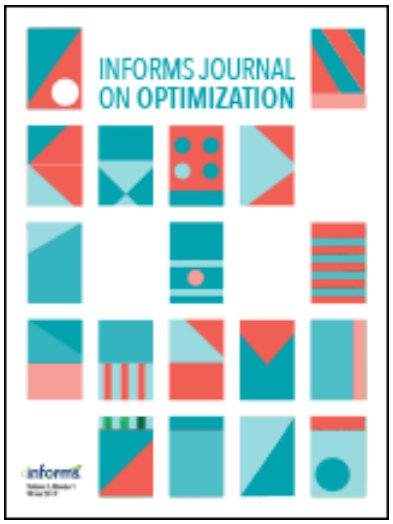


INFORMS Journal on Optimization

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Orbital Conflict: Cutting Planes for Symmetric Integer Programs

Jeff Linderoth, José Núñez Ares, James Ostrowski, Fabrizio Rossi, Stefano Smriglio

To cite this article:
Jeff Linderoth, José Núñez Ares, James Ostrowski, Fabrizio Rossi, Stefano Smriglio (2021) Orbital Conflict: Cutting Planes for Symmetric Integer Programs. INFORMS Journal on Optimization 3(2):139-153. https://doi.org/10.1287/ijoo.2019.0044

http://pubsonline.informs.org/journal/ijoo

INFORMS JOURNAL ON OPTIMIZATION
Vol. 3, No. 2, Spring 2021, pp. 139–153
ISSN 2572-1484 (print), ISSN 2575-1492 (online)

Orbital Conflict: Cutting Planes for Symmetric Integer Programs

Jeff Linderoth,[a] José Núñez Ares,[b] James Ostrowski,[c] Fabrizio Rossi,[d] Stefano Smriglio[d]

[a] Department of Industrial and Systems Engineering & Wisconsin Institutes of Discovery, University of Wisconsin–Madison, Madison, Wisconsin 53706; [b] Department of Biosystems, KU Leuven, 3001 Leuven, Belgium; [c] Department of Industrial and Systems Engineering, University of Tennessee, Knoxville, Tennessee 37996-2315; [d] Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, 67100 L'Aquila, Italy

Contact: linderoth@wisc.edu, https://orcid.org/0000-0003-4442-3059 (JL); jose.nunezares@kuleuven.be, https://orcid.org/0000-0002-8351-2946 (JNA); jostrows@utk.edu, https://orcid.org/0000-0001-5636-555X (JO); fabrizio.rossi@univaq.it, https://orcid.org/0000-0002-7495-390X (FR); stefano.smriglio@univaq.it, https://orcid.org/0000-0001-8152-003X (SS)

Received: March 15, 2020
Revised: June 6, 2020; July 26, 2020
Accepted: August 16, 2020
Published Online in Articles in Advance: March 8, 2021

https://doi.org/10.1287/ijoo.2019.0044

Abstract. 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.

Funding: This work was supported by the U.S. Office of Science [Grant DE-AC02-06CH11347]. J. Ostrowski was supported by the Department of Energy [Grant DE-SC0018175] and the Air Force Office of Scientific Research [Grant FA9550-19-1-0147].

Keywords: integer programming • symmetry • conflict graph

1. Introduction

We focus on binary integer programs (BIPs) of the form

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

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 integer programming solver CPLEX, cutting planes, not branching, have been a major driving force in improving the performance of IP algorithms. 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 reinterpretation 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 BIPs.

Let $\mathcal{F}$ denote the set of feasible solutions to BIPs. A *conflict graph* (CG) can be associated with a 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 1 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 conv($\mathcal{F}$) are both contained in the stable set polytope, denoted as STAB(CG), associated with the conflict graph CG, and valid inequalities for 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, Giandomenico et al. 2013, Correa et al. 2018). The clique inequalities of Padberg (1973) have the form $\sum_{i\in C} x_i \le 1$, 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 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. (2020), 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 this work are the following:

- a hierarchy of cutting planes that spring out of a reinterpretation 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; and
- 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 BIPs. 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.

1.1. Notation

We define $[n] = \{1, 2, \dots, n\}$. 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 $[n] = \{1, \dots, n\}$; 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 $\pi \in \Pi^n$ acts on a vector $\boldsymbol{x} = (x_1, x_2, \dots, x_n)^T \in \mathbb{R}^n$ by permuting its components, $\pi(\boldsymbol{x}) := (x_{\pi^{-1}(1)}, x_{\pi^{-1}(2)}, \dots, x_{\pi^{-1}(n)})^T$. 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:

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

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 $\mathcal{G}$:

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

We sometimes refer to the orbit of a set $S \subseteq [n]$, which is defined as

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

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

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

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 $O = \text{orb}(\text{stab}(F_1^a, \mathcal{G}), \{i\})$ 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 $(\boldsymbol{\lambda}, \boldsymbol{\lambda}_0) \in \mathbb{Z}^{n+1}$ and a base disjunction of the form

$$\left(\boldsymbol{\lambda}^\top \boldsymbol{x} \le \boldsymbol{\lambda}_0\right) \vee \left(\boldsymbol{\lambda}^\top \boldsymbol{x} \ge \boldsymbol{\lambda}_0 + 1\right),$$

constraint orbital branching exploits all symmetrically equivalent forms of $\boldsymbol{\lambda}^\top \boldsymbol{x} \le \boldsymbol{\lambda}_0$. Specifically, either an equivalent form of the inequality $(\boldsymbol{\lambda}^\top \boldsymbol{x} \le \boldsymbol{\lambda}_0)$ holds one of the members of $\{\pi(\boldsymbol{\lambda}) \mid \pi \in \mathcal{G}\} = \text{orb}(\mathcal{G}, \boldsymbol{\lambda})$ or the inequality $\boldsymbol{\lambda}^\top \boldsymbol{x} \ge \boldsymbol{\lambda}_0 + 1$ holds for all of them. This yields the following binary branching disjunction:

$$\left(\lambda^\top x \le \lambda_0\right) \vee \left([\pi(\lambda)]^\top x \ge \lambda_0 + 1 \quad \forall \pi \in \mathcal{G}\right), \tag{3}$$

which is equivalent to

$$\left(\boldsymbol{\lambda}^\top \boldsymbol{x} \le \boldsymbol{\lambda}_0\right) \vee \left(\mu^\top \boldsymbol{x} \ge \boldsymbol{\lambda}_0 + 1 \quad \forall \mu \in \text{orb}(\mathcal{G}, \boldsymbol{\lambda})\right).$$

Note that orbital branching is a special case of constraint orbital branching using the integer vector $(\lambda, \lambda_0) = (e_i, 0)$. The reader may turn to Ostrowski et al. (2008, 2011b) for a proof of validity of the method.

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:

$$\left(x\left(F_1^a \cup i\right) \ge |F_1^a| + 1\right) \vee \left(x\left(F_1^a \cup i\right) \le |F_1^a|\right). \tag{4}$$

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

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

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. Note that adding only a subset of inequalities to the right branch of (5) only weakens the disjunction and thus is valid. 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

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

and the branching inequality

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

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

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

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, that is:

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

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 $k = 1, 2, \ldots, K < |F_1^a|$:

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

Given $T \subset [n]$ and $i \notin T$, characterizing the permutations in $\mathcal{L}_k(T, i)$ exactly appears to be 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,*

$$\bigcup_{\{S \subseteq T : |T \setminus S| \leq k\}} \text{stab}(S, \mathcal{G}) \subseteq \bigcup_{j=0}^{k} \mathcal{L}_j(T, i). \tag{10}$$

Proof of Theorem 1. Let $S \subseteq T$ with $|T \setminus S| = k$ and π in $\text{stab}(S, \mathcal{G})$. We will show 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 subtree rooted at a.

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

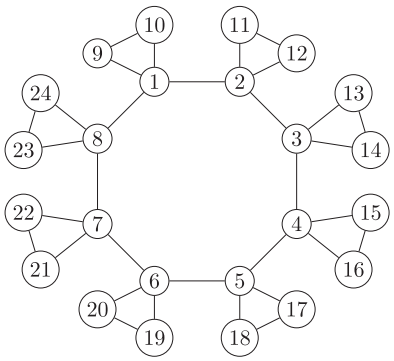
Included in the output of Algorithm 1 is a symmetry-enhanced conflict graph $G(V, E_c)$ on the right branch of the branching disjunction. When node c is to be explored, the clique cut separation heuristic of Marzi et al. (2019) is run to search for clique inequalities violated by the Linear Programming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

Example 1.

Consider the graph $G = (V, E)$ of Figure 1 and define a binary variable x_i for each vertex $i \in V$ assuming value 1 if and only if i belongs to a stable set of G. Computing the stability number of G amounts to solve the following BIP:

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

Figure 1. Example Graph



The optimal value is 8, while the root LP relaxation $\hat{x}$ has objective value 12, with $x_i = 1/2, i \in V$. The symmetry group $\mathcal{G}$ consists of 4,096 permutations yielding two large orbits, corresponding to inner and outer rings. 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

$$(x_9 + x_{11} \geq 2) \vee \left(x_{\pi(9)} + x_{\pi(11)} \leq 1 \quad \forall \pi \in \mathcal{G}\right). \tag{12}$$

Figure 2. Search Tree by Orbital Branching

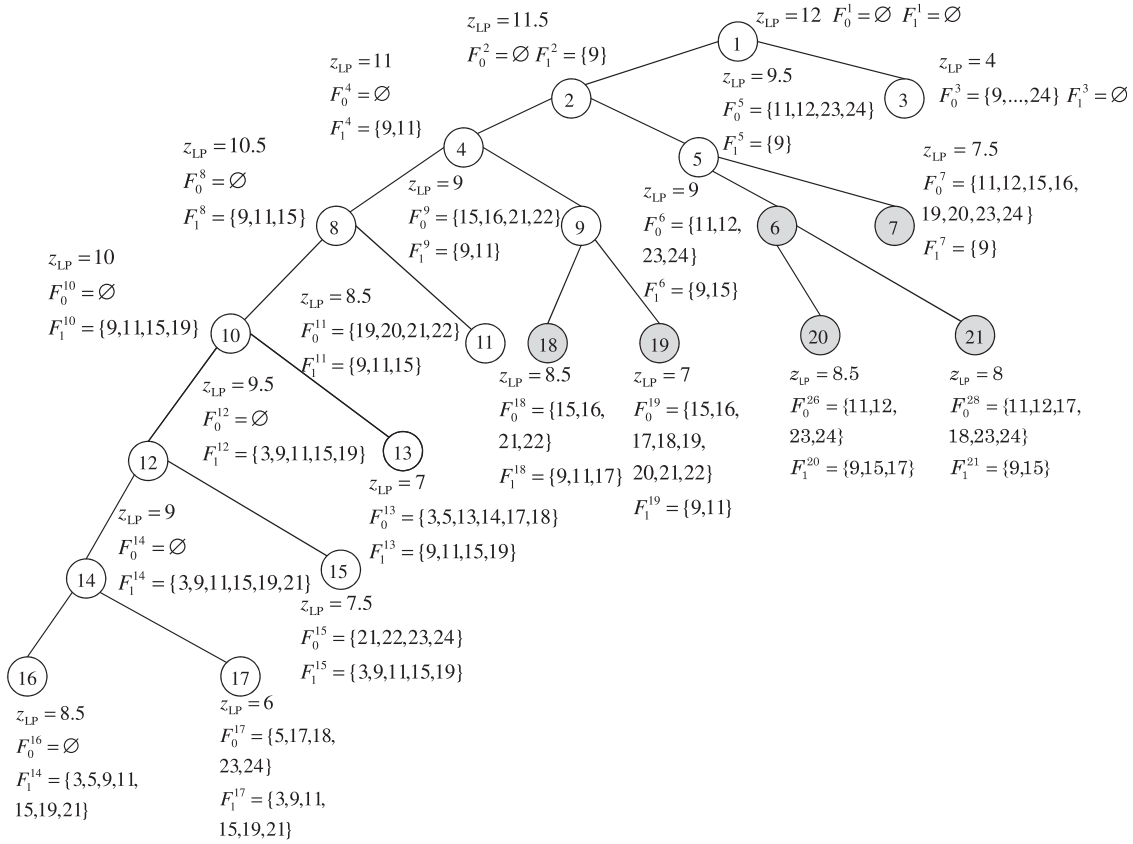
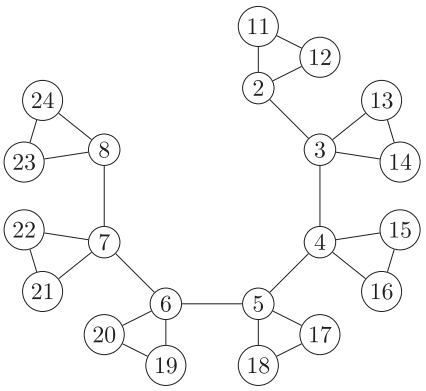


Figure 3. Graph of Subproblem 2



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

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

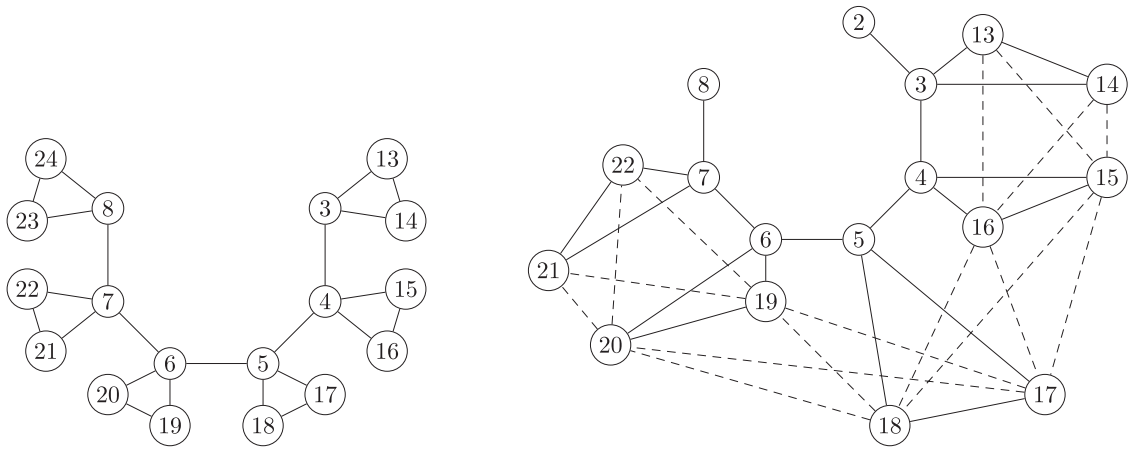
As $|F_1^2| = 1$, these inequalities are either level-0 (written in bold font style) or level-1 (written in plain font style) inequalities. According to Theorem 1, the former are generated by $\bigwedge_{\pi \in \mathrm{stab}(\{9\}, \mathcal{G})} x_{\pi(9)} + x_{\pi(11)} \le 1$. Here $\mathrm{stab}(\{9\}, \mathcal{G})$ consists of 1,024 permutations and describes the graph symmetry with respect to an axis traversing vertices 1 and 5. Note that all of the level-0 constraints are generated, demonstrating that $\mathrm{stab}(\{9\}, \mathcal{G})$ is a good approximate for $\mathcal{L}_0$ in this instance. Observe also that, according to inequality (7), 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. Level-1 inequalities are generated by $\bigwedge_{\pi \in \mathcal{G}} x_{\pi(9)} + x_{\pi(11)} \le 1$ and correspond to pairs of vertices in the outer ring that belong to consecutive triangles.

The graphs of Figure 4 represent the child subproblems that result from the previous disjunction. Note the abundance of four-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 three-cliques. We can use these cliques to strengthen the level-1 inequalities generated by the branching disjunction. In fact, all the generated inequalities will be dominated by the four four-clique inequalities, reducing the size of the resulting formulation while also tightening it. Adding the clique inequalities from the four-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 four more nodes (6,7, 20, and 21).

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

$$(x_9 + x_{11} + x_{15} \ge 3) \vee \left(x_{\pi(9)} + x_{\pi(11)} + x_{\pi(15)} \le 2 \quad \forall \pi \in \mathcal{G}\right). \tag{13}$$

Figure 4. Children of Subproblem 2



Graph of subproblem 4 Graph of subproblem 5

The right side of the disjunction (13) has 128 nonredundant 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:

$$x_9 + x_{11} + x_{15} \le 2 \quad x_9 + x_{11} + x_{16} \le 2$$
$$x_9 + x_{11} + x_{21} \le 2 \quad x_9 + x_{11} + x_{22} \le 2,$$

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

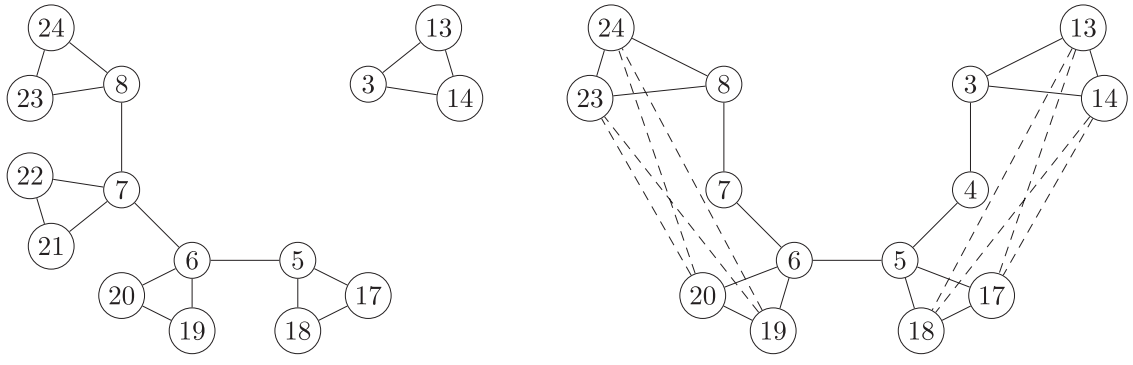
A subset of the level-1 inequalities is formed by looking at the subgroups $\text{stab}(\{9\}, \mathcal{G})$ and $\text{stab}(\{11\}, \mathcal{G})$. The resulting level-1 inequalities generated by $\text{stab}(\{9\}, \mathcal{G}) \cup \text{stab}(\{11\}, \mathcal{G}) \subset \mathcal{L}_0 \cup \mathcal{L}_1$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

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

Note, of course, that because x_9 and x_{11} are fixed to one at node 9, the given constraints reduce to something in the form of $x_i + x_j \le 1$, 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 four-cliques,

Figure 5. Children of Subproblem 4



graph of subproblem 8 graph of subproblem 9

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 with respect to 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:

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

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

When $O \cap (F_0^a \cup F_1^a) = \emptyset$, 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:

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

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 created a branching node with $F_1^a \cup j_i$ as the set of variables fixed to one, then a symmetric set of variables—the lexicographically minimal set in its orbit—will be fixed to one at another node, so there is no need to create a node with the set of variables $F_1^a \cup j_i$ fixed to one, and we can safely fix j_i to zero.

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 (2007) (FLOSN, JGT, MERED, OFSUB9), two stable set instances (KELLER), and minimally aliased response surface designs (OMARS). Files in MPS format for all instances are available at http://homepages.cae.wisc.edu/~linderot/data/oc-instances.tar.gz.

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 of 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), and Karmarkar et al. (1991). For these problems, we considered the complementary formulation, where each variable is substituted by its complement ($x \rightarrow 1 - x$).

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, where 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 one. 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 (Johnson and Trick 1996) 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. (2020).

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 nonisomorphic feasible solutions.

All but the OMARS instances are optimization problems. However, in some cases, we consider optimization instances as enumeration instances by finding all nonisomorphic solutions with a given objective value. Table 1 shows some characteristics of the instances. The columns show the type of problem, the parameters of the considered instance of the problem, the number of columns and rows of the integer programming formulation, the type of inequalities (less or equal (L), equal (E), greater or equal (G)) appearing in the formulation, and the formulation symmetry group size. Additionally, for optimization instances, we list the optimal solution value

Table 1. Characteristics of Test Instances

Type	Instance	No. of columns	No. of rows	L/E/G	Symmetry group size	Optimal value/no. of solutions
COD	8, 3	256	256	L	9.29E+07	4/2
CODBT	0, 5	243	243	G	933,120	27/17
CODBT	4, 2	144	144	G	27,648	20/1
CODBT	4, 3	432	432	G	497,664	44–48?
CODBT	5, 2	288	288	G	276,480	32–36?
CODBT	7, 1	384	384	G	2.32E+07	42–48?
COV	9, 5, 4	126	255	G	362,880	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	60,480	Infeasible
STS	45c	45	330	L	360	15/1
STS	63c	63	651	L	72,576	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	3,840	0/20
MARSD	5, 26, 8, 14	242	61	E	3,840	—/250
MARSD	5, 28, 10, 16	242	61	E	3,840	—/224

as well as the number of nonisomorphic optimal solutions for each instance. For unsolved instances CODBT52, CODBT71, and CODBT43, we report the best known lower and upper bounds.

5.2. Implementation

Building on our software from Ostrowski et al. (2008), we implemented the orbital conflict procedure using the user application functions of MINTO v3.1 (Nemhauser et al. 1994). MINTO is no longer a state-of-the-art framework for mixed integer programming. However, in our experience, its performance (when augmented with symmetry-aware branching methods) on highly symmetric instances is comparable with more modern solvers. 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 (2009), and 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 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 an enumeration version of the instance CODBT05, Table 2 displays the average number of level-$\{1, 2, 3\}$ cuts found at nodes a for different values of $|F_1^a|$. The second column, #nodes, indicates the number of nodes in the branch-and-bound tree for each value of $|F_1^a|$. Our primary observation is the large number of level-k inequalities generated. Note that in this 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.

Table 3 shows the breakdown of computation time for different parts of the algorithm, when implemented on the instance CODBT05. The table demonstrates the significant computational effort required to generate level-k inequalities and how that effort increases with k. The algorithm behavior on CODBT05 is quite typical of all of our computational results, both in terms of the number of level-k inequalities generated and the central processing unit (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.

6.2. Results for Small- and Medium-Sized 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 256 gigabytes of RAM and an 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.

Table 5 presents the enumeration results for our test instances. The instances with a nonzero 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

Table 2. Average Number of One-, Two-, and Three-Level Cuts as a Function of $|F_1^a|$ for the Instance CODBT05

$\lvert F_1^a \rvert$	No. of nodes	Avg. no. one-level	Avg. no. two-level	Avg no three-level
0	1	0.00	0.00	0.00
1	2	9,720.00	0.00	0.00
2	4	1,665.00	136,080.00	0.00
3	24	298.38	9,342.00	670,680.00
4	136	32.78	734.84	14,921.65
5	1,104	14.42	143.35	1,667.92
6	6,074	10.66	71.38	434.04
7	13,405	9.83	50.46	201.78
8	1,135	9.91	47.30	118.85
9	240	11.90	56.16	178.64
10	99	14.44	76.94	218.45
11	81	15.48	277.51	417.52
12	75	12.93	69.12	238.85
13	42	13.88	81.02	295.33
14	34	14.30	93.15	371.00
15	37	17.54	114.00	486.05
16	27	17.26	124.04	574.41
17	26	18.38	142.42	702.46
18	23	24.22	181.17	894.87
19	21	22.05	182.71	1,020.76
20	20	21.25	199.30	1,172.20
21	20	23.30	223.25	1,380.50
22	19	23.47	238.11	1,577.16
23	17	23.76	260.00	1,808.82
24	17	25.94	291.71	2,091.65
25	17	27.47	319.24	2,396.12
26	17	26.00	325.00	2,600.00
27	3	0.00	0.00	0.00

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 perform 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 Mixed Integer Programming

Table 3. Timing of the Different Parts of the Implementation for the Instance CODBT05

Operations	CPU time (sec.)
LP solving	594.9
Clique separation	341.6
Orbital branching/fixing	248.2
Isomorphism fixing	336.5
Level-1 cut generation	1,040.5
Level-2 cut generation	3,478.0
Level-3 cut generation	12,295.8
Total	18,732.8

Table 4. Optimization Results with IP, with and without OC

	No orbital conflict			Orbital conflict		
Instance	No. of cliques	No. of nodes	Time	No. of cliques	No. of nodes	Time
cod83	263	225	30.0	2,604	127	36.6
codbt05	0	56,293	1,849.6	6,372	35,367	1,965.3
codbt42	0	10,073	104.7	3,465	6,251	139.2
cov1075	0	41,843	1,207.6	8,346	21,881	1,132.1
cov954	0	4,777	69.9	1,115	1,793	51.9
flosn52	0	1,371	18.7	66	917	21.9
flosn60	0	3,867	64.9	155	2,787	86.1
flosn84	0	186,475	5,715.1	1,567	132,771	8,336.6
jgt18	102	269	2.2	102	269	3.4
jgt30	4,197	21,301	393.8	2,933	12,265	367.3
keller4a	1,438	79	55.8	2,163	63	59.2
keller4b	1,628	649	155.6	2,352	535	166.7
keller4c	1,874	81	90.3	2,361	59	80.0
mered	0	1,523	262.75	44	237	158.24
ofsub	14,047	37,819	968.4	5,930	17,753	936.1
sts45c	0	6,019	16.1	438	3,689	22.7
sts63c	0	11,415	335.7	3,036	5,365	353.1
sts81c	0	733	475.0	1,010	557	531.5
Geometric mean	—	3,261	150.2	—	1,909	161.8

(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 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 four 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 size fell below $|\text{stab}(F_1^q, \mathcal{G})| \leq 128$. In the table, we show 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

Table 5. Results on Enumeration Instances, with and without OC

	No orbital conflict			Orbital conflict		
Instance	No. of cliques	No. of nodes	Time	No. of cliques	No. of nodes	Time
cod83	259	696	98.4	3,039	382	275.3
codbt05	0	85,141	3,063.2	13,089	45,556	3,075.4
codbt42	0	15,797	163.05	3,951	9,855	240.2
cov1075	0	62,433	1,820.9	12,166	30,861	1,730.8
cov954	0	7,983	152.9	1,713	3,455	159.8
keller4a	1,650	932	208.5	2,228	799	210.6
keller4b	1,707	2,881	386.5	2,513	2,435	418.3
keller4c	2,172	849	318.2	2,533	685	288.9
sts45c	0	8,298	23.4	498	5,383	30.1
sts63c	0	13,948	338.5	3,677	7,198	397.9
sts81c	0	906	682.6	1,181	677	746.9
omars(4, 22, 6, 10)	0	785	3.7	100	589	9.24
omars(5, 24, 10, 16)	0	10,603	220.8	1,129	8,853	546.1
omars(5, 26, 8, 14)	0	219,504	4,950.8	12,220	160,472	11,074.7
omars(5, 28, 10, 16)	0	262,336	5,681.7	19,986	192,954	12,899.1
Geometric mean	—	8,226	323.1	—	5,423.1	461.7

Table 6. Performance of Two-Phase Method on Solving CODBT52

Method	Phase I time	No. of clique	No. of leaves	Phase II CPU time	Leaves avg. time	Phase II wall time	Total wall time
OC	205.5	436	373	2,132.9	5.7	429	634.5
No OC	148.8	0	356	6,883.0	19.3	1,127	1,275.8

Note: Time is measured in seconds.

for CPLEX to solve all leaf nodes from phase I, 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 two, 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.

Table 7 shows the results on two other open binary-ternary covering design problems, which are minimization instances. 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 12,392 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 491,792 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 491,792 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.

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 Equation (7), we obtain a set-covering inequality

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

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.

Table 7. Results on Two Large CODBT Instances

Instance	Known bounds	Fathom group size	Upper bound	No. of cliques	Phase I time	Phase I no. of nodes	No. of leaves	Phase II computation time	Phase II no. of nodes	Phase II wall time
CODBT43	44–48	16	47.1	41,900	4,332	12,392	5,287	563,863	50,752,135	185,309
CODBT71	42–48	8	47.1	338,955	2,090,430	2,961,786	491,712	242,613	17,465,158	68,668

Note: Time is measured in seconds.

Acknowledgments

This work was started while authors F. Rossi and S. Smriglio were visiting the Wisconsin Institute for Discovery at the University of Wisconsin–Madison. This research was performed using the compute resources and assistance of the University of Wisconsin–Madison Center for High Throughput Computing (CHTC) in the Department of Computer Sciences. The CHTC is supported by the University of Wisconsin–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. The suggestions of two anonymous referees significantly enhanced this paper.

References

Achterberg T, Wunderling R (2013) *Mixed Integer Programming: Analyzing 12 Years of Progress. Facets of Combinatorial Optimization* (Springer, Berlin, Heidelberg), 449–482.

Achterberg T, Bixby R, Gu Z, Rothberg E, Weninger D (2020) Presolve reductions in mixed integer programming. *INFORMS J. Comput.* 32(2).

Atamtürk A, Nemhauser G, Savelsbergh M (2000) Conflict graphs in solving integer programming problems. *Eur. J. Oper. Res.* 121(1):40–55.

Correa RC, Donne DD, Koch I, Marenco J (2018) General cut-generating procedures for the stable set polytope. *Discrete Appl. Math.* 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. *Proc. 22nd Annual ACM-SIAM Symp. Discrete Algorithms* (SODA) (Society for Industrial and Applied Mathematics, Philadelphia), 129–135.

Dezső B, Jüttner A, Kovács P (2011) LEMON – An open source C++ graph template library. *Electronic Notes Theoret. Comput. Sci.* 264(5):23–45.

Feo T, Resende M (1989) A probabilistic heuristic for a computationally difficult set covering problem. *Oper. Res. Lett.* 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. Balinski ML, ed. *Approaches to Integer Programming* (Springer, Berlin, Heidelberg), 72–81.

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

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

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

Johnson DJ, Trick M (1996) *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge, October 11–13, 1993* (American Mathematical Society, Providence, RI).

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

Letchford A, Rossi F, Smriglio S (2020) The stable set problem: Clique and nodal inequalities revisited. *Comput. Oper. Res.* 123:105024.

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

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

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

Margot F (2007) Symmetric ILP: Coloring and small integers. Mixed integer programming. *Discrete Optim.* 4(1):40–62.

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 (Springer, Berlin, Heidelberg), 647–686.

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. *J. Symbolic Comput.* 60:94–112.

Nemhauser G, Savelsbergh M, Sigismondi G (1994) MINTO, a mixed INTeger optimizer. *Oper. Res. Lett.* 15(1):47–58.

Núñez-Ares J, Goos P (2019) Enumeration and multicriteria selection of orthogonal minimally aliased response surface designs. *Technometrics* 62(1):21–32.

Ostrowski J (2009) Symmetry in integer programming. Unpublished PhD thesis, Lehigh University.

Ostrowski J, Linderoth J, Rossi F, Smriglio S (2008) Constraint orbital branching. Lodi A, Panconesi A, Rinaldi G, eds. *Proc. 13th Conf. Integer Programming Combin. Optim.* (IPCO) (Springer, Berlin, Heidelberg), 225–239.

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

Ostrowski J, Linderoth J, Rossi F, Smriglio S (2011b) Solving large Steiner triple covering problems. *Oper. Res. Lett.* 39(2):127–131.

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

Pfetsch ME, Rehn T (2019) A computational comparison of symmetry handling methods for mixed integer programs. *Math. Programming Comput.* 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*, vol. 3709 (Springer, Berlin, Heidelberg), 475–489.

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. Combin. Optim.* 21(4):434–457.

Rehn T, Schürmann A (2010) C++ tools for exploiting polyhedral symmetries. Fukuda K, Hoeven J van der, Joswig M, Takayama N, eds. *Proc. 3rd Internat. Congress Math. Software* (Springer, Berlin, Heidelberg), 295–298.

Rotman J (1994) *An Introduction to the Theory of Groups*, 4th ed. (Springer, New York).

Salvagnin D (2005) A dominance procedure for integer programming. Unpublished 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 Comput.* 17(2–4):323–356.