Mathematical Programming manuscript No.
(will be inserted by the editor)

François Margot

Exploiting Orbits in Symmetric ILP

February, 2003

Abstract. This paper describes components of a branch-and-cut algorithm for solving integer linear programs having a large symmetry group. It describes an isomorphism pruning algorithm and variable setting procedures using orbits of the symmetry group. Pruning and orbit computations are performed by backtracking procedures using a Schreier-Sims table for representing the symmetry group. Applications to hard set covering problems, generation of covering designs and error correcting codes are given.

Key words. Branch-and-cut – isomorphism pruning – symmetry

1. Introduction

An integer linear program (ILP) is *symmetric* if its variables can be permuted without changing the structure of the problem. Symmetric ILPs frequently appear when formulating classical problems in combinatorics or optimization (graph coloring problem, scheduling of jobs on parallel identical machines, covering design problems, code construction; see [31] for additional real world examples). Even for relatively modestly sized problems, ILPs with large symmetry groups are difficult to solve using traditional branch-and-bound or branch-and-cut algorithms. (We assume that the reader is familiar with these procedures, as excellent introductions can be found in [10,30,32,33]). The trouble comes from the fact that many subproblems in the enumeration tree are isomorphic, forcing a wasteful duplication of effort.

One way to deal with the symmetry is to try to remove or reduce it by fixing variables and adding inequalities cutting part of the feasible region, while guaranteeing that an optimal solution of the original problem is still feasible [31]. While this is sometimes an efficient approach, it usually involves empirical experimentation, each problem requiring a new study. The approach followed in this paper, along the lines of [23], is to deal with the symmetry by using the symmetry group of the problem to fix or set variables, to generate cuts and to prune the enumeration tree. The symmetry group is assumed to be part of the

François Margot: Department of Mathematics, University of Kentucky, Lexington, KY 40506-0027, e-mail: fmargot@ms.uky.edu

Mathematics Subject Classification (1991): 90C10, 90C27, 90C57

input. If the symmetry group is not completely known, any of its subgroups can be used. Alternatively, software computing the automorphism group of graphs (for example `nauty` [26]) can be used to generate the symmetry group from the ILP formulation.

In [23], an isomorphism pruning algorithm using the symmetry group for a branch-and-cut algorithm was described. One of the main drawbacks of that algorithm is that the branching variable cannot be chosen freely, but always has to be the non-fixed variable with smallest index. In this paper, a modification of the algorithm allows for a more flexible rule to pick the branching variable, called *ranked branching rule*. It also introduces *strict setting algorithms* that can be used to set variables to 0 or 1 without conflicting with the isomorphism pruning. Section 3 defines the ranked branching rule and describes its associated pruning algorithm. Section 4 defines the strict setting algorithms and show how to use orbits of subgroups of G to set additional variables, an operation called *orbit setting*. Section 5 describes the *strong branching* procedure and the corresponding strict setting algorithm. Section 6 briefly presents basic group algorithms and data structures. Finally, Section 7 presents a comparison between several branch-and-cut algorithms to illustrate the effect of two strict setting algorithms when coupled (or not) with orbit setting. The test problems come from three types of applications: set covering problems from Steiner triple systems, covering designs, and error correcting codes.

2. Preliminaries

Let Π^n be the set of all permutations of the ground set $I^n = \{1, \ldots, n\}$. Π^n is known as the symmetric group of I^n. A permutation in Π^n is represented by an n-vector π, with $\pi[i]$ being the image of i under π. If v is an n-vector and $\pi \in \Pi^n$, let $w = \pi(v)$ denote the vector w obtained by permuting the coordinates of v according to π, i.e.,

$$w[\pi[i]] = v[i] \text{ for all } i \in I^n.$$

We consider an integer linear program (ILP) of the form

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \geq b, \\ & x \in \{0,1\}^n \ , \end{aligned} \tag{1}$$

where A is an $m \times n$ matrix. Without loss of generality, we also assume that the entries in A, b, and c are all integers. For a permutation π of the n variables and a permutation σ of the m rows of A let $A(\pi, \sigma)$ be the matrix obtained from A by permuting its columns according to π and its rows according to σ. Let

$$G = \{\pi \mid \pi(c) = c \text{ and there exists } \sigma \text{ s.t. } \sigma(b) = b, A(\pi, \sigma) = A\} \ .$$

Clearly, G is a permutation group of I^n. Moreover, for $\pi \in G$, a point $\bar{x}$ is feasible (resp. optimal) for the linear relaxation of ILP (1) if and only if $\pi(\bar{x})$ is feasible (resp. optimal) for that ILP. Hence, G is a symmetry group of the feasible (and of the optimal) set of the ILP.

Let $S \subseteq I^n$. To simplify the notation, we make no difference between a set S and its characteristic vector and sets containing a single element e are written simply e instead of $\{e\}$. The *orbit* of S under G is

$$orb(S, G) = \{S' \subseteq I^n \mid S' = g(S) \text{ for some } \ g \in G\} \ .$$

The *stabilizer* of S in G is the subgroup of G given by:

$$stab(S, G) = \{g \in G \mid g(S) = S\} \ .$$

For $1 \leq a \leq b \leq n$, we write $v[a..b]$ for the entries $\{v[a], v[a+1], \ldots, v[b]\}$ of v as an unordered set.

If $g_1, \ldots, g_k$ are k permutations of I^n, the permutation $g = g_1 \cdot \ldots \cdot g_k$ is obtained by applying the permutations from right to left, i.e $g(v) = g_1(g_2(\ldots(g_k(v))\ldots))$ for any n-vector v.

The proposed branch-and-cut algorithm will branch by fixing the value of one variable x_j to 0 or 1. We make a difference between a variable *fixed* to 0 or 1 and a variable *set* to 0 or 1: A variable is fixed to a value if this is the result of a branching operation; it is set to a value if, for some reason other than a branching decision (e.g. reduced cost fixing, logical implications), the variable must take that particular value.

Let a be a node of the branch-and-cut enumeration tree. We denote by F_0^a (resp. F_1^a) the set of indices of variables fixed to 0 (resp. to 1) at a and N^a for the set of indices of variables that are not fixed to 0 or 1 at a. We use S_0^a (resp. S_1^a) for the set of indices of variables set to 0 (resp. to 1) at a. Note that $S_0^a \cup S_1^a \subseteq N^a$.

3. Ranked Branching Rule

Let a and b be two nodes of the enumeration tree of a branch-and-cut algorithm. The subproblems associated with nodes a and b of the branch-and-cut are isomorphic if there exists a permutation $g \in G$, such that $g(F_k^a) = F_k^b$ for $k = 0, 1$.

Using this definition to prune nodes is difficult for two reasons: First, for a given pair of nodes a, b, deciding if a suitable permutation $g \in G$ exists is not easy. Second, this decision problem would have to be solved for a large number of pairs of nodes. The goal of this section is to present a practical isomorphism

pruning algorithm. One advantage of the proposed algorithm is that it works with a single node a of the enumeration tree instead of a pair of nodes. This is a very valuable property, as storing nodes that have been already explored is then unnecessary. However, to achieve this, we need to restrict the choice of the branching variable. We then define one *representative* of each class of isomorphic subproblems, guaranteeing that pruning all nodes that are not representatives is valid. The proof of the later is the main result of this section.

In [23], a simple branching rule, called minimum index branching, was used. Unfortunately, this rule is very inflexible: At node a, the branching variable has to be x_f where f is the minimum index in N^a (even if the value of x_f in the current solution of the LP relaxation is 0 or 1). This section presents a relaxation of the rule, leaving more latitude for picking the branching variable at the cost of maintaining a vector R of integers, the *rank vector*, indicating the order in which the variables have been used as branching variables: At the beginning, $R[i] = n + 1$ for $i = 1, \ldots, n$ and $r = 0$. If variable x_f is chosen for branching and $R[f] = n + 1$, then $R[f]$ is set to $r + 1$ and r is increased by one. Note that both R and r are global variables (i.e., the same R and r are in use at each node of the enumeration tree) and that r is never decreased during the whole enumeration.

The rule to select the branching variable x_f at a, called *ranked branching rule*, is then the following:

(i) If there exists $j \in N^a$ with $R[j] < n + 1$, then $f = \arg\min \{R[j] \mid j \in N^a\}$.
(ii) Otherwise, choose freely any index $f \in N^a$.

It follows that if each variable has been used at least once as a branching variable, the resulting rank vector R is a permutation of I^n. The branching rule of [23] is obtained by always choosing the minimum index in N^a in step (ii) above. Note that a variable $i \in S^a_k$ for $k = 0$ or $k = 1$, might be the chosen branching variable. Then the rank vector is updated, a unique son b is created (since the other son could be pruned by infeasibility), and variable i becomes one of the fixed variables.

In the remainder of the paper, we consider a branch-and-cut using a ranked branching rule. The rank vector R at the start of the processing of node a is denoted by R^a. R^a depends on the enumeration strategy, but the results given below are valid for any enumeration strategy.

Let $J = \{j_1, \ldots, j_p\}$ be an unordered multiset of I^{n+1}. Let $\bar{J}$ be the ordered multiset obtained from J by ordering its elements in non-decreasing order. Given two multisets J_1, J_2 in I^{n+1}, we write $J_1 \preceq J_2$ (resp. $J_1 \prec J_2$) if $\bar{J}_1$ is lexicographically smaller or equal to $\bar{J}_2$ (resp. lexicographically strictly smaller than $\bar{J}_2$).

For a given rank vector R, a set J is a *representative* of the sets in its orbit under G if its rank $R(J) := \{R[j] \mid j \in J\}$ is lexicographically minimum among the sets in its orbit under G, i.e.,

$$R(J) \preceq R(g(J)) \; \forall \; g \in G.$$

Notice that, for any rank vector R, there is at least one representative in the orbit of J and, possibly, more than one.

Lemma 1. *Let R_1 and R_2 be two rank vectors obtained during a branch-and-cut using a ranked branching rule and assume that R_2 is obtained after R_1. Then*

(i) if J is not a representative with respect to R_1 then J is not a representative with respect to R_2.
(ii) if J is a representative with respect to R_1 and all the entries in $R(J)$ are strictly smaller than $n+1$ then J is the unique representative of its orbit with respect to R_1.
(iii) if J is a representative with respect to R_1 and all the entries in $R(J)$ are strictly smaller than $n+1$ then J is also a representative with respect to R_2.

Proof. (i): Let $g \in G$ such that $R_1(g(J)) \prec R_1(J)$ and let $p = |J|$. Let $\overline{R_1(J)}$ be the ordered set $(a_1, a_2, \ldots, a_p)$, $\overline{R_2(J)}$ be the ordered set $(a'_1, a'_2, \ldots, a'_p)$, $\overline{R_1(g(J))}$ be the ordered set $(b_1, b_2, \ldots, b_p)$ and $\overline{R_2(g(J))}$ be the ordered set $(b'_1, b'_2, \ldots, b'_p)$. Let k be the index such that $a_i = b_i$ for $i = 1, \ldots, k-1$ and $b_k < a_k$. Then $b_k < n+1$, implying that $a_i = a'_i$ for $i = 1, \ldots, k-1$, $a'_k > b_k$ and $b_i = b'_i$ for $i = 1, \ldots, k$. It follows that $R_2(g(J)) \prec R_2(J)$.

(ii): Let J' be a set in the orbit of J under G. If $R_1(J') = R_1(J)$ then $J' = J$ as R_1 is a bijection between J and $R_1(J)$.

(iii): The orbit of J under G has at least one representative with respect to R_2. Since, by (ii), J is the unique representative with respect to R_1, (i) implies the result. □

The following property is crucial for the validity of the pruning:

Lemma 2. *Let $J \subseteq I^n$ be a representative under G with respect to rank vector R. Let $J' := J - j$ with $j \in \arg\max \{R[i] \mid i \in J\}$. Then J' is also a representative with respect to R.*

Proof. If J' is not a representative, then there exists $g \in G$ such that $R(g(J')) \prec R(J')$. Then $R(g(J)) \prec R(J)$, a contradiction. □

Consider the following *isomorphism pruning* to be applied at node a of the enumeration tree of a branch-and-cut using a ranked branching rule: If F_1^a is not

a representative with respect to R^a, then prune node a. (Node a is said to be *pruned by isomorphism* for short.)

Remark 1. This isomorphism pruning introduces an asymmetry between variables fixed to 1 and those fixed to 0. It would be of course possible to define the representative using F_0^a instead of F_1^a, with similar results. Note however that pruning nodes where at least one of F_0^a and F_1^a is not a representative with respect to R^a would not work. To simplify the following, assume that the branching variable is chosen according to the minimum index rule, implying that R is the identity permutation. Consider the group generated by the permutation $(2, 3, 1)^T$, and suppose that the optimal solution of the ILP (1) is obtained at node a when two of the variables have value 1 and the remaining one has value 0. Then F_1^a is a representative if and only if $F_1^a = \{1, 2\}$, and F_0^a is a representative if and only if $F_0^a = \{1\}$. Since F_1^a and F_0^a are always disjoint, a would be pruned by the isomorphism test working with both F_1^a and F_0^a and all nodes containing the optimal solution would be pruned.

One way to work with both F_1^a and F_0^a would be to say that node a is not pruned if F_1^a is a representative and that F_0^a is lexicographically minimal in its orbit under $stab(F_1^a, G)$. But this results in pruning exactly the same nodes as the proposed isomorphism pruning test, as F_0^a is "filling the gaps" between elements in F_1^a and $F_1^a \cup F_0^a = \{1, 2, \ldots, k\}$ for some k, showing that F_0^a is always lexicographically minimal in the considered orbit. □

Let $\mathcal{B}$ be a branch-and-cut using a ranked branching rule, isomorphism pruning, and a particular enumeration strategy. Let $\mathcal{T}$ be the enumeration tree of $\mathcal{B}$, assuming that nodes are pruned only by isomorphism pruning or when the LP relaxation of the corresponding ILP is infeasible. This implies that even in the case where the linear relaxation associated with node a has an integer optimal solution, $\mathcal{B}$ continues to branch. Pruned nodes are not included in $\mathcal{T}$.

Let $\mathcal{B}'$ be the branch-and-cut obtained from $\mathcal{B}$ by dropping isomorphism pruning, but enumerating the nodes in the same order as $\mathcal{B}$, the remaining nodes being processed arbitrarily after that. Let $\mathcal{T}'$ be the enumeration tree of $\mathcal{B}'$, assuming that nodes are pruned only by infeasibility. Pruned nodes are not included in $\mathcal{T}'$. Note that $\mathcal{T} \subseteq \mathcal{T}'$.

Lemma 3. *Let R be the rank vector obtained at the end of the enumeration for $\mathcal{B}$. Then*

(i) If $a \in \mathcal{T}' - \mathcal{T}$ then F_1^a is not a representative of its orbit with respect to R;
(ii) If $a \in \mathcal{T}$ then F_1^a is the unique representative of its orbit with respect to R;
(iii) $\mathcal{B}$ and $\mathcal{B}'$ return the same optimal value.

Proof. (i): If $a \in \mathcal{T}' - \mathcal{T}$ then a node b on the path between the root of $\mathcal{T}'$ and a is pruned by isomorphism. Then F_1^b is not a representative with respect

to R^b, and thus, by Lemma 1 (i), it is not a representative with respect to R. By definition of a ranked branching rule, we have, for all $i \in F_1^a - F_b^1$, that $R[i] > \max\{R[j] | j \in F_b^1\}$. Then, by Lemma 2, F_1^a cannot be a representative with respect to R.

(ii): As a was not pruned by isomorphism, F_1^a is a representative with respect to R^a. According to the rule for updating the rank vector during the enumeration, we have that $R[i] < n + 1$ for all $i \in F_1^a$. By Lemma 1 (ii) and (iii), F_1^a is the unique representative with respect to R.

(iii): Let a be a node of $\mathcal{T}'$ for which F_1^a is the characteristic vector of an optimal solution to ILP (1) and such that $F_0^a = I^n - F_1^a$. A representative of the orbit of F_1^a under G with respect to R is a set F^*, and, by (i) and (ii), there is a node $b \in \mathcal{T}$ with $F_1^b = F^*$ and $F_0^b = I^n - F^*$. As $\mathcal{B}$ processes node b at some point, $\mathcal{B}$ yields the same optimal value as the one returned by $\mathcal{B}'$. □

Lemma 3 shows the validity of the isomorphism pruning. It should then be obvious that usual techniques such as cutting planes and pruning by bounds can be added to $\mathcal{B}$, keeping a branch-and-cut returning an optimal solution of the problem. On the other hand, setting variables to 0 or 1 requires some care, as explained in the next section.

4. Setting variables

This section shows how to modify standard techniques for setting variables to 0 or 1 (e.g. reduced cost fixing) when coupled with isomorphism pruning. This motivates the definition of a *strict setting algorithm*. Then new results allowing for the setting of additional variables are presented. A consequence of these results is that, at node a, all variables in the same orbit of $stab(F_1^a, G)$ can be set simultaneously to k as soon as it is known that one of them can be set to k by a strict setting algorithm working under symmetry, for $k = 0, 1$.

Let a be a node of the enumeration tree and let z^* be the value of the best known feasible solution to ILP (1) when processing node a. Let ILP^a denote the ILP at node a, i.e., ILP (1) where variables in $F_k^a \cup S_k^a$ take value k, for $k = 0, 1$. It is sometimes possible to identify variables that may be set to 0 or 1 without affecting the optimal solution returned by a branch-and-cut. Usually, if it is possible to show that there exists an optimal solution $\bar{x}$ of ILP^a with $\bar{x}_i = k$, for $k = 0$ or $k = 1$, then it is valid to set that variable to k at a. When using a branch-and-cut with isomorphism pruning, however, this is not true anymore, as the node corresponding to $\bar{x}$ might be pruned by isomorphism and the representative of its orbit might have $x_i = 1 - k$. To avoid this, it is necessary to use a *strict setting algorithm*. This is a procedure used to identify variables that must be 0 (resp. 1) in every optimal solution of ILP (1) having

value less than z^* that is feasible for ILP^a. The procedure then includes the variables in S_0^a (resp. S_1^a) and sets them to 0 (resp. 1). If the algorithm wants to set a variable in $F_k^a \cup S_k^a$ to the value $(1-k)$ for $k = 0$ or $k = 1$, ILP^a is infeasible and a is pruned. For simplicity, this observation is implicit in the remainder of the paper.

For what follows it is essential that the strict setting algorithm works under symmetry: If the setting algorithm is able to set variable x_i in ILP^a then, for any $g \in G$, it is able to set variable $x_{g(i)}$ in the ILP obtained from ILP^a by permuting the variables according to g. This essentially prevents the setting algorithm to work based on conditions linked to the isomorphism pruning, but allows for traditional setting procedures. In the remainder of the paper, we only consider strict setting algorithms working under symmetry.

As an example of a strict setting algorithm, consider the usual reduced cost fixing procedure: Let $\bar{c} \geq 0$ be the reduced costs of the optimal solution of the linear relaxation of ILP^a having value z. If x_i is non basic at its lower bound and $z + \bar{c}_i \geq z^*$ then $x_i = 0$ in every feasible solution of ILP^a with value better than z^*. (A similar test can be used for a non basic variable at its upper bound). Since we assume that c is integer, it is valid to replace the condition by the stronger $z + \bar{c}_i > z^* - 1$.

We consider a branch-and-cut $\mathcal{B}$ using isomorphism pruning, a ranked branching strategy and a strict setting algorithm. Let $\mathcal{T}$ be the nodes in the enumeration tree of $\mathcal{B}$ that are not pruned by infeasibility or isomorphism. For $a \in \mathcal{T}$, let $\mathcal{T}^a$ be the subtree of $\mathcal{T}$ rooted at a. A *feasible leaf* of $\mathcal{T}^a$ is a leaf of $\mathcal{T}^a$ where all variables are fixed to 0 or 1. A *solution* in $\mathcal{T}^a$ is a solution x corresponding to a feasible leaf of $\mathcal{T}^a$. An *optimal solution* in $\mathcal{T}^a$ is a solution in $\mathcal{T}^a$ that is optimal for ILP (1).

The next lemma shows that it is possible to set additional variables to 0, based on the variables fixed to 0 at ancestors nodes. The idea is that if a variable was fixed to 0 at an ancestor d, then either $F_1^d \cup \{i\}$ is not a representative or the nodes f with $F_1^f = F_1^d \cup \{i\}$ are explored in the other subtree in the sons of d. Coupling this observation with some permutations in G, we get the following:

Lemma 4. *Let $a \in \mathcal{T}$ and let d be an ancestor of a in $\mathcal{T}$. Let $g \in G$ such that $g(F_1^d) \subseteq F_1^a \cup S_1^a$ and let $i \in F_0^d$. Then no solution x in $\mathcal{T}^a$ has $x_{g(i)} = 1$.*

Proof. Assume that a feasible leaf b of $\mathcal{T}^a$ has $g(i) \in F_1^b$. As $g(F_1^d) \subseteq F_1^a \cup S_1^a \subseteq F_1^b$, we have that $g(F_1^d) \cup g(i) \subseteq F_1^b$ and thus $F_1^b - F_1^d \neq \emptyset$. Moreover, as a ranked branching rule is used and as i is fixed at d, $R^d(i) < \min\{R^d(j) \mid j \in F_1^b - F_1^d\}$. As $g^{-1}(g(F_1^d) \cup g(i)) = F_1^d \cup i$ and $R^d(F_1^d \cup i) \prec R^d(F_1^b)$, F_1^b is not a representative with respect to R^d. Lemma 1 (i) and Lemma 3 (ii) then show that $b \notin \mathcal{T}$, a contradiction. □

In situations were the strict setting algorithm is an expensive operation, it might be faster to use it only at a subset of the nodes of the enumeration tree. At other nodes, it is possible to use the following result to draw on the knowledge of fixed and set variables at other nodes in the tree. Note also that in certain cases, the strict setting algorithm works with an outside hint and that different conclusions might be obtained with different hints. (For example, for the reduced cost fixing procedure with degenerate LPs, two different optimal LP bases might yield different groups of variables to set to 0 or 1.)

Lemma 5. *Let z^* be the value of the best known feasible solution of ILP (1) while processing node $a \in \mathcal{T}$ and let d be a node of $\mathcal{T}$. Let $g \in G$ such that $g(F_0^d) \subseteq F_0^a \cup S_0^a$ and $g(F_1^d) \subseteq F_1^a \cup S_1^a$. Then no optimal solution x in $\mathcal{T}^a$ with value better than z^* has $x_{g(i)} = 1$ for $i \in S_0^d$ or $x_{g(i)} = 0$ for $i \in S_1^d$.*

Proof. Assume that the indices in $S_0^d \cup S_1^d = \{i_1, \ldots, i_p\}$ are ordered according to the order in which the setting algorithm proved that these variables could be set. Let $b(i_k)$ be the ancestor of d in $\mathcal{T}$ at which variable x_{i_k} was set, for $k = 1, \ldots, p$. Note that $F_0^{b(i_k)} \subseteq F_0^d$ and $F_1^{b(i_k)} \subseteq F_1^d$. We prove by induction on k that $x_{g(i_k)}$ satisfies the statement.

For $k = 1$, observe that a problem isomorphic to $ILP^{b(i_1)}$ is obtained from ILP (1) by assigning the value 0 to all variables in $g(F_0^{b(i_1)})$ and the value 1 to those in $g(F_1^{b(i_1)})$. The choice of g implies that all these variables have the corresponding values in ILP^a. It follows that the setting algorithm applied at node a can prove that $x_{g(i_1)}$ satisfies the statement.

The reasoning for $k > 1$ is similar, since a problem isomorphic to $ILP^{b(i_k)}$ is obtained from ILP (1) by assigning the value 0 to all variables in $g(F_0^{b(i_k)})$ and the value 1 to those in $g(F_1^{b(i_k)})$. Moreover, when the setting algorithm was used at $b(i_k)$ to prove that x_{i_k} could be set, the other variables already set had indices in $\{i_1, \ldots, i_{k-1}\}$. Since all variables in $x_{g(i_q)}$ for $1 \leq q \leq k-1$ are set to the proper value in ILP^a, and thus the strict setting algorithm on ILP^a can prove that $x_{g(i_k)}$ can be set too. □

One difficulty in using Lemma 4 or Lemma 5 to set variables to 0 or 1 is computing the set G^* of all $g \in G$ satisfying the statement. For given nodes a and d, this set has no nice property and might not be a subgroup of G. Note however, that there is no need to compute the whole set G^* as the results remain valid even if setting is done only for a subset of G^*. The following corollaries are easier to use, albeit weaker than the Lemma.

Corollary 1. *Let $i \in F_0^a$. Then all the variables in $orb(i, stab(F_1^a, G))$ may be set to 0 in ILP^a.*

Proof. This is Lemma 4 for $d = a$ and $g \in stab(F_1^a, G)$, i.e., g satisfying $g(F_1^a) = F_1^a$. □

Corollary 2. *For $i \in F_0^a \cup S_0^a$, all the variables in $orb(i, stab(F_1^a, G))$ can be set to 0 in ILP^a. For $i \in S_1^a$, all the variables in $orb(i, stab(F_1^a, G))$ can be set to 1 in ILP^a.*

Proof. For $i \in F_0^a$, the result is Corollary 1. After setting these variables to 0, we have $g(F_0^a) \subseteq F_0^a \cup S_0^a$. Then, for $i \in S_0^a \cup S_1^a$, the result is Lemma 5 for $d = a$ and $g \in stab(F_1^a, G)$. □

Consider the following operations at node $a \in \mathcal{T}$ with rank vector R^a, called an *orbit setting*. Let $set\_alg(a)$ be the strict setting algorithm used at node a with variables in $F_0^a \cup S_0^a$ having value 0 and variables in $F_1^a \cup S_1^a$ having value 1:

(i) Compute all orbits in $stab(F_1^a, G)$.
(ii) For each $i \in S_1^a$, set to 1 all variables in $orb(i, stab(F_1^a, G))$ and update S_1^a accordingly. For each $i \in F_0^a \cup S_0^a$, set to 0 all variables in $orb(i, stab(F_1^a, G))$ and update S_0^a accordingly.
(iii) If additional variables can be set by $set\_alg(a)$ update S_0^a and S_1^a accordingly and go to (ii).
(iv) If $N^a = \emptyset$ then return $n + 1$ and stop.
(v) Let x_f be the variable that would be chosen as the branching variable, according to the ranked branching rule. If $F_1^a \cup f$ is not a representative with respect to R^a, then set to 0 all variables in $orb(f, stab(F_1^a, G))$, update S_0^a accordingly and go to (ii). Otherwise, update R^a, return f and stop.

The output of the orbit setting is the value f in (v) for which $F_1^a \cup f$ is a representative, or $n + 1$ if no such f exists.

The validity of the orbit setting should be clear. Step (ii) is an application of Corollary 2 and the correctness of Step (v) follows from Lemma 1 (i) and Corollary 2.

It remains to show how to compute orbits in $stab(F_1^a, G)$ and how to test if a set is a representative or not. This will be covered in Section 6. If orbit setting is used, the operations performed at node a are:

$f :=$ orbit setting at a;
Repeat until a criterion is met
    solve the LP relaxation of ILP^a;
    generate cuts;
If $f < n + 1$ then create two sons of a by fixing x_f to 0 or 1;

It would of course be possible to apply the orbit setting a second time after cuts have been generated, but the differences are likely to be minimal.

If orbit setting is not used, the branching index f is obtained by repeated application of step (v) of the orbit setting, skipping the setting of variables in $orb(f, stab(F_1^a, G))$ when $F_1^a \cup f$ is not a representative with respect to R^a.

5. Strong Branching

The selection of the branching variable is a crucial component of an efficient branch-and-cut. While a rule of thumb can usually be devised for a particular problem, a universal rule is more elusive. For hard problems, where reducing the size of the enumeration tree is particularly important, the most successful procedure is probably *strong branching*: A list of candidate branching variables is built and, for each variable in the list, the LP relaxation of the two sons that would be created if the variable was selected is solved. The final choice of the variable is then based on the different values of the LP relaxations. Usually the LP is not solved to optimality, as only a fixed number of iterations of the dual simplex algorithms are performed. The maximum size of the candidate list is typically small. In the implemented algorithms, the LP is solved to optimality and the size of the candidate list is at most 10. (See [10] for more background on strong branching.)

After performing an orbit setting at node a of the enumeration tree, the orbits of $stab(F_1^a, G)$ partition N^a into "equivalent" variables: If a variable is set to k, then so are all variables in its orbit. If a variable is not set, then none of the variables in its orbit is set. In the latter case, let i and j be two variables in the same orbit. As there exists $g \in stab(F_1^a, G)$ such that $g(i) = j$, $g(F_1^a) = F_1^a$, $g(S_1^a) = S_1^a$, and $f(F_0^a \cup S_0^a) = F_0^a \cup S_0^a$, including both i and j in the candidate list for strong branching is useless, as the resulting LPs for the potential sons will have exactly the same optimal values. If i (resp. j) can be set then the orbit setting will set j (resp. i) too. It follows that the strong branching candidate list needs to include not more than one variable from each orbit of $stab(F_1^a, G)$. Note that the situation is different for the setting algorithm based on reduced costs: Since the calculations are done with respect to one particular optimal base, it is possible that the algorithm is able to set i but not j.

A side result of the strong branching calculations is the occasional setting of some variables: Let x_i be a candidate for strong branching. If ILP^a with $x_i = k$, for $k = 0$ or $k = 1$, is infeasible or has an optimal value larger or equal to the value z^* of the best known feasible solution (or strictly larger than $z^* - 1$ if c is integer), then x_i may be set to $1 - k$. This is a strict setting algorithm and thus the orbit setting described in Section 4 is still valid. In the computational tests given in the next section, when strong branching is used, it means that the strict setting algorithm described above is used.

6. Group Operations

Following [23], the chosen group representation and algorithms are based on the *Schreier-Sims* representation of G [2–5,14,16–18].

Let $G_0 = G$ and $G_i = stab(i, G_{i-1})$ for $i = 1, \ldots, n$. Observe that $G_0, G_1, \ldots, G_n$ are nested subgroups of G.

For $k = 1, \ldots, n$, let $orb(k, G_{k-1}) = \{j_1, \ldots, j_p\}$ be the orbit of k under G_{k-1}. Then for each $1 \leq i \leq p$, let h_{k,j_i} be any permutation in G_{k-1} sending k on j_i, i.e., $h_{k,j_i}[k] = j_i$. Let $U_k = \{h_{k,j_1}, \ldots, h_{k,j_p}\}$. Note that U_k is never empty as $orb(k, G_{k-1})$ always contains k.

Arrange the permutations in the sets U_k, $k = 1, \ldots, n$ in an $n \times n$ table T, with

$$T_{k,j} = \begin{cases} h_{k,j} & \text{if } j \in orb(k, G_{k-1}), \\ \emptyset & \text{otherwise.} \end{cases}$$

The table T is called the Schreier-Sims representation of G. This table is not uniquely defined, as there is usually a choice for the permutations included in the sets U_k. However, the non-empty entries in the table are always the same in all representations.

It is possible to make a small generalization of the presentation by ordering the points of the ground set in an arbitrary order β, called the *base* of the table. In that case, the subgroups $G(\beta)_k$ for $k = 1, \ldots, n$ are defined as the stabilizer of $\beta[k]$ in $G(\beta)_{k-1}$, with $G(\beta)_0 = G$. The corresponding table is denoted by $T(\beta)$. Row k of $T(\beta)$ corresponds to the element k, $U(\beta)_k$ is the set of non empty entries in row k of $T(\beta)$ and $J(\beta)_k$ denotes the corresponding set of indices $\{j \in I^n \mid T(\beta)[k, j] \neq \emptyset\}$, also called the *basic orbit* of k in T (following the terminology of [18]). When the base β is fixed, we sometimes drop the qualifier (β) in these symbols, but from now on each table T is defined with respect to a base.

Algorithms for creating the table $T(\beta)$ and for changing the base β of the representation can be found in [2,4,5,14,16–18]. The implemented algorithm for creating the table is closest to [16] and runs in $O(n^6 + n^2 \cdot |\mathcal{P}|)$ where $\mathcal{P}$ is a given set of generators of the group. The change of base algorithm is similar to one in [5] and runs in $O(n^6)$. See [23] for details. Although the algorithms are described for a 2-dimensional table T, a more space efficient implementation uses a vector of ordered lists instead, as most entries in the table are usually empty. The actual implementation uses a vector of ordered lists, but algorithms are simpler to describe and understand for the 2-dimensional table.

We use backtracking algorithms to decide if a set is a representative or to compute the orbits in the stabilizer of a set in G. These algorithms take advantage of the fact that we may assume that the base β of the group at node a of the enumeration tree has the following structure: Variables fixed to 1 at a (i.e., F_1^a)

come first in β, then the variables not set to 0 and not fixed ($N^a - S_0^a$), and then the variables fixed or set to 0 at a ($F_0^a \cup S_0^a$).

The data structure associated with group G at node a of the branch-and-cut is the following:

```
integer: bvf
matrix of permutations: T          integer vector: β
integer: fixed_one                 integer vector : part_zero   .
```

In addition a single rank vector R is updated during the whole enumeration according to the rule of Section 3. When processing node a, the current rank vector R corresponds to the vector R^a of the previous sections. The integer bvf is the index of the branching variable of the father of a. The table T is just a Schreier-Sims representation of the group with base β. The variable $fixed\_one$ gives the number of variables in F_1^a and

$$F_1^a = \beta[1..fixed\_one] \qquad \text{with} \qquad R[\beta[1]] < \cdots < R[\beta[fixed\_one]] \ .$$

The vector *part_zero* is used to store information about variables fixed or set to 0. For $i = 1, \ldots, fixed\_one$, $\beta[part\_zero[i]..n]$ are the variables that have been fixed or set to 0 before $\beta[i]$ was fixed to 1. For $i = fixed\_one + 1$, $\beta[part\_zero[i]..n] = F_0^a \cup S_0^a$, i.e., all the variables currently fixed or set to 0 at a. The remaining variables appear in β in increasing order of their rank, after variables in F_1^a and before variables in $F_0^a \cup S_0^a$. This structure of β is not difficult to maintain throughout the branch-and-cut, using the procedure *down*() of [23] and a more general base change algorithm when needed. The procedure *down*() has complexity $O(n^6)$ for downing a point.

This is a compact way to store the sets F_1^a, F_0^a and S_0^a with their history. Note that the set F_0^a can be recovered as

$$\{j \in \beta[part\_zero[fixed\_one + 1]..n] \mid R[j] \leq R[bvf]\}. \tag{2}$$

Another advantage is that if the branching variable at a is chosen as a variable in S_0^a, there is no real need to actually generate the son. This is becaus the data structure and ILP of the son would be identical to those of a, except for bvf which can be updated. However, when branching on a variable in $j \in S_1^a$, we might need to modify the base β by moving j at its appropriate place and updating the vector *part_zero*. Although this could be done at node a, we nevertheless create one son in this situation.

In this section, we consider algorithms for solving questions related to a single node a of the branch-and-cut. To avoid heavy notations, the table associated with a is denoted by T, instead of a more precise notation like $T(a)$ or $a \rightarrow T$.

The same remark applies to the other fields of the data structure associated with a.

We are interested in performing the following operations that were mentioned in Section 4: Computing all orbits in the stabilizer of a set and deciding if the rank of a set is lexicographically minimum in its orbit under G.

6.1. Computing orbits in the stabilizer of a set

If generators $g_1, \ldots, g_p$ of the stabilizer G' are known, a simple algorithm [3] works with the graph with node set I^n and edge set $\{(i,j) \mid g_k(i) = j, \text{ with } 1 \leq k \leq p, \text{ and } i \in I^n\}$. The orbits are then the connected components of the graph. Unfortunately, finding generators of the stabilizer of a set under G is difficult, as it is at least as hard as testing if two graphs are isomorphic [14,19]. We resort to backtracking for finding generators of G'. The resulting algorithm has a complexity exponential in the size of the set to stabilize, but is practical for small sizes.

One property of a Schreier-Sims representation of G is that each $g \in G$ can be uniquely written as

$$g = g_1 \cdot g_2 \cdot \cdots \cdot g_n \tag{3}$$

with $g_i \in U(\beta)_i$ for $i = 1, \ldots n$. Hence the permutations in the table form a set of generators of G. As a consequence, any $g \in G$ can be written as

$$g = g_1 \cdot \cdots \cdot g_{k-1} \cdot g_k \cdot h$$

with $g_i \in U(\beta)_i$ for $i = 1, \ldots k$, and $h \in G_k$. Generators of $stab(\beta[1..k], G)$ can be thus obtained by selecting

a) all permutations $g = g_1 \cdot \cdots \cdot g_{k-1} \cdot g_k$ with $g_i \in U(\beta)_i$ for $i = 1, \ldots k$ such that $g(\beta[1..k]) = \beta[1..k]$.
b) all the entries in $U(\beta)_i$ for $i = k+1, \ldots n$.

Listing all permutations described in a) can be done easily with a backtracking procedure: Observe that $g_2, \ldots, g_k$ all stabilize point $\beta[1]$ and thus we must choose $g_1 \in U(\beta)_1$ such that $g_1[\beta[1]] \in \beta[1..k]$. Once g_1 is chosen, as $g_3, \ldots, g_k$ all stabilize point $\beta[2]$, we must choose $g_2 \in U(\beta)_2$ such that $g_1 \cdot g_2[\beta[2]] \in (\beta[1..k] - g_1[\beta[\beta[1]]])$, i.e., $g_2[2] \in g_1^{-1}(\beta[1..k] - g_1[\beta[1]])$. The same reasoning applies for selecting $g_3, \ldots, g_k$.

The backtracking procedure given below outputs generators of the stabilizer of the points in $\beta[1..k]$. It consists of an initializing procedure stabilizer_gen() that calls a recursive procedure stab_gen().

```
stabilizer_gen(a, k)
/* Outputs generators of stab(β[1..k], G) where G is the group represented by T with base β */
   Output U(β)_i for i = k+1, ..., n;
   ident = identity permutation;
   remain := β[1..k];
   stab_gen(a, k, ident, remain, 1);
```

The parameters of the call to stab_gen() have the following interpretation: ind refers to the point $\beta[ind]$ being treated during the current call; $perm$ is a permutation in G sending $\beta[1..ind-1]$ on a subset $B \subseteq \beta[1..k]$; and $remain$ is the set $perm^{-1}(\beta[1..k] - B)$. The variable i runs through all possible choice for $g_i \in U(\beta)_{ind}$.

```
stab_gen(a, k, perm, remain, ind)
   For each i ∈ remain do
     h := T[β[ind], i];
     If h ≠ ∅ then
        loc_remain := remain − i;
        loc_remain := h^{-1}(loc_remain);
        loc_perm := perm · h;
        If ind < k then
          stab_gen(a, k, loc_perm, loc_remain, ind + 1);
        else
          output perm.
```

Remark 2. When using this algorithm in the first step of the orbit setting, a slight modification may allow us to set more variables to 0 or 1 while computing the generators of the stabilizer: Observe that during a recursive call at depth q, $\beta[1..q] = F_1^d$ and $\beta[part\_zero[q+1]..n] = F_0^d \cup S_0^d$ for some ancestor d of a. It is thus possible to use Lemma 5 and set to 0 all variables $perm[j]$ for $j \in \beta[part\_zero[q+1]..n]$. This is implemented in the codes tested in Section 7. A stronger variant would be to initialize $remain$ as $F_1^a \cup S_1^a$ and then output $perm$ only if it stabilizes F_1^a. Moreover, it would be possible to also set to 1 all variables $perm[j]$ for $j \in S_1^d$, but this would require additional bookkeeping to be able to retrieve the set S_1^d. In most applications that we considered, few variables are ever set to 1, making it likely that very little benefit would be obtained by implementing this. □

6.2. Deciding if a set is a representative or not

For deciding if a set is a representative of its orbit with respect to R, we refer to [23], where procedure $first\_in\_orbit()$ is described. The rank vector and the fact that variables fixed and set to zero are moved to the end of the base (instead of only the variables fixed to 0 as in [23]) require only a small modification of $first\_in\_orbit()$ (line marked (*) below). The justification of this comes directly from the relation (2). The proof of correctness of the procedure is identical to the one in [23], as the ordering of the variables in the base and the test (*) take into account that the lexicographic ordering is done with respect to R. The complexity of the procedure is $O(n \cdot k!)$, where k is the cardinality of the set.

```
first_in_orbit(a, k)

/* Returns "true" if and only if R(β[1..k]) is
lexicographically minimum among all sets in
orb(β[1..k], G) */

    ident := identity permutation;
    remain := β[1..k];
    is_lexmin := true;
    f_in_orb(a, k, ident, remain, 1, is_lexmin);
    return(is_lexmin);
```

The parameter is_lexmin is passed by reference and is used to stop the procedure as soon as it is known that $R(\beta[1..k])$ is not lexicographically minimum among all sets in $orb(\beta[1..k], G)$.

```
f_in_orb(a, k, perm, remain, ind, is_lexmin)

   If is_lexmin = false then return;
   For each i ∈ remain do
        If β^{-1}[i] ≥ part_zero[ind] and R[i] < R[bvf] then (*)
          is_lexmin := false;
          return;
        h := T[β[ind], i];
        If h ≠ ∅ then
          loc_remain := remain − i;
          loc_remain := h^{-1}(loc_remain);
          loc_perm := perm · h;
          If ind < k then
            f_in_orb(a, k, loc_perm, loc_remain, ind + 1,
                                                  is_lexmin);
```

Remark 3. Assume that this algorithm is used to find the branching index f when no orbit setting is used, i.e., the orbits of $stab(\beta[1..k-1], G)$ are not known. It is then possible to compute the orbit (or at least part of it) of $\beta[k]$ in $stab(\beta[1..k-1], G)$ at virtually no cost: During a recursive call of f_in_orb() at depth $k-1$, if the permutation *loc_perm* stabilizes $\beta[1..k-1]$ then $loc\_perm[\beta[k]]$ is in the orbit of $\beta[k]$. If the algorithm returns that $R(\beta[1..k])$ is lexicographically minimum in the orbit of $\beta[1..k]$, it is easy to check that all the points in the orbit of $\beta[k]$ have been found. If $R(\beta[1..k])$ is not lexicographically minimum in the orbit, only a subset Q of the orbit might have been found, but it is possible to set all variables in Q as well as $\beta[k]$ to 0, similarly to what is done in step (v) of the orbit setting. □

7. Applications

We use the software `ABACUS` (version 2.3) originally developed by Thienel [12, 15, 32], now distributed by `OREAS` [29], as generic implementation of all branch-and-cut steps with the LP solver `CPLEX7.1` [11]. The machine used is an HP B2000 running HP-UX11 with a 500MHz PA-8600 CPU. Results on three classical combinatorial problems are presented: set covering problems generated by Steiner triple systems, covering design problems, and error correcting code construction.

All the branch-and-cut, including the one in `CPLEX7.1`, are run in order to prove that no solution with value better than the optimal one exists, i.e., the optimal value is used to prune the enumeration tree from the start. This is done in order to remove the randomness of the time at which an optimal solution is found. Since the optimal value $\hat{z}$ is always an integer for the problems under consideration, the value $\hat{z} - 0.95$ is used as the upper cutoff value. The branching variable order is the minimum index branching variable of [23] described in Section 3. Cutting is used in neither for our algorithms, making them work as Branch-and-Bounds, but the branch-and-cut of `CPLEX7.1` is allowed to use cuts (we use the default settings). Since the goal is to get a better feel for the enumeration tree under different strategies for setting variables, the comparisons are more reliable under these choices.

The three applications and the set of test problems are described briefly below. Table 1 gives characteristics of the test problems. Files of the test problems (in LP format) can be obtained from [21].

Error correcting codes: The *Hamming distance* between two binary n-vectors v and v' is the number of indices $1 \leq i \leq n$ such that $v[i] \neq v'[i]$. An error correcting binary code with distance d and word length w is a collection $\mathcal{C}$ of binary w-vectors such that the Hamming distance between any pair of vectors in $\mathcal{C}$ is at least d ([24], Chapter 9 in [6]). The goal is to find such a collection $\mathcal{C}$ of maximal size. This maximal size is denoted by $A(w, d)$. A simple set packing problem with one variable per binary w-vector yields an ILP named *codwd*. We

report results for *cod*83, *cod*93, and *cod*105. Those ILPs are difficult to solve for the branch-and-cut of `CPLEX7.1` as it runs out of memory after more than two days of CPU time with roughly 80% of the generated nodes still in the tree. Generators of the symmetry group are the $w-1$ permutations corresponding to swapping entry 1 and k in a word, for $k = 2, \ldots, w$, and the 2^w permutations corresponding to complementing a subset of entries. The order of the group is $2^w \cdot w!$. Related reduced problems are denoted by *codwdr*. They are obtained from *codwd* by setting to 1 the variable corresponding to the zero word and deleting all other variables corresponding to words with at most $d-1$ ones. `CPLEX7.1` needs more than 6 hours to solve *cod*83*r*, more than 4 hours to solve *cod*105*r*, and is not done after more than 3 days and 2.25 million nodes (almost none pruned) for *cod*93*r*. Generators of the symmetry group of the reduced problems are the $w-1$ permutations corresponding to swapping entry 1 and k in a word, for $k = 2, \ldots, w$. The order of the group is $w!$.

Set covering from Steiner triple systems: Fulkerson [8] introduced a class of difficult set covering problems obtained from the incidence matrix of Steiner triple systems [9]. These problems, named $STS9, STS15, STS27$ and $STS45$, have 9, 15, 27, and 45 variables respectively and are surprisingly difficult for standard branch-and-cut codes. $STS45$ was first solved by Ratliff in 1979 [1]. As an indication on the difficulty of these problems, Avis [1] showed that any branch-and-bound algorithm using LP relaxations and dominance pruning will enumerate at least $2^{\sqrt{2n/3}}$ nodes for an infinite family of problems $STSn$ with $n \to \infty$. Feo and Resende [7] studied similar problems called $STS81$ and $STS243$, and found good heuristic solutions, but only a few years ago Mannino and Sassano [20] were able to solve $STS81$ to optimality. Their branch-and-bound requires an enumeration tree with more than 900 million nodes. They also introduced the problem $STS135$ for which they could only find an upper bound on the optimal value. We report results for $STS45$, $STS81$ and a problem generated according to the same method that we call $STS63$. Since our branch-and-cut is asymmetric with respect to 0's and 1's in the solution (from isomorphism pruning working with F_1^a, not with F_0^a) and since any optimal solution for $STS45$, $STS63$ or $STS81$ has more than 2/3 of the variables set to 1, it is much faster to solve the problems where all variables are complemented, i.e., x_i is replaced by $(1-x_i)$ for $i = 1, \ldots, n$. The constraints of the resulting ILP become of the form $x_i + x_j + x_k \leq 2$ for the triples $\{i, j, k\}$ generating an inequality $x_i + x_j + x_k \geq 1$ of the corresponding STS problem. The objective is then to minimize $\sum_j -x_j$. The statistics given below are for the modified problems, but numbers in Table 1 relate to the original ILP. We choose not to introduce new names for the modified problems, as identical results could be obtained for the original ILP by running a similar branch-and-cut with the roles of 1's and 0's interchanged. The symmetry groups were computed using the program `nauty` (version 1.5) written by McKay [26]. `CPLEX7.1` is able to prove optimality of the optimal value of $STS45$ in 52 seconds and solves $STS63$ in a little bit more than 3 hours. It fails to do so for $STS81$, as it runs out of memory after 6 hours and having generated more than 9 million nodes (90% of them remaining in the tree). Despite the fact

that $STS81$ is solved easily by our branch-and-cut, the solution of $STS135$ still seems to be out of reach. (Its best known feasible solution has value 103 [28]).

Covering designs: Let V be a set of elements of cardinality v and let k and t be integers such that $v \geq k \geq t \geq 0$. Let $\mathcal{K}$ be the set of all k-subsets of V and $\mathcal{T}$ be the set of all t-subsets of V. A (v, k, t)-*covering design* is a collection $\mathcal{C}$ of sets in $\mathcal{K}$ such that each $t \in \mathcal{T}$ is contained in at least one set of $\mathcal{C}$. A (v, k, t)-covering design $\mathcal{C}$ is *minimum* if the cardinality of $\mathcal{C}$ is as small as possible. Results and optimal values for these problems can be found in [27]. Generators of the symmetry group are the $v - 1$ permutations corresponding to swapping elements 1 and k, for $k = 2, \ldots, v$ and the order of the group is $v!$. The optimal value for the $(10, 5, 4)$-covering design was found by Etzion et al. [13], without optimality proof. The branch-and-cut of [22] proved optimality and generated 40 non-isomorphic optimal solutions. Results for *cov*954, *cov*1053, *cov*1054, *cov*1075, *cov*1076, and *cov*1174 are reported. `CPLEX7.1` solves only *cov*954 (196 seconds) and *cov*1075 (13 hours). The other four problems are not solvable by `CPLEX7.1` even after running for several days (it runs out of memory after 2 days on *cov*1076; it does not make much progress on *cov*1053, *cov*1054 or *cov*1174 after several days and millions of enumerated nodes).

Problem	#variables	Opt	LP	Group order
*cod*83	256	-20	-28.44	10,321,920
*cod*83*r*	219	-19	-25.81	40,320
*cod*93	512	-40	-51.20	185,794,560
*cod*93*r*	466	-39	-47.00	362,880
*cod*105	1024	-12	-18.29	371,5891,200
*cod*105*r*	638	-11	-15.26	3,628,800
$STS45$	45	30	15.00	360
$STS63$	63	45	42.00	72,576
$STS81$	81	61	27.00	1,965,150,720
*cov*954	126	30	25.20	362,880
*cov*1053	252	17	16.00	3,628,800
*cov*1054	252	51	50.00	3,628,800
*cov*1075	120	20	17.14	3,628,800
*cov*1076	120	45	42.86	3,628,800
*cov*1174	330	17	15.71	39,916,800

Table 1. Problem characteristics; "Opt" is the optimal value and "LP" is the value of the LP relaxation of the initial formulation.

We consider four slightly different versions of a branch-and-cut, depending on the choice of the strict setting algorithm and the use of orbit setting.

– BC1 is a branch-and-cut using isomorphism pruning and the strict setting algorithm based on reduced costs (cf. Section 4).

- BC2 is BC1 plus the orbit setting described in Section 4.
- BC3 is BC1 plus the strict setting algorithm based on strong branching described in Section 5. BC3 thus uses two strict setting algorithms.
- BC4 is BC3 plus the orbit setting described in Section 4.

For BC1 and BC3, the additional setting of variables described in Remark 3 is used. For BC2 and BC4, the orbit setting is based on Corollary 2 with the extension described in Remark 2. For BC4, the variables in the strong branching candidate list are selected based on the orbits of $stab(F_1^a, G)$, at most one variable per orbit. The maximum size of the candidate list is 10.

Table 2 gives the total number of nodes in the enumeration trees. Comparing BC1 with BC2 and BC3 with BC4, the orbit setting reduces the size of the enumeration tree significantly. BC2 has a tree on average 16% smaller than BC1 by using orbit setting. The use of the orbit setting in BC4 (and possibly a better choice of the variables in the strong branching candidate list) reduces the tree by an additional factor of roughly 25%. Exceptions are *cod*83*r*, *cod*105, *cod*105*r* and $STS45$, but this probably comes from the fact that *cod*105, *cod*105*r* are solved in a few nodes and the symmetry groups of the other two problems are rather small, implying that only a few non-trivial orbits are exploitable. For $STS45$, the fact that about 1/4 of the variables are in the candidate list for strong branching might also play a role. It is interesting to note that although *cod*83*r* and *cod*93*r* have fewer variables and a symmetry group smaller than *cod*83 and *cod*93, the number of nodes for the reduced problems is larger than for the original problem for all four variants. The opposite is true for *cod*105*r* and *cod*105.

Problem	BC1	BC2	BC3	BC4	`CPLEX7.1`
*cod*83	90	79	38	33	*
*cod*83*r*	135	121	45	39	$1 \cdot 10^6$
*cod*93	671	653	249	203	*
*cod*93*r*	1,365	1,301	295	237	–
*cod*105	19	19	15	15	*
*cod*105*r*	13	13	9	5	14,881
$STS45$	1,571	1,571	515	513	62,934
$STS63$	4,723	4,499	1,477	1,247	$9 \cdot 10^6$
$STS81$	658	503	309	199	*
*cov*954	700	655	159	126	45,139
*cov*1053	685	681	190	111	–
*cov*1054	483	447	172	108	–
*cov*1075	499	470	202	169	50,099
*cov*1076	23,607	22,454	6,392	5,121	*
*cov*1174	87,113	69,036	21,454	16,103	–

Table 2. Enumeration tree size. A '–' means that `CPLEX7.1` did not finish after days of running; A '*' means that it ran out of memory.

Table 3 gives the CPU times of the different algorithms. Comparing BC1 with BC2, and BC3 with BC4, we see that the time spent on the orbit setting is usually more than offset by the time savings due to the reduction in the number of nodes in the enumeration tree. The exception here is $STS81$, although the total CPU time for BC4 is not worse than the one for BC3. Variants BC2 and BC4 seem to dominate BC1 and BC3 respectively. BC2 is faster than BC4 on all problems except $STS81$ and *cov*7114. This is not exactly a surprise, as it is well known that the time to perform strong branching is non-negligible. In addition, all these problems are solved in a few hundred nodes at most by BC2, making it hard for BC4 to recover the cost of reoptimizing 20 problems at each node of the enumeration tree. However, for *cov*1174, where the enumeration tree of BC2 has close to 70,000 nodes, BC4 is significantly faster, despite spending about 2/3 of the CPU time in the strong branching setting algorithm.

Problem	BC1	BC2			BC3			BC4		
*cod*83	13	12	–	1	27	18	–	19	10	0
*cod*83*r*	8	7	–	0	19	15	–	15	11	0
*cod*93	284	284	–	23	867	730	–	651	503	11
*cod*93*r*	327	317	–	7	880	771	–	717	621	2
*cod*105	814	822	–	1	4,175	3,383	–	2,000	1,190	1
*cod*105*r*	75	75	–	0	279	221	–	139	81	0
$STS45$	26	27	–	1	32	23	–	31	22	0
$STS63$	164	169	–	6	167	113	–	120	71	3
$STS81$	61	87	–	50	68	38	–	68	16	32
*cov*954	32	32	–	4	35	26	–	24	16	1
*cov*1053	61	63	–	3	60	35	–	35	16	1
*cov*1054	98	95	–	4	185	143	–	130	96	1
*cov*1075	70	67	–	0	193	162	–	118	92	0
*cov*1076	2,699	2,683	–	153	4,597	3,844	–	3,634	2,979	46
*cov*1174	17,909	14,717	–	555	15,129	10,545	–	11,136	7,431	197

Table 3. CPU times (rounded, in seconds); ordered as: Total time; strong branching time; orbit setting time.

These results indicate that for problems requiring only an enumeration tree of a few hundred nodes, BC2 is probably fastest. However, when BC2 has an enumeration tree much larger, BC4 might be faster. Note that if a problem has an efficient strict setting algorithm faster than the strong branching setting algorithm, the comparison would be more in favor of BC4. This could be simulated by calling the strong branching setting algorithm only at a fraction of the nodes of the enumeration tree.

Acknowledgements. I wish to thank three anonymous referees whose numerous suggestions helped improve the paper substantially.

References

1. Avis D., "A Note on Some Computationally Difficult Set Covering Problems", *Mathematical Programming* 8 (1980), 138–145.
2. Butler G., "Computing in Permutation and Matrix Groups II: Backtrack Algorithm", *Mathematics of Computation* 39 (1982), 671–680.
3. Butler G., *Fundamental Algorithms for Permutation Groups*, *Lecture Notes in Computer Science* 559, Springer (1991).
4. Butler G., Cannon J.J., "Computing in Permutation and Matrix Groups I: Normal Closure, Commutator Subgroups, Series", *Mathematics of Computation* 39 (1982), 663–670.
5. Butler G., Lam W.H., "A General Backtrack Algorithm for the Isomorphism Problem of Combinatorial Objects", *Journal of Symbolic Computation* 1 (1985), 363–381.
6. Conway J.H., Sloane N.J.A., *Sphere Packings, Lattices and Groups*, Springer (1993).
7. Feo T.A., Resende G.C., "A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem", *Operations Research Letters* 8 (1989), 67–71.
8. 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 Triple Systems", *Mathematical Programming Study* 2, (1974), 72–81.
9. Hall M., *Combinatorial Theory*, Wiley (1986).
10. Jünger M., Naddef D., eds., *Computational Combinatorial Optimization*, *Lecture Notes in Computer Science* 2241, Springer (2001).
11. *ILOG CPLEX 7.1 User's Manual*, (2001).
12. Elf M., Gutwenger C., Jünger M., Rinaldi G., "Branch-and-Cut Algorithms for Combinatorial Optimization and their Implementation in ABACUS", in [10], 155–222.
13. Etzion T., Wei V., Zhang Z., "Bounds on the Sizes of Constant Weight Covering Codes", *Designs, Codes and Cryptography* 5 (1995), 217–239.
14. Hoffman C.M., *Group-Theoretic Algorithms and Graph Isomorphism*, *Lecture Notes in Computer Science* 136, Springer (1982).
15. Jünger M., Thienel S., "Introduction to ABACUS – A Branch-And-CUt System", *Operations Research Letters* 22 (1998), 83–95.
16. Kreher D.L., Stinson D.R., *Combinatorial Algorithms, Generation, Enumeration, and Search*, CRC Press (1999).
17. Leon J.S., "On an Algorithm for Finding a Base and a Strong Generating Set for a Group Given by Generating Permutations", *Mathematics of Computation* 35 (1980), 941–974.
18. Leon J.S., "Computing Automorphism Groups of Combinatorial Objects", in *Computational Group Theory*, Atkinson M.D. (ed.), Academic Press (1984), 321–335.
19. Luks E., "Permutation Groups and Polynomial-Time Computation", in *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 11 (1993), *Groups and Computation*, L. Finkelsein, W. Kantor, eds., 139–175.
20. Mannino C., Sassano A., "Solving Hard Set Covering Problems", *Operations Research Letters* 18 (1995), 1–5.
21. `http://www.ms.uky.edu/~fmargot/rec_pub.html`
22. Margot F., "Small Covering Designs by Branch-and-Cut", Research report 2000-27, Department of Mathematics, University of Kentucky. To appear in *Mathematical Programming*.
23. Margot F., "Pruning by Isomorphism in Branch-and-Cut", Research report 2001-08, Department of Mathematics, University of Kentucky.
24. S. Lytsin, "An Updated Table of the Best Binary Codes Known", in *Handbook of Coding Theory*, V.S. Pless, W.C. Huffmann, eds., North-Holland, Elsevier (1998).
25. Martin A., "General Mixed Integer Programming: Computational Issues for Branch-and-Cut Algorithms", in [10], 1–25.
26. McKay B.D., "Nauty User's Guide (Version 1.5)", Computer Science Department, Australian National University, Canberra.
27. Mills W.H., Mullin R.C., "Coverings and Packings", in: *Contemporary Design Theory: A collection of Surveys*, Dinitz J.H., Stinson D.R., eds., Wiley (1992), 371–399.
28. Odijk M.A., van Maaren H., "Improved Solutions to the Steiner Triple Covering Problem", *Information Processing Letters* 65 (1998), 67–69.
29. `http://www.oreas.de`
30. M.W. Padberg, G. Rinaldi, "A Branch-and-Cut Algorithm for the Resolution of Large Scale Symmetric Travelling Salesman Problems", *SIAM Review* 33 (1991), 60–100.

31. Sherali H.D., Smith J.C., "Improving Discrete Model Representations via Symmetry Considerations", *Management Science* 47 (2001) p1396–1407.
32. Thienel S., "ABACUS - A Branch-And-CUt System" Ph.D. Thesis, Universität zu Köln (1995).
33. L.A. Wolsey, *Integer Programming*, Wiley (1998).