Tepper School of Business

11-2009

Symmetry in Integer Linear Programming

François Margot
Carnegie Mellon University, fmargot@andrew.cmu.edu

Symmetry in Integer Linear Programming

François Margot [*]

Abstract An integer linear program (ILP) is symmetric if its variables can be permuted without changing the structure of the problem. Areas where symmetric ILPs arise range from applied settings (scheduling on identical machines), to combinatorics (code construction), and to statistics (statistical designs construction). Relatively small symmetric ILPs are extremely difficult to solve using branch-and-cut codes oblivious to the symmetry in the problem. This paper reviews techniques developed to take advantage of the symmetry in an ILP during its solution. It also surveys related topics, such as symmetry detection, polyhedral studies of symmetric ILPs, and enumeration of all non isomorphic optimal solutions.

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. For example, graph coloring, scheduling of jobs on parallel identical machines, covering design or codes construction are problems involving symmetries. Additional real world examples can be found in [107, 108, 109]. 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 [38, 59, 90, 117].) The trouble comes from the fact that many subproblems in the enumeration tree are isomorphic, forcing a wasteful duplication of effort.

François Margot
Tepper School of Business, Carnegie Mellon University, e-mail: fmargot@andrew.cmu.edu

Even fairly small symmetric ILPs might be difficult to solve using state-of-the-art
ILP solvers. Table 1 gives characteristics of a few problems. The first three problems
are covering, packing, and orthogonal array construction problems (see [15] for de-
tails), the next two are covering problems (see [74] for details), *cod*93 is a binary
error-correcting code construction (see [74] for details), and *STS*81 is a covering
problem with a constraint matrix corresponding to a Steiner triple system (see [74]
for details). Finally, *codbt*24 is the problem of constructing a binary-ternary cov-
ering code of radius one with two binary entries and four ternary ones [9, 28]: For
integers $a, b \geq 0$, let W be the set of words of length $a + b$ where the first a entries in
a word take value in $\{0, 1\}$ and the last b entries take value in $\{0, 1, 2\}$. The problem
codbtab is then the problem of finding a minimum cardinality set $C \subseteq W$ such that
for each $w \in W$, there exists $c \in C$ such that the Hamming distance between w and c
is at most one. (An LP formulation of each problem listed in this paper is available
[71].)

Despite the rather small number of variables and small integrality gap (except
for *cod*93 and *STS*81 where the gap is quite large; $OA_2(6, 3, 3, 2)$ has no gap at
all; it amounts to prove that no integer solution with value 54 exists), these prob-
lems are challenging for state-of-the-art codes. Using one of the leading commer-
cial codes available today, finding an optimal solution (when it exists) is relatively
easy. Proving its optimality, however, is much harder. Setting the upper bound value
to the optimal value (except for the infeasible problem $OA_2(6, 3, 3, 2)$), only prob-
lems $CA_7(7, 2, 4, 7)$ (6 minutes) and $OA_2(6, 3, 3, 2)$ (19 hours) can be solved in less
than 24 hours. On the other hand, a code taking advantage of the symmetry in the
problem (the code of [75], described in Section 9) is able to solve most of them in
under a minute. The exceptions are $PA_7(7, 2, 4, 7)$ (1 hour and 30 minutes), *cov*1174
(1 hour and 15 minutes), and *codbt*24 (10 hours) [2].

Problem	n	Opt.	LP	$\|G\|$
$CA_7(7,2,4,7)$	128	113	112	645,120
$PA_7(7,2,4,7)$	128	-108	-112	645,120
$OA_2(6,3,3,2)$	729	$-$	54	33,592,320
*cov*1054	252	51	50	3,628,800
*cov*1174	330	17	15.71	39,916,800
*cod*93	512	-40	-51.20	185,794,560
*codbt*24	324	36	29.45	248,832
*STS*81	81	61	27	1,965,150,720

Table 1 Problem characteristics; n is the number of variables, Opt. is the optimal value, LP is the
value of the LP relaxation of the initial formulation, and $|G|$ is the number of permutations in the
symmetry group.

Various techniques for dealing with symmetric problems have been studied by several research communities, yielding similar approaches. However, since the favorite algorithmic tools available to each community are different, slightly different algorithms have been developed. In this paper, we survey some of the approaches that have been developed in the Mathematical Programming community for solving symmetric ILPs. Limited comparisons and pointers to similar approaches developed in the Computer Science and Constraint Programming communities are given. A recent survey of techniques developed in the Constraint Programming Community to solve symmetric problems is [49].

The main approaches to deal with symmetries that are discussed here are perturbation (Section 4), fixing variables (Section 5), symmetry breaking inequalities (Section 8), and pruning of the enumeration tree (Section 9). The remainder of the paper covers related topics, such as detecting symmetries (Section 3), symmetric polyhedra (Section 6), partitioning problems (Section 7), enumeration of all non isomorphic solutions (Section 11), further developments using isomorphism pruning (Section 12), choice of formulation (Section 13), and the use of additional symmetries (Section 14). Finally, basic definitions and notation are covered in Section 2, and a very brief introduction to computational group representation is given in Section 10.

2 Preliminaries

In this section, basic definitions and notation are presented. The reader is invited to use standard references on permutation groups [20, 52, 102] to complement the extremely succinct material given here.

An unordered set containing elements $e_1, \ldots, e_n$ is denoted by the notation $\{e_1, \ldots, e_n\}$ while an ordered set of the same elements is denoted by $(e_1, \ldots, e_n)$.

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 $\pi \in \Pi^n$ is represented by an n-vector, with π_i being the image of i under π. The permutation π such that $\pi_i = i$ for $i = 1, \ldots$ is the *identity* permutation and is denoted by I.

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.,

$$w_{\pi_i} = v_i \quad \text{for all } i \in I^n.$$

The *composition* of two permutations $\pi^1, \pi^2 \in \Pi^n$, written $\pi^1 \cdot \pi^2$, is defined as the permutation $h = \pi^1(\pi^2)$. The composition of permutations is associative, i.e. for any $\pi^1, \pi^2, \pi^3 \in G$, we have $(\pi^1 \cdot \pi^2) \cdot \pi^3 = \pi^1 \cdot (\pi^2 \cdot \pi^3)$. The neutral element for the composition is the identity permutation, i.e. for any $\pi \in \Pi^n$, we have $\pi \cdot I = I \cdot \pi = \pi$.

A subset $G \subseteq \Pi^n$ with the composition operation defined above is a *group*, if it contains the identity permutation I and satisfies the following properties:

(i) For any $g^1, g^2 \in G$, we have $g^1 \cdot g^2 \in G$;

(ii) For any permutation $g \in G$, there exists an inverse permutation $g^{-1} \in G$ such that $g \cdot g^{-1} = g^{-1} \cdot g = I$.

If G is a group, the permutation g^{-1} of point (ii) above is unique. The number of permutations in G, denoted by $|G|$, is the *order* of the group. A group is finite if its order is finite. All groups considered in this paper are finite. A *subgroup* of a group G is any subset of G that is a group. To simplify the notation, we make no difference between a set $S \subseteq I^n$ and its characteristic vector. Hence $\pi(S)$ is the subset of I^n containing π_i for all $i \in S$.

Let $K = \{0, 1, \ldots, k\}$ for some positive integer value k. We consider an integer linear program of the form

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

where A is an $m \times n$ matrix, b is an m-vector, c is an n-vector, and all three have rational entries. Let Q be the set of all feasible solutions of ILP (1). Note that sometimes not all n variables are requested to be integer, or some variables have slightly different bounds. These are small extensions of the model, and most results and algorithms can be adapted to handle them. However, symmetry between continuous variables is not always exploited.

The *symmetry group G* of ILP (1) is the set of all permutations π of the n variables mapping Q on itself and mapping each feasible solution on a feasible solution having the same value, i.e.

$$G = \{\pi \in \Pi^n \mid c^T \bar{x} = c^T \pi(\bar{x}) \text{ and } \pi(\bar{x}) \in Q \text{ for all } \bar{x} \in Q\}.$$

Example 1. The following ILP with four binary variables is used to illustrate several definitions.

$$\begin{aligned}
\min \quad & x_1 + x_2 + x_3 + x_4 \\
\text{s.t. } \quad & x_1 + x_2 && \geq 1 \\
& \quad\;\; x_2 + x_3 && \geq 1 \\
& \quad\qquad x_3 + x_4 \geq 1 \\
& x_1 \qquad\quad\; + x_4 \geq 1 \\
& x_1, \; x_2, \; x_3, \; x_4 \in \{0, 1\}.
\end{aligned} \tag{2}$$

The set Q contains 7 solutions: $(1,0,1,0)$, $(0,1,0,1)$, $(1,1,1,0)$, $(1,1,0,1)$, $(1,0,1,1)$, $(0,1,1,1)$, and $(1,1,1,1)$. The symmetry group G of ILP (2) contains eight permutations: the identity permutation I, $(2,3,4,1)$, $(3,4,1,2)$, $(4,1,2,3)$, $(3,2,1,4)$, $(4,3,2,1)$, $(1,4,3,2)$, and $(2,1,4,3)$.

□

It is straightforward to check that G is indeed a group. Note that in most situations G is not known, but a subgroup G' of G is. All the results in this paper hold if G is

replaced by G', but it should be expected that symmetry removal obtained with G' is weaker than the one obtained with G.

The *orbit* of any $v \in \mathbb{R}^n$ under G is

$$orb(v, G) = \{w \in \mathbb{R}^n \mid w = g(v) \text{ for some } g \in G\} \ .$$

The *stabilizer* of any $v \in \mathbb{R}^n$ in G is the subgroup of G given by:

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

Given a set $H = \{g^1, \ldots, g^s\} \subseteq \Pi^n$, the collection of all $g \in \Pi^n$ that can be obtained by composing the given permutations in an arbitrary way (including using any of them more than once) is the group G *generated by* H. The permutations in H are *generators* of G. Any subgroup of Π^n can be generated by a set of $O(n^2)$ generators.

Given a group $G \subseteq \Pi^n$ and a nonempty set $J \subseteq \{1, \ldots, n\}$, the *restriction* of G to J is the set of all permutations of the elements in J that are induced by permutations in $stab(J, G)$. The restriction of G to J is a group.

A vector $v \in \mathbb{R}^n$ is *lexicographically smaller* (resp. *lexicographically larger*) than a vector $w \in \mathbb{R}^n$ if for some $1 \leq p \leq n$ we have $v_i = w_i$ for $i = 1, \ldots, p-1$ and $v_p < w_p$ (resp. $v_p > w_p$). This is denoted by

$$v <_L w \qquad (\text{resp. } v >_L w).$$

Given a group $G \subseteq \Pi^n$, a vector v is *lexicomin* (resp. *lexicomax*) in its orbit under G if there is no $g \in G$ with $g(v) <_L v$ (resp. $g(v) >_L v$).

Example 1 (cont.). The two permutations $(2, 3, 4, 1)$ and $(1, 4, 3, 2)$ form a set of generators for G. The orbit of the vector $v = (0, 1, 0, 1)$ contains only vector $(1, 0, 1, 0)$ in addition to v itself. Vector v is lexicomin in its orbit under G. The stabilizer of v is the subgroup G' of G containing four permutations: I, $(3, 4, 1, 2)$, $(3, 2, 1, 4)$, and $(1, 4, 3, 2)$. The restriction of G to $J = \{2, 4\}$ contains only the identity and the permutation swapping 2 and 4 as $stab(J, G) = stab(v, G) = G'$.

$\square$

3 Detecting symmetries

In most cases where a symmetric ILP occurs, the fact that symmetries are present is known to the modeler. However, sometimes only a subgroup G' of the true symmetry group G is known. Devising techniques to compute generators of the symmetry group of an ILP is thus of practical importance.

The main difficulty to face is that G is implicitly defined in term of the feasible set of the ILP. By definition, an ILP with n variables and an empty feasible set has Π^n for its symmetry group. As deciding if the feasible set of an ILP is empty or not is an NP-complete problem, it is rather easy to show that deciding if $G = \Pi^n$ is also

an NP-complete problem. Indeed, adding two integer variables $0 \leq y_1, y_2 \leq k$ and the constraint $y_1 + y_2 = 1$ to the ILP, we get that the symmetry group of the modified ILP is Π^{n+2} if and only if the original one is infeasible.

While this leaves little hope to find a polynomial-time algorithm for computing generators of G, the situation is in fact much worse. The implicit definition of G makes it difficult to design practical algorithms for this task, even worst-case exponential-time ones. One practical algorithm computes a subgroup G^{LP} of G defined as the symmetry group of the LP relaxation: Consider an ILP of the form (1) with n variables and m constraints. For a permutation $\pi \in \Pi^n$ and a permutation $\sigma \in \Pi^m$ of the m rows, let $A(\pi, \sigma)$ be the matrix obtained from A by permuting its columns according to π and its rows according to σ. The subgroup G^{LP} is given by

$$G^{LP} = \{\pi \in \Pi^n \,|\, \pi(c) = c \text{ and } \exists\, \sigma \in \Pi^m \text{ s.t. } \sigma(b) = b, A(\pi, \sigma) = A\} \;.$$

Example 1 (cont.). For ILP (2), we have $G^{LP} = G$, as permuting both the variables and constraints according to any $\pi \in G$ yields an ILP identical to ILP (2).

However, consider the ILP obtained by adding to ILP (2) the inequality

$$-2x_1 - x_2 - 2x_3 - 2x_4 \geq -6 \;. \tag{3}$$

Let H be its symmetry group and H^{LP} be the symmetry group of its LP relaxation. One can check that the only feasible solution of ILP (2) that becomes infeasible is $(1,1,1,1)$. It follows that H is identical to G. However, adding (3) destroys in the LP relaxation the symmetry between x_2 and the other variables. As a result, we have that H^{LP} contains only I and $(3,2,1,4)$ and $H \neq H^{LP}$.

$\square$

In the case where A is a $(0,1)$-matrix, it is possible to cast the computation of generators of G^{LP} as computing generators of the automorphism group of a bipartite graph with colored vertices. (The automorphism group of a graph is the set of all permutations of its nodes that maps adjacent nodes to adjacent nodes.) Indeed, the matrix

$$\begin{pmatrix} 0 & A^T \\ \hline A & 0 \end{pmatrix}$$

can be seen as the adjacency matrix of a bipartite graph H having one vertex for each column of A and one for each row of A, the two sets of vertices forming the two sides of the bipartition. Two column-vertices (resp. row-vertices) have the same color if and only if their corresponding objective coefficients (resp. right hand side coefficients) are identical. Any automorphism of H that fixes both sides of the bipartition and the color classes induces a permutation of the columns of A that is in G^{LP}. Any permutation of G^{LP} can be extended to an automorphism of H that fixes the bipartition and color classes.

When A is not a $(0,1)$-matrix, it is possible to modify the above construction to get the correct result. Details can be found in [103]. Note that mapping the instance of a problem to a colored graph such that color preserving automorphisms of the graph correspond to symmetries of the problem is standard procedure. For example,

see [1, 98, 99] for satisfiability problems and [95] for Constraint Programming in general.

The computational complexity status of computing generators of the automorphism group of a graph is an open problem as it is equivalent to the complexity status of the famous Graph Isomorphism problem (see [69] for a detailed discussion of complexity of problems related to permutation groups). However, efficient algorithms (such as `nauty` [77], `Saucy` [32], as part of `MAGMA` [10] or `GAP` [113]) are available and perform satisfactorily in many instances.

Another track is to formulate the problem of computing permutations in G^{LP} as an ILP [66], although this approach seems unlikely to be competitive when the order of G^{LP} is large.

While working with G^{LP} instead of G might result in a loss of efficiency, most applications have a known LP formulation for which G^{LP} is either a large subgroup of G or G itself, except for infeasible problems. Usually, the symmetry group used is either built from the modeler knowledge or by computing G^{LP} for some ILP formulation.

4 Perturbation

One of the first idea that comes to mind when facing a symmetric problem is to perturb it slightly to destroy the symmetry or to capture some of the symmetry in the problem. For example, adding a small random perturbation to the coefficients c_i for $i = 1, \ldots, n$ easily destroys the symmetry in the problem. This, in general, does not help much and can even be counter-productive. The main reason is that when ILP (1) is infeasible, the same amount of work has to be done regardless of the objective function. In addition, even if the ILP is feasible, once the algorithm has found an optimal solution, the remainder of the computation can be seen as solving an infeasible ILP. Destroying symmetry by perturbation of the objective function thus achieves very little and using symmetry information in an efficient way while solving the problem is a far superior alternative.

The same is true for the "lexicographic" perturbation $c_i = 2^i$ for $i = 1, \ldots, n$ that can be used for certain binary problems, with the additional caveat that this transformation is numerically unstable and can only be used for very small problems.

Using perturbation is sometimes helpful when trying to find a good solution, but using symmetry information is a superior approach when dealing with an infeasible problem or when proving optimality of a solution.

5 Fixing variables

Another simple idea to reduce symmetry in an ILP is to fix variables. While this could be considered a special case of symmetry breaking inequalities (a topic cov-

ered in Section 8), we treat it separately as it can easily be combined with other techniques for dealing with symmetric ILPs when it is used as a preprocessing step.

Let *ILP* be a particular instance of ILP (1) with symmetry group G. Suppose that it is known that, for some $t \geq 1$, some index set $F = \{i_1, \ldots, i_t\}$ and some values $\{\bar{x}_{i_1}, \ldots, \bar{x}_{i_t}\}$, *ILP* has an optimal solution with $x_i = \bar{x}_i$ for all $i \in F$. Let the *fixed ILP* (*FILP*) be obtained by adding to *ILP* the constraints $x_i = \bar{x}_i$ for all $i \in F$. Let the *reduced ILP* (*RILP*) be obtained from *ILP* by substituting x_i by $\bar{x}_i$ for all $i \in F$. Let G^F (resp. G^R) be the symmetry group of *FILP* (resp. *RILP*). Note that *FILP* is an ILP with n variables, while *RILP* has $n - t$ variables.

Example 2. Consider the ILP with six binary variables

$$
\begin{aligned}
\min \quad & x_1 + x_2 + x_3 + x_4 - x_5 - x_6 \\
\text{s.t.} \quad & x_1 + x_2 \qquad\qquad - x_5 - x_6 \geq -1 \\
& \quad\;\; x_2 + x_3 \qquad\qquad - x_6 \geq 0 \\
& \qquad\;\; x_3 + x_4 - x_5 - x_6 \geq -1 \\
& x_1 \qquad\qquad + x_4 - x_5 \qquad \geq 0 \\
& x_1 + x_2 \qquad\qquad + x_5 \qquad \geq 1 \\
& \qquad\;\; x_3 + x_4 \qquad + x_6 \geq 1 \\
& x_1, \;\; x_2, \;\; x_3, \;\; x_4 \;\; x_5, \;\; x_6 \in \{0,1\}.
\end{aligned}
\tag{4}
$$

Its feasible set contains 36 solutions, 9 with $(x_5, x_6) = (0,0)$, 10 with $(x_5, x_6) = (1,0)$, 10 with $(x_5, x_6) = (0,1)$, and 7 with $(x_5, x_6) = (1,1)$. The symmetry group G of ILP (4) contains only two permutations: I and $(3,4,1,2,6,5)$.

A simple analysis of ILP (4) proves that there exists an optimal solution with $x_5 = 1$ and $x_6 = 1$. Adding these two constraints to the ILP yields *FILP* and its symmetry group G^F is G. Substituting x_5 and x_6, we get *RILP* which is exactly ILP (2) with a symmetry group G^R containing eight permutations generated by $\{(2,3,4,1),(1,4,3,2)\}$.

□

In this section, we discuss properties that might indicate which of the formulations *ILP*, *FILP* or *RILP* should be used. This is of course a difficult question and only partial answers or rules of thumb can be given.

Let v be the n-vector with $v_i = \bar{x}_i$ for all $i \in F$ and $v_i = -1$ otherwise. Assuming that in *FILP* there is at least two possible values for variable x_i for all $i \notin F$, we have that $G^F = stab(v, G)$. It follows that $|G^F| \leq |G|$. On the other hand, G^R contains a subgroup G' that is the restriction of $stab(v, G)$ to the variables indices in the complement of F. As shown in Example 1 at the end of Section 2, the order of a restriction of a group can be smaller than the order of the group itself and thus we might have $|G^R| < |stab(v, G)| = G^F$. However, as shown in Example 2, it is also possible to have $|G^R| > |G|$. As a result, there is no general relation between $|G^R|$ and either $|G^F|$ or $|G|$.

Which of the three ILP formulations to use depends on the solution algorithm $\mathbb{A}$. If $\mathbb{A}$ is a branch-and-bound algorithm oblivious to symmetry, regardless of the sizes of the symmetry groups, using either *FILP* or *RILP* produces similar results, and

this should not be worse than using *ILP*. (A very simple illustration can be found in [57].) On the other hand, if $\mathbb{A}$ uses the symmetry group of the problem, the situation is not so clear cut. If $\mathbb{A}$ is efficient in using the symmetry group, it might be better to solve *ILP* than *FILP* or *RILP*. In the remainder of the section, we give two examples where this happens. In general, however, it should be expected that solving *FILP* or *RILP* is more efficient, in particular if the number of fixed variables is large.

An example where solving *ILP* is easier than solving *FILP* is for an ILP formulation to solve the classical edge coloring problem for a graph H with maximum vertex degree Δ. We want to decide if a coloring of the edges of H with Δ colors exists or not, such that any two edges sharing an endpoint receive distinct colors. A simple ILP formulation for edge coloring (*EC*) uses Δ binary variables x_{ej} for $j = 1, \ldots, \Delta$ for each edge e in the graph with the meaning that $x_{ej} = 1$ if and only if e receives color j. Constraints are simply

$$\sum_{j=1}^{\Delta} x_{ej} = 1 \qquad\qquad \text{for each edge } e \in E(H), \qquad (5)$$

$$\sum_{e \in \delta(v)} x_{ej} \leq 1 \qquad\qquad \text{for all } v \in V(H), \text{ for all } j = 1, \ldots, \Delta, \qquad (6)$$

where $\delta(v)$ is the set of edges incident with vertex v. The symmetry group G of *EC* is the product of the automorphism group of the graph H with the symmetric group on the Δ colors.

Obviously, permuting the Δ colors in any feasible solution of *EC* yields another feasible solution. For a vertex v of maximum degree Δ, all edges in $\delta(v)$ must receive Δ distinct colors. As a result, it is valid to fix the colors on $\delta(v)$ to any valid coloring. This will break the symmetry between the colors, yielding the *FEC* formulation. The symmetry group G^F contains all permutations of G fixing edges in $\delta(v)$ and their colors. Using this ILP formulation for coloring the edges of a clique on 9 nodes and fixing the colors as described above results in a solution time orders of magnitude larger for *FEC* than for *EC* for algorithms of [75].

Going back to the general case, it is also sometimes the case that solving *ILP* is better than solving *RILP*. An example from [74] is the code construction problem *cod*93 listed in Table 1. It has 512 variables, a group with order 185,794,560, an optimal value of -40 and an LP relaxation value of -51.20. By fixing a few variables and substituting them in the formulation, one obtains an equivalent problem *cod*93*r* that has 466 variables, a group order of 362,880, an optimal value of -39 and an LP relaxation value of -47.00. Yet, several algorithms using the symmetry in the problem require a smaller enumeration tree to prove that no solution of value smaller than -40 exists in *cod*93 than to prove that no solution of value smaller than -39 exists in *cod*93*r*.

If *ILP* is binary and an algorithm based on pruning of the enumeration tree (see Section 9) is used, it can be shown that fixing a set F of variables to value 1 and then use the algorithm on *FILP* is never superior to using the algorithm on the original ILP and branching first on the variables in F, creating only one subproblem corresponding to the fixing. A similar result for non binary ILPs can be stated provided

that the variables and the values to which they are fixed satisfy a technical condition. It follows that for these algorithms, solving *FILP* is not a good idea. A similar result holds for comparing *ILP* with *RILP*: the latter might be preferable only if $|G^R| > |stab(v, G)|$.

6 Symmetric polyhedra and related topics

The definition of a symmetric ILP given in Section 2 involves the objective function c. If c in (1) is replaced by the zero vector, the symmetry group G of the corresponding ILP is the *symmetry group of the polyhedron* corresponding to the convex hull of the characteristic vectors of the solutions of the problem. (Note that this group should not be confused with the group of the geometric symmetries of the polyhedron; only symmetries permuting space coordinates are considered here.) Many combinatorial polyhedra have large symmetry groups. Just to cite one example, the polytope associated with the Traveling Salesman Problem (TSP) [3, 63] on the complete graph on n nodes has a symmetry group of order $n!$. This statement might be a little bit misleading, since there are many ILP formulations of the TSP, some of them having less symmetry than others. We are talking here about the most studied formulation using exclusively binary edge variables x_{ij} for all $i, j = 1, \ldots, n$ and $i < j$. This polytope has received a lot of attention and has been the focus of intense computational studies in the last decades with impressive results [3]. However, in these studies, the topic of symmetry is rarely considered, as the objective function used in most instances essentially destroys the symmetry.

Similarly, many polyhedra related to combinatorial problems have large symmetry groups, but studies of their facial structure rarely rely on this knowledge. There are several polytopes closely linked to permutations or permutation groups: The *permutahedron* is the convex hull of all permutations of the entries of the n-vector $(1, 2, 3, \ldots, n)$. Its complete linear description is known [6]. A generalization of this polytope is the *permutahedron of a poset*, the convex hull of all permutations π such that if $i < j$ is the poset, then $\pi_i < \pi_j$. Its complete linear description is known for some classes of posets [4]. The *assignment* polytope is the convex hull of all binary $n \times n$ matrices with exactly one nonzero entry per row and per column. These matrices are in bijection with permutations of n elements: For a matrix M and permutation π, entry $M_{ij} = 1$ if and only if $\pi_i = j$. A complete linear description of the assignment polytope is known [84]. In [13, 14], the *permutation* polytope is studied. This polytope is the convex hull of the vertices of the assignment polytope corresponding to permutations that are in a given group G. While a complete linear description of the permutation polytope is given in [14], that paper also proves that deciding if one of its inequalities is violated by a given matrix $\bar{M}$ is an NP-complete problem.

Let Q be the feasible set of an ILP with variables x and let P be the convex hull of Q. The action of altering the ILP by adding to it a number of variables y, adding a number of constraints and modifying the original constraints such that the projection

of the resulting feasible set on the space of the x variables remains P is known as a *lifting P'* of P. If Q has a symmetry group G, a *symmetric lifting* of P is a lifting P' of P such that G is the restriction of the symmetry group of P' to the x variables. Two important results of Yannakakis [118] are that neither the matching polytope nor the TSP polytope have a symmetric lifting of subexponential size.

One notable exception where the symmetry group G of the polyhedron plays a central role in the study of its facial structure is the problem of obtaining its complete linear description by enumeration. Typically, extreme points of the polyhedron are partitioned into orbits under G and then, for one vertex v in each orbit the facets incident to v are described. These algorithms are based on clever enumeration procedures using the symmetry group of the polyhedron and can be carried out for problems of small dimension. For example, a complete linear description is known for the TSP polytope on a complete graph with up to 10 nodes [27], the Linear Ordering polytope with up to 8 items [27], the Cut polytope on a complete undirected graph with up to 9 nodes [27], the Metric cone and Metric polytope on a complete graph with up to 8 nodes [35, 36].

All enumeration algorithms are limited in the size of instances they can tackle. They might give hints on the form of the complete linear description for all instances but, most of the time, classes of facet defining inequalities for large instances are not facet defining in smaller ones. The search for facet defining inequalities for symmetric polytopes thus requires some effort. One could hope that some generic process could be used to generate strong valid (let alone facet defining) inequalities using the symmetry group of the polytope, but no such process seems to be known. Most derivations of valid inequalities are problem specific and only use the existence of the symmetry group in a limited way. For many symmetric ILPs used to test the existence of combinatorial objects (such as problems similar to those listed in Table 1) the gap between the optimal value of the ILP formulation and its LP relaxation is large. The generation of strong valid inequalities for these problems has received little attention. One recent paper [70] does this for the very hard problem `codbt06`, better known as the Football Pool problem [28, 53] or as the construction of an optimal ternary covering code of length 6. This problem is still open despite extensive efforts for solving it [67, 86].

One main tenet of polyhedral combinatorics is that the knowledge of families of facets of the convex hull P of the feasible set of an ILP is useful for solving the problem, even if the complete linear description of P is not known. If the number of facets in a family F is exponential in the size of the problem encoding, F can still be used in an efficient way providing that it has an efficient *separation algorithm*. Such an algorithm takes a point $\bar{x}$ as input and either outputs one inequality corresponding to a facet in F violated by $\bar{x}$ or guarantees that all such inequalities are satisfied by $\bar{x}$.

Many papers describe facets and separation algorithms for specific symmetric polyhedra. For a symmetric problem with symmetry group G, any valid inequality for P of the form $ax \leq a_0$ generates a collection of "symmetric" inequalities of the form $g(a) \cdot x \leq a_0$ for all $g \in G$. This naturally leads to look for a separation algorithm for this class of inequalities, i.e. an algorithm for the following problem:

LINEAR OPTIMIZATION UNDER SYMMETRY

Input: A group G permuting the elements in I^n given by a set of t generators, a vector $a \in \mathbb{R}^n$ and a point $\bar{x} \in \mathbb{R}^n$;

Output: A permutation $g \in G$ maximizing $g(a) \cdot \bar{x}$.

In practice, the typical situation is to look for the most violated inequality in a class of facets under all possible permutations in G, and numerous examples where this can be done efficiently are known. For example, virtually all exact separation algorithms for facets of the TSP polytope described in [3] fall into this category. Nevertheless, it is straightforward to show that the above problem is NP-hard, using a polynomial transformation from the following problem (a variant of *Problem 5* of [14], variant shown there to be NP-hard):

MAXIMUM OVERLAP UNDER SYMMETRY

Input: A group G permuting the elements in I^n given by a set of t generators, and two functions $\phi_1, \phi_2 : I^n \to \{0,1\}$;

Output: A permutation $g \in G$ maximizing $|\{i \in I^n \mid \phi_1(i) = \phi_2(g(i))\}|$.

Indeed, define $a_i := \phi_2(i)$ and $\bar{x}_i := \phi_1(i)$ for all $i \in I^n$. For $g \in G$, $k, \ell = 0, 1$ and $s = 1, 2$, define

$$y_{k\ell}^g = |\{i \in I^n \mid \phi_1(i) = k, \ \phi_2(g(i)) = \ell\}|$$
$$z_s = |\{i \in I^n \mid \phi_s(i) = 1\}| \ .$$

The first problem asks for a permutation $g \in G$ maximizing y_{11}^g while the second problem asks for maximizing $y_{11}^g + y_{00}^g$. But as $y_{11}^g + y_{10}^g = z_1$ and $y_{10}^g + y_{00}^g = n - z_2$, we have $y_{11}^g + y_{00}^g = 2 y_{11}^g + n - z_1 - z_2$. As the above transformation is polynomial in the size of the instance of the second problem, it is a polynomial time reduction from the second problem to the first one.

7 Partitioning problems

Several classes of symmetric problems arising in practice are of the partitioning type: Given a set S of s elements, find a partition of S into at most t subsets with each of the subsets having to meet the same requirements. There is immediately a symmetry between the subsets in the partition. A typical ILP formulation for such a problem uses binary variables x_{ij} for all $i = 1, \ldots, s$ and $j = 1, \ldots, t$ with the meaning

$$x_{ij} = \begin{cases} 1 & \text{if } i \text{ is assigned to subset } j, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

The problem formulation might use additional variables y_i for $i = 1, \ldots, n_y$ with $y_i \in \mathbb{Z}$ for $i \in Y \subseteq \{1, \ldots n_y\}$. The problem can then be written as

$$\min \quad c^{xT} x + c^{yT} y$$
$$A^x x + A^y y \geq b,$$
$$\sum_{j=1}^{t} x_{ij} = 1, \qquad \text{for all } i = 1, \ldots, s, \qquad (8)$$
$$x_{ij} \in \{0, 1\}, \qquad \text{for all } i = 1, \ldots, s, j = 1, \ldots, t$$
$$y_i \in \mathbb{Z} \qquad \text{for all } i \in Y,$$

where A^x is an $m \times (s \cdot t)$ matrix, A^y is an $m \times n_y$ matrix, c^x is an $(s \cdot t)$-vector, c^y is an n_y-vector, and b is an m-vector and all these matrices and vectors are rational. In addition, we assume that any permutation of the t subsets can be extended to a permutation in the symmetry group of the ILP. In other words, if h is any permutation of I^t, then there exists a permutation h' of I^{n_y} and

$$(x_{11}, \ldots, x_{s1}, x_{12}, \ldots, x_{s2}, \ldots, x_{1t}, \ldots, x_{st}, y_1, \ldots, y_{n_y})$$

is feasible if and only if

$$(x_{1h(1)}, \ldots, x_{sh(1)}, x_{1h(2)}, \ldots, x_{sh(2)}, \ldots, x_{1h(t)}, \ldots, x_{sh(t)}, y_{h'(1)}, \ldots, y_{h'(n_y)})$$

is and both solutions have the same objective value. We also assume that the vector c^x is symmetric with respect to the t subsets, i.e. writing $c(x_{ij})$ for the entry of c^x corresponding to variable x_{ij}, we assume that $c(x_{ij}) = c(x_{ij'})$ for all $j, j' = 1, \ldots, t$.

A few examples of problems fitting this model are bin packing, cutting stock, scheduling on identical machines, graph coloring (either vertex-coloring or edge-coloring), and graph partitioning. Many other practical problems featuring partitions into interchangeable subsets fit this model too (see [108] for examples).

Note that, in addition to the symmetry between the subsets, it is possible that some symmetry between the elements also occurs. Such symmetry occurs for example for graph coloring with a graph with a nontrivial automorphism group, or for bin packing with multiple items having identical dimensions. The material in this section concentrate on dealing with the symmetry between subsets only.

7.1 Dantzig-Wolfe decomposition

To address the symmetry between the subsets of the partition, a Dantzig-Wolfe decomposition approach can be tried: Given the collection of all binary s-vectors z^ℓ for $\ell = 1, \ldots, u$ corresponding to the characteristic vector of a subset of the partition, the above problem can sometimes be reformulated with variables λ^ℓ for $\ell = 1, \ldots, u$ and objective vector c^λ with $c^\lambda_\ell = \sum_{i=1}^{s} c(x_{i1}) z^\ell_i$:

$$\min \quad c^{\lambda T} \lambda + c^{yT} y$$

$$\text{s.t. } A^\lambda\,\lambda + A^{y'}\,y \geq b',$$

$$\sum_{\ell=1}^{u} z_i^\ell\,\lambda^\ell = 1, \qquad \text{for all } i = 1,\ldots,s$$

$$\sum_{\ell=1}^{u} \lambda^\ell = t,$$

$$\lambda^\ell \in \{0,1\} \qquad \text{for } \ell = 1,\ldots,u,$$

$$y_i \in \mathbb{Z} \qquad \text{for all } i \in Y.$$

This reformulation makes the symmetry between the subsets of the partition disappear, as it only asks for t of the given z^ℓ vectors that are disjoint and cover all items in S, without paying attention to their ordering. The disadvantage is of course the large number of variables. This type of formulation requires column generation procedures, as soon as the size of the problem is non trivial. A pricing problem for vectors z^ℓ not included in the formulation then has to be solved.

Examples of successful applications of this approach can be found for the cutting stock problem [114, 115], scheduling [34, 37, 7, 116], edge coloring [83], vertex coloring [79], and graph partitioning [80].

7.2 Partitioning orbitope

Consider the polytope obtained by taking the convex hull of the feasible solutions to the partitioning part of ILP (8), namely:

$$\sum_{j=1}^{t} x_{ij} = 1, \qquad \text{for all } i = 1,\ldots,s, \tag{9}$$

$$x_{ij} \in \{0,1\}, \qquad \text{for all } i = 1,\ldots,s, j = 1,\ldots,t. \tag{10}$$

Assuming that this system is part of a larger problem that has a symmetry between the sets in the partition, one can try to remove that symmetry from the problem by partitioning the feasible set of (9)-(10) in equivalence classes under that symmetry and by selecting one representative of each class. One natural way to do this is to arrange variables x_{ij} for $i = 1,\ldots,s$, $j = 1,\ldots,t$ in a matrix X with x_{ij} being the entry in row i and column j of X. A matrix $\bar{X}$ is *feasible* if its entries $\bar{x}_{ij}$ satisfy the constraints in (9)-(10). The symmetry group G^C considered here is the group that permutes the columns of $\bar{X}$ in any possible way. For column j of $\bar{X}$, define its *value* v_j as

$$v_j = \sum_{i=1}^{s} 2^{s-i} \cdot \bar{x}_{ij}\,.$$

A matrix $\bar{X}$ is then a representative of its equivalence class under G^C if and only if its columns are ordered in non increasing order of their values. The *partitioning orbitope*, introduced in [61], is the polytope obtained as the convex hull of these representative matrices. Its complete linear description is known and is based on *shifted column inequalities* (SCI). An SCI has a *bar* formed by variables $\{x_{\bar{i}\bar{j}}, x_{\bar{i}\bar{j}+1}, \ldots, x_{\bar{i}t}\}$ for some $2 \leq \bar{i} \leq s$, $2 \leq \bar{j} \leq \min\{\bar{i}, t\}$ and a *shifted column* (SC) consisting of $\bar{i} - \bar{j} + 1$ variables, exactly one on each of the diagonals $x_{d+1,1}, x_{d+2,2}, \ldots, x_{d+\bar{j}-1,\bar{j}-1}$ for $d = 0, \ldots, \bar{i} - \bar{j}$ and with the condition that the variable selected in diagonal d has a column index no larger than the one of the variable selected in diagonal $d+1$ for $d = 0, \ldots, \bar{i} - \bar{j} - 1$. See Figure 1. The SCI with bar B and shifted column S is then $x(B) - x(S) \leq 0$, using the notation $x(A) = \sum_{x_{ij} \in A} x_{ij}$.

Theorem 1. *[61] A linear description of the the partitioning orbitope is given by*

$$\sum_{j=1}^{t} x_{ij} = 1, \qquad \text{for all } i = 1, \ldots, s \qquad (11)$$

$$x(B) - x(S) \leq 0, \qquad \text{for all SCI, with } B \text{ its bar and } S \text{ its SC}, \qquad (12)$$

$$x_{ij} = 0, \qquad \text{for } i = 1, \ldots, s, j = i+1, \ldots t, \qquad (13)$$

$$x_{ij} \geq 0, \qquad \text{for } i = 1, \ldots, s, j = 1, \ldots t . \qquad (14)$$

The *packing orbitope* is obtained by replacing in (9) the equality sign by $\leq$. A result similar to Theorem 1 is given in [61] for the packing orbitope.

While the formulation of Theorem 1 has an exponential number of constraints, efficient separation algorithms are given in [61]. Empirical results using this formulation and further development of fixing algorithms based on it can be found in [60].

Note that the ordering of the objects in S, or equivalently the ordering of the rows in the matrix X might have a big influence on the efficiency derived from the partitioning orbitope. An extreme example is solving an edge coloring problem on

Fig. 1 Graphic representation of an SCI with $s = 7$, $t = 6$, $\bar{i} = 6$, and $\bar{j} = 4$. Entries in the shaded rectangle form the bar B and have coefficients 1; exactly one entry in each of the three diagonal segments has a coefficient -1; these three entries form the shifted column S. All other entries have coefficient 0, including entries in the top right shaded part corresponding to constraints (13).
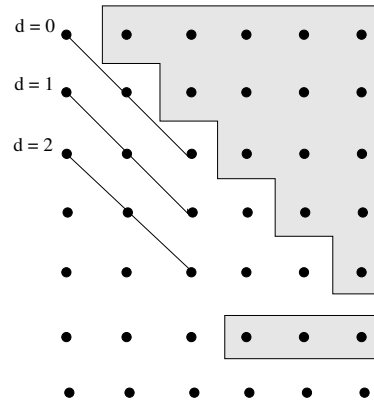
a graph H with maximum vertex degree Δ using the ILP formulation described in Section 5. If the edges corresponding to the first Δ rows of X are edges adjacent to a vertex v of maximum degree in H, the constraints (5)-(6) together with (13)-(14) immediately imply $x_{ii} = 1$ for all $i = 1,\ldots,\Delta$. As a result, all SCI inequalities are satisfied and the gain obtained by using the partitioning orbitope amounts to fixing the colors on the edges adjacent to v. It is likely, however, that a different ordering of the rows of X allows to derive more strength from the SCI. The effect of different orders of the elements in S when using the orbitope has not been investigated.

It could be the case that the partitioning orbitope is particularly useful when it is not clear how to construct a set S of t objects, no two of which can be in the same subset. This is the case for the graph partitioning problem, problem used in the computational experiments of [60].

Alternative integer linear descriptions for the partitioning orbitope are sometimes used and examples can be found in [81, 92]. Generalizing Theorem 1 to the case where the right hand side of (9) is larger than 1 seems difficult, but would have practical implications.

7.3 Asymmetric representatives

Consider the partitioning problem where the goal is to minimize the number of subsets required to partition the s elements, with the additional constraint that the elements in any set U_d from a given list $U = \{U_1,\ldots,U_w\}$ can not all be assigned to the same subset of the partition. (We assume that $|U_d| \geq 2$ for all $d = 1,\ldots,w$ since otherwise the problem is infeasible.) For example, the problem of coloring the nodes of a graph $H = (V,E)$ with the minimum number of colors fits this description, taking V as elements and U as the list of all pairs of elements corresponding to edges in E.

An ILP formulation for this problem (named *Asymmetric Representatives* and developed for node coloring [23, 24, 25] but that can be generalized to handle the description above), uses binary variables z_{ij} for all $i = 1,\ldots,s$ and $j = 1,\ldots,s$ with the meaning

$$z_{ij} = \begin{cases} 1 \text{ if } i \text{ is the representative of } j, \\ 0 \text{ otherwise.} \end{cases} \tag{15}$$

The idea of the formulation is that each element selects an element as its representative, all elements selecting the same representative forming a subset of the partition. Representative elements are elements i with $z_{ii} = 1$. The formulation is as follows.

$$\min \quad \sum_{i=1}^{s} z_{ii}$$

$$\text{s.t.} \quad \sum_{i=1}^{s} z_{ij} = 1, \qquad \text{for all } j = 1,\ldots,s, \qquad (16)$$

$$\sum_{j \in U_d} z_{ij} \leq (|U_d| - 1) \cdot z_{ii}, \qquad \text{for all } d = 1,\ldots,w, \; i = 1,\ldots,s, \qquad (17)$$

$$z_{ij} \in \{0,1\}, \qquad \text{for all } i = 1,\ldots,s, j = 1,\ldots,s. \qquad (18)$$

Inequalities (16) force each element to select exactly one representative, while an inequality (17) prevents the elements in set U_d to all select i as representative and requires that $z_{ii} = 1$ if node i is used as representative of one of the elements in U_d.

This formulation might or might not have symmetries, but the symmetry between the subsets in the partition is destroyed. Nevertheless, this formulation still has several equivalent solutions: for a given partition of the elements, all elements in a subset of the partition can select any of the elements in the subset to get a feasible solution with the same objective value. The formulation can thus also impose that $z_{ij} = 0$ for all $i > j$. (A more general formulation is given in [25], using a partial order on the elements instead of the total order used here.) For coloring the nodes of a graph $H = (V,E)$, the formulation simplifies to

$$\min \quad \sum_{i=1}^{s} z_{ii}$$

$$\text{s.t.} \sum_{(i,j) \notin E} z_{ij} = 1, \quad \text{for all } j = 1,\ldots,s, \qquad (19)$$

$$z_{ij} + z_{ik} \leq z_{ii}, \quad \text{for all distinct } i,j,k, \text{ with } (i,j),(i,k) \notin E, (j,k) \in E, (20)$$

$$z_{ij} \in \{0,1\}, \quad \text{for all } i = 1,\ldots,s, j = 1,\ldots,s, \qquad (21)$$

$$z_{ij} = 0, \qquad \text{for all } i = 1,\ldots,s, j = 1,\ldots,s, \text{ with } i > j, \qquad (22)$$

$$z_{ij} = 0, \qquad \text{for all } (i,j) \in E. \qquad (23)$$

Polyhedral results for this formulation can be found in [23, 24, 25] and [22] gives computational results. Investigation of a similar formulation for the stable set problem is available in [21].

8 Symmetry breaking inequalities

A natural way to get rid of symmetry in a problem is to add symmetry breaking inequalities. There are two main ways to do so. The first one, using *dynamic symmetry breaking inequalities*, is to generate inequalities during the solution process. These inequalities might be invalid for the initial formulation but, due to the development of the enumeration, it is guaranteed that adding them does not prevent the discovery of an optimal solution. The second one, using *static symmetry breaking inequalities*, is to add inequalities to the initial formulation (explicitly or implicitly), cutting some of the symmetric solutions while keeping at least one optimal solution.

An example of static symmetry breaking inequalities are the inequalities describing the partitioning orbitope of Section 7.2.

8.1 Dynamic symmetry breaking inequalities

Several examples of static symmetry breaking inequalities can be found in the Mathematical Programming literature. On the other hand, dynamic symmetry breaking has been investigated mostly in the Constraint Programming literature [43, 45, 50, 91, 94, 96, 97]. The main reason for this is that constraint programming can express constraints in a form different than linear inequalities [111]. Indeed, when the ILP is not binary, some of the constraints described below simply can not be expressed by linear inequalities. For binary problems, however, an example is the *isomorphism inequalities* of [73].

The basis of most dynamic symmetry breaking inequalities is that if a node a of the enumeration tree has a subset of variables $(x_{i_1}, \ldots, x_{i_k})$ fixed respectively to some values $(v_{i_1}, \ldots, v_{i_k})$, then for any $g \in G$ adding an inequality that cuts all solutions with $(x_{g(i_1)}, \ldots, x_{g(i_k)})$ fixed respectively to values $(v_{i_1}, \ldots, v_{i_k})$ can be added at some nodes of the enumeration tree. However, this inequality is not valid for the initial formulation and can only be added during the enumeration.

The drawback of this approach is either the huge number of constraints to handle [50], or the choice of a subset of constraints to use [31, 94, 96], or the design of a separation algorithm. In [45], an implementation based on the computational group theory package `GAP` [113] is presented.

8.2 Static symmetry breaking inequalities

The most general description of static symmetry breaking inequalities for the symmetry group G of ILP (1) is probably the following. A *fundamental region* for G is a closed set F in $\mathbb{R}^n$ such that:

(i) $g(int(F)) \cap int(F) = \emptyset, \ \forall g \in G, \ g \neq I$,

(ii) $\cup_{g \in G} g(F) = \mathbb{R}^n$,

where $int(F)$ denotes the interior of F. Observe that point (i) forces F to be not too large, while point (ii) implies that F contains at least one optimal solution of ILP (1). Indeed, if x^* is an optimal solution, (ii) guarantees that $g(F)$ contains x^* for some $g \in G$ or, equivalently, that $g^{-1}(x^*) \in F$. We thus get:

Theorem 2. *Let G be the symmetry group for ILP (1) and let F be a fundamental region for G. Then an optimal solution to ILP (1) can be found by optimizing over the intersection of the feasible set of ILP (1) with F.*

It turns out that finding a linear description of a fundamental region for G is quite easy, at least in theory. The following result can be found in [52].

Theorem 3. *Let G be the symmetry group of ILP (1) and let $\bar{x} \in \mathbb{R}^n$ such that $g(\bar{x}) \neq \bar{x}$ for all $g \in G, g \neq I$. Then*

$$F = \{x \in \mathbb{R}^n \mid (g(\bar{x}) - \bar{x}) \cdot x \leq 0, \ \forall g \in G, \ g \neq I\} \tag{24}$$

is a fundamental region for G.

The obvious practical weakness of this result is the huge number of inequalities (one for each permutation $g \in G$, except the identity) in this description with many of them not being facet defining for F. Another weakness, shared by almost all practical methods using static symmetry breaking inequalities, is that several isomorphic solutions might still be present in the boundary of F. In practice, relatively simple sets of static symmetry breaking inequalities are used, and most of them can be derived using a weak version of Theorem 2, as given in the next corollary.

Corollary 1. *Theorem 2 remains true when the fundamental region F is replaced by the region obtained from Theorem 3 by relaxing its statement in either of the following ways (or both):*

(i) Inequalities in (24) are written only for a subset of permutations in G;
(ii) The condition that $g(\bar{x}) \neq \bar{x}$ for all $g \in G$, $g \neq I$ is removed.

Proof. The proof of (i) is immediate. For (ii), let F be the feasible set defined by (24) for $\bar{x}$. For any real number $\varepsilon > 0$, define $\bar{x}(\varepsilon) \in \mathbb{R}^n$ such that $\bar{x}(\varepsilon)_i = \bar{x}_i + \varepsilon^i$ for $i = 1, \ldots, n$. For $\varepsilon > 0$ small enough, all components of $\bar{x}(\varepsilon)$ are distinct and thus, using (24), it defines a fundamental region F_ε for G. Let Q be the set of feasible solutions to ILP (1) and let $z \in Q - F$. We now show that there exists $\varepsilon(z) > 0$, such that $z \notin F_\varepsilon$ for all $0 < \varepsilon < \varepsilon(z)$. This implies that F contains all the feasible points from a fundamental region, yielding the result. Define

$$h_z(\varepsilon) = \max_{g \in G} \{(g(\bar{x}_\varepsilon) - \bar{x}_\varepsilon) \cdot z\} \ .$$

Observe that $h_z(\varepsilon)$ is the maximum of a finite number of polynomials in ε of degree n and thus is a continuous function in ε. As $z \notin F$, we have $h_z(0) > 0$, implying that there exists $\varepsilon(z) > 0$ such that $h_z(\varepsilon) > 0$ for all $0 < \varepsilon < \varepsilon(z)$. As Q is a finite set, for $0 < \delta < \min\{\varepsilon(z) \mid z \in Q\}$ we have $F_\delta \cap Q \subseteq F \cap Q$. $\qquad \square$

We list a few applications of Theorem 2 or Corollary 1 below.

i) The ILP has integer variables $0 \leq x_i \leq k$ for $i = 1, \ldots, n$ and G restricted on these variables contains all permutations of I^n. Add inequalities

$$x_1 \geq x_2 \geq \ldots \geq x_n \ .$$

This result is widely known and used routinely. It can be obtained using Corollary 1 with $\bar{x}$ defined by $\bar{x}_i = i$ for $i = 1, \ldots, n$ and observing that the $n-1$ inequalities given above dominate the remainder of the inequalities obtained from the theorem.

ii) The ILP has integer variables $0 \leq x_{ij} \leq k$ for $i = 1, \ldots, s$ and $j = 1, \ldots, t$ and G, when restricted to these variables, contains all permutations of the first indices and all the permutations of the second indices. In other words, when arranging the variables x_{ij} in a two dimensional matrix X as in Section 7.2, all permutations of the rows of X and all the permutations of the columns of X can be extended to permutations in G. Add inequalities expressing that the columns of X must be in non increasing lexicographic order and that the rows of X must be in non increasing lexicographic order [40, 41]. This is the Lex^2 symmetry breaking set, following the terminology of [96]. This result can be obtained from Theorem 2 using a matrix $\bar{X}$ with $\bar{X}_{ij} = (k+1)^{s \cdot t - (i-1) \cdot t - j}$ for $i = 1, \ldots, s$ and $j = 1, \ldots, t$. For example, for $s = 3, t = 4$ and $k = 1$ we get

$$\bar{X} = \begin{pmatrix} 2048 & 1024 & 512 & 256 \\ 128 & 64 & 32 & 16 \\ 8 & 4 & 2 & 1 \end{pmatrix} .$$

Using only the permutations in G that either swap two adjacent columns or swap two adjacent rows, we get the Lex^2 set of constraints. For example, using the permutation swapping the first two columns, we get the inequality

$$-1024\, x_{11} - 64\, x_{21} - 4\, x_{31} + 1024\, x_{12} + 64\, x_{22} + 4\, x_{32} \leq 0 \qquad (25)$$

implying the non increasing lexicographic ordering of these columns. Similarly, using the permutation swapping the first two rows, we get the inequality

$$\begin{aligned} &- 1920\, x_{11} - 960\, x_{12} - 480\, x_{13} - 240\, x_{14} \\ &+ 1920\, x_{21} + 960\, x_{22} + 480\, x_{23} + 240\, x_{24} \leq 0 \end{aligned} \qquad (26)$$

implying the non increasing lexicographic ordering of these rows. Of course, using inequalities (25) or (26) is not advisable in practice when s or t is large, due to the numerical instability introduced by large coefficients. This is an example where Constraint Programming is a more flexible framework than Mathematical Programming to handle symmetry, as the lexicographic orderings on the rows and columns do not have to be expressed as linear inequalities.

However, in the case where $k = 1$ and exactly one entry in each row must have value 1, we can do much better than using the linear inequalities above: To enforce the non increasing lexicographic ordering of the columns, we can use the Shifted Column inequalities of Section 7.2 and to enforce the non increasing lexicographic ordering of the rows, we can use the inequalities

$$x_{ip} \leq \sum_{j=p}^{t} x_{i+1,j} \qquad \text{for } i = 1, \ldots, s-1, \ p = 1 \ldots, t .$$

Note that [41] shows that when $\bar{X}$ must have exactly one 1 per row, the *Lex*^{2} set of constraints can be reinforced by adding the constraints that the sum of the entries in each column is also non increasing, i.e. adding the constraints

$$\sum_{i=1}^{s} x_{ij} \geq \sum_{i=1}^{s} x_{i,j+1} \qquad \text{for } j = 1 \dots, t-1 .$$

This result is of course implied by the inequalities obtained from Theorem 2 (it is implied by the lexicographic ordering of the columns and of the rows) but it does not seem easy to derive it algebraically directly from the inequalities obtained from the theorem.

iii) The generalization of (ii) where the matrix X is d-dimensional with $d \geq 3$ and where any permutation of indices along any dimension of the matrix can be extended to a permutation in G can be handled similarly to (ii). For $i = 1, \dots, d$, define $X_{i,\ell}$ as the $(d-1)$-dimensional matrix obtained from X by selecting all entries whose ith index is equal to ℓ.

Imposing, for each $i = 1, \dots, d$, and each possible value of ℓ that the entries in $X_{i,\ell}$ are lexicographically not smaller than entries in $X_{i,\ell+1}$ is valid [40]. This result can be derived from Theorem 2 similarly to (ii). A weaker result for the case where entries in X are binary, imposing only that the sum of the entries in $X_{i,\ell}$ is not smaller than the sum of the entries in $X_{i,\ell+1}$ is given in [101].

(iv) The matrix X is a $s \times t$ matrix and any permutation of the columns of X can be extended to a permutation in G. A lexicographic ordering on the columns of X can be imposed. This can be done using the inequalities

$$\sum_{i=1}^{s} (k+1)^{s-i} \cdot x_{ij} \geq \sum_{i=1}^{s} (k+1)^{s-i} \cdot x_{i,j+1} \qquad \text{for } j = 1, \dots, t-1,$$

but this is not numerically very stable when s is large. Note that this is identical to case (i) applied to the variables corresponding to $\sum_{i=1}^{s} (k+1)^{s-i} \cdot x_{ij}$ for $j = 1, \dots, t$. Application of this idea can be found [33] for solving a layout problem and in [56] for solving lot-sizing problems. Of course, in special cases, the partitioning orbitope of Section 7.2 for example, better inequalities can be used. Weaker conditions have also been tested on practical problems (see [56, 93, 108] for examples and comparisons) such as the following three possibilities: First,

$$\sum_{i=1}^{s} x_{ij} \geq \sum_{i=1}^{s} x_{i,j+1} \qquad \text{for } j = 1, \dots, t-1 .$$

This can be obtained from Corollary 1 using $\bar{X}_{ij} = j$ for $i = 1, \dots, s$, $j = 1 \dots, t$. Second,

$$\sum_{i=1}^{s} i \cdot x_{ij} \geq \sum_{i=1}^{s} i \cdot x_{i,j+1} \qquad \text{for } j = 1, \dots, t-1,$$

This can be obtained from Corollary 1 using $\bar{X}_{ij} = i \cdot j$ for $i = 1, \ldots, s$, $j = 1 \ldots, t$. Finally,

$$\sum_{i=1}^{s} i^2 \cdot x_{ij} \geq \sum_{i=1}^{s} i^2 \cdot x_{i,j+1} \qquad \text{for } j = 1, \ldots, t-1 \ .$$

This can be obtained from Corollary 1 using $\bar{X}_{ij} = i^2 \cdot j$ for $i = 1, \ldots, s$, $j = 1 \ldots, t$.

It should be noted that the impact of different sets of static symmetry breaking inequalities is difficult to estimate. In most cases, only empirical evaluation of specific implementations for specific classes of problems are available. Very little is known about desirable properties of such a set. Discussion related to the choice of $\bar{x}$ and separation of the inequalities in (24) when G is the symmetric group Π^n or a cyclic group can be found in [44]. In [66], a ranking of sets of static symmetry breaking inequalities is introduced. It is based on the maximum number of points in the orbit of an extreme point of the polytope that are cut by the set of inequalities.

The discussion of efficiency of different sets of static symmetry breaking inequalities when embedded in a branch-and-bound algorithm is muddled by the interaction between the set of inequalities and valid choices for branching decisions. Some experiments have been made [56, 107, 108, 109], but no definite answer is available. There are simply too many variables to consider: problem classes, formulation choice, large or small group order, choice of algorithm, coupling with other symmetry breaking techniques, etc.

Deciding with confidence beforehand that using a given set of dynamic symmetry breaking inequalities is better or worse than using a given set of static symmetry breaking inequalities is extremely difficult. One general rule of thumb is that for problems with a symmetry group of order up to a few thousands of permutations, dynamic symmetry breaking might be very effective. However, when the order of the symmetry groups is larger than, say, a million, dynamic symmetry breaking inequalities can be effective, but only when coupled with other symmetry breaking techniques. Some limited comparisons are reported in [91, 97].

9 Pruning the enumeration tree

A special case of static symmetry breaking inequalities for ILP (1) is obtained from Theorem 2 and Theorem 3 using $\bar{x}_i = (k+1)^{n-i}$, for $= 1, \ldots, n$ in the latter. Then, a vector $z \in \{0, \ldots, k\}^n$ is in the fundamental region F if and only if $(g(\bar{x}) - \bar{x}) \cdot z \leq 0$ for all $g \in G$, which is equivalent to

$$\max\{g(\bar{x}) \cdot z \mid g \in G\} \leq \bar{x} \cdot z \qquad \text{and to} \qquad \max\{\bar{x} \cdot g(z) \mid g \in G\} \leq \bar{x} \cdot z.$$

This last expression is equivalent to say that z is in F if and only if z is lexicomax in its orbit under G, due to the particular choice of $\bar{x}$. This is hardly a surprising or difficult result, and restricting the search for lexicomax (or lexicomin) solutions

in their orbit has been used routinely (see [19, 100, 104, 112], just to name a few, more general expositions and applications can be found in [62, 78]). In [31], in the setting of clausal propositional logic, predicates similar to the constraints above are derived. Results on reducing the number of constraints that must be included to break all symmetries is also discussed. However, the number of constraints that must be included is, in general, too large for this approach to work. Experiments with a small subset of the constraints that does not break all symmetries have been tried [1, 31, 94, 96]. Moreover, even if it were possible to add all these inequalities, the resulting feasible set rarely is an integral polytope, implying that some work is left do be done to solve the problem. An alternative is to handle these constraints by pruning the enumeration tree: Node a of the enumeration tree is pruned if it can be shown that none of the solutions in the subtree rooted at a is lexicomax in its orbit.

A direct use of this idea leads to algorithms that fix (or build) an order on the variables and where lexicomax solutions with respect to that order are sought. These algorithms are presented in Section 9.1. However, it is possible to relax the need of an order on the variables. The resulting algorithms are covered in Section 9.2. Both types of algorithms are similar and it might help to study first algorithms working with a fixed order on the variables, as their description is a little bit simpler.

We focus on three different algorithms for tackling problems with arbitrary symmetry groups. These algorithms were developed independently of each other and look different, although they all use the same basic principles. These algorithms are: Symmetry Backtracking Search (SBS) [11, 12], Symmetry Breaking via Dominance Detection (SBDD) [39, 46, 96], and Isomorphism Pruning (IsoP) [74, 75]. Algorithms for handling special symmetry groups (such as product of several disjoint symmetric groups) have been studied too [42].

Related but simpler and less efficient algorithms for solving quadratic assignment problems can be found in [8, 76]. They essentially just avoid the creation of isomorphic subproblems from the same parent node. Nevertheless, this simple operation makes a difference when solving difficult benchmark quadratic assignment problems [2].

A few definitions are necessary to help in the description of the algorithms.

Let a be a node of the enumeration tree T and, for $i = 1, \ldots, n$, let D_i^a be the possible values for variable x_i at node a. Let ILP^a be the ILP (1) with the additional constraints $x_i \in D_i^a$ for $i = 1, \ldots, n$. The *path of a* in T is the path P_a from the root node of T to a. A node a_1 is a *son* of node a if aa_1 is an edge of T and the path of a_1 goes through a. See Figure 2.

We consider the following three branching rules at node a:

(i) *Partitioning*: Select a variable x_i with $|D_i^a| \geq 2$. Partition D_i^a into $2 \leq \ell \leq |D_i^a|$ non empty sets $D_i^a(1), \ldots, D_i^a(\ell)$. Create the ILP for son a_j for $j = 1, \ldots, \ell$ by replacing $x_i \in D_i^a$ in ILP^a by $x_i \in D_i^a(j)$. To avoid cumbersome notation, we assume that the partition of D_i^a satisfies that, for all $1 \leq j \leq \ell - 1$, if $t \in D_i^a(j)$ and $t' \in D_i^a(j+1)$ then $t < t'$.

(ii) *Splitting*: Select a variable x_i such that $D_i^a = \{v_1, \ldots, v_\ell\}$ with $\ell \geq 2$ and $v_j < v_{j+1}$ for $j = 1, \ldots, \ell - 1$. Create the ILP for son a_j for $j = 1, \ldots, \ell$, by replacing $x_i \in D_i^a$ in ILP^a by $x_i = v_j$.

(iii) *Minimum Index Splitting*: Similar to (ii), the only difference is that i must be the smallest index with $|D_i^a| \geq 2$.

It should be clear that (ii) is more restrictive than (i) and that (iii) is more restrictive than (ii). Note also that if one chooses $\ell = |D_i^a|$ in (i), the resulting rule is (ii). In this section, we assume implicitly that one of these three rules is used, with an arbitrary rule for selecting the branching variable for Partitioning and Splitting. Suppose that a branching rule is fixed and let T be the enumeration tree obtained following that rule, pruning nodes only when they are infeasible (pruned nodes are included in T). Tree T is called the *full enumeration tree* for the selected branching rule. Note that if a is a feasible leaf of T then $|D_i^a| = 1$ for all $i = 1, \ldots, n$. Feasible leaves of T are in bijection with the feasible solutions of the ILP.

9.1 Pruning with a fixed order on the variables

In this section, we assume that a total order on the variables is fixed from the beginning of the algorithm. To simplify notation and without loss of generality we take this order as the natural order defined by the indices on the variables, with x_1 being the first variable in the order.

Note that the material in this section can be adapted so that the order used is built during the execution of the algorithm: The algorithm works with a partial order that is refined during the execution. The initial partial order is an order where all variables have the same priority. Then, each time an operation is performed, it must be valid for an extension of the current partial order and the partial order is modified to include the corresponding constraints. However, the resulting algorithm can be seen as a constrained version of the algorithms presented in Section 9.2 and as such does not deserve much attention.

We know that node a of the enumeration tree can be pruned if none of the optimal solutions in the subtree rooted at a can be lexicomax in its orbit. It is possible to

Fig. 2 Node a and the ordering of its sons; path P_a (in bold); b to the immediate left of P_a; c to the left of P_a.
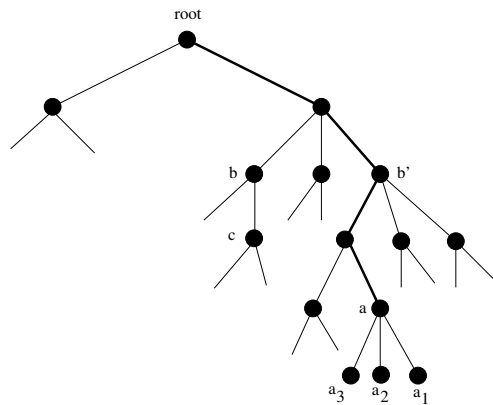
identify situations were this holds using the following extension of lexicographic ordering.

Given two nodes a and b of T, we say that $D^b = (D_1^b, \ldots, D_n^b)$ is *lexico-set larger* than $D^a = (D_1^a, \ldots, D_n^a)$, written $D^b >_{Ls} D^a$, if and only for some $t \in \{1, \ldots, n\}$ we have $\min\{j \mid j \in D_i^b\} \geq \max\{j \mid j \in D_i^a\}$ for $i = 1, \ldots, t-1$ and $\min\{j \mid j \in D_t^b\} > \max\{j \mid j \in D_t^a\}$. By extension, the definition applies to any pair of n-vectors of subsets of I^n. We write $g(D^a)$ for the vector of subsets obtained by permuting the entries of D^a according to g.

Example 3. Consider $D_1^a = \{0,1\}$, $D_2^a = \{0\}$, $D_3^a = \{1\}$, $D_4^a = \{1\}$ and $D_1^b = \{1\}$, $D_2^b = \{1\}$, $D_3^b = \{0\}$, $D_4^b = \{0,1\}$. We have $D^b >_{Ls} D^a$ and for $g = (3,4,2,1)$, we have $g(D^a) = D^b$.

□

We then have:

Theorem 4. *Let a be a node of the full enumeration tree T. If there exists $g \in G$ such that $g(D^a) >_{Ls} D^a$ then node a can be pruned.*

The pruning done by Theorem 4 is called *isomorphism pruning* (*IP*). It is immediate that if IP is able to prune node a, it is also able to prune all sons d of a, as $D_i^d \subseteq D_i^a$ for all $i = 1, \ldots, n$ and thus $g(D^d) >_{Ls} D^d$ if $g(D^a) >_{Ls} D^a$. Nodes of T that are not pruned thus form a subtree of T containing the root of T as well as all solutions that are lexicomax in their orbit. The validity of the pruning follows.

If we assume that at the root r of T we have $D_i^r = \{0, \ldots, k\}$ for all $i = 1, \ldots, n$, IP is virtually useless until branching on variable x_1 has occurred. In general, IP is more efficient when the domains D_i^a for $i = 1, \ldots, t$ are small and t is large. As a consequence, most algorithms using IP also use the Minimum Index Splitting rule [19, 26, 72, 73, 74, 75, 96, 100].

Having the branching order essentially fixed from the beginning is a minor restriction when pure backtracking enumeration algorithms are used, but it is potentially a major drawback when using branch-and-bound algorithms or other domain reduction techniques, as it is well known that a clever branching variable choice can reduce the enumeration tree drastically. Nevertheless, algorithms based on IP and a fixed ordering of the variables have been shown, on many instances, to be orders of magnitude faster than a branch-and-bound algorithm oblivious to existing symmetries. It should be noted, however, that different orderings of the variables produce wildly different performance, transforming a problem that can be solved in seconds into one that is essentially impossible to solve. For many problems, finding a "reasonable" ordering of the variables is not too difficult. However, as mentioned in Section 5, for proving that no proper coloring of the edges of a clique on 9 nodes with 8 colors exists, two "reasonable" ordering of the variables yield running times that are orders of magnitude apart.

9.1.1 Additional domain reduction or additional inequalities

Some care must be taken when additional inequalities or variable domain reduction techniques are used together with IP (or symmetry breaking inequalities, but we focus on IP here). It is valid to use either, provided that it can be shown that at least one optimal solution x^* that is lexicomax in its orbit under G remains feasible. This is usually quite difficult to prove, but the following are examples where this is possible:

(i)   ILP cutting planes: Add any inequality to ILP^a that is valid for the convex hull of the integer solutions of ILP^a; indeed, as no integer point is cut by these, all optimal solutions that are lexicomax in their orbit are kept. This implies that any of the standard cutting plane generators for ILP (Gomory, Cover, Knapsack, etc.) can be used.

(ii) Strict exclusion algorithms: Excluding value v from D_i^a for some i is valid if it is known that no optimal solution $\bar{x}$ of ILP, valid for ILP^a and lexicomax in its orbit, has $\bar{x}_i = v$. This implies that many of the usual techniques for excluding values for variables can be used. It is however necessary to stress that merely guaranteeing that there is an optimal solution $\bar{x}$ of ILP^a with $\bar{x}_i \neq v$ is not enough for having the right of excluding value v from D_i^a.

(iii) Strict exclusion algorithms working under symmetry: Similar to (ii) with the additional constraint that if $x_i \neq v$ is produced by the algorithm for ILP^a then, for any $g \in G$, it can produce that $x_{g(i)} \neq v$ in the ILP obtained by permuting the variables in ILP^a according to g. This requirement prevents the use of exclusion algorithms that use information related to lexicographic order. It is, however, usually met by typical exclusion algorithms that are used in ILP. The interest of imposing the constraint of working under symmetry on the exclusion algorithms is explained later in this section and also in Section 10.

(iv) IP exclusion: Suppose that at node a, for some $g \in G$, and for some $t \in \{1,\dots,n\}$, we have $\min\{j \mid j \in D_{g(i)}^a\} = \max\{j \mid j \in D_i^a\}$ for $i = 1,\dots,t$ and $\max\{j \mid j \in D_{g(t)}^a\} > \max\{j \mid j \in D_t^a\}$. Notice that any feasible solution $\bar{x}$ for ILP^a with $\bar{x}_{g(t)} = v$ for $v = \max\{j \mid j \in D_{g(t)}^a\}$ is not lexicomax in its orbit, as $g^{-1}(\bar{x})$ is lexicographically larger. It is thus valid to exclude v from $D_{g(t)}^a$.

(v)   Orbit exclusion: Suppose that at node a, for some $p \geq 1$, we have $|D_i^a| = 1$ for $i = 1,\dots,p$ and $|D_{p+1}^a| > 1$. Let v be the vector defined by $v_i = D_i^a$ for $i = 1,\dots,p$ and $v_i = -1$ otherwise. Let $\mathbb{O}$ be an orbit under the stabilizer of v in G. Let D be the intersection of the domains D_i^a for all $i \in \mathbb{O}$. Set $D_i^a = D$ for all $i \in \mathbb{O}$.

Note that (ii) and (iii) were introduced in [74, 75] and that particular cases of (iv) have been used in [11, 12] as well as many papers in the Constraint Programming literature (this is a filtering algorithm), as well as under the name *0-fixing* in [74, 75]. Observe that (iv) does not fit the conditions of (iii). It is important to understand also that (v) is valid only if the other exclusion algorithms used satisfy (iii) or (iv). (v) was introduced in [75] under the name *orbit fixing* and a generalization of (v) is also given there.

9.2 Pruning without a fixed order of the variables

The practical need, mentioned in Section 9.1, of branching using the Minimum Index Splitting to perform IP can however be relaxed. A few definitions are needed before continuing the presentation.

Graphically, the full enumeration tree T is drawn with the root at the top and the sons a_j of a for $j = 1, \ldots, \ell$ are drawn from right to left below a, starting with a_1 (see Figure 2). We say that $a_{j'}$ is to the left of a_j if $j < j'$. A node b is *to the immediate left of P_a* if b is the son of a node $c \in P_a - a$ and b is to the left of the son of c that is part of P_a. A node c is *to the left of P_a* if c is in the subtree rooted at a node to the immediate left of P_a.

If we assume that the domains of the variables are modified only by the branching operation, we have [39]:

Theorem 5. *Let a be a node of the full enumeration tree T. If there exists a node b to the immediate left of the path of a and a permutation $g \in G$ such that $D^a_{g(i)} \subseteq D^b_i$ for $i = 1, \ldots, n$ then a can be pruned.*

Proof. All optimal solutions of the ILP are feasible leaves of T. Since T is drawn according to the convention above, the orbit $\mathbb{O}$ under G of any optimal solution has one solution s^* that is to the left of the path to any $s \in \mathbb{O} - s^*$. We claim that no node t in the path to s^* is pruned by the pruning of the statement. Indeed, if t is pruned, then there exists b to the left of the path of t and $g \in G$ with $D^t_{g(i)} \subseteq D^b_i$ for $i = 1, \ldots, n$. But then $g(s^*)$ is a feasible leaf of T to the left of the path to s^*, a contradiction with the definition of s^*.

$\square$

To showcase the connection between Theorem 5 and the lexicographic pruning of Section 9.1, suppose that the branching rule used to produce T in Theorem 5 is the Minimum Index Splitting rule. The pruning made by Theorem 5 is then equivalent to the one done by Theorem 4.

One major difference, however, between the two theorems is that Theorem 4 has the same efficiency when additional exclusion of values are done whereas, as stated, Theorem 5 might miss some pruning due to shrinkage of some domains in b. This motivates the tracking of branching decisions that have been made on the path of a. Let o^a be the *order list* at a, where o^a is simply the ordered list of the indices of the variables used as branching variables on the path of a. In this section, we assume that the Splitting branching rule is used to simplify the presentation, but the algorithms can handle the Partitioning branching rule.

The definition of lexico-set larger given in Section 9.1 can be extended so that comparisons are made according to an ordered list of indices: Let o be an ordered list of p indices with o_i denoting the ith element in the list. We say that, with respect to o, $D^b = (D^b_1, \ldots, D^b_n)$ is *lexico-set larger* than $D^a = (D^a_1, \ldots, D^a_n)$, written $D^b >_{os} D^a$, if and only for some $t \in \{1, \ldots, p\}$ we have $\min\{j \mid j \in D^b_{o_i}\} \geq \max\{j \mid j \in D^a_{o_i}\}$ for $i = 1, \ldots, t-1$ and $\min\{j \mid j \in D^b_{o_t}\} > \max\{j \mid j \in D^a_{o_t}\}$.

We then have [11, 12]:

Theorem 6. *Let a be a node of the full enumeration tree T obtained using the Splitting branching rule and let o^a be the order list at a. If there exists $g \in G$ such that $g(D^a) >_{o^a_S} D^a$ then node a can be pruned.*

9.2.1 Additional domain reduction or additional inequalities

As in Section 9.1, it is possible to couple the pruning of Theorem 6 with domain reduction techniques and cutting planes, with restrictions similar to those listed in Section 9.1.1.

It is easy to overlook that two techniques conflict with each other. An example reported in the literature [96] is the attempt to use jointly the pruning of Theorem 6 with the Lex^2 symmetry breaking constraints described in Section 8.2.

While this is perfectly valid if Theorem 4 or Theorem 6 is used with an ordering of the variable for which lexicomax solutions satisfy the Lex^2 constraints, it might fail when Theorem 6 is used with an arbitrary order.

10 Group representation and operations

While Theorem 5 and Theorem 6 form the basis for pruning algorithms, they are existence results. For a practical implementation, we need an algorithm checking if there exists $g \in G$ satisfying the statement. The implementation of SBS from [11, 12] and implementation of IsoP from [74, 75] work with an arbitrary group given by a collection of generators. The implementations of SBDD from [39, 96] use problem specific algorithms or work only for very simple groups (typically symmetric groups). The SBDD implementation of [46] uses an approach similar to [11, 12], but few details are available, preventing a finer comparison with the implementations of SBS and IsoP. This section covers the basics for handling computational group operations needed in a branch-and-bound algorithm and points differences between the implementations of [11, 12] and [74, 75][3].

The group representation and algorithms are based on the *Schreier-Sims* representation of G, a tool widely used in computational group theory [16, 17, 18, 19, 54, 62, 64, 65]. The reader is referred to [16, 17, 55, 106] for a comprehensive overview of the field.

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

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

---

[3] The algorithms of [74, 75] assume a fixed order of the variables, but as pointed out first by Ostrowski [87] and as the presentation in this paper shows, this requirement can be waived.

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

Example 4. As observed in Example 1, the symmetry group of ILP (2) is $G = \{I, (2,3,4,1), (3,4,1,2), (4,1,2,3), (3,2,1,4), (4,3,2,1), (1,4,3,2), (2,1,4,3)\}$.

We have $G_0 := G$ and $orb(1, G_0) = \{1,2,3,4\}$ with $h_{1,1} = I$, $h_{1,2} = (2,3,4,1)$, $h_{1,3} = (3,4,1,2)$, and $h_{1,4} = (4,1,2,3)$. Then $G_1 := stab(1, G_0) = \{I, (1,4,3,2)\}$ and $orb(2, G_1) = \{2,4\}$ with $h_{2,2} = I$ and $h_{2,4} = (1,4,3,2)$.

Finally, $G_2 := stab(2, G_1) = \{I\}$, $G_3 := stab(3, G_2) = \{I\}$ and $G_4 := stab(4, G_3) = \{I\}$. The corresponding table T is:

	1	2	3	4
1	I	$h_{1,2}$	$h_{1,3}$	$h_{1,4}$
2		I		$h_{2,4}$
3			I	
4				I

$\square$

The table T is called the Schreier-Sims representation of G. 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 $G(\beta)_t$ for $t = 1, \ldots, n$ are defined as the stabilizer of β_t in $G(\beta)_{t-1}$, with $G(\beta)_0 = G$. The corresponding table is denoted by $T(\beta)$. Row t of $T(\beta)$ corresponds to the element t, $U(\beta)_t$ is the set of non empty entries in row t of $T(\beta)$ and $J(\beta)_t$ denotes the corresponding set of indices $\{j \in I^n \mid T(\beta)[t,j] \neq \emptyset\}$, also called the *basic orbit* of t in T (following the terminology of [65]). 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.

The most interesting property of this representation of G is that each $g \in G$ can be uniquely written as

$$g = g_1 \cdot g_2 \cdot \ldots \cdot g_n \tag{27}$$

with $g_i \in U_i$ for $i = 1, \ldots n$. Hence the permutations in the table form a set of generators of G. It is called a strong set of generators, since the equation (27) shows that $g \in G$ can be expressed as a product of at most n permutations in the set.

Given a permutation $g \in G$, it is easy to find the n permutations $g_1, \ldots, g_n$ of equation (27): the permutations $g_2, \ldots, g_n$ all stabilize element 1, forcing g_1 to be $T[1, g(1)]$. Then, as $g_3, \ldots, g_n$ all stabilize element 2, we must have $(g_1 \cdot g_2)(2) = g(2)$, i.e. $g_2(2) = (g_1^{-1} \cdot g)(2)$ and thus $g_2 = T[2, (g_1^{-1} \cdot g)(2)]$. A similar reasoning yields $g_3, \ldots, g_n$.

Algorithms for creating the table $T(\beta)$ and for changing the base β of the representation can be found in [16, 18, 19, 54, 55, 62, 64, 65, 106]. For a group G given by a set T of generators, algorithms for creating the table and with worst-case running time in $O(n^6 + n^2 \cdot |T|)$ [62] have been devised. Faster but more complex algorithms are also known [5, 55, 58, 105, 106]. The complexity of the algorithm of Jerrum

[58] is in $O(n^5 + n^2 \cdot |T|)$ and the one of Babai et al. [5] is in $O(n^4 \cdot \log_c n + n^2 \cdot |T|)$ where c is a constant and one from [106] is $O(n^2 \cdot \log^3 |G| + |T| \cdot n^2 \cdot \log |G|)$. Since we might assume that the permutation group is given by a set of strong generators, the speed of the algorithm for finding the representation of the group is not particularly relevant here. Note also that the cardinality of the ground set of the groups that are usually of interest are small (for computational group theory standards, at least) and that the simpler algorithms perform satisfactorily in the large majority of the cases.

Algorithms for changing the base of the table can be found in [11, 19, 30, 55, 62, 106] with worst-case running time up to $O(n^6)$, while more complex algorithms run in almost linear time. An algorithm with worst case complexity in $O(n^6)$ or even $O(n^4)$ might seem impractical for values of $n \geq 100$. It turns out that the complexity bounds given above are very pessimistic and are usually attained for the symmetric group on n elements. The amount of time spent in the algorithms dealing with the group operations for the applications of [74, 75] stays below 10% of the total cpu time.

Although the algorithms are described here 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. For example, when solving the covering design problem $cov1054$ mentioned in Section 1, $n = 252$, the group has order $10! = 3,628,800$, but the number of entries in the table is, on average, 550. It is worth noting that most of the cpu time used by the group algorithms is spent multiplying permutations. Speedup may be obtained in some cases by keeping permutations in product form (see [55, 106] for details).

Property (27) is the corner stone of the algorithm testing the existence of $g \in G$ satisfying Theorem 6. To simplify the notation, assume that the Minimum Index Splitting rule is used, and thus at node a, for some $p \geq 1$, we have $o_i^a = i$ and $x_i = v_i$ for $i = 1, \ldots, p$.

We use a backtracking algorithm to construct g, if it exists: For $v = 1, \ldots, k$, let F_v^a be the set of indices of variables that have value v at a, i.e. $F_v^a = \{i \in I^n \mid D_i^a = \{v\}\}$.

0) Let $g_0 := I$, let $i = 1$ and let T be a Schreier-Sims table for G with base $(1, 2, \ldots, n)$ and initialize the sets F_v^a for $v = 0, \ldots, k$.
1) Let v be the value that x_i takes at a;
2) If $g_{i-1}^{-1}(i) \in F_w^a$ for some $w > v$ then STOP;
3) For all $j \in g_{i-1}(F_v^a)$ do

    3.1) Let $h_i := T[i, j]$
    3.2) If $h_i \neq \emptyset$ then
        3.2.1) Remove index $g_{i-1}^{-1}(j)$ from F_v^a
        3.2.2) $g_i := h_i^{-1} \cdot g_{i-1}$;
        3.2.3) $i := i + 1$; If $i \leq p$ then go to step 1);

If the algorithm terminates in step 2), then g_{i-1} is a permutation showing that a can be pruned, according to Theorem 6. Although this algorithm is essentially the one used in implementations, important variations occur. The implementations

of `SBS` of [11, 12] and of `IsoP` of [74, 75] differ in modifications reducing the pruning that is done to improve running time. Another important difference is that in [11, 12], the algorithm is called before the branching variable is chosen while in [74, 75] it is called to weed out all values in the domain of the selected branching variable that would lead to a node that could be pruned by isomorphism. Advantages of the former is that information about the orbits of variables can be used to select the branching variable (experiments with different selection rules are described in [88]), while the latter leverages the fact that the operations performed by the algorithm at different sons of a node are very similar and can be collapsed efficiently in one application of the algorithm as described below.

Note that in both implementations of [11, 12] and [74, 75], the sets F_v^a contain only the indices in $\{1, \ldots, p\}$. The motivation for this choice is firstly the orbit exclusion algorithm of Section 9.1.1. Secondly, the fact that the stabilizer used in the orbit exclusion algorithm is a group means that orbit computations can be performed by computing generators of that group.

This last point is the approach taken in [11, 12], where a clever shortcut is used: when step 3.2.2) is reached with $i = p$, orbits are updated to reflect the effect of g_p and backtracking can be made directly to the smallest index i in step 3.2.2) for which $h_i \neq I$. This potentially speeds up the execution, but might miss some pruning that could be obtained from step 2.

In [74, 75] the algorithm is used before branching on variable x_{p+1}. In other words, at node a, assuming that more than one value is in D_{p+1}^a, the index $p + 1$ would appear in F_v^a for all $v \in D_{p+1}^a$. The algorithm is then modified as follows:

(i)   In step 2), if $g_{i-1}^{-1}(i) = p + 1$, then instead of stopping, the value w is removed from D_{p+1}^a and $p + 1$ is removed from F_w^a. This is an application of the domain reduction (iv) of Section 9.1.1.

(ii) If the stopping criterion is met in step 2), but $p + 1$ was the index $g_{i-1}^{-1}(j)$ in an earlier step 3.2.1) used to build the current permutation g_{i-1}, then we can backtrack to that step 3.2.1), remove the value v from D_{p+1}^a, remove $p + 1$ from F_v^a and continue. This is also an application of (iv) of Section 9.1.1.

(iii) In step 3.2.1), if $g_{i-1}^{-1}(j) = p + 1$, then $p + 1$ is removed from all the sets F_w^a.

Using these modifications, the values in D_{p+1}^a at termination of the algorithm are the values that need to be used to create sons by splitting. Of course, if that domain is empty then a is pruned.

Another modification used in [74, 75] is to ignore variables that have been set to 0 by branching. Suppose that variables that have been set to positive values by branching at a are $(i_1, \ldots, i_t)$ and assume that the base β of T starts with these elements in that order. The above algorithm is then run with $i := i_1$ and instead of incrementing i by one, it skips from i_j to i_{j+1}. The justification for this modification is that if $x_{\bar{i}} = 0$ is set for some $\bar{i}$, and a permutation g is obtained in Step 2) of the algorithm, then the minimum value in the domain of $j = g^{-1}(\bar{i})$ can not be smaller than 0. Using the original algorithm, we could exclude from D_j^a all values larger than 0 and continue. This reduction is possibly missed by the modified algorithm, but it remains correct.

The importance of this modification is on display when solving a problem where the variables taking a positive value in any optimal solution is a small subset of the variables. (This is an usual feature of combinatorial problems that are typically solved by ILP.) The depth of the backtracking of the modified algorithm is then much smaller, with a significant impact on the running time.

Unfortunately, there is no direct comparison available between the implementations of [11, 12] and [74, 75], as the only published results for the former are for the well-known Queens problem: On an $n \times n$ chessboard, place n queens that do not attack each other. It would be foolish to draw conclusions on implementations of algorithms designed to handle groups with large orders based only on results on an application where the group order is 8. In Section 13, a comparison (hardly an ideal one but a little bit more meaningful) between implementations of `SBDD` and `IsoP` is given.

11 Enumerating all non isomorphic solutions

One important application of the pruning algorithms of Section 9 is their use for enumerating all non isomorphic solutions to a symmetric ILP. Pruning is the only technique that can reliably list only non isomorphic solutions, as, in general, symmetry breaking inequalities are either too expensive to use or do not remove all symmetry from the problem. There is interest emanating from the Combinatorics and Statistics communities (among others) for enumerating all non isomorphic graphs or matrices with certain properties. For example, covering designs [51, 82] or balanced incomplete block designs [29] have a long history. For small values of the parameters, complete or implicit enumeration algorithms were used to generate solutions. Enumerating all non isomorphic solution is interesting, since these objects are used to build other mathematical objects or used to test conjectures. A few of the enumeration results obtained by pruning algorithms are available in [15, 26, 68, 73, 75, 78, 94, 96, 104]. Note that the solutions obtained by a pruning algorithm are non isomorphic solutions with respect to the symmetry group G used by the algorithm. It might happen that G is only a subgroup of the symmetry group of the feasible set of the problem and thus that isomorphic solutions remain in the output.

Enumerating all non isomorphic solutions also provides a powerful debugging tool. Replicating enumeration results on problems having thousands of non isomorphic solutions is a much better indication that an algorithm is correct than when it is run to find an optimal solution. Indeed, in symmetric problems, a faulty algorithm can still find optimal solutions with surprising ease, whereas it is more likely that at least one of the non isomorphic optimal solutions is missed or that two isomorphic solutions appear in the output when enumerating all non isomorphic solutions.

Some empirical results for enumeration of all non isomorphic solution to a combinatorial problem are given in Section 13.

12 Furthering the reach of isomorphism pruning

Isomorphism pruning and other techniques for handling symmetries in ILP help push the boundaries of the problems that can be solved routinely. However, many symmetric ILPs have a parametrized formulation yielding a never ending stream of challenging problems. When trying to solve an ILP, it is possible to use some kind of partial isomorphism pruning in order to speed up the process. For example, skipping the isomorphism pruning at deep level in the tree is usually beneficial as illustrated on a few examples in [96]. Another idea is to use the orbits of the variables not yet fixed to some values for branching variable selection [88].

Another recent development is the idea of branching on constraints using orbits. Given a branching corresponding to a valid disjunction $a \cdot x \leq b$ or $a \cdot x \geq b + 1$ one son is created with the first constraint, while the second one is generated using $g(a) \cdot x \geq b + 1$ for all $g \in G$. Further, when the disjunction is chosen carefully, it is sometimes possible to enumerate all non isomorphic solution to $a \cdot x = t$ for some values of t and use these solutions for solving the original problem. This technique was pioneered by Östergård and Blass [85] in the combinatorics community and used for improving the lower bound for the Football Pool problem using ILP [67, 86]. In [89], Ostrowski et al. formalize and generalize the technique and apply it successfully to solve Steiner Triple Systems and Covering Design problems that were previously out of reach.

A special situation of practical interest occurs when the symmetry group G is the product of a nontrivial group with a symmetric group. This is the typical situation for partitioning problems of the form considered in Section 7.2 when symmetry between the elements is also present. This happens for example for graph coloring problems where the automorphism group of the graph is non trivial. As symmetric groups are the most challenging ones to handle using the algorithms mentioned in Section 10 while they can be handled easily without complex representation, in [75] hybrid algorithms are used where the symmetric groups are handled directly. Improvement in reported running times are significant.

13 Choice of formulation

When dealing with a symmetric problem, the choice of the formulation might have a big impact on the performances of a solution technique. This is well known for ILPs, and it is even more acute when dealing with symmetries. While tightness of linear relaxation, number of variables, and constraint types can be used as guides for choosing an ILP formulation, things are obscured when dealing with a symmetric ILP, as different variable choices might yield very different symmetry groups. One would expect that fewer variables and a symmetry group with smaller order is better, but as described in Section 5 things are not so simple. In [110], several formulations for constructing a particular class of combinatorial designs are compared.

Just to have a concrete example illustrating how alternative formulations can make a big difference, consider the problem of constructing Balanced Incomplete Block Designs (BIBD). A BIBD is a binary matrix with given dimensions $b \times v$ with exactly k ones per row and constant dot product of value λ between any two of its columns (see [29] for background material). The results of [46, 96] are obtained using a nonlinear formulation with $b \cdot v$ binary variables, one for each entry in the matrix. This yields a formulation where the symmetry group has order $v! \cdot b!$. It is not convenient to use these variables to get an ILP formulation. An alternative linear formulation can be built from a list $\{R_1, \ldots, R_q\}$ of all possible binary rows of length v and having exactly k ones. Define one integer variable x_i taking values between 0 and λ, indicating how many times row R_i appears in the solution, for $i = 1, \ldots, q$. Constraints are that, for each pair j_1, j_2 of columns, the sum of all variables corresponding to rows having 1s in columns j_1 and j_2 should be exactly λ. This yields a formulation with $\frac{v!}{(v-k)! \cdot k!}$ variables, $\frac{v(v-1)}{2}$ constraints and a symmetry group of order $v!$.

For many instances, the latter formulation seems much simpler to solve than the former. The following table compares results for three codes for solving BIBD problems. The comparison is far from ideal, since the three codes are run on different machines and the formulations are not identical. Both GAP-ECLIPSE [46] and SBDD+STAB [96] use the first formulation, while IsoP [75] uses the second one. It is clear that SBDD+STAB outperforms GAP-ECLIPSE, but this should not be too surprising as SBDD+STAB uses the particular (extremely simple) structure of the symmetry group, while GAP-ECLIPSE is a code that can be used with any group. Nevertheless, the difference in running time on the third problem is quite stunning, possibly due to different branching variable choices. On the other hand, on hard problems, IsoP seems significantly faster than SBDD+STAB, even after discounting for the difference in machine speeds. It is likely that part of the difference can be attributed to the formulation IsoP uses.

v k λ	# solutions	GAP-ECLIPSE	SBDD+STAB	IsoP
11 5 2	1	19	0	1
13 4 1	1	42	0	10
13 3 1	2	59,344	0	1
7 3 8	5,413		21,302	115
9 3 3	22,521		34,077	1,147
15 3 1	80		2,522	161
13 4 2	2,461		18,496	5,142
11 5 4	4,393		83,307	3,384

Table 2 Comparison of formulations for BIBD problems. Number of non isomorphic solutions and times (rounded down) in seconds for enumerating them. Empty entries indicate that the corresponding result is not available. Times for GAP-ECLIPSE are from [46], obtained on a 2.6GHz Pentium IV processor. Times for SBDD+STAB are from [96], obtained a 1.4GHz Pentium Mobile laptop running Windows XP. Times for IsoP are obtained on the machine mentioned in Section 1.

14 Exploiting additional symmetries

So far, the only symmetries considered were the symmetries of the original ILP (1). However, while solving the ILP by branch-and-bound, it is sometimes the case that, at node a of the enumeration tree, additional symmetries exist in ILP^a, as seen in Example 2 in Section 5. These symmetries can be used when solving ILP^a, provided that they can be identified. This last point is a big hurdle to clear. As pointed out in Section 3, automatic symmetry detection is a difficult problem in itself. Few papers attempt to use an automatic symmetry detection algorithm at nodes of the tree, and when they do, results are unconvincing [88], as the time spend in searching for additional symmetries is not compensated by a commensurate reduction in the size of the enumeration tree. Successful exploitation of additional symmetries are limited to cases where problem specific rules for generating the symmetries are designed from the start [47]. Development of theory and algorithms for exploiting additional symmetries can be found in [48].

References

1. Aloul F.A., Ramani A., Markov I.L., Sakallah K.A.: Solving Difficult Instances of Boolean Satisfiability in the Presence of Symmetry, IEEE Transactions on CAD **22**, 1117-1137 (2003).
2. Anstreicher K.M.: Recent Advances in the Solution of Quadratic Assignment Problems, Mathematical Programming Ser. B **97**, 27–42 (2003).
3. Applegate D.L., Bixby R.E., Chvátal V., Cook W.J.: The Traveling Salesman Problem, A Computational Study, Princeton (2006).
4. von Arnim A., Schrader R., Wang Y.: The Permutahedron of N-sparse Posets, Mathematical Programming **75**, 1–18 (1996).
5. Babai L., Luks E.M., Seress Á.: Fast Management of Permutation Groups I, SIAM Journal on Computing **26**, 1310–1342 (1997).
6. Balas E.: A Linear Characterization of permutation Vectors, Management Science Research Report 364, Carnegie Mellon University, Pittsburgh, PA (1975).
7. Barnhart C., Johnson E.L., Nemhauser G.L., Savelsbergh M.W.P., Vance P.H: Branch-and-Price: Column Generation for Solving Huge Integer Programs, Operations Research **46**, 316–329 (1998).
8. Bazaraa M.S., Kirca O.: A Branch-and-Bound Based Heuristic for Solving the Quadratic Assignment Problem, Naval Research Logistics Quarterly **30**, 287–304 (1983).
9. Bertolo R., Östergård P., Weakley W.D.: An Updated Table of Binary/Ternary Mixed Covering Codes, Journal of Combinatorial Designs **12**, 157–176 (2004).
10. Bosma W., Cannon J., Playoust C.: The Magma Algebra System. I. The User Language, Journal of Symbolic Computations, **24**, 235-265 (1997).
11. Brown C.A., Finkelstein L., Purdom P.W.: Backtrack Searching in the Presence of Symmetry, Lecture Notes in Computer Science **357**, Springer, 99–110 (1989).
12. Brown C.A., Finkelstein L., Purdom P.W.: Backtrack Searching in the Presence of Symmetry, Nordic Journal of Computing **3**, 203–219 (1996).
13. Buchheim C., Jünger M.: Detecting Symmetries by Branch&Cut, Mathematical Programming Ser. B, **98**, 369–384 (2003).
14. Buchheim C., Jünger M.: Linear Optimization Over Permutation Groups, Discrete Optimization **2**, 308–319 (2005).
15. Bulutoglu D.A., Margot F.: Classification of Orthogonal Arrays by Integer Programming, Journal of Statistical Planning and Inference **138**, 654-666 (2008).

16. Butler G.: Computing in Permutation and Matrix Groups II: Backtrack Algorithm, Mathematics of Computation **39**, 671–680 (1982).
17. Butler G.: Fundamental Algorithms for Permutation Groups, Lecture Notes in Computer Science **559**, Springer (1991).
18. Butler G., Cannon J.J.: Computing in Permutation and Matrix Groups I: Normal Closure, Commutator Subgroups, Series, Mathematics of Computation **39**, 663–670 (1982).
19. Butler G., Lam W.H.: A General Backtrack Algorithm for the Isomorphism Problem of Combinatorial Objects, Journal of Symbolic Computation **1**, 363–381 (1985).
20. Cameron P.J.: Permutation Groups, London Mathematical Society, Student Text **45**, Cambridge University Press, (1999).
21. Campêlo M., Corrêa R.C.: A Lagrangian Decomposition for the Maximum Stable Set Problem, Working Paper (2008), Universidade Federal do Ceará, Brazil.
22. Campêlo M, Campos V.A., Corrêa R.C.: Um Algoritmo de Planos-de-Corte para o Número Cromático Fracionário de um Grafo. To appear in Pesquisa Operacional (2008).
23. Campêlo M., Corrêa R., Frota Y.: Cliques, Holes and the Vertex Coloring Polytope, Information Processing Letters **89**, 159-164 (2004).
24. Campêlo M., Campos V., Corrêa R.: On the Asymmetric Representatives Formulation for the Vertex Coloring Problem, Electronic Notes in Discrete Mathematics **19**, 337-343 (2005).
25. Campêlo M., Campos V., Corrêa R.: On the Asymmetric Representatives Formulation for the Vertex Coloring Problem, Discrete Applied Mathematics **156**, 1097-1111 (2008).
26. Cameron R.D., Colbourn C.J., Read R.C., Wormald N.C.: Cataloguing the Graphs on 10 Vertices, Journal of Graph Theory **9**, 551–562 (1985).
27. Christof T., Reinelt G.: Decomposition and Parallelization Techniques for Enumerating the Facets of Combinatorial Polytopes, International Journal on Computational Geometry and Applications **11**, 423–437 (2001).
28. Cohen G. Honkala I., Litsyn S., Lobstein A.: Covering Codes, North Holland (1997).
29. Colbourn C.J., Dinitz J.H., (eds.): The CRC Handbook of Combinatorial Designs, CRC Press (2007).
30. Cooperman G., Finkelstein L., Sarawagi N.: A Random Base Change Algorithm for Permutation Groups, Proceedings of the International Symposium on Symbolic and Algebraic Computations – ISSAC 90, 161-168, ACM Press (1990).
31. Crawford J., Ginsberg M.L., Luks E., Roy A.: Symmetry-Breaking Predicates for Search Problems, KR'96: Principles of Knowledge Representation and Reasoning, Aiello L.C., Doyle J., Shapiro S. (eds.), 148–159 (1996).
32. Darga P., Liffiton M.H., Sakallah K.A., Markov I.L.: Exploiting Structure in Symmetry Generation for CNF, Proceedings of the 41st Design Automation Conference, San Diego 2004, 530-534 (2004).
33. Degraeve Z., Gochet W., Jans R.: Alternative Formulations for a Layout Problem in the Fashion Industry, European Journal of Operational Research **143**, 80-93 (2002).
34. Desrochers M., Soumis F: A Column Generation Approach to the Urban Transit Crew Scheduling Problem, Transportation Science **23**, 1-13 (1989).
35. Deza A., Fukuda K., Mizutani T., Vo C.: On the Face Lattice of the Metric Polytope, Discrete and Computational Geometry: Japanese Conference (Tokyo, 2002), Lecture Notes in Computer Science **2866**, Springer, 118–128 (2003).
36. Deza A., Fukuda K., Pasechnik D., Sato M.: On the Skeleton of the Metric Polytope, Discrete and Computational Geometry: Japanese Conference (Tokyo, 2000), in Lecture Notes in Computer Science **2098**, Springer, 125–136 (2001).
37. Dumas Y., Desrochers M., Soumis F.: The pickup and delivery problem with time windows, European Journal of Operations Research **54**, 7-22 (1991).
38. Elf M., Gutwenger C., Jünger M., Rinaldi G.: Branch-and-Cut Algorithms for Combinatorial Optimization and their Implementation in ABACUS, in [59], 155–222 (2001).
39. Fahle T., Shamberger S., Sellmann M.: Symmetry Breaking, Proc. 7th International Conference on Principles and Practice of Constraint Programming – CP 2001, Lecture Notes in Computer Science **2239**, Springer, 93–107 (2001).

40. Flener P., Frisch A., Hnich B., Kiziltan Z., Miguel I., Pearson J., Walsh T.: Symmetry in Matrix Models, working paper APES-30-2001, (2001).
41. Flener P., Frisch A., Hnich B., Kiziltan Z., Miguel I., Pearson J., Walsh T.: Breaking Row and Column Symmetries in Matrix Models, Proc. 8th International Conference on Principles and Practice of Constraint Programming – CP 2002, Lecture Notes in Computer Science **2470**, Springer, 462–476 (2002).
42. Flener P., Pearson J., Sellmann M., Van Hentenryck P., Ågren M.: Dynamic Structural Symmetry Breaking for Constraint Satisfaction Problems. To appear in Constraints (2008).
43. Focacci F., Milano M.: Global Cut Framework for Removing Symmetries, Proc. 7th International Conference on Principles and Practice of Constraint Programming – CP 2001, Lecture Notes in Computer Science **2239**, Springer, 77–92 (2001).
44. Friedman E.J.: Fundamental Domains for Integer Programs with Symmetries, Proceedings of COCOA 2007, Lecture Notes in Computer Science **4616**, 146–153 (2007).
45. Gent I. P., Harvey W., Kelsey T.: Groups and Constraints: Symmetry Breaking During Search, Proc. 8th International Conference on Principles and Practice of Constraint Programming – CP 2002, Lecture Notes in Computer Science **2470**, Springer, 415–430 (2002).
46. Gent I. P., Harvey W., Kelsey T., Linton S.: Generic SBDD Using Computational Group Theory, Proc. 9th International Conference on Principles and Practice of Constraint Programming – CP 2003, Lecture Notes in Computer Science **2833**, Springer, 333–347 (2003).
47. Gent I. P., Kelsey T., Linton S., McDonald I., Miguel I., Smith B.: Conditional Symmetry Breaking, Proc. 11th International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science **3709**, 333–347 (2005).
48. Gent I.P., Kelsey T., Linton S.T., Pearson J., Roney-Dougal C.M.: Groupoids and Conditional Symmetry, Proc. 13th International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science **4741**, 823–830 (2007).
49. Gent I.P., Petrie K.E., Puget J.-F.: Symmetry in Constraint Programming, in Handbook of Constraint Programming, Rossi F., van Beek P., Walsh T., (eds.), Elsevier, 329–376 (2006).
50. Gent I.P, Smith B.M.: Symmetry Breaking in Constraint Programming, Proceedings of ECAI-2002, IOS Press, 599-603 (2002).
51. Gordon D.M., Stinson D.R.: Coverings, in The CRC Handbook of Combinatorial Designs, Colbourn C.J., Dinitz J.H., (eds.), CRC Press, 365–372 (2007).
52. Grove L.C., Benson C.T.: Finite Reflection Groups, Springer (1985).
53. Hämäläinen H., Honkala I., Litsyn S., Östergård P.: Football Pools–A Game for Mathematicians, American Mathematical Monthly **102**, 579–588 (1995).
54. Hoffman C.M.: Group-Theoretic Algorithms and Graph Isomorphism, Lecture Notes in Computer Science **136**, Springer (1982).
55. Holt D.F., Eick B., O'Brien E.A.: Handbook of Computational Group Theory, Chapman & Hall/CRC (2004).
56. Jans R.: Solving Lotsizing Problems on Parallel Identical Machines Using Symmetry Breaking Constraints. To appear in INFORMS Journal on Computing (2008).
57. Jans R., Degraeve Z.: A Note on a Symmetrical Set Covering Problem: The Lottery Problem, European Journal of Operational Research **186**, 104–110 (2008).
58. Jerrum M.: A Compact Representation for Permutation Groups, Journal of Algorithms **7**, 60–78 (1986).
59. Jünger M., Naddef D., (eds.): Computational Combinatorial Optimization, Lecture Notes in Computer Science **2241**, Springer (2001).
60. Kaibel V., Peinhardt M., Pfetsch M.E.: Orbitopal Fixing, Proceedings of the 12th International Integer Programming and Combinatorial Optimization Conference, M. Fischetti, D.P. Williamson, (eds.), Lecture Notes in Computer Science **4513**, Springer, 74–88 (2007).
61. Kaibel V., Pfetsch M.E.: Packing and Partitioning Orbitopes, Mathematical Programming **114**, 1–36 (2008).
62. Kreher D.L., Stinson D.R.: Combinatorial Algorithms, Generation, Enumeration, and Search, CRC Press (1999).
63. Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G., Shmoys D.B.: The traveling salesman problem, Wiley (1985).

64. 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**, 941–974 (1980).
65. Leon J.S.: Computing Automorphism Groups of Combinatorial Objects, in Computational Group Theory, Atkinson M.D. (ed.), Academic Press, 321–335 (1984).
66. Liberti L.: Automatic Generation of Symmetry-Breaking Constraints, Proceedings of CO-COA 2008, Lecture Notes in Computer Science **5165**, 328–338 (2008).
67. Linderoth J., Margot F., Thain G.: Improving Bounds on the Football Pool Problem via Symmetry Reduction and High-Throughput Computing, working paper (2007).
68. Luetolf C., Margot F.: A Catalog of Minimally Nonideal Matrices, Mathematical Methods of Operations Research **47**, 221–241 (1998).
69. Luks E.: Permutation Groups and Polynomial-Time Computation, in DIMACS Series in Discrete Mathematics and Theoretical Computer Science **11**, Groups and Computation, L. Finkelstein, W. Kantor (eds.), 139–175 (1993).
70. Marenco J.L., Rey P.A.: The Football Pool Polytope, Electronic Notes in Discrete Mathematics **30**, 75–80 (2008).
71. http://wpweb2.tepper.cmu.edu/fmargot/index.html
72. Margot F.: Pruning by Isomorphism in Branch-and-Cut, Mathematical Programming **94**, 71–90 (2002).
73. Margot F.: Small Covering Designs by Branch-and-Cut, Mathematical Programming Ser. B, **94**, 207-220 (2003).
74. Margot F.: Exploiting orbits in Symmetric ILP, Mathematical Programming Ser. B **98**, 3–21 (2003).
75. Margot F.: Symmetric ILP: Coloring and Small Integers, Discrete Optimization **4**, 40–62 (2007).
76. Mautor T., Roucairol C.: A New Exact Algorithm for the Solution of Quadratic Assignment Problems, Discrete Applied Mathematics **55**, 281-293 (1994).
77. McKay B.D.: Nauty User's Guide (Version 2.2), Computer Science Department, Australian National University, Canberra.
78. McKay D.: Isomorph-Free Exhaustive Generation, Journal of Algorithms **26**, 306–324 (1998).
79. Mehrotra A., Trick M. A.: A Column Generation Approach for Graph Coloring, INFORMS Journal on Computing **8**, 344–354 (1996).
80. Mehrotra A., Trick M. A.: Cliques and Clustering: a Combinatorial Approach, Operations Research Letters **22**, 1–12 (1998).
81. Méndez-Dìaz I., Zabala P.: A Branch-and-Cut Algorithm for Graph Coloring, Discrete Applied Mathematics **154**, 826–847 (2006).
82. 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.
83. Nemhauser G.L., Park S.: A Polyhedral Approach to Edge Colouring, Operations Research Letters **10**, 315-322 (1991).
84. Nemhauser G.L., Wolsey L.A.: Integer and Combinatorial Optimization, Wiley (1988).
85. Östergård P., Blass W.: On the Size of Optimal Binary Codes of Length 9 and Covering Radius 1, IEEE Transactions on Information Theory **47**, 2556–2557 (2001).
86. Östergård P., Wassermann A.: A New Lower Bound for the Football Pool Problem for Six Matches, Journal of Combinatorial Theory Ser. A **99**, 175–179 (2002).
87. Ostrowski J.: Personal communication (2007).
88. Ostrowski J., Linderoth J., Rossi F., Smirglio S.: Orbital Branching, IPCO 2007: The Twelfth Conference on Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science **4513**, Springer, 104–118 (2007).
89. Ostrowski J., Linderoth J., Rossi F., Smirglio S.: Constraint Orbital Branching, IPCO 2008: The Thirteenth Conference on Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science **5035**, Springer, 225–239 (2008).
90. Padberg M.W., Rinaldi G.: A Branch-and-Cut Algorithm for the Resolution of Large Scale Symmetric Travelling Salesman Problems, SIAM Review **33**, 60–100 (1991).

91. Petrie K.E, Smith B.M.: Comparison of Symmetry Breaking Methods in Constraint Programming, In Proceedings of SymCon05, (2005).
92. Plastria F.: Formulating logical Implications in Combinatorial Optimisation, European Journal of Operational Research **140**, 338–353 (2002).
93. Puget J.F.: On the Satisfiability of Symmetrical Constrained Satisfaction Problems, Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems, Lecture Notes in Artificial Intelligence **689**, 350-361, (1993).
94. Puget J.-F.: Symmetry Breaking Using Stabilizers, Proc. 9th International Conference on Principles and Practice of Constraint Programming – CP 2003, Lecture Notes in Computer Science **2833**, Springer, 585–599 (2003).
95. Puget J.-F.: Automatic Detection of Variable and Value Symmetries, Proc. 11th International Conference on Principles and Practice of Constraint Programming – CP 2005, Lecture Notes in Computer Science **3709**, Springer, 475–489 (2005).
96. Puget J.-F.: Symmetry Breaking Revisited, Constraints **10**, 23–46 (2005).
97. Puget J.-F.: A Comparison of SBDS and Dynamic Lex Constraints, In Proceeding of SymCon06, 56–60 (2006).
98. Ramani A., Aloul F.A., Markov I.L., Sakallah K.A.: Breaking Instance-Independent Symmetries in Exact Graph Coloring, Journal of Artificial Intelligence Research **26**, 191–224 (2006).
99. Ramani A., Markov I.L.: Automatically Exploiting Symmetries in Constraint Programming, CSCLP 2004, B. Faltings et al. (eds.), Lecture Notes in Artificial Intelligence **3419**, Springer, 98–112 (2005).
100. Read R.C.: Every One a Winner or How to Avoid Isomorphism Search When Cataloguing Combinatorial Configurations, Annals of Discrete Mathematics **2**, 107-120 (1978).
101. Rey P.A.: Eliminating Redundant Solutions of Some Symmetric Combinatorial Integer Programs, Electronic Notes in Discrete Mathematics **18**, 201-206 (2004).
102. Rotman J.J.: An Introduction to the Theory of Groups, 4th ed., Springer (1994).
103. Salvagnin D.: A Dominance Procedure for Integer Programming, Master's thesis, University of Padova (2005).
104. Seah E., Stinson D.R.: An Enumeration of Non-isomorphic One-factorizations and Howell Designs for the Graph K_{10} minus a One-factor, Ars Combinatorica **21**, 145–161 (1986).
105. Seress Á.: Nearly Linear Time Algorithms for Permutation Groups: An Interplay Between Theory and Practice, Acta Applicandae Mathematicae **52**, 183–207 (1998).
106. Seress Á.: Permutation Group Algorithms, Cambridge Tracts in Mathematics **152**, Cambridge University Press, (2003).
107. Sherali H.D., Fraticelli B.M.P., Meller R.D.: Enhanced Model Formulations for Optimal Facility Layout, Operations Research **51**, 629–644 (2003).
108. Sherali H.D., Smith J.C.: Improving Discrete Model Representations via Symmetry Considerations, Management Science **47**, 1396–1407 (2001).
109. Sherali H.D., Smith J.C., Lee Y.: Enhanced Model Representations for an Intra-Ring Synchronous Optical Network Design Problem Allowing Demand Splitting, INFORMS Journal on Computing **12**, 284–298 (2000).
110. Smith B.: Reducing Symmetry in a Combinatorial Design Problem, Proc. 8th International Conference on Principles and Practice of Constraint Programming – CP 2002, Lecture Notes in Computer Science **2470**, Springer, 207–213 (2002).
111. Smith B.M., Brailsford S.C., Hubbard P.M., Williams H.P.: The Progressive Party Problem: Integer Linear Programming and Constraint Programming Compared, Constraints **1**, 119–138 (1996).
112. Stanton R.G., Bates J.A.: A Computer Search for B-coverings, in Combinatorial Mathematics VII, Lecture Notes in Computer Science **829**, Robinson R.W., Southern G.W., Wallis W.D. (eds.), Springer, 37–50 (1980).
113. The GAP Group: GAP – Groups, Algorithms, and Programming, Version 4.4.10 (2007). (http://www.gap-system.org).
114. Vance P.H.: Branch-and-price Algorithms for the One-dimensional Cutting Stock Problem, Computational Optimization and Applications **9**, 111-228 (1998).

115. Vance P.H., Barnhart C., Johnson E.L., Nemhauser G.L.: Solving Binary Cutting Stock Problems by Column Generation and Branch-and-bound, Computational Optimization and Applications **3**, 111-130 (1994).
116. Vance P.H., Barnhart C., Johnson E.L., Nemhauser G.L.: Airline Crew Scheduling: A New Formulation and Decomposition Algorithm, Operations Research **45**188-200 (1997).
117. Wolsey L.A.: Integer Programming, Wiley (1998).
118. Yannakakis M.: Expressing Combinatorial Optimization Problems by Linear Programs, Journal of Computer and System Sciences **43**, 441–466 (1991).