An Adaptive Prefix-Assignment Technique for Symmetry Reduction

Tommi Junttila, Matti Karppa$^{(\boxtimes)}$, Petteri Kaski, and Jukka Kohonen

Department of Computer Science, Aalto University, Espoo, Finland
{tommi.junttila,matti.karppa,petteri.kaski,jukka.kohonen}@aalto.fi

Abstract. This paper presents a technique for symmetry reduction that adaptively assigns a prefix of variables in a system of constraints so that the generated prefix-assignments are pairwise nonisomorphic under the action of the symmetry group of the system. The technique is based on McKay's canonical extension framework [J. Algorithms 26 (1998), no. 2, 306–324]. Among key features of the technique are (i) adaptability—the prefix sequence can be user-prescribed and truncated for compatibility with the group of symmetries; (ii) parallelisability—prefix-assignments can be processed in parallel independently of each other; (iii) versatility—the method is applicable whenever the group of symmetries can be concisely represented as the automorphism group of a vertex-colored graph; and (iv) implementability—the method can be implemented relying on a canonical labeling map for vertex-colored graphs as the only nontrivial subroutine. To demonstrate the tentative practical applicability of our technique we have prepared a preliminary implementation and report on a limited set of experiments that demonstrate ability to reduce symmetry on hard instances.

1 Introduction

Symmetry Reduction. Systems of constraints often have substantial symmetry. For example, consider the following system of Boolean clauses:

$$(x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_3 \vee \bar{x}_5) \wedge (x_2 \vee \bar{x}_4 \vee \bar{x}_6). \tag{1}$$

The associative and commutative symmetries of disjunction and conjunction induce symmetries between the variables of (1), a fact that can be captured by stating that the group Γ generated by the two permutations $(x_1\ x_2)(x_3\ x_4)(x_5\ x_6)$ and $(x_4\ x_6)$ consists of all permutations of the variables that map (1) to itself. That is, Γ is the *automorphism group* of the system (1), cf. Sect. 2.

Known symmetry in a constraint system is a great asset from the perspective of solving the system, in particular since symmetry enables one to disregard partial solutions that are *isomorphic* to each other under the action of Γ on the space of partial solutions. Techniques for such *isomorph rejection*[1] [42] (alternatively, *symmetry reduction* or *symmetry breaking*) are essentially mandatory if

---

[1] A term introduced by J.D. Swift [42]; cf. Hall and Knuth [20] for a survey on early work on exhaustive computer search and combinatorial analysis.

one desires an exhaustive traversal of the (pairwise nonisomorphic) solutions of a highly symmetric system of constraints, or if the system is otherwise difficult to solve, for example, with many "dead-end" partial solutions compared with the actual number of solutions.

A prerequisite to symmetry reduction is that the symmetries are known. In many cases it is possible to automatically discover and compute these symmetries to enable practical and automatic symmetry reduction. In this context the dominant computational approach for combinatorial systems of constraints is to represent Γ via the automorphism group of a vertex-colored graph that captures the symmetries in the system. Carefully engineered tools for working with symmetries of vertex-colored graphs [15,27,35,37] and permutation group algorithms [10,40] then enable one to perform symmetry reduction. For example, for purposes of symmetry computations we may represent (1) as the following vertex-colored graph:

$$(2)$$
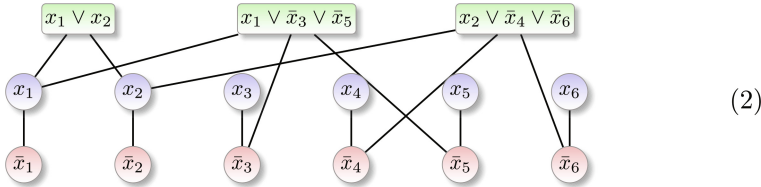
In particular, the graph representation (2) enables us to discover and reduce symmetry to avoid redundant work when solving the underlying system (1).

Our Contribution. The objective of this paper is to document a novel technique for symmetry reduction on systems of constraints. The technique is based on adaptively assigning values to a prefix of the variables so that the obtained prefix-assignments are pairwise nonisomorphic under the action of Γ. The technique can be seen as an instantiation of McKay's [36] influential canonical extension framework for isomorph-free exhaustive generation.

To give a brief outline of the technique, suppose we are working with a system of constraints over a finite set U of variables that take values in a finite set R. Suppose furthermore that $\Gamma \leq \mathrm{Sym}(U)$ is the automorphism group of the system. Select k distinct variables $u_1, u_2, \ldots, u_k$ in U. These k variables form the *prefix sequence* considered by the method. The technique works by assigning values in R to the variables of the prefix, in prefix-sequence order, with u_1 assigned first, then u_2, then u_3, and so forth, so that at each step the partial assignments so obtained are pairwise nonisomorphic under the action of Γ. For example, in (1) the partial assignments $x_1 \mapsto 0$, $x_2 \mapsto 1$ and $x_1 \mapsto 1$, $x_2 \mapsto 0$ are isomorphic since $(x_1\ x_2)(x_3\ x_4)(x_5\ x_6) \in \Gamma$ maps one assignment onto the other; in total there are three nonisomorphic assignments to the prefix x_1, x_2 in (1), namely (i) $x_1 \mapsto 0$, $x_2 \mapsto 0$, (ii) $x_1 \mapsto 0$, $x_2 \mapsto 1$, and (iii) $x_1 \mapsto 1$, $x_2 \mapsto 1$. Each partial assignment that represents an isomorphism class can then be used to reduce redundant work when solving the underlying system by standard techniques—in the nonincremental case, the system is augmented with a symmetry-breaking predicate requiring that one of the nonisomorphic partial assignments holds,

while in the incremental setting [22,43] the partial assignments can be solved independently or even in parallel.

Our contribution in this paper lies in how the isomorph rejection is implemented at the level of isomorphism classes of partial assignments by careful reduction to McKay's [36] isomorph-free exhaustive generation framework. The key technical contribution is that we observe how to generate the partial assignments in a normalized form that enables both *adaptability* (that is, the prefix $u_1, u_2, \ldots, u_k$ can be arbitrarily selected to match the structure of Γ) and precomputation of the extending variable-value orbits along a prefix.

Among further key features of the technique are:

1. *Implementability.* The technique can be implemented by relying on a canonical labeling map for vertex-colored graphs (cf. [27,35,37]) as the only nontrivial subroutine that is invoked once for each partial assignment considered.
2. *Versatility.* The method is applicable whenever the group of symmetries can be concisely represented as a vertex-colored graph; cf. (1) and (2). This is useful in particular when the underlying system has symmetries that are not easily discoverable from the final constraint encoding, for example, due to the fact that the constraints have been compiled or optimized[2] from a higher-level representation in a symmetry-obfuscating manner. A graphical representation can represent such symmetry directly and independently of the compiled/optimized form of the system.
3. *Parallelisability.* As a corollary of implementing McKay's [36] framework, the technique does not need to store representatives of isomorphism classes in memory to perform isomorph rejection, which enables easy parallelisation since the partial assignments can be processed independently of each other.

The main technical contribution of this paper is developed in Sect. 4 where we present the prefix-assignment technique. The required mathematical preliminaries on symmetry and McKay's framework are reviewed in Sects. 2 and 3, respectively. Our development in Sect. 4 relies on an abstract group Γ, with the understanding that a concrete implementation can be designed e.g. in terms of a vertex-colored graph representation, as will be explored in Sect. 5.

Preliminary Implementation and Experiments. To demonstrate the tentative practical applicability of our technique we have prepared a preliminary experimental implementation.[3] The implementation is structured as a preprocessor that works with an explicitly given graph representation and utilizes the *nauty* [35,37] canonical labeling software for vertex-colored graphs as a subroutine to prepare an exhaustive collection of nonisomorphic prefix assignments relative to a user-supplied or heuristically selected prefix of variables. In Sect. 6 we

---

[2] For a beautiful illustration, we refer to Knuth's [31, Sect. 7.1.2, Fig. 10] example of optimum Boolean chains for 5-variable symmetric Boolean functions—from each optimum chain it is far from obvious that the chain represents a symmetric Boolean function. (See also Example 2.)

[3] This implementation can be found at https://github.com/pkaski/reduce/.

report on a limited set of experiments with solving systems of Boolean polynomial equations that demonstrate the ability to reduce symmetry on hard instances and give an initial favorable comparison with earlier techniques.

Earlier Work. A classical way to exploit symmetry in a system of constraints is to augment the system with so-called symmetry-breaking predicates (SBP) that eliminate either some or all symmetric solutions [5,14,19,39]. Such constraints are typically lexicographic leader (lex-leader) constraints that are derived from a generating set for the group of symmetries Γ. Among recent work in this area, Devriendt *et al.* [16] extend the approach by presenting a more compact way for expressing SBPs and a method for detecting "row interchangeabilities". Itzhakov and Codish [26] present a method for finding a set of symmetries whose corresponding lex-leader constraints are enough to completely break symmetries in search problems on small (10-vertex) graphs; this approach is extended by Codish *et al.* [12] by adding pruning predicates that simulate the first iterations of the equitable partition refinement algorithm of *nauty* [35,37]. Heule [23] shows that small complete symmetry-breaking predicates can be computed by considering arbitrary Boolean formulas instead of lex-leader formulas.

Our present technique can be seen as a method for producing symmetry-breaking predicates by augmenting the system of constraints with the disjunction of the nonisomorphic partial assignments. The main difference to the related work above is that our technique does not produce the symmetry-breaking predicate from a set of generators for Γ but rather the predicate is produced recursively, and with the possibility for parallelization, by classifying orbit representatives up to isomorphism using McKay's [36] framework. As such our technique breaks all symmetry with respect to the prescribed prefix, but comes at the cost of additional invocations of graph-automorphism and canonical-labeling tools. This overhead and increased symmetry reduction in particular means that our technique is best suited for constraint systems with hard combinatorial symmetry that is not easily capturable from a set of generators, such as symmetry in combinatorial classification problems [28]. In addition to McKay's [36] canonical extension framework, other standard frameworks for isomorph-free exhaustive generation in this context include *orderly algorithms* due to Faradžev [18] and Read [38], as well as the homomorphism principle for group actions due to Kerber and Laue [30].

It is also possible to break symmetry within a constraint solver during the search by dynamically adding constraints that rule out symmetric parts of the search space (cf. [11,19] and the references therein). If we use the nonisomorphic partial assignments produced by our technique as assumption sequences (cubes) in the incremental cube-and-conquer approach [22,43], our technique can be seen as a restricted way of breaking the symmetries in the beginning of the search, with the benefit—as with cube-and-conquer—that the portions of the search space induced by the partial assignments can be solved in parallel, either with complete independence or with appropriate sharing of information (such as conflict clauses) between the parallel nodes executing the search.

2   Preliminaries on Group Actions and Symmetry

This section reviews relevant mathematical preliminaries and notational conventions for groups, group actions, symmetry, and isomorphism for our subsequent development. (Cf. [10, 17, 25, 28, 29, 40] for further reference.)

Groups and Group Actions. Let Γ be a finite group and let Ω be a finite set (the domain) on which Γ acts. For two groups Λ and Γ, let us write $\Lambda \leq \Gamma$ to indicate that Λ is a subgroup of Γ. We use exponential notation for group actions, and accordingly our groups act from the right. That is, for an object $X \in \Omega$ and $\gamma \in \Gamma$, let us write X^γ for the object in Ω obtained by acting on X with γ. Accordingly, we have $X^{(\beta\gamma)} = (X^\beta)^\gamma$ for all $\beta, \gamma \in \Gamma$ and $X \in \Omega$. For a finite set V, let us write $\mathrm{Sym}(V)$ for the group of all permutations of V with composition of mappings as the group operation.

Suppose that Γ acts on two sets, Ω and Σ. We extend the action to the Cartesian product $\Omega \times \Sigma$ elementwise by defining $(X, S)^\gamma = (X^\gamma, S^\gamma)$ for all $(X, S) \in \Omega \times \Sigma$ and $\gamma \in \Gamma$. Isomorphism extends accordingly; for example, we say that (X, S) and (Y, T) are isomorphic and write $(X, S) \cong (Y, T)$ if there exists a $\gamma \in \Gamma$ with $Y = X^\gamma$ and $T = S^\gamma$. Suppose that Γ acts on a set U. We extend the action of Γ on U to an elementwise action of Γ on subsets $W \subseteq U$ by setting $W^\gamma = \{w^\gamma : w \in W\}$ for all $\gamma \in \Gamma$ and $W \subseteq U$.

Orbit and Stabilizer, Automorphisms, Isomorphism. For an object $X \in \Omega$ let us write $X^\Gamma = \{X^\gamma : \gamma \in \Gamma\}$ for the *orbit* of X under the action of Γ and $\Gamma_X = \{\gamma \in \Gamma : X^\gamma = X\} \leq \Gamma$ for the *stabilizer* subgroup of X in Γ. Equivalently we say that Γ_X is the *automorphism group* of X and write $\mathrm{Aut}(X) = \Gamma_X$ whenever Γ is clear from the context; if we want to stress the acting group we write $\mathrm{Aut}_\Gamma(X)$.

We write $\Omega/\Gamma = \{X^\Gamma : X \in \Omega\}$ for the set of all orbits of Γ on Ω. For $\Lambda \leq \Gamma$ and $\gamma \in \Gamma$, let us write $\Lambda^\gamma = \gamma^{-1}\Lambda\gamma = \{\gamma^{-1}\lambda\gamma : \lambda \in \Lambda\} \leq \Gamma$ for the *γ-conjugate* of Λ. For all $X \in \Omega$ and $\gamma \in \Gamma$ we have $\mathrm{Aut}(X^\gamma) = \mathrm{Aut}(X)^\gamma$. That is, the automorphism groups of objects in an orbit are conjugates of each other.

We say that two objects are *isomorphic* if they are on the same orbit of Γ in Ω. In particular, $X, Y \in \Omega$ are isomorphic if and only if there exists an *isomorphism* $\gamma \in \Gamma$ from X to Y that satisfies $Y = X^\gamma$. An isomorphism from an object to itself is an *automorphism*. Let us write $\mathrm{Iso}(X, Y)$ for the set of all isomorphisms from X to Y. We have that $\mathrm{Iso}(X, Y) = \mathrm{Aut}(X)\gamma = \gamma\mathrm{Aut}(Y)$ where $\gamma \in \mathrm{Iso}(X, Y)$ is arbitrary. Let us write $X \cong Y$ to indicate that X and Y are isomorphic. If we want to stress the group Γ under whose action isomorphism holds, we write $X \cong_\Gamma Y$.

Canonical Labeling and Canonical Form. A function $\kappa : \Omega \to \Gamma$ is a *canonical labeling map* for the action of Γ on Ω if

(K)   for all $X, Y \in \Omega$ it holds that $X \cong Y$ implies $X^{\kappa(X)} = Y^{\kappa(Y)}$
(canonical labeling).

For $X \in \Omega$ we say that $X^{\kappa(X)}$ is the *canonical form* of X in Ω. From (K) it follows that isomorphic objects have identical canonical forms, and the canonical labeling map gives an isomorphism that takes an object to its canonical form.

We assume that the act of computing $\kappa(X)$ for a given X produces as a side-effect a set of generators for the automorphism group $\mathrm{Aut}(X)$.

3   McKay's Canonical Extension Method

This section reviews McKay's [36] canonical extension method for isomorph-free exhaustive generation. Mathematically it will be convenient to present the method so that the isomorphism classes are captured as orbits of a group action of a group Γ, and extension takes place in one step from "seeds" to "objects" being generated, with the understanding that the method can be applied inductively in multiple steps so that the "objects" of the current step become the "seeds" for the next step. We stress that all material in this section is well known (Cf. [28].).

Objects and Seeds. Let Ω be a finite set of *objects* and let Σ be a finite set of *seeds*. Let Γ be a finite group that acts on Ω and Σ. Let κ be a canonical labeling map for the action of Γ on Ω.

Extending Seeds to Objects. Let us connect the objects and the seeds by means of a relation $e \subseteq \Omega \times \Sigma$ that indicates which objects can be built from which seeds by extension. For $X \in \Omega$ and $S \in \Sigma$ we say that X *extends* S and write XeS if $(X, S) \in e$. We assume the relation e satisfies

(E1) e is a union of orbits of Γ, that is, $e^\Gamma = e$ (invariance), and

(E2)   for every object $X \in \Omega$ there exists a seed $S \in \Sigma$ such that XeS (completeness).

For a seed $S \in \Sigma$, let us write $e(S) = \{X \in \Omega : XeS\}$ for the set of all objects that extend S.

Canonical Extension. Next let us associate with each object a particular isomorphism-invariant extension by which we want to extend the object from a seed. A function $M : \Omega \to \Sigma$ is a *canonical extension map* if

(M1)   for all $X \in \Omega$ it holds that $(X, M(X)) \in e$ (extension), and

(M2)   for all $X, Y \in \Omega$ we have that $X \cong Y$ implies $(X, M(X)) \cong (Y, M(Y))$ (canonicity).

That is, (M1) requires that X is in fact an extension of $M(X)$ and (M2) requires that isomorphic objects have isomorphic canonical extensions. In particular, $X \mapsto (X, M(X))$ is a well-defined map from Ω/Γ to e/Γ.

Generating Objects from Seeds. Let us study the following procedure, which is invoked for exactly one representative $S \in \Sigma$ from each orbit in Σ/Γ:

(P)   Let $S \in \Sigma$ be given as input. Iterate over all $X \in e(S)$. Perform zero or more isomorph rejection tests on X and S. If the tests indicate we should accept X, visit X.

Let us first consider the case when there are no isomorph rejection tests. Here and in what follows we indicate with the "†"-symbol that a proof of a claim can be found in the full version of this conference abstract.[4]

Lemma 1 (†). *The procedure (P) visits every isomorphism class of objects in Ω at least once.*

Let us next modify procedure (P) so that any two visits to the same isomorphism class of objects originate from the same procedure invocation. Let $M : \Omega \to \Sigma$ be a canonical extension map. Whenever we construct X by extending S in procedure (P), let us visit X if and only if

(T1) $(X, S) \cong (X, M(X))$.

Lemma 2 (†). *The procedure (P) equipped with the test (T1) visits every isomorphism class of objects in Ω at least once. Furthermore, any two visits to the same isomorphism class must (i) originate by extension from the same procedure invocation on input S, and (ii) belong to the same $\mathrm{Aut}(S)$-orbit of this seed S.*

Let us next observe that the outcome of test (T1) is invariant on each $\mathrm{Aut}(S)$-orbit of extensions of S.

Lemma 3 (†). *For all $\alpha \in \mathrm{Aut}(S)$ we have that (T1) holds for (X, S) if and only if (T1) holds for (X^α, S).*

Lemma 3 in particular implies that we obtain complete isomorph rejection by combining the test (T1) with a further test that ensures complete isomorph rejection on $\mathrm{Aut}(S)$-orbits. Towards this end, let us associate an arbitrary order relation on every $\mathrm{Aut}(S)$-orbit on $e(S)$. Let us perform the following further test:

(T2) $X = \min X^{\mathrm{Aut}(S)}$.

The following lemma is immediate from Lemmas 2 and 3.

Lemma 4. *The procedure (P) equipped with the tests (T1) and (T2) visits every isomorphism class of objects in Ω exactly once.*

A Template for Canonical Extension Maps. We conclude this section by describing a template of how to use an arbitrary canonical labeling map $\kappa : \Omega \to \Gamma$ to construct a canonical extension map $M : \Omega \to \Sigma$.

---

[4] Available at https://arxiv.org/abs/1706.08325.

For $X \in \Omega$ construct the canonical form $Z = X^{\kappa(X)}$. Using the canonical form Z only, identify a seed T with ZeT. In particular, such a seed must exist by (E2). (Typically this identification can be carried out by studying Z and finding an appropriate substructure in Z that qualifies as T. For example, T may be the minimum seed in Σ that satisfies ZeT. Cf. Lemma 9.) Once T has been identified, set $M(X) = T^{\kappa(X)^{-1}}$.

Lemma 5 (†). *The map $X \mapsto M(X)$ above is a canonical extension map.*

4   Generation of Partial Assignments

This section describes an instantiation of McKay's method that generates partial assignments of values to a set of variables U one variable at a time following a *prefix sequence* at the level of isomorphism classes given by the action of a group Γ on U. Let R be a finite set where the variables in U take values.

Partial Assignments, Isomorphism, Restriction. For a subset $W \subseteq U$ of variables, let us say that a *partial assignment* of values to W is a mapping $X : W \to R$. Isomorphism for partial assignments is induced by the following group action.[5] Let $\gamma \in \Gamma$ act on $X : W \to R$ by setting $X^\gamma : W^\gamma \to R$ where X^γ is defined for all $u \in W^\gamma$ by

$$X^\gamma(u) = X(u^{\gamma^{-1}}). \tag{3}$$

Lemma 6 (†). *The action* (3) *is well-defined.*

For an assignment $X : W \to R$, let us write $\underline{X} = W$ for the underlying set of variables assigned by X. Observe that the underline map is a homomorphism of group actions in the sense that $\underline{X^\gamma} = \underline{X}^\gamma$ holds for all $\gamma \in \Gamma$ and $X : W \to R$. For $Q \subseteq \underline{X}$, let us write $X|_Q$ for the restriction of X to Q.

The Prefix Sequence and Generation of Normalized Assignments. We are now ready to describe the generation procedure. Let us begin by prescribing the prefix sequence. Let $u_1, u_2, \ldots, u_k$ be k distinct elements of U and let $U_j = \{u_1, u_2, \ldots, u_j\}$ for $j = 0, 1, \ldots, k$. In particular we observe that

$$U_0 \subseteq U_1 \subseteq \cdots \subseteq U_k$$

with $U_j \setminus U_{j-1} = \{u_j\}$ for all $j = 1, 2, \ldots, k$.

---

[5] For conciseness and accessibility, the present conference version develops the method only for variable symmetries, and accordingly the group action (3) acts only on the variables in U and not on the values in R. However, the method does extend to capture symmetries on both variables in U and values in R (essentially by consideration of the Cartesian product $U \times R$ in place of U), and such an extension will be developed in a full version of this paper.

For $j = 0, 1, \ldots, k$ let Ω_j consist of all partial assignments $X : W \to R$ with $W \cong U_j$. Or what is the same, using the underline notation, Ω_j consists of all partial assignments X with $\underline{X} \cong U_j$.

We rely on canonical extension to construct exactly one object from each orbit of Γ on Ω_j, using as seeds exactly one object from each orbit of Γ on Ω_{j-1}, for each $j = 1, 2, \ldots, k$. We assume the availability of canonical labeling maps $\kappa : \Omega_j \to \Gamma$ for each $j = 1, 2, \ldots, k$.

Our construction procedure will work with objects that are in a normal form to enable precomputation for efficient execution of the subsequent tests for isomorph rejection. Towards this end, let us say that $X \in \Omega_j$ is *normalized* if $\underline{X} = U_j$. It is immediate from our definition of Ω_j and (3) that each orbit in Ω_j / Γ contains at least one normalized object.

Let us begin with a high-level description of the construction procedure, to be followed by the details of the isomorph rejection tests and a proof of correctness. Fix $j = 1, 2, \ldots, k$ and study the following procedure, which we assume is invoked for exactly one normalized representative $S \in \Omega_{j-1}$ from each orbit in Ω_{j-1} / Γ:

(P') Let a normalized $S \in \Omega_{j-1}$ be given as input. For each $p \in u_j^{\mathrm{Aut}(U_{j-1})}$ and each $r \in R$, construct the assignment

$$X : U_{j-1} \cup \{p\} \to R$$

defined by $X(p) = r$ and $X(u) = S(u)$ for all $u \in U_{j-1}$. Perform the isomorph rejection tests (T1') and (T2') on X and S. If both tests accept, visit $X^{\nu(p)}$ where $\nu(p) \in \mathrm{Aut}(U_{j-1})$ normalizes X.

From an implementation perspective it is convenient to precompute the orbit $u_j^{\mathrm{Aut}(U_{j-1})}$ together with group elements $\nu(p) \in \mathrm{Aut}(U_{j-1})$ for each $p \in u_j^{\mathrm{Aut}(U_{j-1})}$ that satisfy $p^{\nu(p)} = u_j$. Indeed, a constructed X with $\underline{X} = U_{j-1} \cup \{p\}$ can now be normalized by acting with $\nu(p)$ on X to obtain a normalized $X^{\nu(p)}$ isomorphic to X.

The Isomorph Rejection Tests. Let us now complete the description of procedure (P') by describing the two isomorph rejection tests (T1') and (T2'). This paragraph only describes the tests with an implementation in mind, the correctness analysis is postponed to the following paragraph.

Let us assume that the elements of U have been arbitrarily ordered and that $\kappa : \Omega_j \to \Gamma$ is a canonical labeling map. Suppose that X has been constructed by extending a normalized S with $\underline{X} = \underline{S} \cup \{p\} = U_{j-1} \cup \{p\}$. The first test is:

(T1') Subject to the ordering of U, select the minimum $q \in U$ such that $q^{\kappa(X)^{-1}\nu(p)} \in u_j^{\mathrm{Aut}(U_j)}$. Accept if and only if $p \cong_{\mathrm{Aut}(X)} q^{\kappa(X)^{-1}}$.

From an implementation perspective we observe that we can precompute the orbit $u_j^{\mathrm{Aut}(U_j)}$. Furthermore, the only computationally nontrivial part of the test is the computation of $\kappa(X)$ since we assume that we obtain generators for

$\mathrm{Aut}(X)$ as a side-effect of this computation. Indeed, with generators for $\mathrm{Aut}(X)$ available, it is easy to compute the orbits $U/\mathrm{Aut}(X)$ and hence to test whether $p \cong_{\mathrm{Aut}(X)} q^{\kappa(X)^{-1}}$. Let us now describe the second test:

(T2')   Accept if and only if $p = \min p^{\mathrm{Aut}(S)}$ subject to the ordering of U.

From an implementation perspective we observe that since S is normalized we have $\mathrm{Aut}(S) \leq \mathrm{Aut}(\underline{S}) = \mathrm{Aut}(U_{j-1})$ and thus the orbit $u_j^{\mathrm{Aut}(U_{j-1})}$ considered by procedure (P') partitions into one or more $\mathrm{Aut}(S)$-orbits. Furthermore, generators for $\mathrm{Aut}(S)$ are readily available (due to S itself getting accepted in the test (T1') at an earlier level of recursion), and thus the orbits $u_j^{\mathrm{Aut}(U_{j-1})}/\mathrm{Aut}(S)$ and their minimum elements are cheap to compute. Thus, a fast implementation of procedure (P') will in most cases execute the test (T2') before the more expensive test (T1').

Correctness. This section establishes the correctness of procedure (P') together with the tests (T1') and (T2') by reduction to McKay's framework and Lemma 4. Fix $j = 1, 2, \ldots, k$. Let us start by defining the extension relation $e \subseteq \Omega_j \times \Omega_{j-1}$ for all $X \in \Omega_j$ and $S \in \Omega_{j-1}$ by setting XeS if and only if

there exists a $\gamma \in \Gamma$ such that $\underline{X}^\gamma = U_j, \underline{S}^\gamma = U_{j-1}$, and $X^\gamma|_{U_{j-1}} = S^\gamma.$ (4)

This relation is well-defined in the context of McKay's framework:

Lemma 7 (†). *The relation* (4) *satisfies (E1) and (E2).*

The following lemma establishes that the iteration in procedure (P') constructs exactly the objects $X \in e(S)$; cf. procedure (P).

Lemma 8 (†). *Let $S \in \Omega_{j-1}$ be normalized. For all $X \in \Omega_j$ we have XeS if and only if there exists a $p \in u_j^{\mathrm{Aut}(U_{j-1})}$ with $\underline{X} = U_{j-1} \cup \{p\}$ and $X|_{U_{j-1}} = S$.*

Next we show the correctness of the test (T1') by establishing that it is equivalent with the test (T1) for a specific canonical extension function M. Towards this end, let us use the assumed canonical labeling map $\kappa : \Omega_j \to \Gamma$ to build a canonical extension function M using the template of Lemma 5. In particular, given an $X \in \Omega_j$ as input with $\underline{X} = U_{j-1} \cup \{p\}$, first construct the canonical form $Z = X^{\kappa(X)}$. In accordance with (T1'), select the minimum $q \in U$ such that $q^{\kappa(X)^{-1}} \nu(p) \in u_j^{\mathrm{Aut}(U_j)}$. Now construct $M(X)$ from X by deleting the value of $q^{\kappa(X)^{-1}}$.

Lemma 9 (†). *The mapping $X \mapsto M(X)$ is well-defined and satisfies both (M1) and (M2).*

To complete the equivalence between (T1') and (T1), observe that since X and p determine S by $X|_{\underline{X}\setminus\{p\}} = S$, and similarly X and $q^{\kappa(X)^{-1}}$ determine $M(X)$ by $X|_{\underline{X}\setminus\{q^{\kappa(X)^{-1}}\}} = M(X)$, the test (T1) is equivalent to testing whether

$(X, p) \cong (X, q^{\kappa(X)^{-1}})$ holds, that is, whether $p \cong_{\mathrm{Aut}(X)} q^{\kappa(X)^{-1}}$ holds. Observe that this is exactly the test (T1').

It remains to establish the equivalence of (T2') and (T2). We start with a lemma that captures the $\mathrm{Aut}(S)$-orbits considered by (T2).

Lemma 10 (†). *For a normalized $S \in \Omega_{j-1}$ the orbits in $e(S)/\mathrm{Aut}(S)$ are in a one-to-one correspondence with the elements of $(u_j^{\mathrm{Aut}(U_{j-1})}/\mathrm{Aut}(S)) \times R$.*

Now order the elements $X \in e(S)$ based on the lexicographic ordering of the pairs $(p, X(p)) \in u_j^{\mathrm{Aut}(U_{j-1})} \times R$. Since the action (3) fixes the values in R elementwise, we have that (T2') holds if and only if (T2) holds for this ordering of $e(S)$. The correctness of procedure (P') equipped with the tests (T1') and (T2') now follows from Lemma 4.

Selecting a Prefix. This section gives a brief discussion on how to select the prefix. Let $U_k = \{u_1, u_2, \ldots, u_k\}$ be the set of variables in the prefix sequence. It is immediate that there exist $|R|^k$ distinct partial assignments from U_k to R. Let us write R^{U_k} for the set of these assignments. The group Γ now partitions R^{U_k} into orbits via the action (3), and it suffices to consider at most one representative from each orbit to obtain an exhaustive traversal of the search space, up to isomorphism. Our goal is thus to select the prefix U_k so that the setwise stabilizer Γ_{U_k} has comparatively few orbits on R^{U_k} compared with the total number of such assignments. In particular, the ratio of the number of orbits $|R^{U_k}/\Gamma_{U_k}|$ to the total number of mappings $|R|^k$ can be viewed as a proxy for the achieved symmetry reduction and as a rough[6] proxy for the speedup factor obtained compared with no symmetry reduction at all.

Subroutines. By our assumption, the canonical labeling map κ produces as a side-effect a set of generators for the automorphism group $\mathrm{Aut}(X)$ for a given input X. We also assume that generators for the groups $\mathrm{Aut}(U_j)$ for $j = 0, 1, \ldots, k$ can be precomputed by similar means. This makes the canonical labeling map essentially the only nontrivial subroutine needed to implement procedure (P'). Indeed, the orbit computations required by tests (T1') and (T2') are implementable by elementary permutation group algorithms [10, 40]. The next section describes how to implement κ by reduction to vertex-colored graphs.[7]

---

[6] Here it should be noted that executing the symmetry reduction carries in itself a nontrivial computational cost. That is, there is a tradeoff between the potential savings in solving the system gained by symmetry reduction versus the cost of performing symmetry reduction. For example, if the instance has no symmetry and Γ is a trivial group, then symmetry reduction merely makes it more costly to solve the system.

[7] Reduction to vertex-colored graphs is by no means the only possibility to obtain the canonical labeling map to enable (P'), (T1'), and (T2'). Another possibility would be to represent Γ directly as a permutation group and use dedicated permutation-group algorithms [33, 34]. Our present choice of vertex-colored graphs is motivated by easy availability of carefully engineered implementations for working with vertex-colored graphs.

5   Representation Using Vertex-Colored Graphs

This section describes one possible approach to represent the group of symmetries $\Gamma \leq \mathrm{Sym}(U)$ of a system of constraints over a finite set of variables U taking values in a finite set R. Our representation of choice will be vertex-colored graphs over a fixed finite set of vertices V. In particular, isomorphisms between such graphs are permutations $\gamma \in \mathrm{Sym}(V)$ that map edges onto edges and respect the colors of the vertices; that is, every vertex in V maps to a vertex of the same color under γ. It will be convenient to develop the relevant graph representations in steps, starting with the representation of the constraint system and then proceeding to the representation of setwise stabilizers and partial assignments. These representations are folklore (see e.g. [28]) and are presented here for completeness of exposition only.

Representing the Constraint System. To capture $\Gamma \cong \mathrm{Aut}(G)$ via a vertex-colored graph G with vertex set V, it is convenient to represent the variables U directly as a subset of vertices $U \subseteq V$ such that no vertex in $V \setminus U$ has a color that agrees with a color of a vertex in U. We then seek a graph G such that $\mathrm{Aut}(G) \leq \mathrm{Sym}(U) \times \mathrm{Sym}(V \setminus U)$ projected to U is exactly Γ. In most cases such a graph G is concisely obtainable by encoding the system of constraints with additional vertices and edges joined to the vertices representing the variables in U. We discuss two examples.

Example 1. Consider the system of clauses (1) and its graph representation (2). The latter can be obtained as follows. First, introduce a blue vertex for each of the six variables of (1). These blue vertices constitute the subset U. Then, to accommodate negative literals, introduce a red vertex joined by an edge to the corresponding blue vertex representing the positive literal. These edges between red and blue vertices ensure that positive and negative literals remain consistent under isomorphism. Finally, introduce a green vertex for each clause of (1) with edges joining the clause with each of its literals. It is immediate that we can reconstruct (1) from (2) up to labeling of the variables even after arbitrary color-preserving permutation of the vertices of (2). Thus, (2) represents the symmetries of (1).

Let us next discuss an example where it is convenient to represent the symmetry at the level of original constraints rather than at the level of clauses.

Example 2. Consider the following system of eight cubic equations over 24 variables taking values modulo 2:

$$x_{11}y_{11}z_{11} + x_{12}y_{12}z_{12} + x_{13}y_{13}z_{13} = 0 \qquad x_{21}y_{11}z_{11} + x_{22}y_{12}z_{12} + x_{23}y_{13}z_{13} = 0$$
$$x_{11}y_{11}z_{21} + x_{12}y_{12}z_{22} + x_{13}y_{13}z_{23} = 0 \qquad x_{21}y_{11}z_{21} + x_{22}y_{12}z_{22} + x_{23}y_{13}z_{23} = 1$$
$$x_{11}y_{21}z_{11} + x_{12}y_{22}z_{12} + x_{13}y_{23}z_{13} = 1 \qquad x_{21}y_{21}z_{11} + x_{22}y_{22}z_{12} + x_{23}y_{23}z_{13} = 1$$
$$x_{11}y_{21}z_{21} + x_{12}y_{22}z_{22} + x_{13}y_{23}z_{23} = 1 \qquad x_{21}y_{21}z_{21} + x_{22}y_{22}z_{22} + x_{23}y_{23}z_{23} = 1$$

This system seeks to decompose a $2 \times 2 \times 2$ tensor (whose elements appear on the right hand sides of the equations) into a sum of three rank-one tensors.

The symmetries of addition and multiplication modulo 2 imply that the symmetries of the system can be represented by the following vertex-colored graph:
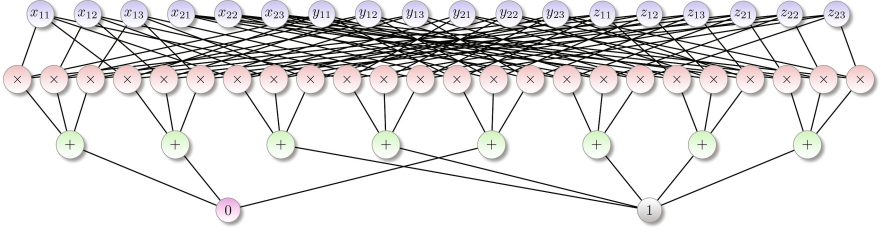
Indeed, we encode each monomial in the system with a product-vertex, and group these product-vertices together by adjacency to a sum-vertex to represent each equation, taking care to introduce two uniquely colored constant-vertices to represent the right-hand side of each equation.

The representation built directly from the system of polynomial equations in Example 2 concisely captures the symmetries in the system independently of the final encoding of the system (e.g. as CNF) for solving purposes. In particular, building the graph representation from such a final CNF encoding (cf. Example 1) results in a less compact graph representation and obfuscates the symmetries of the original system, implying less efficient symmetry reduction.

Representing the Values. In what follows it will be convenient to assume that the graph G contains a uniquely colored vertex for each value in R. (Cf. the graph in Example 2.) That is, we assume that $R \subseteq V \setminus U$ and that $\mathrm{Aut}(G)$ projected to R is the trivial group.

Representing Setwise Stabilizers in the Prefix Chain. To enable procedure (P') and the tests (T1') and (T2'), we require generators for $\mathrm{Aut}(U_j) \leq \Gamma$ for each $j = 0, 1, \ldots, k$. More generally, given a subset $W \subseteq U$, we seek to compute a set of generators for the setwise stabilizer $\mathrm{Aut}_\Gamma(W) = \Gamma_W = \{\gamma \in \Gamma : W^\gamma = W\}$, with $W^\gamma = \{w^\gamma : w \in W\}$. Assuming we have available a vertex-colored graph G that represents Γ by projection of $\mathrm{Aut}_{\mathrm{Sym}(V)}(G)$ to U, let us define the graph $G \upharpoonright W$ by selecting one vertex $r \in R$ and joining each vertex $w \in W$ with an edge to the vertex r. It is immediate that $\mathrm{Aut}_{\mathrm{Sym}(V)}(G \upharpoonright W)$ projected to U is precisely $\mathrm{Aut}_\Gamma(W)$.

Representing Partial Assignments. Let $X : W \to R$ be an assignment of values in R to variables in $W \subseteq U$. Again to enable procedure (P') together with the tests (T1') and (T2'), we require a canonical labeling $\kappa(X)$ and generators for the automorphism group $\mathrm{Aut}(X)$. Again assuming we have a vertex-colored graph G that represents Γ, let us define the graph $G \upharpoonright X$ by joining each vertex $w \in W$ with an edge to the vertex $X(w) \in R$. It is immediate that $\mathrm{Aut}_{\mathrm{Sym}(V)}(G \upharpoonright X)$ projected to U is precisely $\mathrm{Aut}_\Gamma(X)$. Furthermore, a canonical labeling $\kappa(X)$ can be recovered from $\kappa(G \upharpoonright X)$ and the canonical form $(G \upharpoonright X)^{\kappa(G \upharpoonright X)}$.

Using Tools for Vertex-Colored Graphs. Given a vertex-colored graph G as input, practical tools exist for computing a canonical labeling $\kappa(G) \in \mathrm{Sym}(V)$ and a set of generators for $\mathrm{Aut}(G) \leq \mathrm{Sym}(V)$. Such tools include *bliss* [27], *nauty* [35, 37], and *traces* [37]. Once the canonical labeling and generators are available in $\mathrm{Sym}(V)$ it is easy to map back to Γ by projection to U so that corresponding elements of Γ are obtained.

6   Preliminary Experimental Evaluation

This section documents a preliminary and limited experimental evaluation of an implementation of the adaptive prefix-assignment technique. The implementation is written in C and structured as a preprocessor that works with an explicitly given graph representation and utilizes the *nauty* [35, 37] canonical labeling software for vertex-colored graphs as a subroutine.

As instances we study systems of polynomial equations aimed at discovering the tensor rank of a small $m \times m \times m$ tensor $T = (t_{ijk})$ modulo 2, with $t_{ijk} \in \{0, 1\}$ and $i, j, k = 1, 2, \ldots m$. Computing the rank of a given tensor is NP-hard [21].[8] In precise terms, we seek to find the minimum r such that there exist three $m \times r$ matrices $A, B, C \in \{0, 1\}^{m \times r}$ such that for all $i, j, k = 1, 2, \ldots, m$ we have

$$\sum_{\ell=1}^{r} a_{i\ell} b_{j\ell} c_{k\ell} = t_{ijk} \pmod{2}. \tag{5}$$

Such instances are easily compilable into CNF with A, B, C constituting three matrices of Boolean variables so that the task becomes to find the minimum r such that the compiled CNF instance is satisfiable. Independently of the target tensor T such instances have a symmetry group of order at least $r!$ due to the fact that the columns of the matrices A, B, C can be arbitrarily permuted so that (5) maps to itself. In our experiments we select the entries of T uniformly at random so that the number of 1s in T is exactly n.

Hardware and Software Configuration. The experiments were performed on a cluster of Dell PowerEdge C4130 compute nodes, each equipped with two Intel Xeon E5-2680v3 CPUs and 128 GiB of main memory, running Linux version `3.10.0514.10.2.el7.x86_64`. All experiments were executed by allocating a single core on a single CPU of a compute node. Other unrelated compute load was in general present on the nodes during the experiments. A time-out of five hours of CPU time was applied.

Symmetry Reduction Tools and SAT Solvers. We report on three methods for symmetry reduction on aforementioned tensor-rank instances: (1) no reduction

---

[8] Yet considerable interest exists to determine tensor ranks of small tensors, in particular tensors that encode and enable fast matrix multiplication algorithms; cf. [1–4, 8, 9, 13, 24, 32, 41, 44].

("raw"), (2) `breakid` version 2.2 [16], (3) our technique ("`reduce`") with a user-selected prefix consisting of $2r$ variables that constitute the first two rows of the matrix A, and a graph representation of (5) as in Example 2. Three different SAT solvers were used in the experiments: `lingeling` and `ilingeling` version `bbc-9230380` [7], and `glucose` version 4.1 [6]. We use the incremental solver `ilingeling` together with the incremental CNF output of `reduce`.

Results. Table 1 shows the results of a tensor rank computation modulo 2 for two random tensors T with $m = 5$, $n = 16$ and $m = 5$, $n = 18$. In both cases we observe that the rank is 9. The running times displayed in the table are in seconds, with "t/o" indicating a time-out. For both tensors we observe decreased running time due to symmetry reduction, with `reduce` performing better than `breakid`, but also taking more time to perform the preprocessing (indicated in columns labelled with "prep.") due to repeated calls to canonical labeling.

We would like to highlight that the comparison between `reduce` and `breakid` in particular illustrates the relevance of a graph representation of the symmetries in (5), which are not easily discoverable from the compiled CNF. In our experiments `reduce` receives as input the graph representation of the system (5) constructed as in Example 2, whereas `breakid` works on the compiled CNF input alone. This demonstrates the serendipity and versatility of using an auxiliary graph to represent the symmetries so that the symmetries are easily accessible. In fact, if we rely on the standard representation (2) built from the compiled CNF, the standard representation does *not* represent all the symmetries in (5).

Table 1. Computing tensor rank for two $5 \times 5 \times 5$ tensors

m	r	n	raw		prep. breakid	breakid		prep. reduce	reduce			Sat?
			glucose	lingeling		glucose	lingeling		glucose	lingeling	ilingeling	
5	6	16	60.02	114.14	0.06	36.01	81.86	0.29	3.70	9.67	7.28	UNSAT
5	7	16	t/o	t/o	0.19	t/o	t/o	1.07	372.43	614.04	249.10	UNSAT
5	8	16	t/o	t/o	0.12	t/o	t/o	2.56	t/o	t/o	15316.01	UNSAT
5	9	16	3.20	23.78	0.07	1.20	158.44	5.03	0.72	149.82	1.16	SAT
5	10	16	0.99	12.56	0.07	0.69	21.49	9.06	30.64	166.37	0.80	SAT
5	11	16	0.16	33.17	0.28	0.09	52.71	16.24	1.37	6.36	1.48	SAT
5	6	18	14.37	31.34	0.23	11.05	23.96	0.53	2.79	2.90	4.25	UNSAT
5	7	18	4606.30	t/o	0.18	3604.87	t/o	1.01	114.77	116.80	94.89	UNSAT
5	8	18	t/o	t/o	0.06	t/o	t/o	3.23	6901.79	t/o	3867.00	UNSAT
5	9	18	395.75	444.84	0.06	2.68	209.66	5.93	40.32	39.21	1.32	SAT
5	10	18	13.05	22.03	0.07	3.79	356.18	10.46	7.75	11.67	1.45	SAT
5	11	18	19.47	43.68	0.22	27.29	859.40	17.11	31.86	112.62	1.42	SAT

References

1. Alekseev, V.B.: On bilinear complexity of multiplication of 5×2 matrix by 2×2 matrix. Physics and mathematics, Uchenye Zapiski Kazanskogo Universiteta. Seriya Fiziko-Matematicheskie Nauki, vol. 156, pp. 19–29. Kazan University, Kazan (2014)

2. Alekseev, V.B.: On bilinear complexity of multiplication of $m \times 2$ and 2×2 matrices. Chebyshevskii Sb. **16**(4), 11–27 (2015)

3. Alekseev, V.B., Smirnov, A.V.: On the exact and approximate bilinear complexities of multiplication of 4×2 and 2×2 matrices. Proc. Steklov Inst. Math. **282**(1), 123–139 (2013)

4. Alekseyev, V.B.: On the complexity of some algorithms of matrix multiplication. J. Algorithms **6**(1), 71–85 (1985)

5. Aloul, F.A., Sakallah, K.A., Markov, I.L.: Efficient symmetry breaking for boolean satisfiability. In: Proceedings of the IJCAI 2003, pp. 271–276. Morgan Kaufmann (2003)

6. Audemard, G., Simon, L.: Extreme cases in SAT problems. In: Creignou, N., Le Berre, D. (eds.) SAT 2016. LNCS, vol. 9710, pp. 87–103. Springer, Cham (2016). doi:10.1007/978-3-319-40970-2_7

7. Biere, A.: Splatz, Lingeling, Plingeling, Treengeling, YalSAT entering the SAT Competition 2016. In: Proceedings of SAT Competition 2016 - Solver and Benchmark Descriptions. Department of Computer Science Series of Publications B, vol. B-2016-1, pp. 44–45. University of Helsinki (2016)

8. Bläser, M.: Lower bounds for the multiplicative complexity of matrix multiplication. Comput. Complex. **8**(3), 203–226 (1999)

9. Bläser, M.: On the complexity of the multiplication of matrices of small formats. J. Complex. **19**(1), 43–60 (2003)

10. Butler, G. (ed.): Fundamental Algorithms for Permutation Groups. LNCS, vol. 559. Springer, Heidelberg (1991)

11. Chu, G., de la Banda, M.G., Mears, C., Stuckey, P.J.: Symmetries, almost symmetries, and lazy clause generation. Constraints **19**(4), 434–462 (2014)

12. Codish, M., Gange, G., Itzhakov, A., Stuckey, P.J.: Breaking symmetries in graphs: the nauty way. In: Rueher, M. (ed.) CP 2016. LNCS, vol. 9892, pp. 157–172. Springer, Cham (2016). doi:10.1007/978-3-319-44953-1_11

13. Courtois, N.T., Hulme, D., Mourouzis, T.: Multiplicative complexity and solving generalized Brent equations with SAT solvers. In: Proceedings of the Third International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking (COMPUTATION TOOLS 2012), pp. 22–27 (2012)

14. Crawford, J.M., Ginsberg, M.L., Luks, E.M., Roy, A.: Symmetry-breaking predicates for search problems. In: Proceedings of the KR 1996, pp. 148–159. Morgan Kaufmann (1996)

15. Darga, P.T., Liffiton, M.H., Sakallah, K.A., Markov, I.L.: Exploiting structure in symmetry detection for CNF. In: Proceedings of the DAC 2004, pp. 530–534. ACM (2004)

16. Devriendt, J., Bogaerts, B., Bruynooghe, M., Denecker, M.: Improved static symmetry breaking for SAT. In: Creignou, N., Le Berre, D. (eds.) SAT 2016. LNCS, vol. 9710, pp. 104–122. Springer, Cham (2016). doi:10.1007/978-3-319-40970-2_8

17. Dixon, J.D., Mortimer, B.: Permutation Groups. Graduate Texts in Mathematics, vol. 163. Springer, New York (1996)

18. Faradžev, I.A.: Constructive enumeration of combinatorial objects. In: Problèmes Combinatoires et Théorie des Graphes, pp. 131–135. No. 260 in Colloq. Internat. CNRS, CNRS, Paris (1978)
19. Gent, I.P., Petrie, K.E., Puget, J.: Symmetry in constraint programming. In: Handbook of Constraint Programming. Foundations of Artificial Intelligence, vol. 2, pp. 329–376. Elsevier (2006)
20. Hall Jr., M., Knuth, D.E.: Combinatorial analysis and computers. Amer. Math. Monthly **72**(2, Part 2), 21–28 (1965)
21. Håstad, J.: Tensor rank is NP-complete. J. Algorithms **11**(4), 644–654 (1990)
22. Heule, M.J.H., Kullmann, O., Wieringa, S., Biere, A.: Cube and conquer: guiding CDCL SAT solvers by lookaheads. In: Eder, K., Lourenço, J., Shehory, O. (eds.) HVC 2011. LNCS, vol. 7261, pp. 50–65. Springer, Heidelberg (2012). doi:10.1007/978-3-642-34188-5_8
23. Heule, M.J.H.: The quest for perfect and compact symmetry breaking for graph problems. In: Proceedings of the SYNASC 2016, pp. 149–156. IEEE Computer Society (2016)
24. Hopcroft, J.E., Kerr, L.R.: On minimizing the number of multiplications necessary for matrix multiplication. SIAM J. Appl. Math. **20**(1), 30–36 (1971)
25. Humphreys, J.F.: A Course in Group Theory. Oxford University Press, Oxford (1996)
26. Itzhakov, A., Codish, M.: Breaking symmetries in graph search with canonizing sets. Constraints **21**(3), 357–374 (2016)
27. Junttila, T., Kaski, P.: Engineering an efficient canonical labeling tool for large and sparse graphs. In: Proceedings of the ALENEX 2007. SIAM (2007)
28. Kaski, P., Östergård, P.R.J.: Classification Algorithms for Codes and Designs. Algorithms and Computation in Mathematics, vol. 15. Springer, Heidelberg (2006)
29. Kerber, A.: Applied Finite Group Actions. Algorithms and Combinatorics, vol. 19, 2nd edn. Springer, Heidelberg (1999)
30. Kerber, A., Laue, R.: Group actions, double cosets, and homomorphisms: unifying concepts for the constructive theory of discrete structures. Acta Appl. Math. **52**(1–3), 63–90 (1998)
31. Knuth, D.E.: The Art of Computer Programming. Combinatorial Algorithms, vol. 4A, Part 1. Addison-Wesley, Reading (2011)
32. Laderman, J.D.: A noncommutative algorithm for multiplying 3×3 matrices using 23 multiplications. Bull. Amer. Math. Soc. **82**, 126–128 (1976)
33. Leon, J.S.: Permutation group algorithms based on partitions, I: theory and algorithms. J. Symbol. Comput. **12**(4–5), 533–583 (1991)
34. Leon, J.S.: Partitions, refinements, and permutation group computation. In: Groups and Computation, II, pp. 123–158. No. 28 in DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society (1997)
35. McKay, B.D.: Practical graph isomorphism. Congressus Numerantium **30**, 45–87 (1981)
36. McKay, B.D.: Isomorph-free exhaustive generation. J. Algorithms **26**(2), 306–324 (1998)
37. McKay, B.D., Piperno, A.: Practical graph isomorphism. II. J. Symb. Comput. **60**, 94–112 (2014)
38. Read, R.C.: Every one a winner; or, how to avoid isomorphism search when cataloguing combinatorial configurations. Ann. Discrete Math. **2**, 107–120 (1978)

39. Sakallah, K.A.: Symmetry and satisfiability. In: Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 289–338. IOS Press (2009)
40. Seress, Á.: Permutation Group Algorithms. Cambridge University Press, Cambridge (2003)
41. Strassen, V.: Gaussian elimination is not optimal. Numer. Math. **13**(4), 354–356 (1969)
42. Swift, J.D.: Isomorph rejection in exhaustive search techniques. In: Combinatorial Analysis, pp. 195–200. American Mathematical Society (1960)
43. Wieringa, S.: The icnf file format (2011). http://www.siert.nl/icnf/. Accessed April 2017
44. Winograd, S.: On multiplication of 2×2 matrices. Linear Algebra Appl. **4**(4), 381–388 (1971)