Computer Sciences Department

Orbital Branching

James Ostrowski
Jeff Linderoth
Fabrizio Rossi
Stefano Smriglio

Technical Report #1641

August 2008

UNIVERSITY OF WISCONSIN MADISON

University of Wisconsin, Dept. of Computer Science
Technical Report

James Ostrowski[1] · Jeff Linderoth[2] · Fabrizio Rossi[3] · Stefano Smriglio[4]

Orbital Branching

August 3, 2008

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.

Key words. Integer programming – symmetry – branch-and-bound algorithms

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 $A(\sigma, \pi)$ be the matrix obtained by permuting the rows of A by σ and the columns of A by π, i.e. $A(\sigma, \pi) = P_\sigma A P_\pi$, where P_σ and P_π are the permutation matrices corresponding to σ and π respectively. The *symmetry group* $\mathcal{G}$ of the matrix A is the set of permutations

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

Department of Industrial and Systems Engineering, Lehigh University e-mail: jao204@lehigh.edu
Department of Computer Science, University of Wisconsin-Madison, e-mail: linderoth@wisc.edu
Dipartimento di Informatica, Università di L'Aquila e-mail: rossi@di.univaq.it
Dipartimento di Informatica, Università di L'Aquila e-mail: smriglio@di.univaq.it

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. Furthermore, symmetric solutions appear again and again all over the tree. 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 [1,7,17]. In addition, there is a body of research on valid inequalities that can help exclude symmetric feasible solutions [12,21,23]. Kaibel and Pfetsch [9] 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 [13,14] in the context of IP and dating back to Bazaraa and Kirca [2], 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 only one node in the equivalence class of the group will 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.

A preliminary version of this work has been published in the conference proceedings of the the Twelfth Conference on Integer Programming and Combinatorial Optimization (IPCO) [20]. The remainder of the paper is divided into six sections. In Section 2 we give some mathematical preliminaries. Orbital branching is introduced and formalized in Section 3. Enhancements to orbital branching are discussed in Section 4, and a more complete comparison to isomorphism pruning is also presented there. Implementation details are provided in Section 5, and computational results are presented in Section 6. Conclusions about the impact of orbital branching and future research directions are given in Section 7.

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 $\varGamma$, but for concreteness, the reader may consider the group $\varGamma$ 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 $\varGamma$ is the set of all subsets of I^n to which S can be sent by permutations in $\varGamma$, i.e.,

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

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 $j \in \mathrm{orb}(\{k\},\varGamma)$, then $k \in \mathrm{orb}(\{j\},\varGamma)$, i.e. the variable x_j and x_k share the same orbit. Therefore, the union of the orbits

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

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

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

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

The stabilizer of S is a subgroup of $\varGamma$.

Throughout this paper, we display permutations in *cyclic notation*. The expression $(a_1,\ a_2,\ \ldots,\ a_k)$ denotes a cycle which sends a_i to a_{i+1} for $i = 1,\ \ldots,\ k-1$ and sends a_k to a_1. Some permutations may be written as a product of cycles. We will omit all 1-element cycles from our display.

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 $(k+1)^{\text{st}}$ 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.

3.1. Orbital Branching: Description

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). When we remove columns from the matrix, we do not change the index on any of the remaining columns. Unfortunatly, performing this processing at every nodes adds some notational difficulties as the dimension of the constraint matrix is always changing. It should be clear that the mapping $\phi : (0,1)^n \rightarrow (0,1)^{|N^a|}$ with $\phi(x)_i = x_i$ for all $i \in N^a$ maps feasible solutions with respect to A to feasible solutions with respect to $A(F_1^a, F_0^a)$. Similarly, any permutation $\pi \in \mathcal{G}(A(F_1^a, F_0^a))$, which permutes only the set of elements N^a, can be extended to permute the set of elements I^n by, for every $i \in F_1^a \cup F_0^a$ having π map element i to itself. For this reason as well as the sake of clarity, we will think of all permutations as acting on the set I^n and we will not differentiate solutions with are feasible at node a with solutions feasible with resepect to $A(F_1^a, F_0^a)$.

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. If $a \neq \{b, c\}$ then there $\nexists x \in \mathcal{F}(b)$ such that $\exists \pi \in \mathcal{G}(A(F_1^a, F_0^a))$ with $\pi(x) \in \mathcal{F}(c)$.*

Proof. Suppose not, i.e., that there $\exists x \in \mathcal{F}(b)$ and a permutation $\pi \in \mathcal{G}(A(F_1^a, F_0^a))$ 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 $k \in O_i$, that is, b is in the left branch of a. 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.

3.2. Orbital Branching: An Illustrative Example

Example 1. In order to demonstrate the effects of orbital branching, consider the graph $G = (V, E)$ of Figure 1 and the associated PIP:

$$\begin{aligned} \max \sum_{i \in V} & x_i \\ x_i + x_j &\leq 1 \qquad \forall \{i, j\} \in E, \\ x_i &\in \{0, 1\} \qquad \forall i \in V \end{aligned}$$

which corresponds to computing the stability number of G.

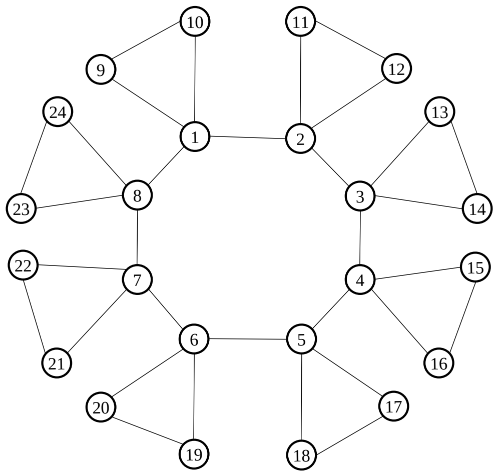


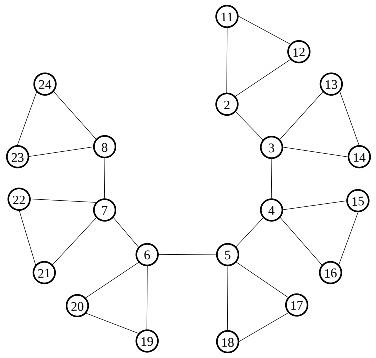
Fig. 1. Example

Applying Step 1 of Algorithm 1 at the root subproblem $F_1^a = F_0^a = \emptyset$ results in a group $\mathcal{G}(A)$ containing 4096 permutations and an orbital partition $\mathcal{O}(\mathcal{G}(A))$ containing two orbits, namely, $O_1 = \{1,\ldots,8\}$ and $O_2 = \{9, \ldots, 24\}$. Thanks to the structure of the matrix A, in which each constraint corresponds to an edge of G, the orbits of $\mathcal{G}(A)$ can be intuitively visualized on the graph.

Step 2 of Algorithm 1 selects an orbit on which to base the branching dichotomy. Suppose the largest orbit O_2 is chosen, and the branching index $k = 9 \in O_2$ is used. Then, two subproblems b and c are generated as follows: $F_1^b = \{9\}$ and $F_0^b = \emptyset$; $F_1^c = \emptyset$ and $F_0^c = \{9, \ldots, 24\}$. The structure of subproblems b and c, where fixed variables have been removed, is drawn in Figure 2.

The advantage of orbital branching over classical branching on a variable is highlighted by completely executing two branch-and-bound algorithms on the PIP of Example 1. We assume that a feasible solution of (optimal) value 8 is found at the root node. In the first algorithm the branching decision is carried out by orbital branching where Step 2 selects the largest orbit. In the second algorithm, ordinary branching is performed on the variable corresponding to the vertex of G with maximum degree in the remaining graph, typically effective for stable set problems [22]. In Figures 3 and 4, the complete enumeration trees obtained respectively by orbital branching and branching on variable are drawn. At each node a, we report the variables fixed (F_1^a, F_0^a) and the value of the LP relaxation z_{LP}. Orbital branching results in fewer evaluated subproblems: 21 vs. 49 for the variable-branching dichotomy.

An insightful explanation of orbital branching's improved performance is obtained by examining the structure of subproblems. For instance, Figure 5 shows the graphs remaining at subproblems 9 and 19 of the variable-branching enumeration tree. The



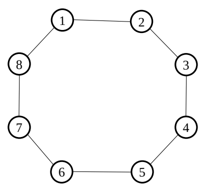


Fig. 2. Child subproblems

graphs are isomorphic, but both subproblems are evaluated when branching on variables. On the contrary, orbital branching breaks such a symmetry at the root subproblem. The complete catalog of graphs and orbital partitions for each subproblem in the orbital branching branch-and-bound tree is reported in the Appendix. Looking at the catalog of subproblems, one can observe that no isomorphic subproblems are evaluated when orbital branching is used on this example. This is not, however, true in general.

4. Enhancements to Orbital Branching

In this section, we demonstrate how additional variables may be fixed during branch and bound by considering the implications of symmetry. We also discuss how to perform orbital branching by considering a subgroup of the original symmetry group. We compare orbital branching to a related technique for combating symmetry in integer programs, isomorphism pruning. The section concludes with a brief discussion on how to most effectively employ orbital branching on integer programs whose optimal solution has a large support.

4.1. 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.

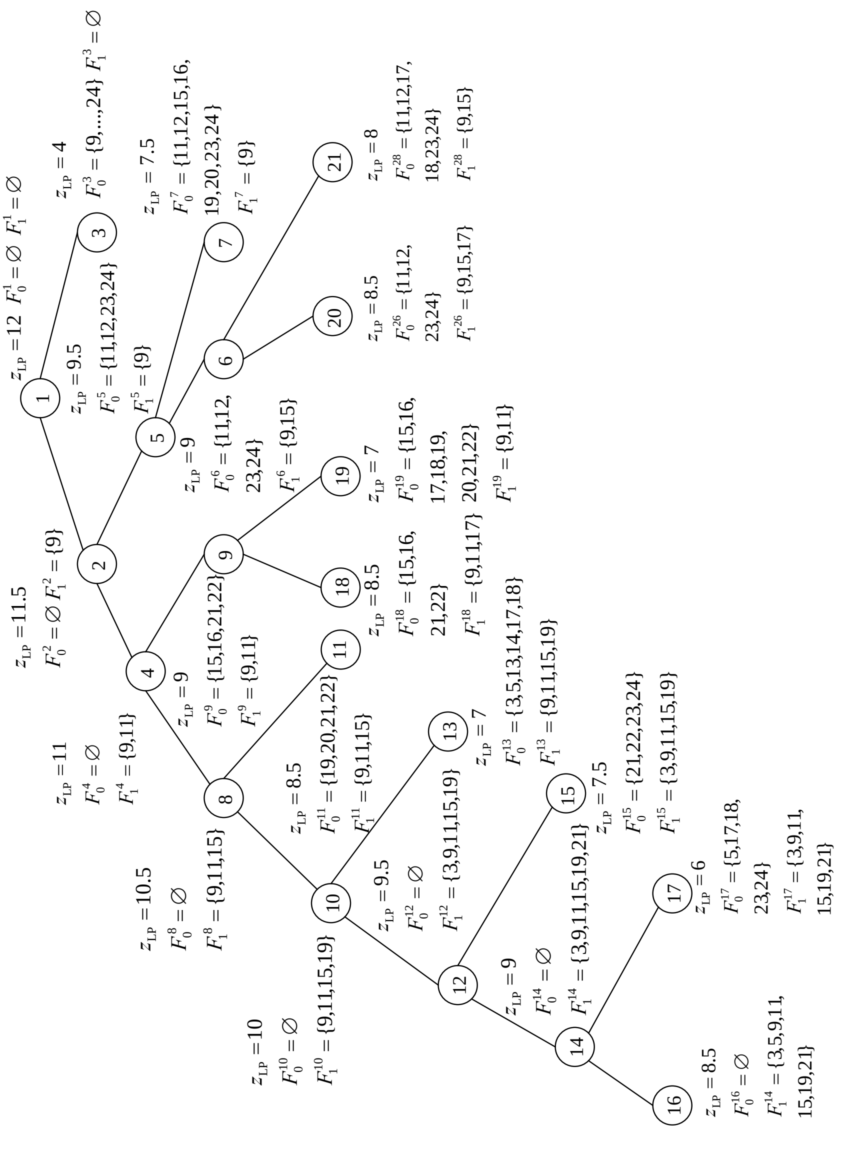


Fig. 3. Enumeration tree with orbital branching

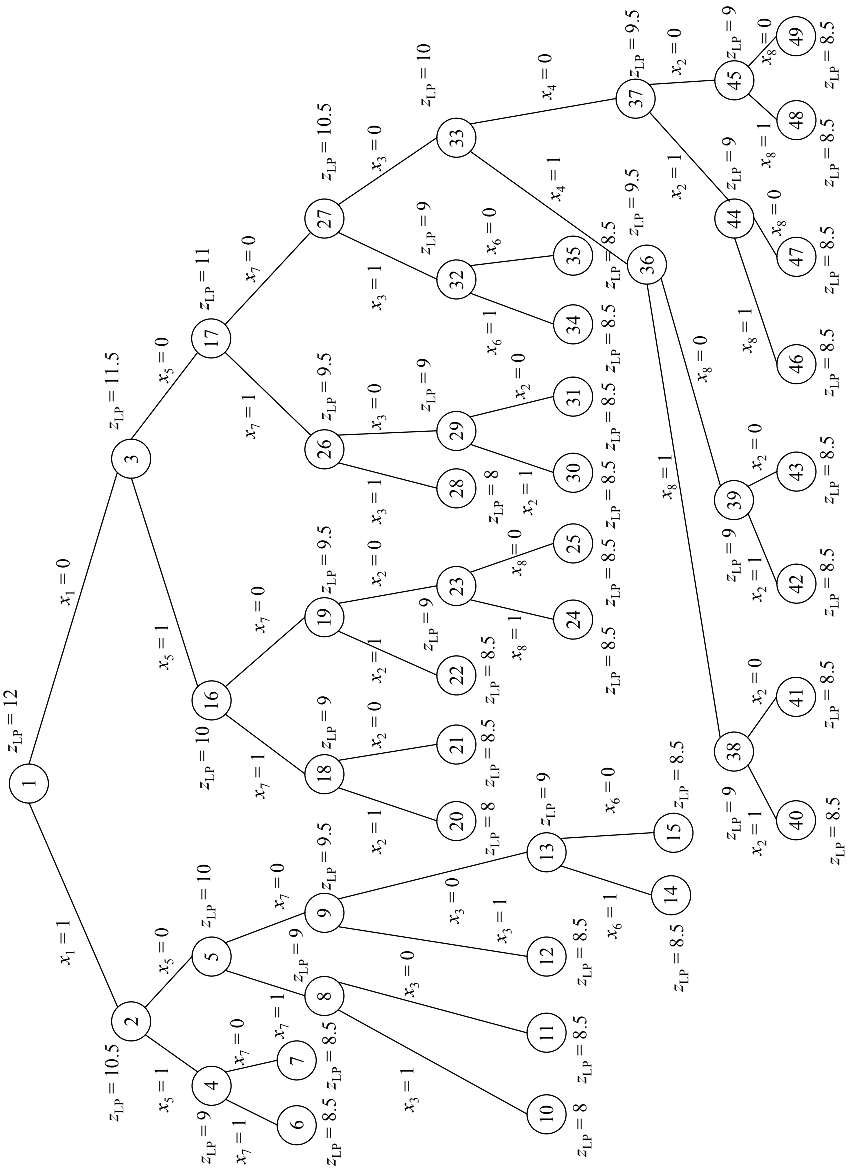
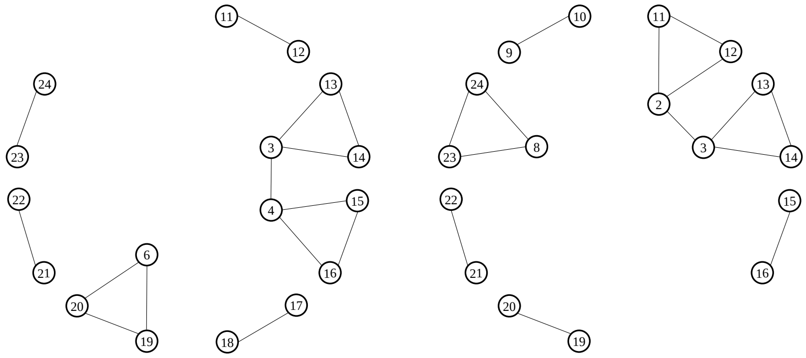


Fig. 4. Enumeration tree with branching on variable



subproblem 9 subproblem 19

Fig. 5. isomorphic subproblems from branching on variable

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

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

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 either there exists a focus node for a with respect to x or x is not an optimal solution.

Proof:

Let $S \neq \emptyset$. Then, there exist $j \in F_0^a$ and $i \in S$ such that $i \in \mathrm{orb}(j, \mathcal{G}(A(F_1^a, \emptyset)))$, 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 first subproblem in which j is fixed. Let $\rho(c)$ be the parent node of c. By our choice of j as the first fixed variable, $\{\pi(i) | \forall i \text{ s.t.} x_i = 1 \text{ and } \pi \in \mathcal{G}(A(F_1^a, \emptyset))\} \cap F_0^{\rho(c)} = \emptyset$. Therefore, $\pi(x)$ is not feasible in a since it does not satisfy the bounds, but is feasible in $\rho(c)$ and has the same objective value of x.

The variable x_j could have been fixed either (i) as a result of a branching decision, or (ii) it was deduced that no optimal solution exists with $x_j = 1$ at node $\rho(c)$ (and the fixing applied to the child nodes), or (iii) by orbital fixing (at ρ).

– (i) If j was fixed by orbital branching then the left child of $\rho(c)$ has $x_h = 1$ for some $h \in \operatorname{orb}(j, \mathcal{G}(A(F_1^{\rho(c)}, F_0^{\rho(c)})))$. 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.
– (ii) If it was deduced that no optimal solution feasible at $\rho(c)$ exists with $x_j = 1$, then, since $\pi(x)$ is feasible in $\rho(c)$ with $x_j = 1$, and π preserves objective value, x cannot be an optimal solution.
– (iii) Lastly, j could have been fixed by orbital fixing. This implies that the set S is nonempty in $\rho(c)$ and the argument can be repeated until the first ancestor d of a is reached such that F_0^d does not contain variables fixed by orbital fixing. Therefore, a sequence of permutations $\pi^1, \ldots, \pi^r$ have been found such that $\pi^r\pi^{r-1}\ldots\pi^1\pi(x)$ is feasible in d and has the same value of x.
Then, either argument (i) or (ii) can be applied, that is, either there is a focus node f of d with respect to $\pi^r\pi^{r-1}\ldots\pi^1\pi(x)$ (which would also be a focus node for a with respect to x), or j was fixed by an optimality condition (which implies $\pi^r\pi^{r-1}\ldots\pi^1\pi(x)$ and thus x are not optimal).

There may be elements in S which do not share an orbit with j. One can show that these elements can also be fixed by adding the fixed variables to F_0, updating S, and repeating the argument. As long as S is non-empty, each iteration will fix at least one variable.

An immediate consequence of Theorem 3 is that for all $i \in F_0^a$ and for all $j \in \operatorname{orb}(i, \mathcal{G}(A(F_1^a, \emptyset)))$ 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 depends on 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.

Example (continued) When orbital branching with orbital fixing is applied to the PIP of Example 1, it generates the enumeration tree drawn in Figure 6. Orbital fixing is performed at subproblem 6, a node that has $F_0^6 = \{11, 12, 23, 24\}$ and $F_1^6 = \{9, 15\}$. The group $\mathcal{G}(A(F_1^6, \emptyset))$ yields the orbits: $\{2, 3\}$ $\{5, 8\}$ $\{6, 7\}$ $\{11, 12, 13, 14\}$

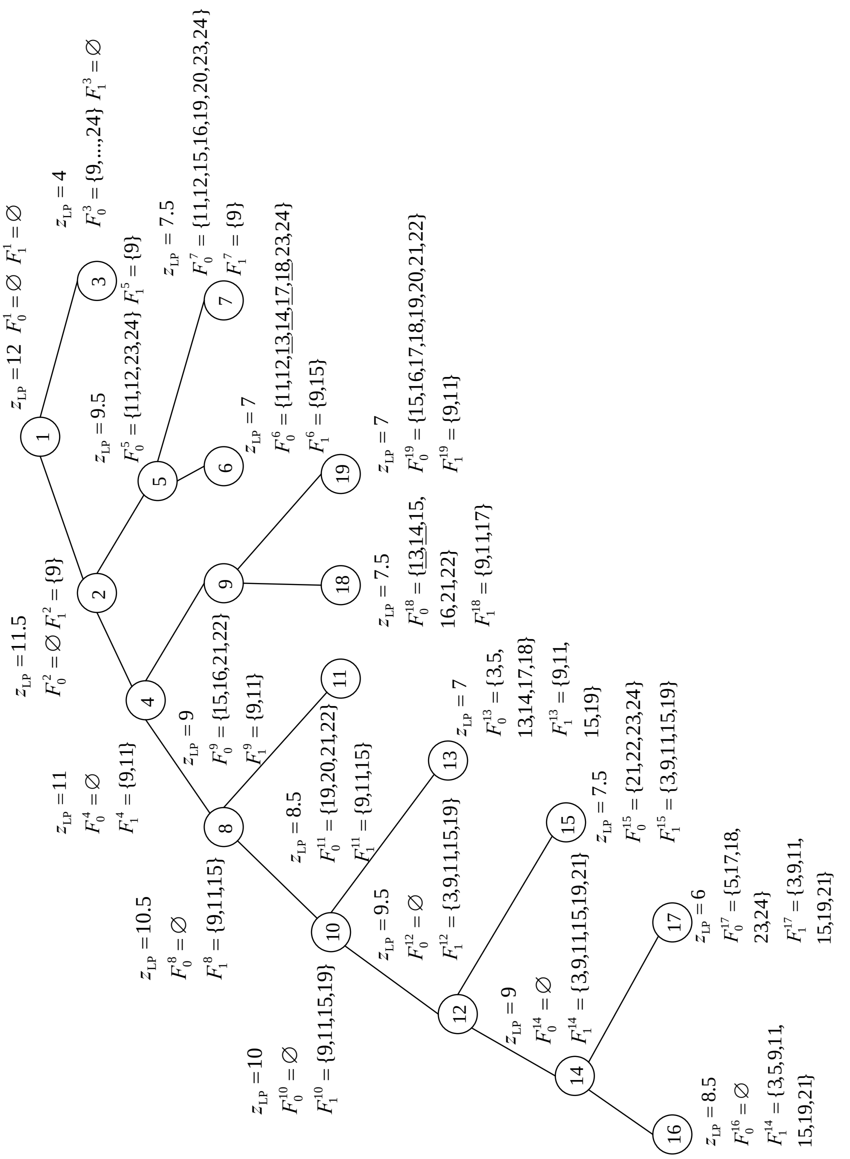


Fig. 6. Enumeration tree with orbital branching and orbital fixing

$\{17, 18, 23, 24\}$ $\{19, 20, 21, 22\}$. The orbit $\{11, 12, 13, 14\}$ contains variables that have already been set to zero: $\{11, 12, 13, 14\} \cap F_0^a = \{13, 14\}$. Therefore, the variables x_{13} and x_{14} are fixed to 0 by orbital fixing. In the same way, looking at the orbit $\{17, 18, 23, 24\}$, orbital fixing sets variables x_{17} and x_{18} to 0. All the variables fixed to 0 by orbital fixing are underlined in Figure 6.

The effect of orbital fixing is clear at subproblem 6, where the optimal value of the LP relaxation reduces from 9 to 7, as compared to the algorithm without orbital fixing, avoiding further branching (see the tree of Figure 3).

The example also helps illustrate the existence of a focus node if orbital fixing is performed (Theorem 3). Define a as the subproblem found at node 6. The set of variables fixed by orbital fixing is $S = \{13, 14, 17, 18\}$. Consider the solution $x \in \mathcal{F}(a)$: $x_2 = x_5 = x_8 = x_9 = x_{13} = x_{15} = x_{19} = x_{21} = 1$, and all other variables set to 0. Following the proof of Theorem 3, we have $i = 13$ and $j \in \mathrm{orb}(\{i\}, \mathcal{G}(A(F_1^a, \emptyset))$, i.e., $j = 12$. A permutation $\pi \in \mathcal{G}(A(F_1^a, \emptyset))$ such that $\pi(i) = j$ is: $[(2, 3), (12, 13), (11, 14)]$. We have $\bar{x} = \pi(x)$, that is, $\bar{x}_2 = \bar{x}_5 = \bar{x}_8 = \bar{x}_{12} = \bar{x}_{19} = \bar{x}_{21} = 1$, and all other variables set to 0. Notice that $\bar{x} \notin \mathcal{F}(a)$, since $x_{12} = 1$. By definition, subproblem 5 is the subproblem c in the proof of Theorem 3, and subproblem 2 is the subproblem $\rho(c)$. Then, we have $h = 11$ and π' can be defined as: $(11, 12)$. Finally, $\tilde{x} = \pi'(\pi(x))$ is: $\bar{x}_3 = \bar{x}_7 = \bar{x}_9 = \bar{x}_{11} = \bar{x}_{15} = \bar{x}_{17} = \bar{x}_{19} = \bar{x}_{23} = 1$, and all other variables set to 0. This is feasible for subproblem 4. Thus, 4 is a focus node for a.

4.2. Using a Subgroup of the Original Symmetry Group

We delay discussion of the computation of the symmetry groups $\mathcal{G}(A(F_1^a, F_0^a))$ until Section 5.1, but we simply note at this point that all known algorithms which compute the symmetry group of a given graph have exponential running time. Thus, computing the symmetry group $\mathcal{G}(A(F_1^a, F_0^a))$ at each node a may be computationally prohibitive. We will show via computational results in Section 6 that this is often not the case. In the case that recomputing the full symmetry group $\mathcal{G}(A(F_1^a, F_0^a))$ is too costly, there is an alternative. Instead, orbital branching can use the symmetry group $\mathrm{stab}(F_1^a, \mathcal{G}(A))$ to create orbits at every node in the tree. In this method, the original, global, symmetry group $\mathcal{G}(A)$ is only computed once, at the root node, and the stabilizers are computed given the original symmetry group. This is typically more computationally efficient than re-computing the symmetry groups from scratch. In the following, in order to distinguish between the two symmetry groups that could be used in orbital branching at node a, we will refer to branching using $\mathrm{stab}(F_1^a, \mathcal{G}(A))$ as *global branching* (because we use only the symmetry group found at the root node), and branching using $\mathcal{G}(A(F_1^a, F_0^a))$ as *local branching*.

The decreased computational overhead in orbit calculations when using the global symmetry group comes at a price. As Theorems 4 and 5 demonstrate, the orbits from the global group $\mathrm{stab}(F_1^a, \mathcal{G}(A))$ are a subdivision of the orbits used for orbital fixing and orbital branching, so the branching dichotomy and fixing mechanisms are weaker.

Theorem 4. *If $O \in \mathcal{O}(\mathrm{stab}(F_1^a, \mathcal{G}(A)))$ and $O \cap F_1^a = \emptyset$, then $\exists\, O' \in \mathcal{O}(\mathcal{G}(A(F_1^a, \emptyset)))$ with $O \subseteq O'$.*

Proof. We prove Theorem 4 by proving the following equivalent statement: if $\exists \pi \in \mathrm{stab}(F_1^a, \mathcal{G}(A))$ with $\pi(i) = j$ and $i, j \notin F_1^a$, then $\exists \pi' \in \mathcal{G}(A(F_1^a, \emptyset))$ with $\pi(i) = j$. Let $\pi \in \mathrm{stab}(F_1^a, \mathcal{G}(A))$ be such that $\pi(i) = j$ and $i, j \notin F_1^a$. Since $\pi(F_1^a) = F_1^a$ we can restrict π by ignoring its action on F_1^a. Let C^a be the collection of inequalities which have been removed (become redundant) either at a or any parent of a. Since $\pi \in \mathrm{stab}(F_1^a, \mathcal{G}(A))$, there exists a $\sigma \in \Pi^m$ such that $A(\sigma, \pi) = A$. Each constraint, $c^T x \geq (\leq) 1$, in C^a contains at least one variable in F_1^a. This constraint gets mapped to $\sigma(c)^T \pi(x) \geq (\leq) 1$, a constraint still containing at least one variable in F_1^a, hence a constraint in C^a. We can then restrict σ by ignoring its action on the constraint set C^a. Call the pair of restricted permutations π' and σ'. These permutations act on the same set of variables and constraints as $\mathcal{G}(A(F_1^a, \emptyset))$. We also have that $A(F_1^a, \emptyset)(\pi', \sigma') = A(F_1^a, \emptyset)$, so $\pi' \in \mathcal{G}(A(F_1^a, \emptyset))$ with $\pi(i) = j$. □

Orbital fixing does not change the result of Theorem 4. Specifically, if S^a is the set of indices of variables fixed to zero by orbital fixing at node a, then the orbits from the group $\mathcal{G}(A(F_1^a, \emptyset))$ are a subdivision of orbits from the group $\mathcal{G}(A(F_1^a, F_0^a \cup S^a))$.

Theorem 5. *Let S^a be the set of variables fixed to zero by orbital fixing at node a. If $O \in \mathcal{O}(\mathcal{G}(A(F_1^a, \emptyset))), \exists O' \in \mathcal{O}(\mathcal{G}(A(F_1^a, F_0^a \cup S^a)))$ with $O \subseteq O'$.*

Proof. We prove Theorem 5 by proving the equivalent statement that if $\exists \pi \in \mathcal{G}(A(F_1^a, \emptyset))$ with $\pi(i) = j$ and $i, j \notin S^a$, then $\exists \pi' \in \mathcal{G}(A(F_1^a, F_0^a \cup S^a))$ with $\pi(i) = j$. Let $\pi \in \mathcal{G}(A(F_1^a, \emptyset))$ with $\pi(i) = j$. We can restrict π by ignoring its actions on the set $F_0^a \cup S^a$. Call the restricted permutation π'. Let C^a be the collection of inequalities which have been removed (become redundant) either at a or any parent of a. We know that there exists a $\sigma \in \Pi^{m-|C^a|}$ such that $A(F_1^a, \emptyset)(\pi, \sigma) = A(F_1^a, \emptyset)$. Since $A(F_1^a, F_0^a)$ contains the same rows as $A(F_1^a, \emptyset)$, we have that $A(F_1^a, F_0^a)(\pi', \sigma) = A(F_1^a, F_0^a)$. □

4.3. Comparison to Isomorphism Pruning

The fundamental idea behind isomorphism pruning is that for each node $a = (F_1^a, F_0^a)$, the orbits $\mathrm{orb}(F_1^a, \mathcal{G}(A))$ 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 $F_1^b \in \mathrm{orb}(F_1^a, \mathcal{G}(A))$, 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 variables fixed to 1 are isomorphic will be evaluated. One disadvantage of this method is that computing $\mathrm{orb}(F_1^a, \mathcal{G}(A))$ can require significant computational effort. Further the set $\mathrm{orb}(F_1^a, \mathcal{G}(A))$ 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 [13,14]. The key idea introduced is to declare one *unique representative* among the members of $\mathrm{orb}(F_1^a, \mathcal{G}(A))$, and if F_1^a is not the unique representative, then the node a may safely be pruned. The oracle that checks if F_1^a a unique representative among $\mathrm{orb}(F_1^a, \mathcal{G}(A)$ runs in polynomial time. 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 [14] and *orbital fixing* in this work (see Sec. 4.1).

4.4. Reversing Orbital Branching

One of the advantages of orbital branching is that the "right" branch, in which all variables in the branching orbit O are fixed to zero, typically changes the optimal value of the LP relaxation significantly, and the left branch, in which one variable in O is fixed to one also has a significant impact on the problem. In some classes of PIP or CIP, fixing a variable to zero can have more impact than fixing a variable to one. This is typically true in instance in which the number of ones in an optimal solution is larger than 1/2 the number of variables. In such cases, 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 [14] also makes a similar observation for his isomorphism pruning algorithm, and he solves the complemented versions of such instances. In orbital branching, we opt for the former way of exploiting this fact, and the "left" branch fixes one variable to zero, and orbital fixing fixes variables to one instead of zero.

5. Implementation

The orbital branching method has been implemented using the user application functions of MINTO v3.1 [19]. 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 Section 5.2 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). In this section, we discuss the features of the implementation that are specific to orbital branching—the computation of the symmetry groups and orbital branching rules.

5.1. 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, vertex set $M = \{n+1, n+2, \ldots, n+m\}$ represents the constraints, and 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 [16], by McKay, has been shown to be quite effective [4], 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.

5.2. Branching Rules

The orbital branching rule introduced in Section 3 leaves significant freedom in choosing the orbit on which to base the branching (Step 2 of Algorithm 1). 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 $O_1, O_2, \dots O_p$ (consisting of all currently free variables) of the orbital partition $\mathcal{O}(\mathcal{G}(A(F_1^a, F_0^a)))$ 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^*}:

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

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 $\hat{x}$:

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

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:

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

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 Section 4.1, 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:

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

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:

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

Name	Variables
cod83	256
cod93	512
cod105	1024
cov1053	252
cov1054	2252
cov1075	120
cov1076	120
cov954	126
f5	243
sts45	45
sts63	63
sts81	81

Table 1. Symmetric Integer Programs

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:

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

6. 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 [11], the instances whose names begin with `cov` are covering designs [18], the instance `f5` is the "football pool problem" on five matches [6], and the instances `sts` are used to compute the incidence width of the well-known Steiner-triple systems [5]. The `cov` formulations have been strengthened with a number of Schöenheim inequalities, as derived by Margot [15]. The `sts` instances typically have roughly 2/3 of the variables equal to one in an optimal solution, so for these instances, we reverse the orbital branching dichotomy, as explained in Section 4.4. 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. For each instance, the (known) optimal solution value was set a priori to aid pruning and reduce the random impact of finding a feasible solution in the search. Nodes were searched in a depth-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 in performing orbital branching. In this case, we use MINTO's default branching strategy [10]. If orbital branching is not performed at a node, then there is little likelihood that it will be effective at the node's children. In this case, we save the computational overhead of re-computing the symmetry group, and simply allow MINTO to choose a branching variable. The CPU time was limited in all cases to four hours.

Table 2 shows the results of an experiment designed to compare the performance of the six different orbital branching rules introduced in Section 5.2. In this experiment, reduced cost fixing, orbital fixing, and the local symmetry group $\mathcal{G}(A(F_1^a, F_0^a))$ were used, and the CPU time required (in seconds) for orbital branching to solve each instance in the test suite for the six different is reported. 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 for a variety of parameter settings is shown in Table 6 in the Appendix.

Instance	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6
cod83	11	4	5	6	8	5
cod93	1677	1557	2368	3269	242	399
cod105	239	238	345	255	424	229
cov954	5	4	24	8	17	5
cov1053	103	617	768	346	105	90
cov1054	14400	14400	14431	14400	181	14400
cov1075	69	50	216	14400	210	128
cov1076	14400	14400	14400	14400	1560	14400
f5	64	80	668	42	34	64
sts45	8	8	95	8	8	8
sts63	93	91	1132	1630	161	137
sts81	127	164	13465	3423	434	3371

Table 2. CPU Time for Orbital Branching Using Local Symmetry Group

In order to succinctly present many of our computational results, we use performance profiles of Dolan and Moré [3]. 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. Methods whose corresponding profile lines are the highest are the most effective. Figure 7 shows a performance profile of the results of the first experiment, the CPU times in Table 2.

The most effective branching method is **Rule 5**—the method that keeps the size of the symmetry group large on the left branch. (This method gives the "highest" line in Fig. 7). In fact, this branching method is the only one that is able to solve all of the instances in the test suite within the four hour time limit. This result is somewhat surprising. Anecdotally, symmetry has long been thought to be a significant hurdle for solving

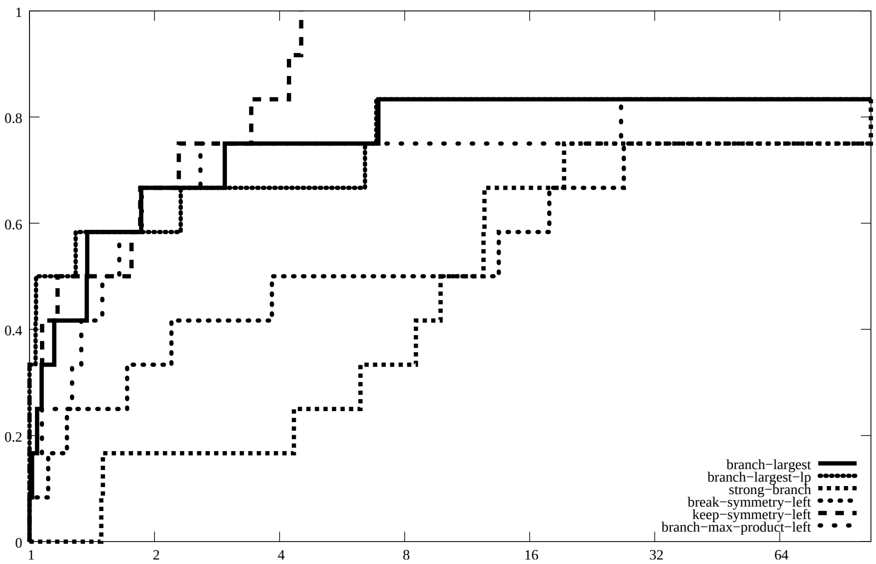


Fig. 7. Performance Profile of Branching Rules

integer programs. One might expect that methods in which symmetry was *removed* as quickly as possible would have been the most effective. Our results go counter to this intuition. Instead, if effective methods for *exploiting* problem symmetry (like those in orbital branching) are present, the results indicate that one should attempt to keep a large amount of symmetry in the subproblems.

A second experiment was aimed at measuring the impact of using the local symmetry group $\mathcal{G}(A(F_1^a, F_0^a))$ instead of the global symmetry group $\mathrm{stab}(F_1^a, \mathcal{G}(A))$ (discussed in Section 4.2) when making a branching decision. Table 3 shows the CPU time (in seconds) orbital branching, equipped with reduced cost fixing and orbital fixing, required on the instances in the test suite, for the different branching rules employing the global symmetry group.

Instance	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6
cod83	10	3	5	1	1	5
cod93	1677	1556	2361	166	167	396
cod105	237	237	359	234	242	237
cov954	5	4	23	13	6	5
cov1053	103	619	761	280	240	89
cov1054	14400	14400	14405	14400	179	14400
cov1075	55	42	202	14400	152	95
cov1076	14400	14400	14404	14400	1415	14400
f5	64	79	664	44	45	64
sts45	8	8	50	8	8	8
sts63	104	90	101	20	20	81
sts81	29	28	73	39	39	3383

Table 3. CPU Time for Orbital Branching Using Global Symmetry Group

Again, branching **Rule 5** that keeps symmetry on the left child node, was by far the most effective. A side-by-side comparison of Tables 2 and 3 indicates that in general using the global symmetry group is more effective than attempting to exploit symmetry that may only be locally present at a node. Figure 8 shows a performance profile comparing the CPU time required to solve the instances using branching **Rule 5** with both the local and global symmetry groups. Surprisingly, the improved performance of the global symmetry group comes not only from improved efficiency of the branching calculations, but in many cases the number *nodes* is reduced, as shown in Table 4. These computational results run counter to Theorem 4, which states that orbits from the global symmetry group are a subdivision of orbits from the local group. Since the orbits of the local group are no smaller, one would expect that orbital branching's enumeration tree would also be smaller in this case.

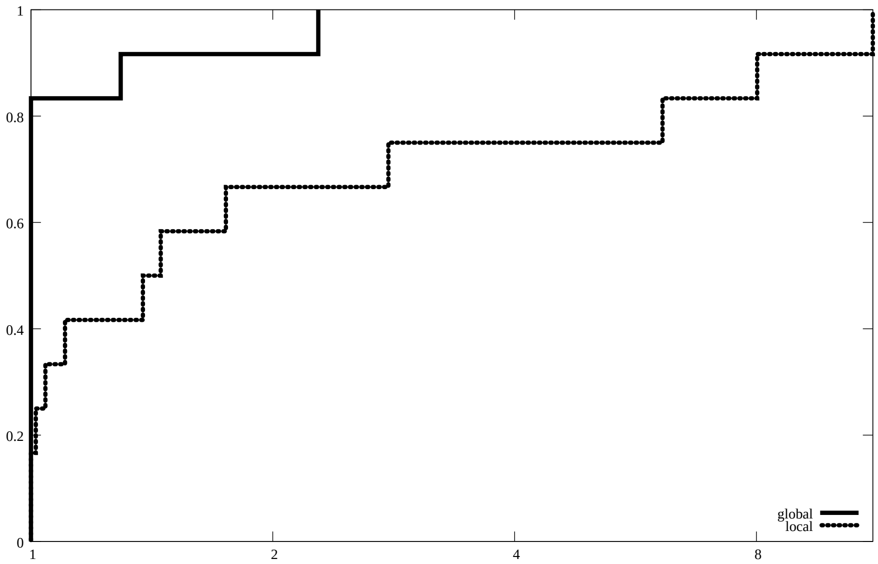


Fig. 8. Performance Profile of Local versus Global Symmetry Groups

A third comparison worthy to note is the impact of performing orbital fixing, as introduced in Section 4.1. Using branching **Rule 5**, each instance in Table 1 was run both with and without orbital fixing. Figure 9 shows a performance profile comparing the results in the two cases. The results shows that orbital fixing has a *significant* positive impact.

The final comparison we make here is between orbital branching (using branching **Rule 5** and the global symmetry group), the isomorphism pruning algorithm of Margot, and the commercial solver CPLEX version 10.1, which has features for symmetry detection and handling. Table 5 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" [14]. The paper [14] does not report results on `f5`. The CPLEX results were obtained on an Intel Pentium 4 CPU clocked at 2.40GHz.

Instance	Local Symmetry	Global Symmetry
cod83	195	25
cod93	1577	1361
cod105	23	11
cov954	449	249
cov1053	3139	9775
cov1054	1249	1249
cov1075	381	381
cov1076	31943	31943
f5	717	1125
sts45	4507	4709
sts63	9993	5533
sts81	83961	6293

Table 4. Number of Nodes in Orbital Branching Enumeration Tree with Different Symmetry Groups

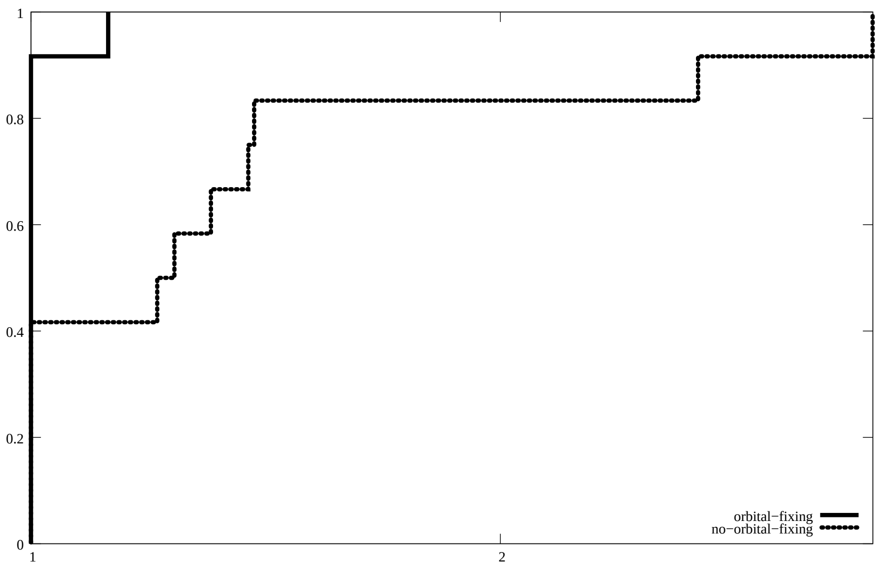


Fig. 9. Performance Profile of Impact of Orbital Fixing

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 and CPU times required to solve the instances are quite comparable. Orbital branching proves to be faster than CPLEX in all but one case, while in all cases the number of evaluated nodes is remarkably smaller.

7. Conclusions

In this work, we presented a simple way to capture and exploit the symmetry of an integer program when branching. We showed through a set of experiments that the new

	Orbital Branching		Isomorphism Pruning		CPLEX v10.1	
Instance	Time	Nodes	Time	Nodes	Time	Nodes
cod83	1	25	19	33	391	32077
cod93	167	1361	651	103	**fail**	488136
cod105	242	11	2000	15	1245	1584
cov954	6	249	24	126	9	1514
cov1053	240	9775	35	111	937	99145
cov1054	179	1249	130	108	**fail**	239266
cov1075	152	381	118	169	141	10278
cov1076	1415	31943	3634	5121	**fail**	1179890
f5	45	1125	-	-	1150	54018
sts45	8	4709	31	513	24	51078
sts63	20	5533	120	1247	3414	4974655
sts81	39	6293	68	199	**fail**	12572533

Table 5. Comparison of Orbital Branching, Isomorphism Pruning, and CPLEX v10.1

method, orbital branching, outperforms CPLEX, a state-of-the-art solver, when a high degree of symmetry is present. Orbital branching also seems to be of comparable quality to the isomorphism pruning method of Margot [14]. 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. A majority of this work was completed while author Linderoth was a faculty member at Lehigh University. Author Ostrowski is supported by the NSF through the IGERT Grant DGE-9972780.

References

1. C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch and Price: Column generation for solving huge integer programs. *Operations Research*, 46:316–329, 1998.
2. M. S. Bazaraa and O. Kirca. A branch-and-bound heuristic for solving the quadratic assignment problem. *Naval Research Logistics Quarterly*, 30:287–304, 1983.
3. Elizabeth Dolan and Jorge Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002.
4. P. Foggia, C. Sansone, and M. Vento. A preformance comparison of five algorithms for graph isomorphism. *Proc. 3rd IAPR-TC15 Workshop Graph-Based Representations in Pattern Recognition*, pages 188–199, 2001.
5. D. R. Fulkerson, G. L. Nemhauser, and L. E. Trotter. 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, 1973.

6. H. Hamalainen, I. Honkala, S. Litsyn, and P. Östergård. Football pools—A game for mathematicians. *American Mathematical Monthly*, 102:579–588, 1995.

7. S. Holm and M. Sørensen. The optimal graph partitioning problem: Solution method based on reducing symmetric nature and combinatorial cuts. *OR Spectrum*, 15:1–8, 1993.

8. V. Kaibel, M. Peinhardt, and M.E. Pfetsch. Orbitopal fixing. In *IPCO 2007: The Twelfth Conference on Integer Programming and Combinatorial Optimization*. Springer, 2007. To appear.

9. V. Kaibel and M.E. Pfetsch. Packing and partitioning orbitopes. *Mathemathical Programming*, 2007. To appear.

10. J. T. Linderoth and M. W. P. Savelsbergh. A computational study of search strategies in mixed integer programming. *INFORMS Journal on Computing*, 11:173–187, 1999.

11. S. Litsyn. An updated table of the best binary codes known. In V. S. Pless and W. C. Huffman, editors, *Handbook of Coding Theory*, volume 1, pages 463–498. Elsevier, Amsterdam, 1998.

12. E. M. Macambira, N. Maculan, and C. C. de Souza. 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.

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

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

15. F. Margot. Small covering designs by branch-and-cut. *Mathematical Programming*, 94:207–220, 2003.

16. B. D. McKay. *Nauty User's Guide (Version 1.5)*. Australian National University, Canberra, 2002.

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

18. W. H. Mills and R. C. Mullin. Coverings and packings. In *Contemporary Design Theory: A Collection of Surveys*, pages 371–399. Wiley, 1992.

19. G. L. Nemhauser, M. W. P. Savelsbergh, and G. C. Sigismondi. MINTO, a Mixed INTeger Optimizer. *Operations Research Letters*, 15:47–58, 1994.

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

21. E. Rothberg. Using cuts to remove symmetry. Presented at the 17^{th} International Symposium on Mathematical Programming.

22. E. C. Sewell. A branch-and-bound algorithm for the stability number of a sparse graph. *INFORMS Journal on Computing*, 10:438–447, 1998.

23. H. D. Sherali and J. C. Smith. Improving zero-one model representations via symmetry considerations. *Management Science*, 47(10):1396–1407, 2001.

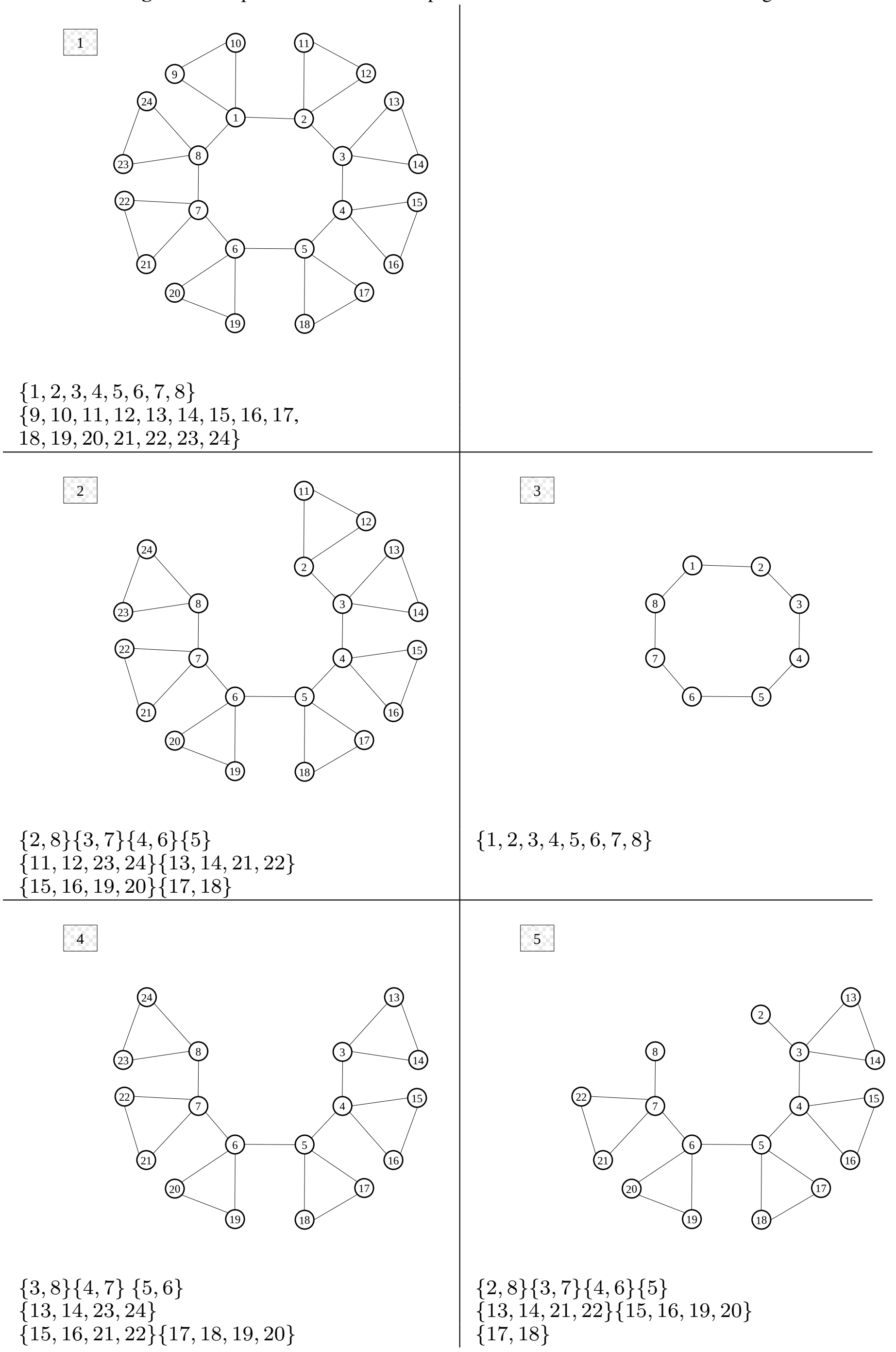
Instance	Branching Rule	Time	Nodes	Nauty Time	# Fixed by RCF	# Fixed by OF	#Fixed by SBF	Deepest Orbital Level
cod105	Break Symmetry	254.52	17	14.984722	0	1020	0	7
cod105	Keep Symmetry	423.92	23	21.618711	216	1228	0	8
cod105	Branch Largest LP Solution	237.95	7	4.165367	0	0	0	2
cod105	Branch Largest	239.25	9	6.418025	0	0	0	3
cod105	Max Product Orbit Size	229.46	9	6.118071	1	960	0	3
cod105	Strong Branch	344.67	7	4.183364	0	1024	1532	2
cod83	Break Symmetry	6.15	143	1.256809	325	548	0	15
cod83	Keep Symmetry	8.19	195	2.266657	251	942	0	18
cod83	Branch Largest LP Solution	3.58	57	0.548917	328	864	0	7
cod83	Branch Largest	10.57	193	0.431934	233	588	0	7
cod83	Max Product Orbit Size	4.77	105	0.612906	69	642	0	11
cod83	Strong Branch	5.33	21	0.354945	16	762	412	6
cod93	Break Symmetry	3268.83	37297	58.355112	106725	6202	0	26
cod93	Keep Symmetry	242.49	1577	32.926999	11473	2422	0	44
cod93	Branch Largest LP Solution	1557.11	14461	1.917708	201292	348	0	7
cod93	Branch Largest	1677.35	16439	2.212663	205636	1060	0	7
cod93	Max Product Orbit Size	398.91	3503	6.821963	41907	704	0	25
cod93	Strong Branch	2367.93	161	8.245748	437	2400	13478	15
cov1053	Break Symmetry	345.82	15321	28.748629	0	2418	0	35
cov1053	Keep Symmetry	105.41	3139	18.710157	0	1696	0	31
cov1053	Branch Largest LP Solution	616.70	20725	2.074679	0	988	0	19
cov1053	Branch Largest	103.47	3437	1.902710	0	1094	0	17
cov1053	Max Product Orbit Size	90.22	2859	2.476628	0	1466	0	20
cov1053	Strong Branch	768.40	777	14.072857	0	2834	16462	43
cov1054	Break Symmetry	14400	110116	0.199969	0	0	0	0
cov1054	Keep Symmetry	181.26	1249	18.497186	0	454	0	15
cov1054	Branch Largest LP Solution	14400	104126	1.055841	56	88	0	5
cov1054	Branch Largest	14400	105500	1.710738	0	0	0	7
cov1054	Max Product Orbit Size	14400	104172	2.030689	0	176	0	8
cov1054	Strong Branch	14400	846	79.314951	0	220	12846	57
cov1075	Break Symmetry	14400	408822	0.837873	862268	0	0	0
cov1075	Keep Symmetry	209.74	381	189.816146	413	962	0	15
cov1075	Branch Largest LP Solution	49.78	495	23.338451	1400	520	0	9
cov1075	Branch Largest	68.56	461	44.260274	1333	900	0	13
cov1075	Max Product Orbit Size	128.41	543	102.014486	1028	1090	0	21
cov1075	Strong Branch	215.54	71	37.435308	126	92	1858	10
cov1076	Break Symmetry	14400	496533	0.735888	720913	0	0	0
cov1076	Keep Symmetry	1559.87	31943	656.975116	21902	960	0	20
cov1076	Branch Largest LP Solution	14400	498573	15.820595	631691	222	0	7
cov1076	Branch Largest	14400	504396	33.967836	495631	388	0	9
cov1076	Max Product Orbit Size	14400	498258	110.187249	638795	532	0	18
cov1076	Strong Branch	14400	4989	2327.428166	2798	1256	71682	27
cov954	Break Symmetry	8.41	237	4.447316	423	272	0	11
cov954	Keep Symmetry	17.27	449	11.000322	677	948	0	15
cov954	Branch Largest LP Solution	3.83	153	0.664898	638	0	0	6
cov954	Branch Largest	5.26	249	1.183821	818	304	0	12
cov954	Max Product Orbit Size	4.85	217	1.091832	699	132	0	11
cov954	Strong Branch	23.99	63	1.904713	65	160	1724	11
f5	Break Symmetry	42.46	995	2.473627	3515	1356	0	14
f5	Keep Symmetry	34.50	717	1.529766	2102	598	0	14
f5	Branch Largest LP Solution	79.76	2573	0.596910	7660	252	0	8
f5	Branch Largest	64.08	1829	0.626903	9710	430	0	11
f5	Max Product Orbit Size	64.28	1835	0.694894	9678	418	0	13
f5	Strong Branch	668.16	123	1.096838	169	736	8610	15
sts45	Break Symmetry	7.59	4571	0.719893	1	0	0	4
sts45	Keep Symmetry	8.14	4507	1.288806	2	0	0	6
sts45	Branch Largest LP Solution	7.85	4683	0.609907	3	0	0	3
sts45	Branch Largest	8.12	4917	0.393939	1	0	0	2
sts45	Max Product Orbit Size	8.13	4917	0.396940	1	0	0	2
sts45	Strong Branch	94.53	1417	42.961484	0	0	7984	16
sts63	Break Symmetry	1630.34	666623	6.867958	720	126	0	43
sts63	Keep Symmetry	160.85	9993	135.706374	12	0	0	11
sts63	Branch Largest LP Solution	91.37	32627	12.596084	7	0	0	9
sts63	Branch Largest	92.68	33785	9.120613	19	0	0	7
sts63	Max Product Orbit Size	136.77	31261	57.272287	48	0	0	10
sts63	Strong Branch	1132.09	3157	913.579109	0	0	16858	24
sts81	Break Symmetry	3422.66	1000000	2.360643	235	0	0	4
sts81	Keep Symmetry	434.08	83961	128.024537	8	0	0	15
sts81	Branch Largest LP Solution	164.01	25739	68.663563	5	0	0	13
sts81	Branch Largest	126.96	11323	84.573144	0	0	0	13
sts81	Max Product Orbit Size	3370.85	1000000	0.134980	200	0	0	0
sts81	Strong Branch	13465.36	11291	12074.918282	1	0	62098	30

Table 6. Performance of Orbital Branching Rules (Local Symmetry) on Symmetric IPs

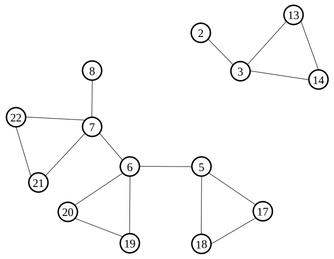
Instance	Branching Rule	Time	Nodes	Nauty Time	# Fixed by RCF	# Fixed by OF	# Fixed by SBF	Deepest Orbital Level
cod105	Break Symmetry	234.13	11	7.124916	0	1020	0	4
cod105	Keep Symmetry	242.48	11	7.141914	0	1020	0	4
cod105	Branch Largest LP Solution	237.18	7	3.612450	0	0	0	2
cod105	Branch Largest	237.32	9	5.472168	0	0	0	3
cod105	Max Product Orbit Size	237.46	9	5.337189	1	960	0	3
cod105	Strong Branch	359.10	7	3.611451	0	1024	1532	2
cod83	Break Symmetry	1.33	25	0.326951	37	906	0	7
cod83	Keep Symmetry	1.34	25	0.327950	37	906	0	7
cod83	Branch Largest LP Solution	3.42	57	0.356948	328	864	0	7
cod83	Branch Largest	10.39	193	0.309953	233	588	0	7
cod83	Max Product Orbit Size	4.62	105	0.428935	69	642	0	11
cod83	Strong Branch	5.23	21	0.265958	16	762	412	6
cod93	Break Symmetry	165.69	1361	8.448719	7397	3378	0	14
cod93	Keep Symmetry	167.16	1361	8.438727	7397	3378	0	14
cod93	Branch Largest LP Solution	1555.70	14461	1.529769	201292	348	0	7
cod93	Branch Largest	1677.24	16439	1.702739	205636	1060	0	7
cod93	Max Product Orbit Size	395.66	3503	4.070383	41907	704	0	25
cod93	Strong Branch	2361.04	161	3.825411	437	2400	13478	15
cov1053	Break Symmetry	280.49	11271	23.658413	0	3454	0	33
cov1053	Keep Symmetry	240.18	9775	4.668290	0	724	0	25
cov1053	Branch Largest LP Solution	619.27	20903	1.084832	0	988	0	19
cov1053	Branch Largest	102.56	3437	1.234820	0	1094	0	17
cov1053	Max Product Orbit Size	89.23	2859	1.578751	0	1466	0	20
cov1053	Strong Branch	760.98	777	7.565862	0	2830	16464	43
cov1054	Break Symmetry	14400	110307	0.178972	0	0	0	0
cov1054	Keep Symmetry	178.78	1249	15.193699	0	454	0	15
cov1054	Branch Largest LP Solution	14400	104161	0.906862	56	88	0	5
cov1054	Branch Largest	14400	105846	1.449778	0	0	0	7
cov1054	Max Product Orbit Size	14400	104184	1.711737	0	176	0	8
cov1054	Strong Branch	14400	846	52.745968	0	220	12846	57
cov1075	Break Symmetry	14400	410572	0.769883	865517	0	0	0
cov1075	Keep Symmetry	152.18	381	132.985777	413	962	0	15
cov1075	Branch Largest LP Solution	41.87	495	15.746605	1400	520	0	9
cov1075	Branch Largest	54.65	461	30.613345	1333	900	0	13
cov1075	Max Product Orbit Size	95.19	543	69.173483	1028	1090	0	21
cov1075	Strong Branch	201.61	71	23.901366	126	92	1858	10
cov1076	Break Symmetry	14400	495919	0.708892	719961	0	0	0
cov1076	Keep Symmetry	1414.99	31943	516.150545	21902	960	0	20
cov1076	Branch Largest LP Solution	14400	496393	13.137002	628579	222	0	7
cov1076	Branch Largest	14400	504849	26.244011	496164	388	0	9
cov1076	Max Product Orbit Size	14400	497593	86.502848	637905	532	0	18
cov1076	Strong Branch	14400	5280	1692.859650	2971	1288	76298	27
cov954	Break Symmetry	12.67	373	7.068934	632	524	0	13
cov954	Keep Symmetry	6.20	249	1.926707	748	48	0	11
cov954	Branch Largest LP Solution	3.65	153	0.516923	638	0	0	6
cov954	Branch Largest	4.99	249	0.930857	818	304	0	12
cov954	Max Product Orbit Size	4.57	217	0.837871	699	132	0	11
cov954	Strong Branch	23.46	63	1.335799	65	160	1724	11
f5	Break Symmetry	44.49	1125	4.553311	2983	2994	0	17
f5	Keep Symmetry	44.62	1125	4.565308	2983	2994	0	17
f5	Branch Largest LP Solution	79.49	2573	0.389942	7660	252	0	8
f5	Branch Largest	63.75	1829	0.440937	9710	430	0	11
f5	Max Product Orbit Size	63.94	1835	0.516921	9678	418	0	13
f5	Strong Branch	664.39	123	0.412933	169	736	8610	15
sts45	Break Symmetry	7.89	4709	0.750886	0	0	0	6
sts45	Keep Symmetry	7.81	4709	0.748884	0	0	0	6
sts45	Branch Largest LP Solution	7.81	4683	0.517921	3	0	0	3
sts45	Branch Largest	8.10	4917	0.372943	1	0	0	2
sts45	Max Product Orbit Size	8.12	4917	0.370943	1	0	0	2
sts45	Strong Branch	49.91	1287	3.219508	0	148	7150	16
sts63	Break Symmetry	20.15	5533	5.517160	1	308	0	11
sts63	Keep Symmetry	20.06	5533	5.550147	1	308	0	11
sts63	Branch Largest LP Solution	90.08	36579	1.722739	19	32	0	9
sts63	Branch Largest	103.76	43349	1.678746	17	32	0	7
sts63	Max Product Orbit Size	81.08	30133	4.614296	47	176	0	8
sts63	Strong Branch	101.43	1377	10.535395	0	676	6710	24
sts81	Break Symmetry	39.06	6293	14.333820	0	670	0	17
sts81	Keep Symmetry	38.87	6293	14.287827	0	670	0	17
sts81	Branch Largest LP Solution	27.87	5649	5.967096	0	562	0	14
sts81	Branch Largest	28.87	5823	5.698133	0	410	0	14
sts81	Max Product Orbit Size	3382.52	1000000	0.133979	200	0	0	0
sts81	Strong Branch	73.45	573	19.796985	0	1112	2514	22

Table 7. Performance of Orbital Branching Rules (Global Symmetry) on Symmetric IPs

Fig. 10. Example 1: Structure of Subproblems and Orbits in Orbital Branching.

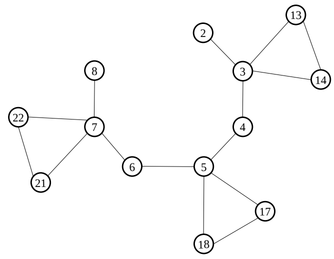


6



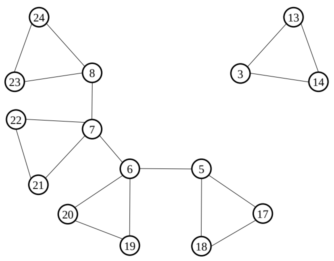
$\{2\}\{3\}\{5\}\{8\}\{6,7\}$
$\{13,14\}\{17,18\}\{19,20,21,22\}$

7



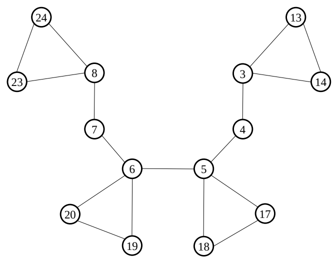
$\{2,8\}\{3,7\}\{4,6\}\{5\}$
$\{13,14,21,22\}\{17,18\}$

8



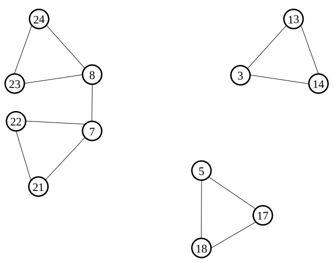
$\{3,13,14\}\{5,8\}\{6,7\}$
$\{17,18,23,24\}\{19,20,21,22\}$

9



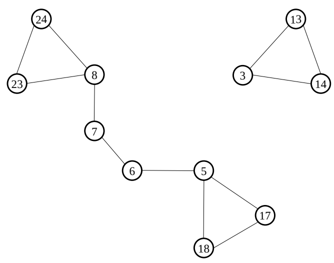
$\{3,8\}\{4,7\}$ $\{5,6\}$
{13,14,23,24}{17,18,19,20}

10



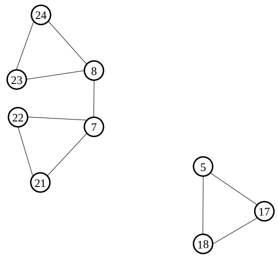
$\{3,13,14,5,17,18\}\{7,8\}$
$\{21,22,23,24\}$

11



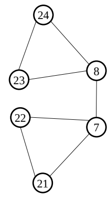
$\{3,13,14\}\{5,8\}\{6,7\}$
{17,18,23,24}

12



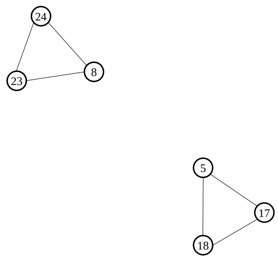
$\{5, 17, 18\}\{7, 8\}$
$\{21, 22, 23, 24\}$

13



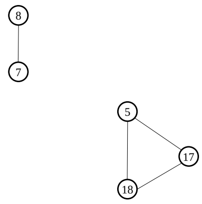
$\{7, 8\}$
$\{21, 22, 23, 24\}$

14



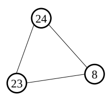
$\{5, 17, 18, 23, 24\}$

15



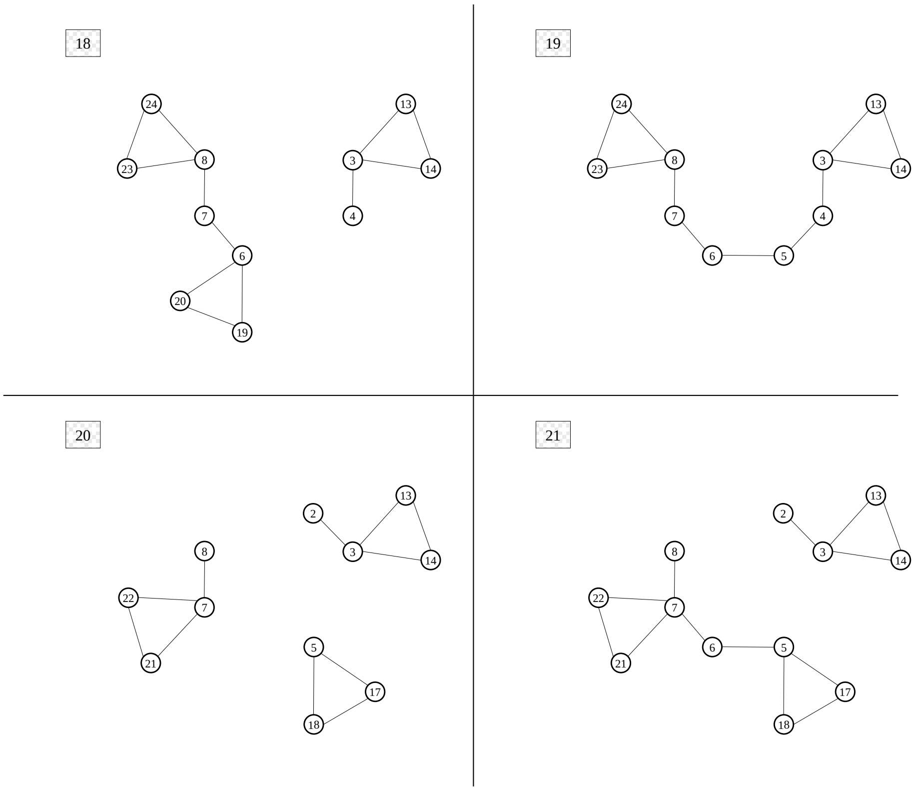
$\{5, 17, 18\}\{7, 8\}$

16



$\{8, 23, 24\}$

17


18
24
23
8
7
6
20
19
13
3
14
4
19
24
23
8
7
6
5
4
3
13
14
20
8
22
7
21
2
3
13
14
5
17
18
21
8
22
7
21
6
5
17
18
2
3
13
14