subproblems, the CPU times should be interpreted appropriately.

The results show that the number of subproblems evaluated by orbital branching is smaller than isomorphism pruning in three cases, and in nearly all cases, the number of nodes is comparable. For the instance `cov1076`, which is not solved by orbital branching, a large majority of the CPU time is spent computing symmetry groups at each node. In a variant of orbital branching that

Table 2. Comparison of Orbital Branching, Isomorphism Pruning, and CPLEX v10.1

	Orbital Branching		**Isomorphism Pruning**		**CPLEX v10.1**	
Instance	Time	Nodes	Time	Nodes	Time	Nodes
cod83	2	25	19	33	391	32077
cod93	176	539	651	103	**fail**	488136
cod105	306	11	2000	15	1245	1584
cov1053	50	745	35	111	937	99145
cov1054	2	11	130	108	**fail**	239266
cov1075	292	377	118	169	141	10278
cov1076	**fail**	13707	3634	5121	**fail**	1179890
cov954	22	401	24	126	9	1514
f5	66	935	-	-	1150	54018
sts27	1	71	-	-	0	1647
sts45	3302	24317	31	513	24	51078

Table 3. Performance of Orbital Branching Rules on Symmetric IPs

Instance	Branching Rule	Time	Nodes	Nauty Time	# Fixed by RCF	# Fixed by OF	# Fixed by SBF	Deepest Orbital Level
cod105	Break Symmetry	305.68	11	22.88	0	1020	0	4
cod105	Keep Symmetry	306.47	11	22.92	0	1020	0	4
cod105	Branch Largest LP Solution	283.54	7	11.87	0	0	0	2
cod105	Branch Largest	283.96	9	18.01	0	0	0	3
cod105	Max Product Orbit Size	302.97	9	17.41	0	920	0	3
cod105	Strong Branch	407.14	7	11.85	0	1024	1532	2
cod83	Break Symmetry	2.35	25	1.09	44	910	0	7
cod83	Keep Symmetry	2.38	25	1.10	44	910	0	7
cod83	Branch Largest LP Solution	8.81	93	2.76	209	534	0	6
cod83	Branch Largest	10.03	113	3.41	183	806	0	14
cod83	Max Product Orbit Size	9.39	115	4.59	109	634	0	11
cod83	Strong Branch	9.44	23	0.97	27	878	394	6
cod93	Break Symmetry	175.47	529	75.15	3382	3616	0	17
cod93	Keep Symmetry	175.58	529	75.31	3382	3616	0	17
cod93	Branch Largest LP Solution	3268.89	12089	1326.26	181790	3756	0	29
cod93	Branch Largest	2385.80	8989	920.90	142351	4986	0	49
cod93	Max Product Orbit Size	587.06	2213	215.68	28035	1160	0	29
cod93	Strong Branch	2333.22	161	19.76	380	2406	13746	14
cov1053	Break Symmetry	50.28	745	27.51	0	836	0	33
cov1053	Keep Symmetry	50.31	745	27.54	0	836	0	33
cov1053	Branch Largest LP Solution	1841.41	23593	990.12	0	5170	0	71
cov1053	Branch Largest	148.37	2051	70.73	0	1504	0	36
cov1053	Max Product Orbit Size	192.18	2659	91.72	0	1646	0	68
cov1053	Strong Branch	1998.55	1455	53.96	0	5484	34208	54
cov1054	Break Symmetry	1.77	11	0.85	0	186	0	4
cov1054	Keep Symmetry	1.76	11	0.85	0	186	0	4
cov1054	Branch Largest LP Solution	14400	54448	7600.80	0	814	0	35
cov1054	Branch Largest	14400	54403	7533.80	0	1452	0	49
cov1054	Max Product Orbit Size	14400	52782	7532.77	0	1410	0	38
cov1054	Strong Branch	14400	621	87.76	0	204	4928	32
cov1075	Break Symmetry	14400	9387	13752.11	37121	0	0	2
cov1075	Keep Symmetry	291.85	377	268.45	379	926	0	15
cov1075	Branch Largest LP Solution	906.48	739	861.57	1632	716	0	23
cov1075	Branch Largest	268.49	267	248.45	793	1008	0	13
cov1075	Max Product Orbit Size	395.11	431	366.24	1060	1066	0	21
cov1075	Strong Branch	223.53	67	60.71	106	128	1838	10
cov1076	Break Symmetry	14400	8381	13853.35	2	0	0	3
cov1076	Keep Symmetry	14400	13707	13818.47	11271	1564	0	26
cov1076	Branch Largest LP Solution	14400	6481	13992.74	10	116	0	14
cov1076	Branch Largest	14400	6622	13988.71	0	176	0	13
cov1076	Max Product Orbit Size	14400	6893	13967.86	71	580	0	14
cov1076	Strong Branch	14400	1581	3255.74	5	164	58	23
cov954	Break Symmetry	21.72	401	14.81	570	1308	0	14
cov954	Keep Symmetry	21.70	401	14.83	570	1308	0	14
cov954	Branch Largest LP Solution	11.30	175	7.03	498	48	0	5
cov954	Branch Largest	15.69	265	10.51	671	212	0	12
cov954	Max Product Orbit Size	14.20	229	9.25	602	212	0	11
cov954	Strong Branch	17.55	45	1.74	50	100	1084	8
f5	Break Symmetry	65.86	935	23.25	2930	2938	0	17
f5	Keep Symmetry	65.84	935	23.26	2930	2938	0	17
f5	Branch Largest LP Solution	91.32	1431	28.95	7395	272	0	8
f5	Branch Largest	100.66	1685	30.75	7078	434	0	11
f5	Max Product Orbit Size	102.54	1691	30.96	7230	430	0	13
f5	Strong Branch	671.51	123	2.59	187	760	8586	15
sts27	Break Symmetry	0.84	71	0.71	0	8	0	10
sts27	Keep Symmetry	0.83	71	0.71	0	8	0	10
sts27	Branch Largest LP Solution	2.33	115	2.12	3	86	0	14
sts27	Branch Largest	0.97	73	0.83	1	28	0	13
sts27	Max Product Orbit Size	2.88	399	2.42	1	888	0	11
sts27	Strong Branch	1.63	75	1.15	2	76	0	14
sts45	Break Symmetry	3302.70	24317	3230.12	12	0	0	4
sts45	Keep Symmetry	3301.81	24317	3229.88	12	0	0	4
sts45	Branch Largest LP Solution	4727.29	36583	4618.66	25	0	0	2
sts45	Branch Largest	4389.80	33675	4289.45	36	0	0	2
sts45	Max Product Orbit Size	4390.39	33675	4289.79	36	0	0	2
sts45	Strong Branch	1214.04	7517	884.79	2	144	45128	21

uses a symmetry group that is smaller but much more efficient to compute (and which space prohibits us from describing in detail here), cov1076 can be solved in 679 seconds and 14465 nodes. Since in any optimal solution to the Steiner triple systems, more than 2/3 of the variables will be set to 1, orbital branching would be much more efficient if all variables were complemented, or equivalently if the orbital branching dichotomy (2) was replaced by its complement. Margot [10] also makes a similar observation, and his results are based on using the complemented instances, which may account for the large gap in performance of the two methods on `sts45`. We are currently instrumenting our code to deal with instances for which the number of ones in an optimal solution is larger than 1/2. Orbital branching proves to be faster than CPLEX in six cases, while in all cases the number of evaluated nodes is remarkably smaller.

6 Conclusion

In this work, we presented a simple way to capture and exploit the symmetry of an integer program when branching. We showed through a suite of experiments that the new method, orbital branching, outperforms state-of-the-art solvers when a high degree of symmetry is present. In terms of reducing the size of the search tree, orbital branching seems to be of comparable quality to the isomorphism pruning method of Margot [10]. Further, we feel that the simplicity and flexibility of orbital branching make it an attractive candidate for further study. Continuing research includes techniques for further reducing the number of isomorphic nodes that are evaluated and on developing branching mechanisms that combine the child bound improvement and change in symmetry in a meaningful way.

Acknowledgments

The authors would like to thank Kurt Anstreicher and François Margot for inspiring and insightful comments on this work. In particular, the name *orbital branching* was suggested by Kurt. Author Linderoth would like to acknowledge support from the US National Science Foundation (NSF) under grant DMI-0522796, by the US Department of Energy under grant DE-FG02-05ER25694, and by IBM, through the faculty partnership program. Author Ostrowski is supported by the NSF through the IGERT Grant DGE-9972780.

References

1. Barnhart, C., Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W.P., Vance, P.H.: Branch and Price: Column generation for solving huge integer programs. Operations Research **46** (1998) 316–329
2. Holm, S., Sørensen, M.: The optimal graph partitioning problem: Solution method based on reducing symmetric nature and combinatorial cuts. OR Spectrum **15** (1993) 1–8
3. Méndez-Díaz, I., Zabala, P.: A branch-and-cut algorithm for graph coloring. Discrete Applied Mathematics **154**(5) (2006) 826–847
4. Macambira, E.M., Maculan, N., de Souza, C.C.: Reducing symmetry of the SONET ring assignment problem using hierarchical inequalities. Technical Report ES-636/04, Programa de Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro (2004)
5. Rothberg, E.: Using cuts to remove symmetry. Presented at the 17$^{\text{th}}$ International Symposium on Mathematical Programming
6. Sherali, H.D., Smith, J.C.: Improving zero-one model representations via symmetry considerations. Management Science **47**(10) (2001) 1396–1407

7. Kaibel, V., Pfetsch, M.: Packing and partitioning orbitopes. Mathemathical Programming (2007) To appear.
8. Kaibel, V., Peinhardt, M., Pfetsch, M.: Orbitopal fixing. In: IPCO 2007: The Twelfth Conference on Integer Programming and Combinatorial Optimization, Springer (2007) To appear.
9. Margot, F.: Pruning by isomorphism in branch-and-cut. Mathematical Programming **94** (2002) 71–90
10. Margot, F.: Exploiting orbits in symmetric ILP. Mathematical Programming, Series B **98** (2003) 3–21
11. Bazaraa, M.S., Kirca, O.: A branch-and-bound heuristic for solving the quadratic assignment problem. Naval Research Logistics Quarterly **30** (1983) 287–304
12. Nemhauser, G.L., Savelsbergh, M.W.P., Sigismondi, G.C.: MINTO, a Mixed INTeger Optimizer. Operations Research Letters **15** (1994) 47–58
13. McKay, B.D.: Nauty User's Guide (Version 1.5). Australian National University, Canberra. (2002)
14. Foggia, P., Sansone, C., Vento, M.: A preformance comparison of five algorithms for graph isomorphism. Proc. 3rd IAPR-TC15 Workshop Graph-Based Representations in Pattern Recognition (2001) 188–199
15. Litsyn, S.: An updated table of the best binary codes known. In Pless, V.S., Huffman, W.C., eds.: Handbook of Coding Theory. Volume 1. Elsevier, Amsterdam (1998) 463–498
16. Mills, W.H., Mullin, R.C.: Coverings and packings. In: Contemporary Design Theory: A Collection of Surveys. Wiley (1992) 371–399
17. Hamalainen, H., Honkala, I., Litsyn, S., Östergård, P.: Football pools—A game for mathematicians. American Mathematical Monthly **102** (1995) 579–588
18. Fulkerson, D.R., Nemhauser, G.L., Trotter, L.E.: Two computationally difficult set covering problems that arise in computing the 1-width of incidence matrices of Steiner triples. Mathematical Programming Study **2** (1973) 72–81
19. Margot, F.: Small covering designs by branch-and-cut. Mathematical Programming **94** (2003) 207–220
20. Dolan, E., Moré, J.: Benchmarking optimization software with performance profiles. Mathematical Programming **91** (2002) 201–213