Breaking Symmetries with Involutions

Michael Codish
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Mikoláš Janota
CIIRC, Czech Technical University in Prague, Czech Republic

Abstract

Symmetry breaking for graphs and other combinatorial objects is notoriously hard. On the one hand, complete symmetry breaks are exponential in size. On the other hand, current, state-of-the-art, partial symmetry breaks are often considered too weak to be of practical use. Recently, the concept of graph patterns has been introduced and provides a concise representation for (large) sets of non-canonical graphs, i.e. graphs that are not lex-leaders and can be excluded from search. In particular, four (specific) graph patterns apply to identify about 3/4 of the set of all non-canonical graphs. Taking this approach further, we discover that graph patterns that derive from permutations that are involutions play an important role in the construction of symmetry breaks for graphs. We take advantage of this to guide the construction of partial and complete symmetry-breaking constraints based on graph patterns. The resulting constraints are small in size and strong in the number of symmetries they break.

2012 ACM Subject Classification Computing methodologies; Theory of computation → Constraint and logic programming

Keywords and phrases graph symmetry, patterns, permutation, Ramsey graphs, greedy, CEGAR

Digital Object Identifier 10.4230/LIPIcs.CP.2025.8

Funding The results were supported by the Ministry of Education, Youth and Sports within the dedicated program ERC CZ under the project *POSTMAN* no. LL1902 and co-funded by the European Union under the project *ROBOPROX* (reg. no. CZ.02.01.01/00/22_008/0004590).

Acknowledgements The authors would like to thank the anonymous reviewers for their valuable comments and constructive feedback, which helped improve the quality and clarity of this paper. This work began at the *Dagstuhl-Seminar 23261 SAT Encodings and Beyond*.

1 Introduction

Graph search problems are about finding simple graphs with desired structural properties. Such problems arise in many real-world applications and are fundamental in graph theory. Solving graph search problems is typically hard due to the enormous search space and the large number of symmetries in graph representation: Any graph obtained by permuting the vertices of a solution (or a non-solution) is also a solution (or a non-solution), and is considered isomorphic, or "symmetric". The set of all such isomorphic graphs forms an isomorphism class. To optimize search, we aim to restrict it to focus on one "canonical" graph from each isomorphism class.

One common approach to eliminate symmetries is to add *symmetry breaking constraints* that are satisfied by at least one member of each isomorphism class [9, 24, 26]. A symmetry-breaking constraint is called *complete* if it is satisfied by exactly one member of each isomorphism class and *partial* otherwise. In many cases, symmetry-breaking constraints, complete or partial, are expressed in terms of "lex-constraints". Each lex-constraint corresponds to one symmetry, σ, which is a permutation on vertices, and restricts the search space to consider assignments that are lexicographically smaller than their permuted form obtained

according to σ. If one considers the set of lex-constraints corresponding to all n-factorial permutations, then the corresponding symmetry break is complete but too large to be of practical use.

Itzhakov and Codish [14] observe that a complete symmetry break can be defined in terms of a number of lex-constraints which is considerably smaller than n-factorial. They succeed in computing complete symmetry-breaking constraints of practical size for graphs with 10 or less vertices. Dančo et al. obtain similar results in the context of finite models [10]. But this approach does not scale. Itzhakov *et al.* [4] introduce the notion of "lex-implications", which are a refinement of lex-constraints. In the case of graphs, they compute complete symmetry breaks which are much smaller in size. But still, this approach does not scale beyond graphs with 11 vertices. It is known that breaking symmetry by adding constraints to eliminate symmetric solutions is intractable in general [1, 9]. So, we do not expect to find a complete symmetry break of polynomial size that identifies canonical graphs that are lex-leaders. Hence, the interest in partial symmetry breaks.

Codish *et al.* [7] introduce a partial symmetry break, which is equivalent to considering the quadratic number of permutations that swap a pair of vertices (transpositions). Rintanen *et al.* [23] enhance this approach for directed graphs. This approach is widely applied and turns out to work well in practice, despite eliminating only a small portion of the symmetries. However, when dealing with hard instances of graph search problems, this constraint does not suffice. Over the past decade, there has been little progress in the research of partial symmetry-breaking constraints for graph search problems. Some attempts are made in [15] and in [5]. However, there are no references in the literature to the applications that make use of the symmetry breaks defined in these papers.

In a recent paper, Codish and Janota [6] introduce a set-covering perspective on symmetry breaking. A permutation "covers" a graph if its application on the graph yields a smaller graph with respect to a given (lexicographic) order. A complete symmetry break is then a set of permutations that covers all non-canonical graphs. In that paper, the authors introduce the notion of a "pattern" which provides a concise representation for the set of graphs covered by a permutation.

In this paper, we pick up on the notion of "patterns" which we call here "graph patterns" and show how they can be used to provide complete and also partial symmetry breaks which improve on the current state-of-the-art. Motivated by the set-covering perspective presented in [6], we implement a greedy algorithm to cover the set of all non-canonical graphs using graph patterns. Although this approach does not scale, it provides a test bed to study the structure of the graph patterns selected in the greedy approach. We observe that the first four graph patterns selected in the greedy approach already cover 3/4 of the set of all graphs (and slightly more of the set of all non-canonical graphs). Our study also reveals the importance of a specific type of permutation when breaking symmetries on graphs. This is the class of permutations that are equal to their inverse, also called *involutions*. This class includes the classes of transpositions (which swap a pair of indices) and of consecutive transpositions (which swap a pair of consecutive vertices) that are at the basis of the partial symmetry breaks introduced in [8, 7].

We implement an algorithm that guides the search for a complete symmetry break based on the structure of graph patterns. Along the way, we obtain a chain of partial symmetry breaks of increasing precision. We present experimental results to evaluate these symmetry breaks. These results show that in the setting of a counterexample-guided abstraction-based (CEGAR) addition of symmetry breaks, focusing on "involutions first" reduces the number of iterations of the CEGAR-loop. Broadly speaking, this shows that it is beneficial to look for symmetry-breaking constraints systematically, rather than arbitrarily.

The rest of the paper is structured as follows. Section 2 provides definitions and notations that are used throughout. A series of examples are provided to demonstrate the concepts upon which we build. Section 3 builds on the set-covering approach to symmetry breaking and presents a greedy approach to cover all of the non-canonical graphs with graph patterns. While this approach does not scale, it provides a test bed with which to study the structure of the ("best") graph patterns selected. Section 4 applies the lessons learned from the greedy approach to guide a CEGAR-based algorithm to collect specific types of graph patterns that derive from various types of involutions. Section 5 describes a series of experiments in which we provide complete and partial symmetry breaks to a specific graph search problem and evaluate their performance. Finally, Section 6 concludes and presents future work.

2 Preliminaries and Notation

Representing Graphs. Throughout this paper, we consider simple graphs, i.e. undirected graphs with no self-loops. The adjacency matrix of a graph G is an $n \times n$ Boolean matrix. The element at row i and column j is *true* if and only if (i, j) is an edge. The list of edges of a graph G is denoted $edges(G)$. It consists of the $\binom{n}{2}$ elements obtained as the concatenation of the columns of the upper triangle of G (or any other predetermined order). In abuse of notation, we let G denote a graph in any of its representations. An *unknown graph* of order-n is represented as an $n \times n$ adjacency matrix of Boolean variables which is symmetric and has the values *false* (denoted by 0) on the diagonal, or as the corresponding list of edges.

Ordering Graphs. Let G_1, G_2 be known or unknown graphs with n vertices. Then, $G_1 \leq G_2$ if and only if $edges(G_1) \leq_{lex} edges(G_2)$ where $\leq_{lex}$ denotes the standard lexicographic ordering. When G_1 and G_2 are unknown graphs, then the lexicographic ordering, $G_1 \leq G_2$, can be viewed as specifying a *lexicographic order constraint* over the variables in G_1 and G_2. For Boolean strings $\bar{a} = \langle a_1, \ldots, a_m \rangle$ and $\bar{b} = \langle b_1, \ldots, b_m \rangle$ and $1 \leq i \leq m$, we denote

$$\bar{a} <^i_{lex} \bar{b} \Leftrightarrow \{\ (a_1 = b_1), \ldots, (a_{i-1} = b_{i-1}), (a_i = 0), (b_i = 1)\ \} \tag{1}$$

We denote $G_1 <^i G_2$ if $edges(G_1) <^i_{lex} edges(G_2)$.

Permutations. The group of permutations on $\{1 \ldots n\}$ is denoted S_n. We represent a permutation $\pi \in S_n$ as a sequence of length n where the ith element indicates the value of $\pi(i)$. For example: the permutation $[2, 3, 1] \in S_3$ maps as follows: $\{\ 1 \mapsto 2, 2 \mapsto 3, 3 \mapsto 1\ \}$. We will refer to the following types of permutations:

Transpositions. A *transposition* is a permutation that swaps two elements and leaves all other in place. For example, $[4, 2, 3, 1]$ is the transposition that swaps elements 1 and 4. A transposition that swaps a pair of consecutive elements, is called a *consecutive transposition*. For example, $[1, 3, 2, 4]$ is a consecutive transposition.

Involutions. An *involution* is a permutation π such that $\pi \circ \pi$ is the identity. Involutions are permutations that swap any set of pairs of disjoint elements. For example, $[4, 3, 2, 1]$ is the involution which swaps the pairs $\{\ 1, 4\ \}$ and $\{\ 2, 3\ \}$. Transpositions are a special case of involutions. A *consecutive involution* is an involution that swaps any number of consecutive pairs. For example $[2, 1, 4, 3]$ is a consecutive involution. We say that an involution is *intersecting* if it swaps (among others) the pairs $\{\ i, j\ \}$ and $\{\ k, l\ \}$ with $i < j$ and $k < l$ such that the intervals $[i, j]$ and $[k, l]$ are intersecting. If an involution is not intersecting, then we say that it is *disjoint*. For example, $[3, 4, 1, 2]$ (in cyclic notation

$(1\ 3)(2\ 4)$) is a disjoint involution because the intervals $[1,3]$ and $[2,4]$ are intersecting. The number of involutions is given as the sequence `A000085` of the Online Encyclopedia of Integer Sequences. Note that an involution is always a composition of finitely many disjoint transpositions.

Permutations act on graphs and on unknown graphs in the natural way. For a graph and also for an unknown graph G, viewing G as an adjacency matrix, given a permutation $\pi \in S_n$, then $\pi(G)$ is the adjacency matrix obtained by mapping each element at position (i,j) to position $(\pi(i),\pi(j))$ (for $1 \leq i,j \leq n$). The composition of permutations is defined in the natural way. Two graphs G, H are *isomorphic* if there exists a permutation $\pi \in S_n$ such that $G = \pi(H)$. Note that applying an involution to the vertices of a graph (directed or undirected) induces an involution on its edges because $(\pi \circ \pi(i), \pi \circ \pi(j)) = (i,j)$ for any involution π.

Symmetry Breaks. A *symmetry break* for graph search problems is a predicate, $\psi(G)$, on a graph G, which is satisfied by at least one graph in each isomorphism class of graphs. If ψ is satisfied by exactly one graph in each isomorphism class, then we say that ψ is a *complete symmetry break*. Otherwise, it is *partial*.

Heule [13] defines the notion of *redundancy ratio*, which we denote $\rho(\psi)$, to measure the precision of ψ. This is the ratio between the number of graphs that satisfy $\psi(G)$ and the number of isomorphism classes. One can view $\rho(\psi)$ as the average number of graphs per isomorphism class that are not eliminated by ψ.

In our setting, the canonical graphs, are the minimal (lexleader) graphs of the isomorphism classes of graphs. The following example is adapted from [6] and demonstrates some of the concepts introduced so far. Notice that, in this paper, edge variables are ordered by columns.

▶ **Example 1.** The following depicts an unknown, order-4, graph G, its permutation $\pi(G)$, for $\pi = [1,2,4,3]$, and their representations as lists of edges. The lex-constraint $G \leq \pi(G)$ can be simplified as described by Frisch *et al.* [11] to: $\langle x_2, x_3 \rangle \leq_{lex} \langle x_4, x_5 \rangle$.

$$\mathbf{G} = \begin{bmatrix} 0 & x_1 & x_2 & x_4 \\ x_1 & 0 & x_3 & x_5 \\ x_2 & x_3 & 0 & x_6 \\ x_4 & x_5 & x_6 & 0 \end{bmatrix} \quad \pi(\mathbf{G}) = \begin{bmatrix} 0 & x_1 & x_4 & x_2 \\ x_1 & 0 & x_5 & x_3 \\ x_4 & x_5 & 0 & x_6 \\ x_2 & x_3 & x_6 & 0 \end{bmatrix} \quad \begin{array}{l} \text{edges}(G) = \langle x_1, x_2, x_3, x_4, x_5, x_6 \rangle \\ \text{edges}(\pi(G)) = \langle x_1, x_4, x_5, x_2, x_3, x_6 \rangle \end{array}$$

There are 64 graphs of order 4. Eleven of these are canonical and the other 53 are non-canonical. The canonical graphs, represented as lists of edges, are detailed below.

```
[0,0,0,0,0,0] [0,0,0,0,0,1] [0,0,0,0,1,1] [0,0,0,1,1,1] [0,0,1,0,1,1] [0,0,1,1,0,0]
[0,0,1,1,0,1] [0,0,1,1,1,1] [0,1,1,1,1,0] [0,1,1,1,1,1] [1,1,1,1,1,1]
```

Graph Patterns. A *graph pattern* is a partially instantiated graph G (some elements are variables) such that all instances of G are non-canonical. We typically represent a graph pattern G using the list notation edges(G).

▶ **Example 2.** The following six graph patterns describe all of the 53 non-canonical graphs of order 4.

```
[1,0,A,B,C,D] [A,1,0,B,C,D] [A,1,B,0,C,D]
[A,B,1,B,0,C] [A,A,B,C,1,0] [A,B,B,1,0,C]
```

So, an order 4 graph is canonical if and only if it is not an instance of any of these six graph patterns. One can check that this is the case for the 11 canonical graphs detailed in Example 1.

For demonstration, it is easy to see why the first graph pattern of Example 2 is a valid pattern. If the vector representation of the graph G starts with 1,0, which correspond to the edges $\{1,2\},\{1,3\}$, these can be swapped by the transposition $[1,3,2,4]$ (swap vertices 2 and 3) resulting in a smaller graph (this will possibly affect other edges, but these are no longer important for the lexicographic comparison). So, G is not canonical.

Formally, graph patterns derive from permutations as stated in the following definition which is adapted from [6] where edge variables are ordered by rows (in this paper we order the edges by columns).

▶ **Definition 3** ([6]). *Let π be a permutation, G be an unknown graph of order-n with $edges(G) = \langle x_1,\ldots,x_m\rangle$, $edges(\pi(G)) = \langle y_1,\ldots,y_m\rangle$, and let $1 \le i \le m$. The graph pattern, $\mathsf{pat}_i(\pi)$, is the result of applying the most general unifier to the equations from the right side in Equation (1) for the case of $edges(\pi(G)) <^i_{lex} edges(G)$ (i.e., the most general substitution that makes both sides of the equations identical). If the equations have no solution, then we denote $\mathsf{pats}_i(\pi) = \bot$. The set of all graph patterns of order n is denoted: $AllPats(n) = \{\ \mathsf{pat}_i(\pi) \neq \bot \mid \pi \in S_n, 1 \le i \le \binom{n}{2}\ \}$.*

The graph pattern $\mathsf{pat}_i(\pi)$ represents the set of graphs that get smaller at position i with π. If π is a particular type of permutation, e.g., a transposition, then we say that $\mathsf{pat}_i(\pi)$ is a pattern of that type. Note that $\mathsf{pat}_i(\pi)$ is a legal graph pattern because graphs that get smaller under some permutation are trivially non-canonical. Observe that $\mathsf{pat}_3([1,3,2,4]) = \bot$, as in this case $edges(G) = [x_1,x_2,x_3,x_4,x_5,x_6]$, $edges(\pi(G)) = [x_2,x_1,x_3,x_4,x_6,x_5]$ and $edges(\pi(G)) <^3_{lex} edges(G)$ has no solution (because $x_3 \not< x_3$). Observe also, that in some cases, different permutations lead to identical patterns. For example, $\mathsf{pat}_3([2,5,1,3,4]) = \mathsf{pat}_3([2,5,1,4,3]) = [x_1,x_1,1,x_2,x_3,x_4,0,x_1,x_5,x_6]$.

▶ **Example 4.** The following details how the graph patterns from Example 2 are derived from permutations.

$\mathsf{pat}_1([1,3,2,4]) = [1,0,A,B,C,D]$ $\quad$ $\mathsf{pat}_2([2,1,3,4]) = [A,1,0,B,C,D]$
$\mathsf{pat}_2([1,2,4,3]) = [A,1,B,0,C,D]$ $\quad$ $\mathsf{pat}_3([1,2,4,3]) = [A,B,1,B,0,C]$
$\mathsf{pat}_4([2,1,3,4]) = [A,B,B,1,0,C]$ $\quad$ $\mathsf{pat}_5([1,3,2,4]) = [A,A,B,C,1,0]$

▶ **Definition 5** (cover). *Let $\pi \in S_n$, $1 \le i \le \binom{n}{2}$, and let $\mathsf{pat}_i(\pi) \neq \bot$. Then, $\mathsf{pat}_i(\pi)$ covers a graph G if G is an instance of $\mathsf{pat}_i(\pi)$. Equivalently, $\mathsf{pat}_i(\pi)$ covers G if $\pi(G) <^i G$. We denote the set $cover(\mathsf{pat}_i(\pi)) = \{\ G \mid \pi(G) <^i G\ \}$. Sometimes we write also $cover(\pi,i)$. If $\mathsf{pat}_i(\pi) = \bot$ then it has no instances and so in this case, we define $cover(\mathsf{pat}_i(\pi)) = \emptyset$. The number of graphs covered by a graph pattern p is 2^k where k is the number of variables in p.*

Some graph patterns "dominate" others.

▶ **Definition 6** (dominate). *We say that graph pattern p_1 dominates graph pattern p_2 if $cover(p_2) \subseteq cover(p_1)$. Given a set S of graph patterns, $dominators(S)$ denotes the set of dominating atoms in S. We denote $DomPats(n) = dominators(AllPats(n))$.*

Viewing graph patterns as first-order logic terms, p_1 dominates p_2 if p_1 is more general than p_2. In Prolog, one can implement a test for "p_1 dominates p_2" using the built-in operator `subsumes_term(p1,p2)`.

▶ **Example 7.** There are 59 elements in $AllPats(4)$ (after removing redundancies). The set $DomPats(4)$ contains only 18 elements. For instance,

$\mathsf{pat}_3([1,2,4,3]) = [A,B,1,B,0,C]$ dominates $\mathsf{pat}_3([1,4,2,3]) = [A,A,1,A,0,B]$, and
$\mathsf{pat}_2([2,1,4,3]) = [A,1,B,C,0,D]$ dominates $\mathsf{pat}_3([3,4,2,1]) = [A,1,1,B,0,A]$.

Table 1 Numbers of graph patterns (and dominating graph patterns) for various types of permutations: consecutive transpositions `(ct)`, transpositions `(t)`, consecutive involutions with transpositions `(ci+t)`, disjoint involutions `(di)`, involutions `(i)`, and all permutations `(all)`.

order	ct	t	ci+t	di	i	all
4	6 (6)	12 (12)	14 (14)	14 (14)	17 (16)	59 (18)
5	12 (12)	30 (30)	40 (39)	47 (46)	80 (75)	550 (163)
6	20 (20)	60 (60)	92 (88)	136 (130)	348 (327)	4610 (1648)
7	30 (30)	105 (105)	187 (176)	361 (339)	1451 (1369)	43065 (17945)
8	42 (42)	168 (168)	354 (329)	906 (842)	6055 (5762)	421435 (199509)

Table 1 details the total number of graph patterns and the number of dominating graph patterns (in parentheses) for various types of permutations with small values of n. For consecutive transpositions, and for transpositions, all of the patterns are dominating. For other types of involutions, a large majority of the patterns are dominating.

Some graph patterns are orthogonal to others.

▶ **Definition 8** (orthogonal). *Let p_1 and p_2 be (non-$\perp$) graphs patterns. We say that p_1 is orthogonal to p_2 if $cover(p_1) \cap cover(p_2) = \emptyset$.*

In Prolog, one can implement a test for "p_1 is orthogonal to p_2" using the built-in operator for "not unifiable".

CEGAR. In [14] and in [4] the authors compute complete symmetry breaks for graphs based respectively on permutations and on a refinement of permutations which they term "implications". A similar approach is applied in [10] to break the symmetries for finite models. Common to all of these works is an algorithm based on *counter-example guided abstraction refinement (CEGAR)* [3].

In a nutshell, and in its simplest form, the CEGAR-based algorithm performs as follows: Let Ψ denote a set of permutations, which is initially empty. The algorithm repeatedly seeks a counter-example to the statement: "Ψ *is a complete symmetry break*". A counter-example takes the form: graph G and permutation π such that

$$\pi(G) < G \wedge \bigwedge_{\pi' \in \Psi} G \leq \pi'(G).$$

If such a counter-example is found then $\Psi = \Psi \cup \{\pi\}$. If no such counter-example is found, then Ψ is a complete symmetry break. The search for a counter-example is implemented using a SAT encoding and incremental SAT solving.[1]

An important detail is that Ψ may contain redundant elements. For example, if a permutation added at some point becomes redundant in view of permutations added later. A second phase of the algorithm iterates on the elements of Ψ to remove redundant permutations (similar to the iterative algorithm for a minimally unsatisfiable set or monotone predicates in general [21, 20]). It is important to note that the time to perform the second phase is costly. For example, in [14], the authors report that the time to compute the complete symmetry break Ψ for order 10 graphs is close to 10 hours and the time to reduce it is 84 hours.

In this paper, we focus on symmetry breaks defined in terms of graph patterns. We seek a set of graph patterns that covers all of the non-canonical graphs. The set $AllPats(n)$ of all graph patterns clearly covers all non-canonical graphs and hence it is a complete symmetry

[1] The SAT solver cadical-2.1.0 [2] was used in all our experiments.

break. But this set is too large to be of practical use. We adapt the CEGAR approach to graph patterns. In this setting, Ψ is a set of graph patterns, and we repeatedly seek a counter-example: a graph G and a graph pattern $\mathsf{pat}_i(\pi)$ such that G is covered by $\mathsf{pat}_i(\pi)$ but is not covered by any of the graph patterns in Ψ. If such a counter-example is found then $\Psi = \Psi \cup \{\mathsf{pat}_i(\pi)\}$. If no such counter-example is found, then Ψ is a complete symmetry break.

Let $p = [p_1, \ldots, p_m]$ be a graph pattern and let $G = [x_1, \ldots, x_m]$ denote an unknown graph. Let I_1 denote the set of induces in p which contain a zero, I_2 denote the set of indices in p which contain a one, and I_3 denote the set of pairs of indices that contain equal variables. The single clause:

$$c = (\bigvee_{i \in I_1} x_i) \vee (\bigvee_{i \in I_2} \neg x_i) \vee (\bigvee_{(i,j) \in I_3} x_i \oplus x_j)$$

encodes that G is not covered by p. Strictly speaking, c is not a clause due to the `xor` operations. It is straightforward to replace $x_i \oplus x_j$ by a fresh variable x_{ij} and to add clauses for $x_{ij} \leftrightarrow x_i \oplus x_j$ (note that these fresh variables are reused across the encoding of different graph patterns).

▶ **Example 9.** let $p = [A, B, 1, B, 0, C]$. The clause $(x_5 \vee \neg x_3 \vee x_2 \oplus x_4)$ encodes that $G = [x_1, x_2, x_3, x_4, x_5, x_6]$ is not covered by p.

In [4], the authors present a CEGAR based approach to derive complete symmetry breaks based on "lex-implications". Essentially, lex-implications are constraints of the form detailed in Equation (1). Similar to graph patterns, lex-implications can be represented as pairs of the form (π, i). The novelty in the presentation in this paper stems from the graph pattern perspective where it is natural to talk about concepts such as cover, dominance and orthogonality. In the graph pattern perspective, any set S of graph patterns is a symmetry break in the sense that a graph G is canonical, *only if* it is not an instance of an element in S. Moreover, S is a complete symmetry break if also the inverse *(if)* statement holds.

3 Breaking Symmetries with Graph Patterns: A Greedy Approach

The CEGAR approach outlined above breaks symmetries as they are suggested by the SAT solver as counter-examples. This might lead to unnecessarily large sets of graph patterns as the SAT solver makes arbitrary choices relatively to the ultimate objective. In this section, we apply a greedy approach to derive symmetry breaks consisting of graph patterns. At the core of the approach is the notion of ranking a graph pattern in view of the set Ψ of graph patterns selected so far.

▶ **Definition 10.** *Let Ψ be a set of graph patterns (a symmetry break) and let p be a graph pattern. We define ranking$_\Psi(p)$ to be the number of graphs covered by p but not covered by any of the elements of Ψ*

The basic idea is presented as Algorithm 1. At Line 2, The set S of candidate graph patterns is initialized to the set of dominating graph patterns. The symmetry break Ψ is initialized to the empty set. At each step of the algorithm, a graph pattern p with maximal ranking is selected and the sets S and Ψ are updated.

In the implementation, ranking the current set S of candidate graph patterns (Line 5 in Algorithm 1) is a bottleneck. Each graph pattern is ranked using a sat encoding with a model counter. As an optimization, in the loop at Lines 4–6, we remove from S all graph

Algorithm 1 Greedily compute symmetry break for order n graphs.

```
1:  procedure SYMBREAK(n)
2:      S ← DomPats(n); Ψ ← ∅;
3:      repeat
4:          select p ∈ S such that
5:                r = max_{p'∈S} ranking_Ψ(p');
6:                ranking_Ψ(p) = r;
7:          if r>0 then
8:              S ← S \ {p}; Ψ ← Ψ ∪ {p};
9:      until r = 0;
10:     return Ψ;                          ▷ Ψ is a complete symmetry break
```

patterns p such that $\text{ranking}_\Psi(p) = 0$. Moreover, whenever we select a pattern $p \in S$ at Line 4, we also select a set $S' \subseteq S$ (as large as possible) so that $\{p\} \cup S'$ are mutually orthogonal (as prescribed by Definition 8). The selection of S' is greedy: When selecting $p \in S$ at Line 4, initialize $S' = \{p\}$ and iterate over the patterns from S, adding a pattern to S' if it is orthogonal to those already in S'.

▶ **Example 11.** The following details the run of Algorithm 1 for $n = 4$. The algorithm selects 6 graph patterns in 3 rounds. The column labeled Δ indicates the number of graphs covered by this graph pattern and not covered by those above. In this example, the second graph pattern added in each round is orthogonal to that from the first added in the round. The selected graph patterns are the same as those detailed in Example 2. Note that the sum of the numbers in the Δ column is 53, corresponding to the number of non-canonical graphs of order 4.

round	pattern	Δ
1	[1,0,A,B,C,D]	16
1	[A,1,0,B,C,D]	16
2	[A,1,B,0,C,D]	8
2	[A,B,1,B,0,C]	6
3	[A,A,B,C,1,0]	5
3	[A,B,B,1,0,C]	2

The following proposition identifies four specific graph patterns that cover 3/4 of the total number of graphs. Observe that the permutations detailed in the proposition are consecutive transpositions.

▶ **Proposition 12.** *For $n \geq 5$, the four top ranking graph patterns always take the following particular form where $m = \binom{n}{2} - 2$. Below, we adopt the cycle notation for permutations. The permutation (j, k) is the permutation which swaps elements j and k and leaves all other elements fixed.*

1. $\mathsf{pat}_1((2,3)) = [1, 0, x_1, x_2, x_3, x_4, x_5, x_6, \ldots, x_m]$
2. $\mathsf{pat}_2((1,2)) = [x_1, 1, 0, x_2, x_3, x_4, x_5, x_6, \ldots, x_m]$
3. $\mathsf{pat}_2((3,4)) = [x_1, 1, x_2, 0, x_3, x_4, x_5, x_6, \ldots, x_m]$
4. $\mathsf{pat}_4((4,5)) = [x_1, x_2, x_3, 1, x_4, x_5, 0, x_6, \ldots, x_m]$

The first two graph patterns are orthogonal and each covers 2^{m-2} graphs. The second two are also orthogonal and each covers 2^{m-3} graphs from those not covered by the first two. Together they cover 3/4 of the total number of graphs.

Table 2 Profiling partial and complete symmetry breaks obtained from greedy algorithm.

n	symmetry break	total	ct	t	ci+t	di	i	ρ	% ncc
7	greedy complete	116	28	30	59	60	80	1.00	100.00
	greedy partial	58	28	28	48	49	51	1.27	99.99
	greedy ct partial	13	13	0	0	0	0	24.38	98.78
	cegar complete	136	15	16	44	48	62	1.00	100.00
	cegar partial	68	15	16	25	27	30	1035.25	51.56
	involutions partial	113	28	30	60	70	113	1.01	99.99
8	greedy complete	439	40	46	120	124	235	1.00	100.00
	greedy partial	219	40	44	115	116	153	1.07	99.99
	greedy ct partial	18	18	0	0	0	0	52.63	99.76
	cegar complete	492	19	24	104	114	194	1.00	100.00
	cegar partial	246	19	24	63	71	95	9710.21	44.66
	involutions partial	396	40	47	127	158	396	1.02	99.99

Proof (sketch). The proof follows from a theorem presented in [18], which states that if G is an order n lex-leader canonical graph where the order of edges is column-wise, then so is the subgraph $G(k)$ on the first k vertices for $1 \leq k < n$. It follows that for any position $i < \binom{n}{2}$, the graph pattern $\mathsf{pat}_i((j,k))$ on order-n graphs is also a graph pattern for order-$n+1$ graphs. ◀

Table 2 compares 6 symmetry breaks and profiles them with respect to 5 types of permutations. The first three symmetry breaks are obtained using Algorithm 1: `greedy complete` is the complete symmetry break, `greedy partial` is the partial symmetry break consisting of the first (highest ranking) 50% of the patterns obtained from Algorithm 1, and `greedy ct partial` is the partial symmetry break consisting of the (longest) prefix of patterns obtained which are all consecutive transpositions. The next two symmetry breaks are obtained using the CEGAR algorithm described in [4]: `cegar complete` is the complete symmetry break obtained, and `cegar partial` is the partial symmetry break consisting of the first 50% of the patterns obtained from the CEGAR algorithm. We will come back to the fifth symmetry break, `involutions partial` later.

The column labeled `total` details the total number of patterns in the symmetry break. The next 5 columns detail the number of patterns in the symmetry break derived from a permutation of a given type: `(ct)` consecutive transpositions, `(t)` all transpositions, `(ci+t)` consecutive involutions and transpositions, `(di)` disjoint involutions, and `(i)` all involutions. All of these types of permutations are specific forms of involutions. The first two: `(ct)` and `(t)` are those widely applied in the symmetry breaks defined in [8, 7].

The penultimate column, labeled ρ, shows the redundancy ratio. This is the ratio between the number of graphs that are not covered by the selected patterns (symmetries not broken) and the number of isomorphism classes (all symmetries broken). If the set of patterns is a complete symmetry break, then $\rho = 1$. The smaller this number is, the better the symmetry break. The last column, labeled `% ncc`, details the percentage of non-canonical graphs covered by the symmetry break. If the set of patterns is a complete symmetry break, then this number is 100.

What we learn from Table 2:

The first $4-10\%$ of the patterns selected are all derived from consecutive transpositions. These patterns alone already cover $> 98\%$ of the non-canonical graphs. Moreover,

as stated in Proposition 12, the first four patterns always cover a total of 3/4 of the total number of graphs which is also about 75% of the total number of non-canonical graphs.

About half of the patterns selected in `greedy complete` are involutions; These are highly ranked. More than 60% are in the top half (by ranking) of the patterns. In contrast, for $n = 7$ less than 5% of all (n-factorial) permutations are involutions, and for $n = 8$ less than 2%. The number of involutions is given as sequence `A000085` of the Online Encyclopedia of Integer Sequences.

Greedy selection pays off. Comparing the redundancy ratios for `cegar partial` and for `greedy partial` shows this. Basically, the CEGAR-based algorithm selects an arbitrary pattern which covers some graph that is not yet covered. In contrast to the greedy algorithm where the best such pattern is chosen.

There are a small number of graph patterns that cover a large portion of the search space. All of these patterns derive from consecutive transpositions. This is apparent from the last column in the table and the rows corresponding to `greedy ct partial`: for n=7, 13 graph patterns cover $> 98\%$ of the non-canonical graphs, and for $n = 8$, 18 graph patterns cover $> 99\%$. We propose to consider three numbers when evaluating the quality of a symmetry break: (1) the redundancy ratio, (2) the percentage of non-canonical graphs covered, and (3) the size of the symmetry break. It is easy to obtain perfect values for the first two numbers when the third is large.

Given the apparent importance of various types of involutions, we consider a fifth symmetry break in Table 2: `involutions partial`. Here, we take the set of all dominating patterns derived from involutions. These are reduced to remove redundant patterns (a pattern in a set is redundant if all of the graphs that it covers are covered by other patterns in the set). For example, when $n = 7$, there are 1369 dominating patterns for permutations which are involutions (see Table 1) and these can be reduced to 113 patterns after removing redundancies. As indicated in Table 2, adopting involutions gives a close-to-perfect symmetry break for small values of n.

The greedy approach does not scale. Algorithm 1 relies on the expensive application of a model counter to rank the candidate patterns. Also, the computation of all dominating patterns is too time-consuming. This motivates the approach taken in the next section.

4 Tweaking CEGAR for Better Partial and Complete Symmetry Breaks

In this section we take the lessons learned from Table 2 and apply them to guide a CEGAR-based algorithm to make better selections. We layer the selection of counter-examples with layers corresponding to the five types of patterns considered in the profiling of Table 2. This means that in each layer, each iteration of the CEGAR loop searches for a graph G and a permutation π s.t. $\pi(G) < G$ and π is the permutation used in that layer. This yields a graph pattern generated from π and covering graph G.[2]

In the first layer `(ct)`, we seek counter-examples that are consecutive transpositions. When no further counter-examples of this type remain, we proceed to layer `(t)` to collect counter-examples which are transpositions, and so on for the layers `(ci+t)` introducing consecutive

[2] Graph patterns generated by a single permutation are disjoint, hence only one can cover the graph G.

Table 3 Comparing partial symmetry breaks obtained by layered CEGAR.

ord	ct		t		ci+t		di		i		compl
	size	ratio	size	ratio	size	ratio	size	ratio	size	ratio	size
4	6	1.00	6	1.00	6	1.00	6	1.00	6	1.00	6
5	12	1.35	13	1.26	13	1.06	13	1.06	14	1.00	14
6	20	2.08	24	1.77	29	1.16	30	1.12	36	1.00	36
7	30	3.87	40	3.02	64	1.46	70	1.31	111	1.01	115
8	42	7.27	62	5.39	137	1.95	167	1.53	397	1.02	444
9	56	13.05	91	9.42	269	2.76	401	1.83	2,024	1.03	2,760
10	72	21.53	128	15.34	526	3.97	1,001	2.20	12,644	1.04	24,993
11	90	33.23	174	23.52	1,008	5.71	2,523	2.65	81,522	≈1.14	289,863
12	110	48.97	230	34.53	1,896	≈7.39	6,275	≈3.12	n/a	n/a	n/a

involutions, `(di)` for disjoint involutions, and `(i)` for all other involutions. In a final, sixth layer, we seek arbitrary counter-examples. To ensure that counter-examples are found of the required type, we encode in the SAT solver an additional condition in a straightforward fashion. For example, involutions are encoded as the implications $\pi(i) = j \Rightarrow \pi(j) = i$, for $i \neq j$.

The layered CEGAR-based algorithm can be applied in two different capacities: First, to provide partial symmetry breaks as obtained at the end of each layer of the run. Second, to guide the search for a complete symmetry break. Experimentation indicates that making a better selection of counter-examples early on reduces the number of iterations applied in the course of the algorithm and reduces the number of redundant patterns that need to be removed in the second phase of the algorithm.

Table 3 addresses the first capacity. Each pair of columns details the partial symmetry break obtained after the corresponding layer in the revised CEGAR algorithm. For each layer, we present the size (number of patterns) of the corresponding partial symmetry break and the corresponding redundancy ratio. For the last layer (in the last column), the symmetry break is complete and the redundancy ratio is always 1.00. The first two pairs of columns `ct` and `t` correspond to the symmetry breaks described in [8]. To compute the redundancy ratios, the knowledge-compilation-based tool `d4` [19] was applied for model counting.[3] In cases where the model counter was not able to compute the number of graphs we applied the approximate model counter `approx` [25]. These cases are marked by the symbol ≈.

Now we look at the effect of layering when CEGAR is used to calculate a complete symmetry break. Figure 1 summarizes the comparison of the layered approach to CEGAR and the standard one. Figure 1 (on the left), details numbers of iterations (and patterns). The two upper curves detail the number of iterations required to calculate a complete cover of all graphs of order $n \in \{8..11\}$. Each iteration of the CEGAR loop adds one pattern to the cover. However, some patterns might become redundant due to the addition of a stronger pattern later on. The lower two curves detail the number of patterns in the irredundant cover obtained by reducing the output of the CEGAR algorithm (irredundant cover is calculated by standard means [20]). Note that the size of the irredundant cover must always be smaller or equal to the number of iterations of the CEGAR iterations.

[3] Other model counters were considered but `d4` gave better performance.

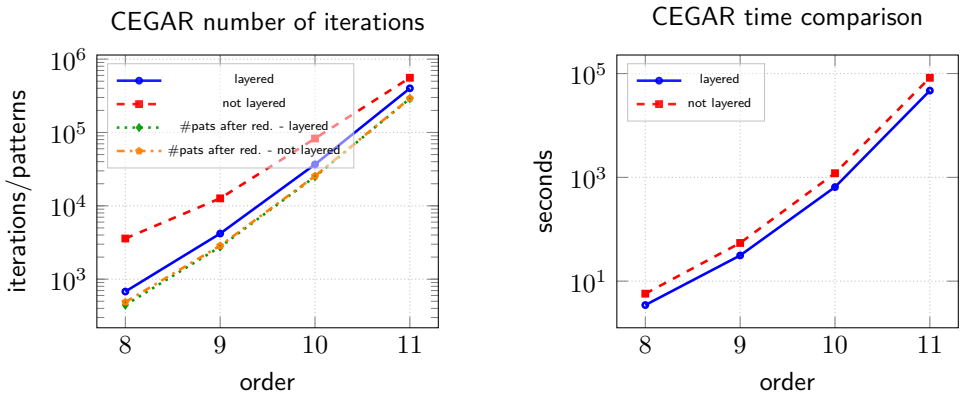

Figure 1 Comparison of CEGAR and Layered CEGAR (log scale).

The number of iterations is always lower for the layered approach than for the standard one. This typically also reduces the overall time. For $n = 11$, CEGAR needed 400,016 iterations in the layered approach, contrasting with 557,279 iterations in the non-layered (standard) approach. Interestingly, both were reduced to covers of similar sizes, 289,863 in the layered approach and 291,888 in the non-layered one. This indicates that the layered approach guides CEGAR *more precisely*.

Figure 1 (on the right) complements this information with the total CPU time needed to calculate the cover, and it can be observed that lowering the iterations of the loop also reduces the total time.

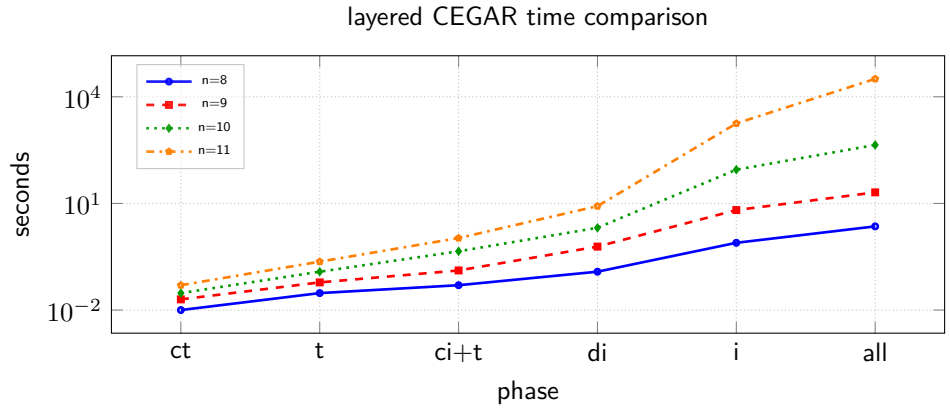

Figure 2 Computation time for partial symmetry breaks with layered CEGAR (log scale).

Figure 2 details the computation times for partial symmetry breaks obtained in the different phases of the layered CEGAR algorithm. For $n = 11$, computing the partial symmetry breaks defined in terms of disjoint involutions and in terms of all involutions is respectively 3 and 1 orders of magnitude faster than computing the complete symmetry break. Given the fact that these symmetry breaks are relatively small and have good redundancy ratios indicates that they provide a good choice when balancing time (to compute them) and size with precision.

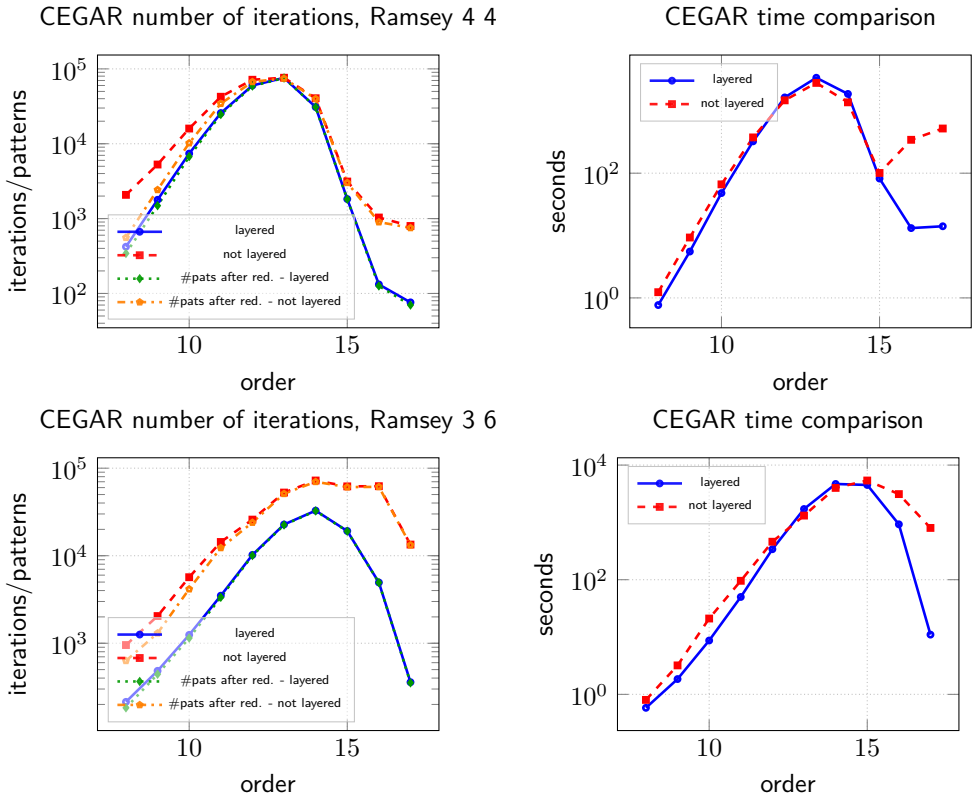

Figure 3 Comparison of CEGAR's Performance on Ramsey graphs (log scale).

5 Symmetry Breaks for a Specific Graph Search Problem

In this section, we describe the computation of partial and complete symmetry breaks tailored for a specific graph search problem. In this context, when seeking a counter-example in the CEGAR loop, we restrict the search to graphs that satisfy the constraints of the given search problem. We apply this approach for the layered approach to CEGAR and the standard one.

We consider a classic example: the search for *Ramsey graphs* [22]. The graph $R(s,t;n)$ is a simple graph with n vertices that contains no clique of size s, and no independent set of size t. The *Ramsey number* $R(s,t)$ is the smallest number n for which there is no $R(s,t;n)$ graph. We also describe the impact of using these symmetry breaks.

Figure 3 summarizes the comparison of the computation of symmetry breaks in this context. We focus on symmetry breaks for $R(4,4;n)$ (two upper plots) and $R(3,6;n)$ (two lower plots).[4] We study only the values of n until the respective Ramsey numbers, which are both 18. The number of Ramsey graphs peaks at some point, at which it is also difficult to calculate the cover. The two plots on the left detail the numbers of iterations in the layered/non-layered approaches as well as the numbers of patterns removed after the CEGAR loops (upper for $Ramsey(4,4;n)$ and lower for $Ramsey(3,6;n)$). The two plots on the right detail total CPU times (upper for $Ramsey(4,4;n)$ and lower for $Ramsey(3,6;n)$).

[4] These enable us to perform a precise analysis, however, unsatisfiability alone for larger Ramsey numbers is famously difficult [12].

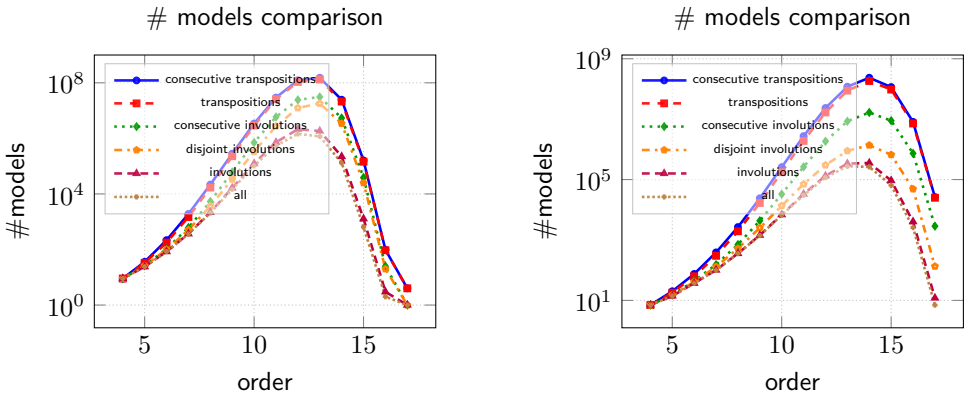

Figure 4 The impact of various partial symmetry breaks on Ramsey graphs (log scale).

The number of iterations is always lower for the layered approach than for the standard one. Also, the number of patterns removed in the reduce phase is lower. This typically reduces the overall time. This is, however, not always the case as individual SAT calls are more expensive.

Here, in comparison to Figure 1, the number of patterns removed after the CEGAR loop is much smaller. Interestingly, in the case of $R(3,6;n)$, not only that the number of iterations is significantly smaller in the layered approach, but it also remains so after reduction. This suggests that the two runs of CEGAR+reduction reach completely different local optima.

Figure 4 depicts the number of Ramsey graphs found when applying the various proposed symmetry breaks: $Ramsey(4,4;n)$ on the left and $Ramsey(3,6;n)$ on the right. The curves in both plots from highest to lowest, correspond precisely and in order to the 6 layers: `(ct)`, `(t)`, `(ci+t)`, `(di)`, `(i)`, and `(all)` (the complete symmetry break). The two upper curves describe the application of the widely applied partial symmetry breaks defined in [8]. The lowest curve describes the application of a complete symmetry break. General transpositions (`t`) do not give much advantage over just consecutive transpositions `(ct)`. However, both are by orders of magnitude worse than the other classes of permutations. Note the proximity of the curve for all involutions (`i`) to the lowest curve (the complete symmetry break). This suggests that focusing only on involutions gives a symmetry break very close to the complete break.

6 Conclusions and Future Work

The objective of this paper is to approach the search for graph symmetry breaks in a systematic way. We study the structure of graph patterns that are selected in a greedy algorithm to break all symmetries for graphs of small orders. We learn that graph patterns that derive from a specific class of permutations, called involutions, play an important role. Involutions generalize the class of transpositions, which play an important role in breaking symmetries for graphs. We then refine a CEGAR-based approach to compute symmetry breaks introducing a layered approach. In this way, we can guide the CEGAR-based search for a complete symmetry break and also provide a series of partial symmetry breaks of increasing precision.

It is important to reflect on why we construct partial symmetry breaks in a CEGAR-based algorithm, in contrast to simply collecting all permutations of the restricted types. For patterns based on transpositions, the straightforward construction is doable. However,

for patterns based on the other types of involutions, there are too many of them and the CEGAR-based approach enables us to select those that contribute to the corresponding symmetry breaks.

We do not expect to find a complete symmetry break of polynomial size that identifies canonical graphs that are lex-leaders, cf. [16]. However, we still aim to find small, perhaps even polynomial in size, *partial* symmetry breaks that break a majority of the symmetries on graphs. Coming back to the greedy algorithm presented in Section 3, for $n = 8$, 439 graph patterns are found to cover all of the 268,423,110 non-canonical graphs of order 8. The last 163 graph patterns selected in the greedy algorithm, cover a negligible total of 382 of these 268,423,110 graphs. In much the same way that Proposition 12 points to four specific graph patterns that cover 75% of the non-canonical graphs, the *holy grail* for symmetry breaking for graphs is to specify a small symmetry break that breaks a significant portion of the symmetries.

This paper opens a number of avenues for future work. Since focusing on specific permutation types positively impacts CEGAR, we plan to also apply the lessons learned in this paper in the context of a dynamic symmetry breaking for graph generation. In this approach, for example as performed in [17], symmetries are detected and broken during the generation (and enumeration) of the solutions of a given graph search problem. Another direction of research is to identify further sub-classes of permutations by either refining the framework proposed here or looking for completely different classes. Such division could also be problem-specific. A more theoretical direction of research is to find a justification for why involutions lead to better symmetry breaks.

References

1 László Babai and Eugene M. Luks. Canonical labeling of graphs. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 171–183. ACM, 1983. doi:10.1145/800061.808746.

2 Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL 2.0. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, volume 14681 of *Lecture Notes in Computer Science*, pages 133–152. Springer, 2024. doi:10.1007/978-3-031-65627-9_7.

3 Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In E. Allen Emerson and Aravinda Prasad Sistla, editors, *Computer Aided Verification*, pages 154–169, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. doi:10.1007/10722167_15.

4 Michael Codish, Thorsten Ehlers, Graeme Gange, Avraham Itzhakov, and Peter J. Stuckey. Breaking symmetries with lex implications. In John P. Gallagher and Martin Sulzmann, editors, *Functional and Logic Programming - 14th International Symposium, FLOPS 2018, Nagoya, Japan, May 9-11, 2018, Proceedings*, volume 10818 of *Lecture Notes in Computer Science*, pages 182–197. Springer, 2018. doi:10.1007/978-3-319-90686-7_12.

5 Michael Codish, Graeme Gange, Avraham Itzhakov, and Peter J. Stuckey. Breaking symmetries in graphs: The nauty way. In Michel Rueher, editor, *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*, volume 9892 of *Lecture Notes in Computer Science*, pages 157–172. Springer, 2016. doi:10.1007/978-3-319-44953-1_11.

6 Michael Codish and Mikoláš Janota. Breaking symmetries from a set-covering perspective. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 22nd International Conference, CPAIOR, Melbourne, Australia November 10-13, 2025 Proceedings*, Lecture Notes in Computer Science. Springer, 2025.

7 Michael Codish, Alice Miller, Patrick Prosser, and Peter J. Stuckey. Constraints for symmetry breaking in graph representation. *Constraints*, 24(1):1–24, 2019. doi:10.1007/s10601-018-9294-5.

8 Michael Codish, Alice Miller, Patrick Prosser, and Peter James Stuckey. Breaking symmetries in graph representation. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 510–516, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6480.

9 James M. Crawford, Matthew L. Ginsberg, Eugene M. Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. In Luigia Carlucci Aiello, Jon Doyle, and Stuart C. Shapiro, editors, *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96)*, pages 148–159. Morgan Kaufmann, 1996.

10 Marek Dančo, Mikoláš Janota, Michael Codish, and João Jorge Araújo. Complete symmetry breaking for finite models. In *AAAI 2025*, 2025. doi:10.48550/arXiv.2502.10155.

11 Alan M. Frisch and Warwick Harvey. Constraints for breaking all row and column symmetries in a three-by-two matrix. In *Proceedings of SymCon'03*, 2003.

12 Thibault Gauthier and Chad E. Brown. A formal proof of R(4, 5)=25. In Yves Bertot, Temur Kutsia, and Michael Norrish, editors, *15th International Conference on Interactive Theorem Proving, ITP 2024, September 9-14, 2024, Tbilisi, Georgia*, volume 309 of *LIPIcs*, pages 16:1–16:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ITP.2024.16.

13 Marijn J. H. Heule. Optimal symmetry breaking for graph problems. *Mathematics in Computer Science*, 2019.

14 Avraham Itzhakov and Michael Codish. Breaking symmetries in graph search with canonizing sets. *Constraints*, pages 1–18, 2016.

15 Avraham Itzhakov and Michael Codish. Breaking symmetries with high dimensional graph invariants and their combination. In André A. Ciré, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 20th International Conference, CPAIOR 2023, Nice, France, May 29 - June 1, 2023, Proceedings*, volume 13884 of *Lecture Notes in Computer Science*, pages 133–149. Springer, 2023. doi:10.1007/978-3-031-33271-5_10.

16 George Katsirelos, Nina Narodytska, and Toby Walsh. On the complexity and completeness of static constraints for breaking row and column symmetry. In David Cohen, editor, *Principles and Practice of Constraint Programming - CP 2010 - 16th International Conference, CP 2010, St. Andrews, Scotland, UK, September 6-10, 2010. Proceedings*, volume 6308 of *Lecture Notes in Computer Science*, pages 305–320. Springer, 2010. doi:10.1007/978-3-642-15396-9_26.

17 Markus Kirchweger and Stefan Szeider. SAT modulo symmetries for graph generation and enumeration. *ACM Trans. Comput. Log.*, 25(3):1–30, 2024. doi:10.1145/3670405.

18 Vladmír Kvasnička and Jiří Pospíchal. Canonical indexing and constructive enumeration of molecular graphs. *J. Chem. Inf. Comput. Sci.*, 30(2):99–105, April 1990. doi:10.1021/CI00066A001.

19 Jean-Marie Lagniez and Pierre Marquis. An improved decision-DNNF compiler. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 667–673, 2017. doi:10.24963/ijcai.2017/93.

20 João Marques-Silva, Mikoláš Janota, and Anton Belov. Minimal sets over monotone predicates in boolean formulae. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 592–607. Springer, 2013. doi:10.1007/978-3-642-39799-8_39.

21 Alexander Nöhrer, Armin Biere, and Alexander Egyed. Managing SAT inconsistencies with HUMUS. In Ulrich W. Eisenecker, Sven Apel, and Stefania Gnesi, editors, *Sixth International Workshop on Variability Modelling of Software-Intensive Systems, Leipzig, Germany, January 25-27, 2012. Proceedings*, pages 83–91. ACM, 2012. doi:10.1145/2110147.2110157.

22 Stanislaw P. Radziszowski. Small Ramsey numbers. *Electronic Journal of Combinatorics*, 1994. Revision #14: January, 2014. URL: http://www.combinatorics.org/.

23 Jussi Rintanen and Masood Feyzbakhsh Rankooh. Symmetry-breaking constraints for directed graphs. In Ulle Endriss, Francisco S. Melo, Kerstin Bach, Alberto José Bugarín Diz, Jose Maria Alonso-Moral, Senén Barro, and Fredrik Heintz, editors, *ECAI 2024 - 27th European Conference on Artificial Intelligence, 19-24 October 2024, Santiago de Compostela, Spain - Including 13th Conference on Prestigious Applications of Intelligent Systems (PAIS 2024)*, volume 392 of *Frontiers in Artificial Intelligence and Applications*, pages 4248–4253. IOS Press, 2024. doi:10.3233/FAIA240998.

24 Ilya Shlyakhter. Generating effective symmetry-breaking predicates for search problems. *Discrete Applied Mathematics*, 155(12):1539–1548, 2007. doi:10.1016/J.DAM.2005.10.018.

25 Mate Soos and Kuldeep S. Meel. BIRD: engineering an efficient CNF-XOR SAT solver and its applications to approximate model counting. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019*, pages 1592–1599. AAAI Press, 2019. doi:10.1609/AAAI.V33I01.33011592.

26 Toby Walsh. General symmetry breaking constraints. In *Principles and Practice of Constraint Programming - CP 2006, 12th International Conference, CP 2006, Nantes, France, September 25-29, 2006, Proceedings*, pages 650–664, 2006. doi:10.1007/11889205_46.