Orbital Conflict: Cutting Planes for Symmetric Integer Programs

Jeff Linderoth
Department of Industrial and Systems Engineering, University of Wisconsin-Madison linderoth@wisc.edu

José Núñez Ares
Department of Biosystems, KU Leuven jose.nunezares@kuleuven.be

James Ostrowski
Department of Industrial and Systems Engineering, University of Tennessee jostrows@utk.edu

Fabrizio Rossi, Stefano Smriglio
Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, fabrizio.rossi@univaq.it stefano.smriglio@univaq.it

Cutting planes have been an important factor in the impressive progress made by integer programming (IP) solvers in the past two decades. However, cutting planes have had little impact on improving performance for symmetric IPs. Rather, the main breakthroughs for solving symmetric IPs have been achieved by cleverly exploiting symmetry in the enumeration phase of branch and bound. In this work, we introduce a hierarchy of cutting planes that arise from a reinterpretation of symmetry-exploiting branching methods. There are too many inequalities in the hierarchy to be used efficiently in a direct manner. However, the lowest levels of this cutting-plane hierarchy can be implicitly exploited by enhancing the conflict graph of the integer programming instance and by generating inequalities such as clique cuts valid for the stable-set relaxation of the instance. We provide computational evidence that the resulting symmetry-powered clique cuts can improve state-of-the-art symmetry exploiting methods. The inequalities are then employed in a two-phase approach with high-throughput computations to solve heretofore unsolved symmetric integer programs arising from covering designs, establishing for the first time the covering radii of two binary-ternary codes.

Key words: Integer programming – symmetry – conflict graph
History:

1. Introduction

We focus on binary integer programs(BIP)s of the form

$$\max_{x\in\{0,1\}^n} \{c^T x \mid Ax \leq b\}, \tag{1}$$

where $A \in \mathbb{R}^{m\times n}$, $c \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$. We are interested in BIPs containing a great deal of *symmetry*—instances where permuting some of its variables yields an equivalent problem. The presence of symmetry significantly reduces the effectiveness of standard integer programming (IP) solution techniques, and problems of relatively limited size can be very challenging to solve. Several techniques have been investigated in order to overcome this drawback, ranging from problem

reformulation to the recognition of symmetric subproblems in the enumeration tree. These techniques are illustrated in the excellent survey of Margot (2009) to which we refer the reader for an exhaustive introduction. In addition, the work of Pfetsch and Rehn (2019) is a comprehensive computational evaluation of many of the state-of-the-art symmetry-handling techniques in IP. Most effective computational methods for dealing with symmetry involve exploiting the symmetry in the branching phase to reduce the combinatorial enumeration. On the other hand, as documented by Achterberg and Wunderling (2013) with respect to the state-of-the-art commercial solver integer programming solver CPLEX, cutting planes, not branching, have been a major driving force in improving performance of IP algorithm. Cutting planes have had little impact on improving performance for symmetric IPs. In this work, we offer some ideas on how to exploit symmetry to generate cutting planes. The cutting planes come from a re-interpretation of the standard 0-1 branching variable disjunction through the lens of the constraint orbital branching paradigm introduced by Ostrowski et al. (2008). The method yields far too many cutting planes to be used efficiently, but the cutting planes can be naturally arranged in a hierarchy, and inequalities coming from the lowest levels of the hierarchy can be implicitly employed. Key to our idea is the well-known notion of the conflict graph and the stable-set relaxation of BIP.

Let $\mathcal{F}$ denote the set of feasible solutions to (BIP). A *conflict graph* (CG) can be associated with (BIP) that describes the logical relations between the variables (Atamtürk et al. 2000). The conflict graph contains one vertex for each variable and one for its complement. Two vertices are adjacent in the conflict graph if the corresponding variables are *in conflict*–that is, cannot both take the value one in a feasible solution. By construction, any $x \in \mathcal{F}$ is the incidence vector of a stable set in the conflict graph. Therefore, $\mathcal{F}$ and $\text{conv}(\mathcal{F})$ are both contained in the stable set polytope $\text{STAB}(CG)$ associated with the the conflict graph CG, and valid inequalities for $\text{STAB}(CG)$ are also valid for $\mathcal{F}$. The stable set polytope has been intensively studied from both theoretical (Grötschel et al. 1988) and computational perspectives (Rebennack et al. 2011, Correa et al. 2018, Giandomenico et al. 2013). The *clique inequalities* of Padberg (1973) have the form $\sum_{i \in C} x_i \leq 1$, where $C \subset V$ induces a maximal clique (i.e. set of pairwise adjacent vertices) in the graph G. Clique inequalities are both strong, being facet-inducing for $\text{STAB}(CG)$, and numerically-safe to employ in an IP solver. Despite the fact that the associated separation problem for clique inequalities is strongly NP-hard (Grötschel et al. 1988), computational evidence has established that fast separation heuristics perform well in practice (Marzi et al. 2019). State-of-the-art commercial IP solvers all employ clique inequalities from the conflict graph.

However, as pointed out by both Atamtürk et al. (2000) and Achterberg et al. (2019), care must be taken in the construction and management of the conflict graph. Specifically, edges in the conflict graph are often added during a probing phase, where the logical implications of fixing a variable

to zero or one are sequentially considered. This can be computationally demanding in practice. In this paper we investigate a mechanism called *orbital conflict* to populate conflict graphs based on the symmetries present in the problem. The major contributions of the work are the following:

- a hierarchy of cutting planes which spring out of a re-interpretation of symmetry-exploiting branching methods;
- an implicit description of the lowest levels of this hierarchy through new edges of the CG;
- a computational assessment of the implication information implied by branching through symmetry-powered clique cuts;
- a computational evidence that these can improve state-of-the-art symmetry-exploiting methods;
- a two-phase solution methodology that combines our symmetry-exploiting methods with parallel processing to solve heretofore unsolved instances of symmetric BIPs.

In Section 2 we review common notions in symmetry-enhanced branching methods and recall the concept of constraint orbital branching. In Section 3 we apply constraint orbital branching to the standard 0-1 variable branching disjunction to derive a family of symmetry-induced cutting planes for BIP. In Section 4 we describe how we employ these cutting planes implicitly through their addition to the conflict graph of the BIP instance, and we provide an example to demonstrate their potential. Section 5 describes our computational setting, and Section 6 consists of a set of experiments aimed at demonstrating the impact of employing orbital conflict. By using orbital conflict, in conjunction with high-throughput computing, we are able to solve for the first time three symmetric BIP arising from covering designs.

Notation: We define $[n] = \{1, 2, \ldots, n\}$. For any node a of the branch-and-bound tree we let F_1^a and F_0^a denote the indices of variables fixed to one and zero, respectively. We represent the conflict graph for (1) at node a as $G(V, E_a)$. Given a set $T \subseteq [n]$ and an element $i \in [n]$, we often abuse notation and write $T \cup i$ rather than the correct $T \cup \{i\}$. Given $x \in \mathbb{R}^n$ and $T \subseteq [n]$, we often use the shorthand notation $x(T) \stackrel{\text{def}}{=} \sum_{i \in T} x_i$.

2. Symmetry-exploiting branching

Describing symmetry in IP requires some notions from group theory. We briefly introduce these notions here in order to make the presentation self-contained, but refer the reader to standard references such as (Rotman 1994) for an exhaustive treatment. Let us denote by Π^n the set of all permutations of $[n] = \{1, \ldots, n\}$; that is, the symmetric group of $[n]$. Permutations $\pi \in \Pi^n$ are represented by $n-$vectors, where $\pi(i)$ represents the image of i under π. The notation is extended to the case of permutations acting on sets $S \subseteq [n]$; that is, $\pi(S) = \{\pi(i), i \in S\} \subseteq [n]$. A permutation $\pi \in \Pi^n$ is said to be a *symmetry* of the IP instance if it maps any feasible solution into another

feasible solution with the same value. The set of symmetries of an instance forms the *symmetry group* G:

$$\mathcal{G} \stackrel{\text{def}}{=} \{\pi \in \Pi^n \mid c^T x = c^T \pi(x),\ \pi(x) \in \mathcal{F} \quad \forall x \in \mathcal{F}\}.$$

For a point $z \in \mathbb{R}^n$, the *orbit* of z under $\mathcal{G}$ is the set of all points to which z can be sent by permutations in $\mathcal{G}$:

$$\text{orb}(\mathcal{G}, z) \stackrel{\text{def}}{=} \{\pi(z) \mid \pi \in \mathcal{G}\}.$$

Finally, the *stabilizer* of a set S in $\mathcal{G}$ is the set of permutations in $\mathcal{G}$ that send S to itself:

$$\text{stab}(S, \mathcal{G}) = \{\pi \in \mathcal{G} \mid \pi(S) = S\}.$$

Note that $\text{stab}(S, \mathcal{G})$ is a subgroup of $\mathcal{G}$.

Our work builds on and reinterprets the symmetry-exploiting branching methods known as *Orbital Branching* (OB) (Ostrowski et al. 2011a) and *Constraint Orbital Branching* (COB) (Ostrowski et al. 2008). Suppose that we plan to apply the standard 0-1 branching dichotomy

$$(x_i = 1) \vee (x_i = 0) \tag{2}$$

at subproblem a of the branch-and-bound tree. Let $O = \text{orb}(\text{stab}(F_1^a, \mathcal{G}), x_i) = \{h_1, h_2, \ldots, h_{|O|}\}$ be the orbit of the variable x_i in the stabilizer of F_1^a in $\mathcal{G}$. Orbital branching prescribes to apply a strengthened right branch where all the variables in the orbit of x_i can be fixed to zero. Specifically, the branching disjunction

$$(x_i = 1) \vee \left(\sum_{h \in O} x_h = 0 \right),$$

can be enforced, and at least one optimal solution is contained in one of the two created child subproblems.

Constraint orbital branching extends the rationale of orbital branching to general disjunctions. Given an integer vector $(\lambda, \lambda_0) \in \mathbb{Z}^{n+1}$ and a base disjunction of the form

$$(\lambda^\top x \leq \lambda_0) \vee (\lambda^\top x \geq \lambda_0 + 1),$$

constraint orbital branching exploits all symmetrically equivalent forms of $\lambda^\top x \leq \lambda_0$ so as to obtain the following binary branching disjunction:

$$\Big(\lambda^\top x \leq \lambda_0\Big) \vee \Big([\pi(\lambda)]^\top x \geq \lambda_0 + 1 \quad \forall \pi \in \mathcal{G}\Big), \tag{3}$$

which is equivalent to

$$\Big(\lambda^\top x \leq \lambda_0\Big) \vee \Big(\mu^\top x \geq \lambda_0 + 1 \quad \forall \mu \in \text{orb}(\mathcal{G}, \lambda)\Big).$$

Note that orbital branching is a special case of constraint orbital branching using the integer vector $(\lambda, \lambda_0) = (e_i, 0)$.

Constraint orbital branching can be very powerful, as demonstrated by its successful application to solve instances of the highly-symmetric *Steiner triple covering problems* (Ostrowski et al. 2011b). However, exploiting its potential in general purpose solvers requires problem-specific knowledge of good branching vectors (λ, λ_0) and clever mechanisms for using and managing the potentially huge number of symmetric constraints on the right-branching child node in the disjunction (3). We will introduce the concept of *orbital conflict* in Section 4 as a mechanism for implicitly managing the large collection of inequalities coming from one particular application of constraint orbital branching.

3. Cutting planes from branching disjunctions

In this section, we employ the COB branching disjunction (3) on an augmented form of the standard variable branching disjunction (2), and we suggest a computationally useful mechanism for categorizing the family of resulting symmetric branching inequalities. We assume that we are branching on variable x_i at subproblem a. Since the variables in F_1^a are fixed to one at node a, the standard $0-1$ branching disjunction (3) can be reinterpreted as being obtained from the following base disjunction:

$$\Big(x(F_1^a \cup i) \geq |F_1^a| + 1\Big) \vee \Big(x(F_1^a \cup i) \leq |F_1^a|\Big). \tag{4}$$

By applying COB (3) to the base disjunction (4) we obtain

$$\Big(x(F_1^a \cup i) \geq |F_1^a| + 1\Big) \vee \Big(x(\pi(F_1^a \cup i)) \leq |F_1^a| \quad \forall \pi \in \mathcal{G}\Big). \tag{5}$$

If $|\mathcal{G}|$ is large, then adding all of the symmetric inequalities to the child node on the right-branch of the disjunction (5) is not practical. Instead, we will focus on identifying a subset of permutations likely to produce useful inequalities. To that end, we make the following definition.

Definition 1. Let $\mathcal{G} \subseteq \Pi^n$ be a permutation group, $T \subseteq [n]$, and $i \notin T$. The permutation $\pi \in \mathcal{G}$ is a *level$-k$ permutation* for (T, i) if $|\pi(T \cup i) \cap T| = |T| - k$.

We define $\mathcal{L}_k(T, i) \subseteq \mathcal{G}$ to be the set of all level$-k$ permutations for (T, i) in $\mathcal{G}$ and note that $\mathcal{G}$ is partitioned into these sets:

$$\mathcal{G} = \bigcup_{k=0}^{|T|} \mathcal{L}_k(T, i).$$

Let $\pi \in \mathcal{L}_k(F_1^a, i)$ be a level$-k$ permutation for the set of variables fixed to one at node a and branching index i. By definition, $|\pi(F_1^a \cup i) \cap F_1^a| = |F_1^a| - k$, so

$$x(\pi(F_1^a \cup i)) = x(\pi(F_1^a \cup i) \cap F_1^a) + x(\pi(F_1^a \cup i) \setminus F_1^a) = |F_1^a| - k + x(\pi(F_1^a \cup i) \setminus F_1^a),$$

and the branching inequality

$$x(\pi(F_1^a \cup i)) \leq |F_1^a| \tag{6}$$

associated with π in the disjunction (5) is equivalent to

$$x(\pi(F_1^a \cup i) \setminus F_1^a) \leq k. \tag{7}$$

We refer to (7) as a *level−k branching inequality* for node a and branching index i. The branching disjunction (5) is equivalent to applying the level-k branching inequalities for all possible values of k, i.e:

$$(x_i \geq 1) \vee \Big(x(\pi(F_1^a \cup i)) \setminus F_1^a) \leq k \ \forall k = 0, 1, 2, \ldots, |F_1^a|, \ \forall \pi \in \mathcal{L}_k(F_1^a, i)\Big). \tag{8}$$

For $\pi \in \mathcal{L}_k(F_1^a, i)$, the set $\pi(F_1^a \cup i) \setminus F_1^a$ has cardinality $k+1$ and there are $k+1$ elements on the left-hand side of (7). Therefore, branching inequalities (6) will be strongest for permutations $\pi \in \mathcal{L}_k(F_1^a, i)$ with small values of k. To mitigate the impact of dealing with the potentially large number of inequalities in (5), we propose to employ level$-k$ branching inequalities for only small values of $k = 1, 2, \ldots, K < F_1^a$.

$$(x_i \geq 1) \vee \Big(x(\pi(F_1^a \cup i)) \setminus F_1^a) \leq k \ \forall k = 0, 1, 2, \ldots, K, \ \forall \pi \in \mathcal{L}_k(F_1^a, i)\Big). \tag{9}$$

Given, $T \subset [n]$ and $i \notin T$, characterizing exactly the permutations in $\mathcal{L}_k(T, i)$ appears to difficult. However, we can calculate a subset of these permutations with the following theorem.

Theorem 1. *Let $\mathcal{G} \subseteq \Pi^n$ be a permutation group, $T \subseteq [n]$, $i \notin T$, and $\mathcal{L}_k(T, i)$ be the set of all level−k permutations for (T, i) in $\mathcal{G}$. Then,*

$$\bigcup_{\{S \subseteq T : |T \setminus S| \leq k\}} \text{stab}(S, \mathcal{G}) \subset \bigcup_{j=1}^{k} \mathcal{L}_j(T, i). \tag{10}$$

Proof of Theorem 1 Let $S \subseteq T$ with $|T \setminus S| = k$ and π in $\text{stab}(S, \mathcal{G})$. We will shows that π is at most a level-k permutation for (T, i). First we note that

$$\pi(T \cup i) \cap T = (\pi(S) \cap T) \cup (\pi(T \setminus S) \cap T) \cup (\pi(i) \cap T).$$

Since π is a permutation, this is a union of three disjoint sets, and we have

$$|\pi(T \cup i) \cap T| = |(\pi(S) \cap T)| + |(\pi(T \setminus S) \cap T)| + |(\pi(i) \cap T)|. \tag{11}$$

As $\pi \in \text{stab}(S, \mathcal{G})$, we have that $\pi(S) = S$, so $|(\pi(S) \cap T)| = |S \cap T| = |T| - k$. Plugging this into (11) yields $|\pi(T \cup i) \cap T| \geq |T| - k$, so π is at most a level-k permutation for (T, i). Q.E.D.

For $k = 0$ and $T = F_1^a$, the permutations in the subgroup on the left-hand-side of (10) are precisely those used to define the orbits of the variables in orbital branching, so an immediate consequence of Theorem 1 is that the branching disjunction used by orbital branching is dominated by branching disjunction (9), even for $K = 0$.

Using Theorem 1, we can for a fixed k enumerate the appropriate subsets of T and perform the stabilizer computations in (10). Note that as one moves deeper in the tree, the number of stabilizer computations used to approximate $\mathcal{L}_k$ can increase considerably, since it is dependent on $\binom{|F_1^a|}{k}$. Thus, the size of $\mathcal{L}_k$ is expected to increase dramatically as k increases. In Section 6.1, we show a small experiment that documents the number of level$-k$ inequalities we obtain for a symmetric IP. In our work, we do not choose to directly employ the level-k branching inequalities, or even use the inequalities to separate a given fractional solution $\hat{x}$. Rather, in the next section, we show how to use level-1 branching inequalities to augment the conflict graph of the problem, and we use well-known techniques to separate fractional solutions from its stable-set relaxation.

4. Orbital Conflict

The level-1 inequalities (7) are of the form $x_i + x_j \leq 1$ for some $i \in [n], j \in [n]$. Thus, information from these inequalities need not be added directly to the formulation, but can be handled implicitly by adding the edges (i, j) to the (local) conflict graph of the problem at a node. Algorithm (1) describes the generation of level-1 inequalities at a node a of the enumeration tree. Note that the level-1 inequalities generated at ancestors of a remain valid for the entire sub-tree rooted at a.

Included in the output of Algorithm 1 is a symmetry-enhanced conflict graph $G(V, E_c)$ on the right branch of the branching disjunction. When node c is to be explored, the clique cut separation heuristic of Marzi et al. (2019) is run to search for clique inequalities violated by the LP relaxation of node c. In the following example we compare a branch-and-cut algorithm based on OB with one enhanced by OC.

Algorithm 1: Orbital Conflict (OC)

Input: Subproblem $a = (F_1^a, F_0^a, G(V, E_a))$, branching variable index i

Output: Two child subproblems b and c

begin

/* Initialize */

$F_0^c := F_0^a$

$E_c := E_a$

/* Zero-Fixing/Orbital Branching: */

for $\pi \in \text{stab}(F_1^a, \mathcal{G})$ **do**

$F_0^c = F_0^c \cup \{\pi(x_i)\}$

end

/* Populate Local Conflict Graph */

for $S \subset F_1^a$ *with* $|S| = |F_1^a| - 1$ **do**

for $\{u, v\} = F_1^a \cup \{i\} \setminus S$ *and* $\pi \in \text{stab}(S, \mathcal{G})$ **do**

$E_c = E_c \cup \{(\pi(u), \pi(v))\}$

end

end

/* Branching */

$b = (F_1^a \cup \{i\}, F_0^a, G(V, E_a))$

$c = (F_1^a, F_0^c, G(V, E_c))$

return: b, c

end

EXAMPLE 1. Consider the graph $G = (V, E)$ of Figure 1 and the associated BIP:

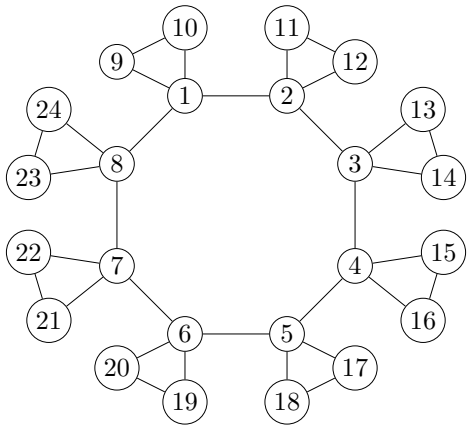


Figure 1 **Example graph**

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

which corresponds to computing the stability number of G. The optimal value is 8, while the root LP relaxation $\hat{x}$ has objective value 12, with $x_i = \frac{1}{2}, i \in V$. If a branch-and-bound algorithm is executed based on OB the search tree of Figure 2 is explored.

Let us focus on subproblem 2 at depth 1, with $F_1^2 = \{9\}$ and having the conflict graph shown in Figure 3. Suppose we choose to branch on variable x_{11}. The resulting COB disjunction (5) is

$$\Big(x_9 + x_{11} \ge 2\Big) \vee \Big(x_{\pi(9)} + x_{\pi(11)} \le 1 \quad \forall \pi \in \mathcal{G}\Big). \tag{12}$$

Enumerating all the constraints on the right-hand-side of (12) yields the following inequalities:

$$\begin{array}{ll}
\mathbf{x_9 + x_{11} \le 1} & \mathbf{x_9 + x_{12} \le 1} \\
\mathbf{x_9 + x_{23} \le 1} & \mathbf{x_9 + x_{24} \le 1} \\
x_{10} + x_{11} \le 1 & x_{10} + x_{12} \le 1 \\
x_{10} + x_{23} \le 1 & x_{10} + x_{24} \le 1 \\
x_{11} + x_{13} \le 1 & x_{11} + x_{14} \le 1 \\
x_{12} + x_{13} \le 1 & x_{12} + x_{14} \le 1 \\
x_{13} + x_{15} \le 1 & x_{13} + x_{16} \le 1 \\
x_{14} + x_{15} \le 1 & x_{14} + x_{16} \le 1 \\
x_{15} + x_{17} \le 1 & x_{15} + x_{18} \le 1 \\
x_{16} + x_{17} \le 1 & x_{16} + x_{18} \le 1 \\
x_{17} + x_{19} \le 1 & x_{17} + x_{20} \le 1 \\
x_{18} + x_{19} \le 1 & x_{18} + x_{20} \le 1 \\
x_{19} + x_{21} \le 1 & x_{19} + x_{22} \le 1 \\
x_{20} + x_{21} \le 1 & x_{20} + x_{22} \le 1 \\
x_{21} + x_{23} \le 1 & x_{21} + x_{24} \le 1 \\
x_{22} + x_{23} \le 1 & x_{22} + x_{24} \le 1.
\end{array}$$

As $|F_1^2| = 1$, these inequalities are either level-0 inequalities or level-1 inequalities. To distinguish the two, the level-0 constraints are written in bold font style and level-1 constraints are in black. Note that the level-0 constraints reduce to fixing x_{11}, x_{12}, x_{23}, and x_{24} to zero when x_9 is fixed to one. Also note that all of the level-0 constraints are generated by $\bigwedge_{\pi \in \mathrm{stab}(\{9\},\ \mathcal{G})} x_{\pi(9)} + x_{\pi(11)} \le 1$, demonstrating that $\mathrm{stab}(\{9\}, \mathcal{G})$ is a good approximate for $\mathcal{L}_0$ in this instance.

The graphs of Figure 4 represent the child subproblems that result from the above disjunction. Note the abundance of 4-cliques in subproblem 5 that are formed by adding the level-1 inequalities (the newly added constraints are represented by the dashed edges), whereas the parent subproblem only had 3-cliques. We can use these cliques to strengthen the level-1 inequalities generated by

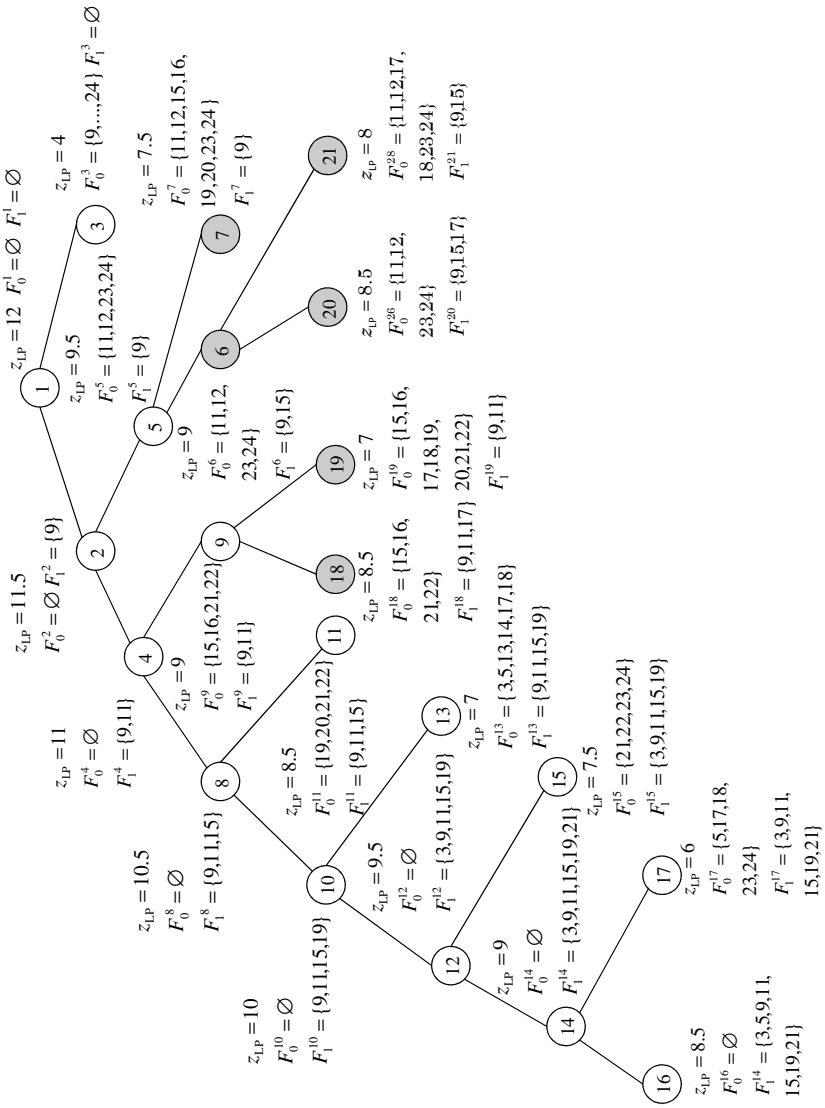


Figure 2 Search tree by orbital branching

the branching disjunction. In fact, all the generated inequalities will be dominated by the four 4-clique inequalities, reducing the size of the resulting formulation while also tightening it. Adding the clique inequalities from the 4-cliques to the LP formulation improves the bound from 9.5 to 8, allowing it to be pruned by bound. Without these clique inequalities, we would have to solve 4 more nodes (6,7, 20, and 21).

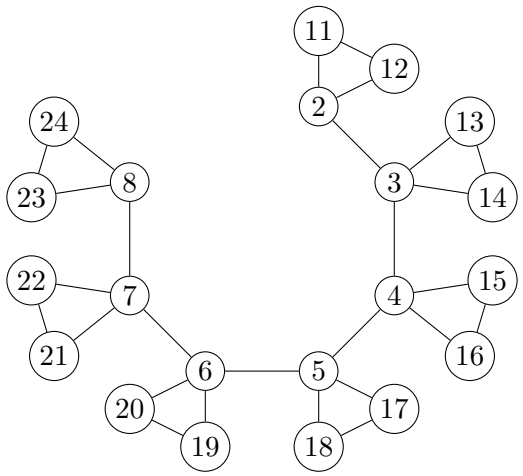
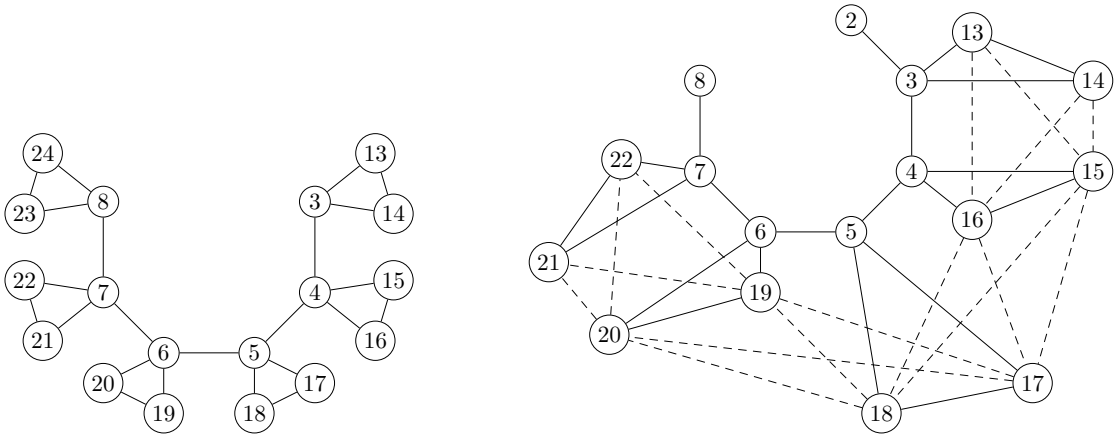


Figure 3 Graph of subproblem 2



Graph of subproblem 4 Graph of subproblem 5

Figure 4 Children of subproblem 2

Let us now apply the branching disjunction (5) at subproblem 4, with $F_1^4 = \{9, 11\}$ using the branching variable x_{15}. The resulting disjunction is

$$\Big(x_9 + x_{11} + x_{15} \geq 3\Big) \vee \Big(x_{\pi(9)} + x_{\pi(11)} + x_{\pi(15)} \leq 2 \quad \forall \pi \in \mathcal{G}\Big). \tag{13}$$

The right side of the disjunction (13) has 128 non-redundant inequalities that will not be enumerated in this example. However, we can enumerate all level-0 and level-1 inequalities. Note that $\text{stab}(\{9,\ 11\}), \mathcal{G}) \subset \mathcal{L}_0$ contains the symmetry that reflects the graph across its vertical axis as well as the permutations that permute the pairs of adjacent vertices on the graph's outer ring (excepting vertices 9-12). Thus, the permutations in $\text{stab}(\{9,\ 11\})$ generate the following level-0 constraints:

$$\begin{aligned} x_9 + x_{11} + x_{15} &\leq 2 & \qquad x_9 + x_{11} + x_{16} &\leq 2 \\ x_9 + x_{11} + x_{21} &\leq 2 & \qquad x_9 + x_{11} + x_{22} &\leq 2, \end{aligned}$$

which results in fixing x_{15}, x_{16}, x_{21}, and x_{22} to zero (which would have occurred via orbital branching).

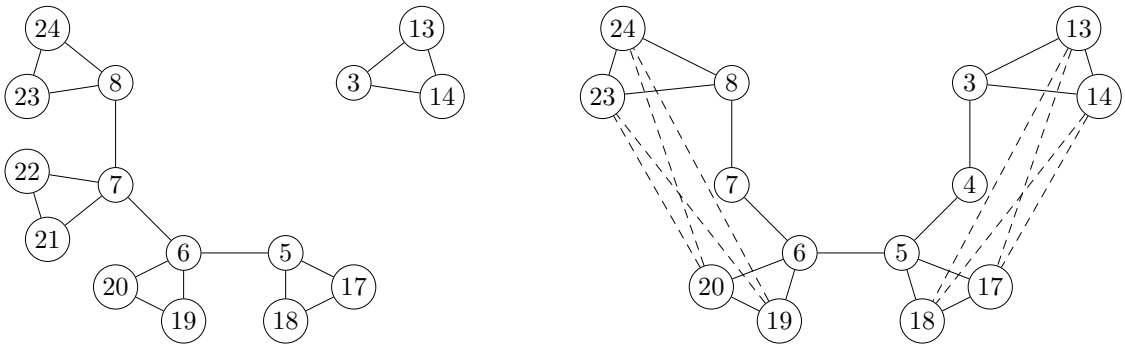


Figure 5 Children of subproblem 4

A subset of the level-1 inequalities are formed by looking at the subgroups $\text{stab}(\{9\},\mathcal{G})$ and $\text{stab}(\{11\},\mathcal{G})$. The resulting level-1 inequalities generated by $\text{stab}(\{9\},\mathcal{G}) \cup \text{stab}(\{11\},\mathcal{G}) \subset \mathcal{L}_0 \cup \mathcal{L}_1$ are the following:

$\pi \in \text{stab}(\{9\},\mathcal{G})$:	$\pi \in \text{stab}(\{11\},\mathcal{G})$:
$x_9 + x_{19} + x_{23} \leq 2$	$x_{11} + x_{13} + x_{17} \leq 2$
$x_9 + x_{19} + x_{24} \leq 2$	$x_{11} + x_{13} + x_{18} \leq 2$
$x_9 + x_{20} + x_{23} \leq 2$	$x_{11} + x_{14} + x_{17} \leq 2$
$x_9 + x_{20} + x_{24} \leq 2$	$x_{11} + x_{14} + x_{18} \leq 2.$

Note, of course, that because x_9 and x_{11} are fixed to one at node 9, the above constraints reduce to something in the form of $x_i + x_j \leq 1$, and thus can be added to the conflict graph of the child subproblem.

The graphs of Figure 5 represent the subproblems formed by our branching disjunction. Note again that the additional constraints added to the right branch, represented by the dashed edges, form 4-cliques, again allowing us to strengthen the symmetry-exploiting constraints and replace them with clique-based constraints. Enforcing the two corresponding clique inequalities in the linear description the LP relaxation yields an integer optimal solution of value 4 and the subproblem is pruned. Notice that adding these constraints avoids generating subproblems 18 and 19 in the OB tree. The overall saving produced by OC w.r.t. OB corresponds to the gray nodes of Figure 2.

This example demonstrates the potential positive impact of employing the symmetry-induced branching inequalities we propose. We next perform a suite of computational experiments to further test their utility.

5. Experimental setting

In order to produce a fair evaluation of the orbital conflict algorithm, we also implemented other symmetry-exploiting techniques, such as isomorphism pruning (Margot 2002) and orbital fixing (Margot 2003). Isomorphism pruning is an approach that prunes the branching tree in such a way

that it only contains one element per each orbit of optimal solutions. Let $\mathcal{O}$ be the set of orbits defined by $\text{stab}(F_1^a, \mathcal{G})$ at a node a. Orbital fixing acts on each orbit $O \in \mathcal{O}$ as follows:

$$\texttt{if } O \cap F_0^a \neq \varnothing \rightarrow x_i = 0, \forall i \in O \tag{14}$$

$$\texttt{if } O \cap F_1^a \neq \varnothing \rightarrow x_i = 1, \forall i \in O \tag{15}$$

When $O \cap (F_0^a \cup F_1^a) = \varnothing$, then O is called a free orbit.

The OB scheme branches on one of the free orbits at a feasible node a. Let $O_1, O_2, \ldots, O_p$ be the free orbits at node a. For our computational experiments, the branching rule was set to choose a variable belonging to the largest orbit, this is

$$x_i \in O_{j^*} \text{ s.t. } j^* \in \arg\max_{j \in \{1,\ldots,p\}} |O_j|.$$

We implement isomorphism pruning in an equivalent way that we call *isomorphism fixing*. For each free orbit O_i, let $j_i \in O_i$ be its variable with minimum index. If the set $F_1^a \cup j_i$ is not the lexicographically minimal element in its orbit, we set all variables in O_i to be zero. The argument is that if we did create a branching node with $F_1^a \cup j_i$ as the set of variables fixed to one, then a symmetric set of variables fixed to one *will* be used at another node, so we could safely prune the node.

In the original paper on isomorphism pruning of Margot (2002), the branching scheme was rigid, as it was only possible to branch on the variable with a minimum index across all free orbits. The thesis of Ostrowski (2009) proved that other branching rules were also valid when performing an additional conjugation of the group before implementing group operations. See also Pfetsch and Rehn (2019) for a nice description of the method. We implement the flexible version of isomorphism pruning in our implementation so that it can be combined with orbital branching, branching on a variable that appears in a largest orbit.

5.1. Set of instances

To test the performance of orbital conflict, we prepared a collection of instances where symmetry is known to be present. These instances belong to the following problem types: Steiner triple systems (STS), covering designs (COV), binary coding (COD), binary-ternary covering codes (CODBT), infeasible instances from Margot website (FLOSN, JGT, MERED, OFSUB9), two stable set instances (KELLER), and minimally aliased response surface designs (OMARS).

A n-STS instance is based on a Steiner triple system of order n, which is a collection of triples from n elements such that every pair of distinct elements appears together in only one of the triples. A n-STS instance consists on finding the smallest set of elements that covers all the triples of a given Steiner triple system of order n. This is known as the incidence width of the system.

These instances were introduced in (Fulkerson et al. 1974, Feo and Resende 1989, Karmarkar et al. 1991). For these problems, we considered the complementary formulation, where each variable is substituted by its complement ($x \to 1 - x$).

A (v, k, t)-COV instance is a collection of k-subsets of v elements such that every t-element subset is contained in at least one of the k-subsets in the collection. See (Schönheim 1964) for an analysis of the properties of these designs. The problems that we considered are minimization problems on the number of k-subsets. The repository at `https://www.ccrwest.org/cover.html` points to references for different instances and indicates which problems are not yet solved.

A (n, R)-COD instance is a collection of codewords in a binary code of length n such that any element of the binary code of length n is at a distance of at most R of a codeword of the collection. R is called the covering radius. The paper of Graham and Sloane (1985) is recommended for further information. The link `http://old.sztaki.hu/~keri/codes/index.htm` contains an updated repository of the current solvability status of different COD instances.

A (b, t)-CODBT instance is a binary-ternary covering code, that is, a collection of vectors of length $b + t$ with b binary coordinates and t ternary coordinates such that, for every possible binary-ternary vector defined by b and t there is a vector in the collection at a distance of at most 1. When $b = 0$, this problem is also known as the football pool problem (Linderoth et al. 2009). The same link quoted for the COD instances has results for mixed binary-ternary codes. In Section 6.3, we will use our symmetry-enhanced integer programming methods to solve for the first time three of these instances.

KELLER integer programs correspond to computing the stability number of *Keller graphs*. These arise in the reformulation of Keller's cube-tiling conjecture (Debroni et al. 2011). Vertices of a d-dimensional Keller graph correspond to the 4^d d-digit numbers (d-tuples) on the alphabet $\{0, 1, 2, 3\}$. Two vertices are adjacent if their labels differ in at least two positions, and in at least one position the difference in the labels is two modulo four. Keller graphs were introduced in the second DIMACS max-clique challenge and represent meaningful benchmark instances for the max-clique/stable set community. The instance are available at `ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique`. Note that since we are interested in computing stable sets, we consider complemented versions of the clique problems. We test three formulations for graph Keller 4, based on clique and nodal inequalities. Details can be found in (Letchford et al. 2018).

A (m, n, α, β)-OMARS instance is an orthogonal minimally-aliased response surface design (see (Núñez-Ares and Goos 2019)), which is an experimental design with m factors and n runs with sparsity properties α and β that have desirable statistical estimation properties. These instances are enumeration instances, so we will enumerate all non-isomorphic feasible solutions.

All but the OMARS instances are optimization problems. However, in some cases, we consider optimization instances as enumeration instances by finding all non-isomorphic solutions with a given objective value. Table 1 shows some characteristics of the instances. In addition to the instance size, we list the type of inequalities present in the problem and its symmetry group size. For optimization instances, we list the optimal solution value as well as the number of non-isomorphic optimal solutions for each instance. For unsolved instances CODBT52,CODBT71,CODBT43 we report the best known lower and upper bounds.

Type	Instance	#cols	#rows	L/E/G	Symmetry Group size	optimal value # Sols
COD	8,3	256	256	L	9.29E+07	4/2
CODBT	0,5	243	243	G	933120	27/17
CODBT	4,2	144	144	G	27648	20/1
CODBT	4,3	432	432	G	497664	44-48?
CODBT	5,2	288	288	G	276480	32-36?
CODBT	7,1	384	384	G	2.32E+07	42-48?
COV	9,5,4	126	255	G	362880	30/3
COV	10,7,5	120	637	G	3.63E+06	20/3
FLOSN	52	234	234	L/E	312	infeasible
FLOSN	60	270	270	L/E	360	infeasible
FLOSN	84	378	378	L/E	504	infeasible
JGT	18	132	105	L/E	48	infeasible
JGT	30	228	177	L/E	96	infeasible
KELLER	4A	171	881	L	384	11/8
KELLER	4B	171	473	L	384	11/8
KELLER	4C	171	477	L	384	11/8
MERED		560	420	L/E	9.29E+11	infeasible
OFSUB	9	203	92	L/E	60480	infeasible
STS	45c	45	330	L	360	15/1
STS	63c	63	651	L	72576	18/7
STS	81c	81	1080	L	7.97E+09	20/1
MARSD	4,22,6,10	80	37	E	384	-/5
MARSD	5,24,10,16	242	61	E	3840	-/20
MARSD	5,26,8,14	242	61	E	3840	-/250
MARSD	5,28,10,16	242	61	E	3840	-/224

Table 1 Characteristics of Test Instances

5.2. Implementation

Our approach was implemented using the user application functions of MINTO v3.1 (Nemhauser et al. 1994), while the clique separation algorithm used the LEMON library (Dezső et al. 2011), and the necessary symmetry calculations (group operations) used the PERMLIB library (Rehn and Schürmann 2010). To compute the generators of the symmetry group of the problem instance, we relied on the usual approach of building a graph such that the formulation symmetries correspond

one-to-one to the graph automorphisms. The generators were calculated using the NAUTY software (McKay and Piperno 2014). For a more complete description of computing the symmetry group, the reader is invited to turn to (Puget 2005, Salvagnin 2005, Margot 2010, Pfetsch and Rehn 2019). To minimize the performance variability induced by the timing at which feasible solutions are found by branch and bound, we input the optimal value of the instance to the solver to use as a cutoff. We also employ reduced-cost fixing.

6. Computational experiments

Our computational results are divided into three parts. First, we demonstrate the prevalence of the level-k inequalities on one symmetric instance. Next, we measure the computational impact of adding level-1 inequalities to the conflict graph, as described in Algorithm 1. We finish by using the symmetry-enhanced conflict graph in combination with a two-phase approach and parallel computing to solve unsolved symmetric IP instances arising from covering designs.

6.1. Frequency of Level-k Inequalities

Our first experiment demonstrates the prevalence of (unique) level-k inequalities for small values of k and the relative computational time of implementing the different features of our algorithm. For the instance CODBT52, Table 2 displays the average number of level-$\{1,2,3\}$ cuts found at nodes a for different values of $|F_1^a|$ as well as the average group size of $\text{stab}(F_1^a, \mathcal{G})$. Our first observation is the large number of level-k inequalities generated. In our computation, we do *not* add the level-k inequalities to the linear programming relaxation at the child nodes, since adding that many inequalities would quickly overwhelm the linear programming solver. Another interesting observation from Table 2 is that level-k inequalities can be generated at nodes that do not contain any apparent symmetry, as evident by the last row of the table. In some sense, level-k inequalities "look back" in the tree and determine cuts that would have been generated if the branching order had been different.

Table 3 shows the breakdown of computation time for different parts of the algorithm, when implemented on the instance CODBT52. The table demonstrates the significant computational effort required to general level-k inequalities and how that effort increases with k. The algorithm behavior on CODBT52 is quite typical of all of our computational results, both in terms of the number of level-k inequalities generated and the CPU time required to generate them. Thus, in subsequent experiments, we focus on assessing the impact of implicitly employing the level-1 inequalities as edges of a local conflict graph as described in Algorithm 1.

$\lvert F_1^a \rvert$	#nodes	1-level	2-level	3-level	$\lvert \text{stab}(F_1^a, \mathcal{G}) \rvert$
0	1	0	0	0	933120
1	3	5265	0	0	3840
2	11	440.73	26987.7	0	436.36
3	45	90.69	1594.42	79812.0	168
4	197	46.77	310.15	3711.6	40.19
5	530	23.43	126.88	628.17	18.78
6	272	15.11	55.07	193.21	12.68
7	176	13.46	46.86	122.08	2.59
8	76	18.4	69.47	168.64	3.37
9	14	26.0	98.71	244.64	9.07
10	36	16.14	64	189.08	1.14
11	48	11.35	63.04	192.83	1.04
12	44	12.25	67.41	232.02	1.02
13	8	13.0	78.25	287.75	1

Table 2 Density of the 1-,2- and 3-level cuts as a function of $|F_1^a|$ for the instance codbt52

Operations	**CPU Time**
LP Solving	139.0
Clique Separation	27.2
Orbital Branching/Fixing	39.6
Isomorphism Fixing	53.9
Level-1 Cut Generation	234.5
Level-2 Cut Generation	906.2
Level-3 Cut Generation	3658.8
Total	5152.5

Table 3 Timing of the different parts of the implementation for the instance codbt52

6.2. Results for small and medium size instances

For our next experiment, we solved the optimization instances in Table 1 both with and without adding the level-1 inequalities to the local conflict graph. Computations were completed on a DellR810 machine with 256G of RAM and E7-4850 Xeon processor, and Table 4 shows the results of this experiment. The table contains the time, number of nodes, and total number of clique inequalities found for both cases for each instance. Table 4 shows that some significant decrease in branch-and-bound tree size can be obtained when using the orbital conflict procedure. In fact, for each of the 18 test instances, the tree size was smaller when using orbital conflict, resulting in a tree that was on average only 58% of the size of the tree without the orbital conflict-induced clique inequalities. However, the results in the table also indicate that the computational effort required for the reduced tree size is significant. Specifically, even though the number of nodes is significantly reduced, the CPU time on average is increased by 7.7% when doing orbital conflict. In six of the 18 instances, the CPU time was reduced.

	No Orbital Conflict			Orbital Conflict		
Instance	**# clq**	**#node**	**Time**	**# clq**	**#node**	**Time**
cod83	263	225	30.0	2604	127	36.6
codbt05	0	56293	1849.6	6372	35367	1965.3
codbt42	0	10073	104.7	3465	6251	139.2
cov1075	0	41843	1207.6	8346	21881	1132.1
cov954	0	4777	69.9	1115	1793	51.9
flosn52	0	1371	18.7	66	917	21.9
flosn60	0	3867	64.9	155	2787	86.1
flosn84	0	186475	5715.1	1567	132771	8336.6
jgt18	102	269	2.2	102	269	3.4
jgt30	4197	21301	393.8	2933	12265	367.3
keller4a	1438	79	55.8	2163	63	59.2
keller4b	1628	649	155.6	2352	535	166.7
keller4c	1874	81	90.3	2361	59	80.0
mered	0	1523	262.75	44	237	158.24
ofsub	14047	37819	968.4	5930	17753	936.1
sts45c	0	6019	16.1	438	3689	22.7
sts63c	0	11415	335.7	3036	5365	353.1
sts81c	0	733	475.0	1010	557	531.5
Geo. Mean	-	3261	150.2	-	1909	161.8

Table 4 Optimization results with IP, with and without OC

	No Orbital Conflict			Orbital Conflict		
Instance	**# clq**	**#node**	**Time**	**# clq**	**#node**	**Time**
cod83	259	696	98.4	3039	382	275.3
codbt05	0	85141	3063.2	13089	45556	3075.4
codbt42	0	15797	163.05	3951	9855	240.2
cov1075	0	62433	1820.9	12166	30861	1730.8
cov954	0	7983	152.9	1713	3455	159.8
keller4a	1650	932	208.5	2228	799	210.6
keller4b	1707	2881	386.5	2513	2435	418.3
sts45c	0	8298	23.4	498	5383	30.1
sts63c	0	13948	338.5	3677	7198	397.9
sts81c	0	906	682.6	1181	677	746.9
omars(4,22,6,10)	0	785	3.7	100	589	9.24
omars(5,24,10,16)	0	10603	220.8	1129	8853	546.1
omars(5,26,8,14)	0	219504	4950.8	12220	160472	11074.7
omars(5,28,10,16)	0	262336	5681.7	19986	192954	12899.1
Geo. Mean	-	9675	323.4	-	6289	477.4

Table 5 Results on enumeration instances, with and without OC

Table 5 presents the enumeration results for our test instances. The instances with a non-zero objective function were turned into enumeration instances by enumerating all optimal solutions to the instance. The results are similar to the results when optimizing shown in Table 4. When doing orbital conflict, the enumeration tree size is only 65% of the enumeration tree size required without

orbital conflict. However, again, the computational effort required to implement OC outweighs the tree size improvement, result in an average increase in CPU time of 47%. The CPU time is reduced in only one of the 14 enumeration test instances.

6.3. Results for large instances

Despite the fact that the CPU times in general increased when using orbital conflict for the small-to-medium sized instances, the significant reduction in tree size led us to believe that performing the computationally costly orbital conflict procedure at the top of the branch and bound tree could help us to solve larger symmetric instances.

With that in mind, we developed the following two-phase approach to tackling difficult, symmetric instances. In Phase I, we do the orbital conflict procedure on the instance as usual, but we prune a feasible node a if the size of $\text{stab}(F_1^a, \mathcal{G})$ falls below a certain threshold. These pruned nodes (which we call leaves) are saved to MPS files together with the edges of the conflict graph valid at that node. In Phase II, each one of these MPS files can be solved in parallel using a commercial, state-of-the art MIP solver. The hope is that through the combination of branching, symmetry-induced variable fixing, and enhanced conflict-graph information, solving each of the active leaf node instances will be significantly easier than solving the original problem directly. In addition, each of the active leaf node instances can be solved in parallel. The jobs on Phase II are executed on a High Throughput Computing (HTC) Grid managed by the scheduling software HTCondor (Thain et al. 2005).

To demonstrate the promise of this two-phase approach, we employ it to solve the instance CODBT52. The optimization software CPLEX 12.7, tuned to aggressively tackle symmetry, can solve this instance in 14,087 seconds (roughly 4 hours) and 28,817,719 nodes. Note that this in itself is an achievement, as this instance has not been reported solved in the literature. However, the two-phase approach we propose can solve this instance much more effectively. In Table 6, we show the computational behavior of a two-phase approach both with and without employing orbital conflict, fathoming all nodes and writing to MPS files once the group group size fell below $|\text{stab}(F_1^a, \mathcal{G})| \leq 128$. In the table, we shows the CPU time in Phase I, the number of clique inequalities obtained, the number of active leaves that need to be solved at the end of Phase I, the total CPU time for CPLEX to solve all leaf nodes in Phase II, the total wall clock time in Phase II, and the total Wall Time (combining Phase I CPU/Wall Time with Phase II Wall Time). In Phase II, we limit the number of threads that CPLEX can use to 2, so as not to overwhelm the shared resources provided by HTCondor. From the table, we see that *significant* speedup is possible with this two-phase approach, solving the same instance in only 634 seconds, so we employed it in an effort to solve some heretofore unsolved instances of binary-ternary covering.

Method	Phase I Time	Num clique	Num leaves	Phase II CPU Time	Leaves Ave Time	Phase II Wall Time	Total Wall Time
OC	205.5	436	373	2132.9	5.7	429	634.5
No OC	148.8	0	356	6883.0	19.3	1127	1275.8

Table 6 Performance of two-phase method on solving codbt52. Time is measured in seconds.

Table 7 shows the results on two other open binary-ternary covering design problems. The optimal solution of CODBT43 is known to lie in the interval $[44 - 48]$. We run our two-phase algorithm with orbital conflict, pruning and saving all nodes a such that $\text{stab}(F_1^a, \mathcal{G}) < 16$. In total, we generated $1,208$ such nodes, which were then solved with CPLEX, setting an upper value of 47.1. All of the active leaf node instances were infeasible, so the optimal solution to CODBT43 is 48. For the instance CODBT71, we employed our two-phase approach with a fathoming group size of 32, resulting in a branch-and-bound tree with $111,318$ nodes left to be evaluated. All of these instances were solved in parallel by CPLEX, using an upper bound value of 47.1 as a cutoff value. Since all $111,318$ instances were infeasible, our computations have verified that the optimal solution for the instance CODBT71 has value 48. To our knowledge the optimal solutions for instances CODBT52,CODBT71 and CODBT43 are all new.

instance	known bounds	Fathom Group Size	UB	#clq	Phase I Time	Phase I Num Nodes	Num leaves	Phase II Comp Time	Phase I Num Nodes	Phase I Walltime
CODBT43	44-48	16	47.1	41900	4332	12392	5287	563863	50752135	185309
CODBT52	32-36	32	35.1	2708	938	3847	1191	1830.5	251380	568
CODBT71	42-48	8	47.1	338955	2090430	2961786	491712	242613	17465158	68668

Table 7 Results on two large codbt instances. Time is measured in seconds.

7. Conclusions

In this work, we introduce a hierarchy of cutting planes for symmetric integer programs that arise from a reinterpretation of symmetry-exploiting branching methods. We show how to implicitly and effectively utilize the cutting planes from level-1 of this hierarchy to populate the conflict graph in the presence of symmetry. Our orbital conflict method outperforms the state-of-the-art solver CPLEX 12.7 for large symmetric instances, proving that augmenting the instance conflict graph via symmetry considerations can be beneficial. For small and medium sized instances, the time spent in the additional group operations required to make use of the deeper symmetry operations appears to outweigh its benefit, at least in our implementation.

Future work consists of refining the orbital conflict algorithm to make it more computationally efficient. For example, we could consider doing fast, partial group operations to compute the symmetries in (10), or not performing the calculations at all nodes of the enumeration tree. Another extension of the work is to make better computational use of the level-k inequalities for $k \geq 2$. By complementing the variables in the Equation (7), we obtain a set-covering inequality

$$\bar{x}(\pi(F_1^a \cup i) \setminus F_1^a) \geq 1.$$

So these inequalities could be used in a set-covering relaxation for the instance. As seen in Table 2, our procedure can generate a considerable number of k-level cuts for $k \geq 2$, so there may be some benefit to this approach.

Acknowledgments

This work was started while authors Rossi and Smriglio were visiting the Wisconsin Institute for Discovery of University of Madison. This research was performed using the compute resources and assistance of the UW-Madison Center For High Throughput Computing (CHTC) in the Department of Computer Sciences. The CHTC is supported by UW-Madison, the Advanced Computing Initiative, the Wisconsin Alumni Research Foundation, the Wisconsin Institutes for Discovery, and the National Science Foundation, and is an active member of the Open Science Grid, which is supported by the National Science Foundation and the U.S. Department of Energy's Office of Science.

References

Achterberg T, Bixby R, Gu Z, Rothberg E, Weninger D (2019) Presolve reductions in mixed integer programming. *INFORMS Journal on Computing* URL `http://dx.doi.org/https://doi.org/10.1287/ijoc.2018.0857`, to appear.

Achterberg T, Wunderling R (2013) Mixed integer programming: Analyzing 12 years of progress. *Facets of Combinatorial Optimization*, 449–482 (Springer).

Atamtürk A, Nemhauser G, Savelsbergh M (2000) Conflict graphs in solving integer programming problems. *European Journal of Operational Research* .

Correa RC, Donne DD, Koch I, Marenco J (2018) General cut-generating procedures for the stable set polytope. *Discrete Applied Mathematics* 245:28 – 41.

Debroni J, Eblen JD, Langston MA, Myrvold W, Shor P, Weerapurage D (2011) A complete resolution of the Keller maximum clique problem. *Proceedings of the Twenty-second Annual ACM-SIAM Symposium on Discrete Algorithms*, 129–135, SODA '11 (Philadelphia, PA, USA: Society for Industrial and Applied Mathematics).

Dezső B, Jüttner A, Kovács P (2011) Lemon–an open source C++ graph template library. *Electronic Notes in Theoretical Computer Science* 264(5):23–45.

Feo T, Resende M (1989) A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters* 8(2):67–71.

Fulkerson D, Nemhauser G, Trotter L (1974) Two computationally difficult set covering problems that arise in computing the 1-width of incidence matrices of Steiner triple systems. *Approaches to integer programming*, 72–81 (Springer).

Giandomenico M, Rossi F, Smriglio S (2013) Strong lift-and-project cutting planes for the stable set problem. *Math. Progr.* 141:165–192, ISSN 0025-5610.

Graham R, Sloane N (1985) On the covering radius of codes. *IEEE Transactions on Information Theory* 31(3):385–401.

Grötschel M, Lovász L, Schrijver A (1988) Stable sets in graphs. *Geometric Algorithms and Combinatorial Optimization*, 272–303 (Springer).

Karmarkar N, Resende M, Ramakrishnan K (1991) An interior point algorithm to solve computationally difficult set covering problems. *Mathematical Programming* 52(1-3):597–618.

Letchford A, Rossi F, Smriglio S (2018) The stable set problem: Clique and nodal inequalities revisited. `http://www.optimization-online.org/DB_HTML/2018/05/6612.html`.

Linderoth J, Margot F, Thain G (2009) Improving bounds on the football pool problem via symmetry reduction and high-throughput computing. *INFORMS Journal on Computing* 21:445–457.

Margot F (2002) Pruning by isomorphism in branch-and-cut. *Mathematical Programming* 94(1):71–90.

Margot F (2003) Exploiting orbits in symmetric ILP. *Mathematical Programming* 98(1):3–21.

Margot F (2009) Symmetry in Integer Linear Programming. Jünger M, Liebling T, Naddef D, Nemhauser GL, Pulleyblank WR, Reinelt G, Rinaldi G, Wolsey LA, eds., *50 Years of Integer Programming 1958-2008*, 647–686 (Springer), ISBN 978-3-540-68274-5.

Margot F (2010) Symmetry in integer linear programming. *50 Years of Integer Programming 1958-2008*, 647–686 (Springer).

Marzi F, Rossi F, Smriglio S (2019) Computational study of separation algorithms for clique inequalities. *Soft Comput.* 23(9):3013–3027.

McKay B, Piperno A (2014) Practical graph isomorphism, II. *Journal of Symbolic Computation* 60:94–112.

Nemhauser G, Savelsbergh M, Sigismondi G (1994) MINTO, a Mixed INTeger Optimizer. *Operations Research Letters* 15(1):47–58.

Núñez-Ares J, Goos P (2019) Enumeration and multicriteria selection of orthogonal minimally aliased response surface designs. *Technometrics* 0(0):1–32.

Ostrowski J (2009) *Symmetry in Integer Programming*. Ph.D. thesis, Lehigh University.

Ostrowski J, Linderoth J, Rossi F, Smriglio S (2008) Constraint orbital branching. *In IPCO 2008: The Thirteenth Conference on Integer Programming and Combinatorial Optimization*, 225–239 (Springer).

Ostrowski J, Linderoth J, Rossi F, Smriglio S (2011a) Orbital branching. *Math. Program.* 126(1):147–178.

Ostrowski J, Linderoth J, Rossi F, Smriglio S (2011b) Solving large Steiner Triple Covering Problems. *Operations Research Letters* 39(2):127 – 131.

Padberg M (1973) On the facial structure of set packing polyhedra. *Mathematical Programming* 5(1):199–215.

Pfetsch ME, Rehn T (2019) A computational comparison of symmetry handling methods for mixed integer programs. *Mathematical Programming Computation* 11(1):37–93.

Puget JF (2005) Automatic Detection of Variable and Value Symmetries. Beek P, ed., *Lecture Notes in Computer Science, Principles and Practice of Constraint Programming-CP 2005*, volume 3709, 475–489 (Berlin Heidelberg: Springer).

Rebennack S, Oswald M, Theis D, Seitz H, Reinelt G, Pardalos P (2011) A Branch and Cut solver for the maximum stable set problem. *J. Comb. Optim.* 21(4):434–457.

Rehn T, Schürmann A (2010) C++ Tools for Exploiting Polyhedral Symmetries. *ICMS*, 295–298 (Springer).

Rotman J (1994) *An introduction to the theory of groups* (Springer), 4 edition.

Salvagnin D (2005) *A Dominance Procedure for Integer Programming*. Master's thesis, University of Padova, Padova, Italy.

Schönheim J (1964) On coverings. *Pacific J. Math.* 14(4):1405–1411.

Thain D, Tannenbaum T, Livny M (2005) Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience* 17(2-4):323–356.