



Compact Partial Symmetry Breaking for Graph Search Problems

Michael Codish¹, Mikoláš Janota², and Peter J. Stuckey^{3,4}

¹ Ben-Gurion University of the Negev, Israel

² Czech Technical University, Czechia

³ Monash University, Australia

⁴ OPTIMA ARC Industrial Training and Transformation Centre, Australia

Abstract

Symmetry breaking in graph search problems remains challenging: complete symmetry breaking constraints are typically exponential in size, while partial approaches trade precision for scalability. We introduce a strength-driven, redundancy-aware methodology for constructing compact partial symmetry breaking constraints based on graph patterns. Each pattern corresponds to a Boolean clause over equality literals and can be integrated directly into CP or SAT models. Our approach incrementally selects strong, non-redundant graph patterns, yielding partial symmetry breaks that achieve substantial pruning of non-canonical graphs while remaining compact in size. This provides a principled framework for deriving lightweight partial symmetry breaking constraints which are significantly more precise than all published state-of-the-art alternatives. We evaluate the proposed constraints on graph instances with up to 25 vertices using a redundancy ratio metric. The resulting partial symmetry breaks constitute only a small fraction of the size of complete symmetry breaking constraints, while delivering significant performance improvements across several graph search benchmarks compared to previously defined partial symmetry breaking methods.

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), which is isomorphic, or “symmetric”. To optimize the 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 [11, 26, 28]. 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 permutations, then the corresponding symmetry break is complete but too large to be of practical use.

Itzhakov and Codish [15] observe that a complete symmetry break for order- n graphs 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 fewer vertices. Dančo et al. [12] obtain similar results in the context of finite models. However, this approach does not scale. In subsequent work, Itzhakov *et al.* [5] introduce the notion of “lex-implications”, which refine lex-constraints, and they compute complete symmetry breaks that are considerably smaller in size. However, this approach still does not scale beyond graphs with 11 vertices. It is known that breaking symmetry by adding lex-constraints to eliminate symmetric solutions is intractable in general [1, 11]. 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.* [10, 9] 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.* [24] 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 [17] and in [6]. However, there are no references in the literature to applications which make use of the symmetry breaks defined in these papers.

In a recent paper, Codish and Janota [8] introduce the notion of a graph pattern for symmetry breaking. Graph patterns are simple atoms which provide a concise representation for (large) sets of non-canonical graphs. This also facilitates a set-covering perspective on symmetry breaking as described in [7]. A complete symmetry break should cover all of the non-canonical graphs. A partial symmetry break should cover “most” of the non-canonical graphs.

This paper is about scalable and precise partial symmetry breaks for graphs constructed using graph patterns. The idea is to repeatedly select graph patterns based on two notions: strength of patterns, and non-redundancy of patterns with respect to those already selected. The proposed construction results in symmetry breaks which are scalable both in terms of the cost to compute them as well as in terms of their size. Precision is measured in terms of redundancy ratio [14], ρ , that is the ratio between the number of graphs satisfying the symmetry break, and the number of canonical graphs. This means that, for example, when enumerating the solutions of a graph search problem with a symmetry breaking predicate with redundancy ratio ρ , we can expect to encounter on average ρ solutions for every canonical solution.

Applying the techniques proposed in [10, 9] and in [5], one can obtain complete symmetry breaks for graphs of order up to $n = 10$ and $n = 11$, respectively. For $n = 10$, the approach of [5] results in a complete symmetry break which consists of 25,263 graph patterns which is basically the same number of CNF clauses. Using involutions as reported in [8], one can obtain a partial symmetry break for order-10 graphs with 12,616 graph patterns and a redundancy ratio of 1.04. But this approach also does not scale. In this paper we show a scalable symmetry break consisting of 2,273 graph patterns for $n = 10$ with a redundancy ratio, $\rho = 1.34$. That means on average, for every canonical graph we get 1.34 graphs instead of 1. Moreover, we demonstrate the feasibility of our approach for graphs of sizes up to 25.

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 edge variables of a graph G is denoted $edges(G)$. It consists of the elements in the upper (or lower) triangle of the matrix representation in some fixed order. Throughout this paper we assume that the list consists of the $\binom{n}{2}$ elements obtained as the concatenation of the *columns* of the upper triangle of G . We will clarify the motivation for the column-wise ordering later in this section. 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 edge variables. The following example specifies two order-4 unknown graphs as adjacency matrices and as lists of edge variables.

Example 1. *The following depicts two unknown, order-4, graphs G_1 and G_2 and their representations as lists of edges.*

$$\mathbf{G}_1 = \begin{bmatrix} 0 & x_1 & x_2 & x_4 \\ x_1 & 0 & x_3 & x_5 \\ x_2 & x_3 & 0 & x_6 \\ x_4 & x_5 & x_6 & 0 \end{bmatrix} \quad \mathbf{G}_2 = \begin{bmatrix} 0 & x_1 & x_4 & x_2 \\ x_1 & 0 & x_5 & x_3 \\ x_4 & x_5 & 0 & x_6 \\ x_2 & x_3 & x_6 & 0 \end{bmatrix} \quad \begin{aligned} edges(G_1) &= \langle x_1, x_2, x_3, x_4, x_5, x_6 \rangle \\ edges(G_2) &= \langle x_1, x_4, x_5, x_2, x_3, x_6 \rangle \end{aligned}$$

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, $G_1 \leq G_2$, specifies a *lexicographic order constraint* over the variables in G_1 and G_2 .

Example 2. *Consider the two graphs, G_1 and G_2 from Example 1. The lexicographic order constraint $G_1 \leq G_2$ is $\langle x_1, x_2, x_3, x_4, x_5, x_6 \rangle \leq_{lex} \langle x_1, x_4, x_5, x_2, x_3, x_6 \rangle$ which can be simplified as described by Frisch et al. [13] to: $\langle x_2, x_3 \rangle \leq_{lex} \langle x_4, x_5 \rangle$.*

For Boolean strings $\bar{a} = \langle a_1, \dots, a_m \rangle$ and $\bar{b} = \langle b_1, \dots, b_m \rangle$ and $1 \leq i \leq m$, we denote

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

We sometimes refer to $\bar{a} <_{lex}^i \bar{b}$ as the set of equations on the right side of Equation (1). We denote $G_1 <^i G_2$ if $edges(G_1) <_{lex}^i edges(G_2)$.

Example 3. *Consider the two graphs, G_1 and G_2 from Example 1. Then,*

$$G_1 <^3 G_2 \Leftrightarrow \{ (x_1 = x_1), (x_2 = x_4), (x_3 = 0), (x_5 = 1) \}$$

$$G_1 <^5 G_2 \Leftrightarrow \{ (x_1 = x_1), (x_2 = x_4), (x_3 = x_5), (x_4 = x_2), (x_5 = 0), (x_3 = 1) \}$$

Observe that the second constraint ($G_1 <^5 G_2$) is not satisfiable.

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

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. 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 $\{2, 3\}$.

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 $1 \leq i, j \leq n$). Two graphs G, H are *isomorphic* if there exists a permutation $\pi \in S_n$ such that $G = \pi(H)$.

Let G be an order- n graph and π_v a permutation on the vertices of G . Let $\text{edges}(G) = \langle x_1, \dots, x_m \rangle$ and $\text{edges}(\pi(G)) = \langle y_1, \dots, y_m \rangle$ (for $m = \binom{n}{2}$). The permutation π_e such that $\pi_e(\langle x_1, \dots, x_m \rangle) = \langle y_1, \dots, y_m \rangle$ is called the edge permutation corresponding to π_v . Often we write π_v (or π_e) to indicate that a permutation applies on the vertices (or edges) of a graph.

Example 4. Consider the two graphs, G_1 and G_2 from Example 1 and consider the permutation $\pi_v = [1, 2, 4, 3]$. Then $G_2 = \pi(G_1)$. Recall that $\text{edges}(G_1) = \langle x_1, x_2, x_3, x_4, x_5, x_6 \rangle$ and $\text{edges}(G_2) = \langle x_1, x_4, x_5, x_2, x_3, x_6 \rangle$. So, $\pi_e = [1, 4, 5, 2, 3, 6]$ is the edge permutation corresponding to π_v .

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*. In our setting, the canonical graphs, are the minimal (lexleader) graphs of the isomorphism classes of graphs. In this setting, when lexicographic order is defined in terms of the column-wise concatenation of edge variables, canonical graphs of order n can be extended to canonical graphs of order $n+1$ [16]. This facilitates an incremental approach when generating symmetry breaking constraints.

Heule [14] 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 .

Table 1 illustrates (some of) the current “state of the art” for partial and complete lexleader static symmetry breaks. All of the results presented in this table are obtained within a 24 hour cutoff. The missing entries are timeouts. On the left, the columns titled **trans** are about symmetry breaks based on transpositions as presented in [10, 9]. These are the most widely applied partial symmetry breaks because (1) they are scalable (polynomial in size), and (2) they are easy to implement by imposing a lexicographic order on the rows (or columns) of the adjacency matrix. The problem with this symmetry break is that it is very imprecise.

On the right, the column titled **comp** is about the complete symmetry breaks described in [5]. These are complete, so their redundancy ratio is 1.00. We have crossed out the entry for $n = 11$. This result is given in [5] where it is reported to take 8.44 days to compute.

In the middle, the columns titled **con inv**, **disj inv**, and **inv** are about symmetry breaks based on graph patterns derived from specific types of permutations: consecutive, disjoint involutions and involutions, as defined in [8]. For each of the symmetry breaks, the columns titled “size” report the number of lex constraints of the form $G <^i \pi(G)$. The columns titled “ ρ ” detail the redundancy ratio of the corresponding symmetry break. For smaller values of n , ρ is computed using an exact model counter **ganak-2.5.2** [25], and for larger values of n , ρ is computed using an approximate model counter **approx-4.2** [27]. In the latter case, the value is annotated in the table by the symbol “ $\approx$ ”.

Table 1: State-of-the-art symmetry breaks compared in terms of size (number of lex constraints $G <^i \pi(G)$) and redundancy ratio (sometimes approximately computed).

n	trans [9]		con inv [8]		disj inv [8]		inv [8]		comp [5]
	size	ρ	size	ρ	size	ρ	size	ρ	size
4	6	1.00	6	1.00	6	1.00	6	1.00	6
5	13	1.26	13	1.06	15	1.06	16	1.00	17
6	24	1.77	29	1.16	32	1.12	39	1.00	42
7	40	3.02	64	1.46	72	1.31	113	1.01	127
8	62	5.39	134	1.95	166	1.53	399	1.02	473
9	91	9.42	269	2.76	399	1.83	2014	1.03	2826
10	128	15.34	526	3.97	996	2.20	12616	1.04	25263
11	174	23.52	1005	5.71	2499	2.65	81439	≈ 1.12	~~274109~~
12	230	≈ 32.47	1891	≈ 7.39	6244	≈ 3.02			
13	297	≈ 48.77	3511	≈ 12.19	15447	≈ 4.18			
14	376	≈ 69.44	6442	≈ 16.74	37721	≈ 4.26			
15	468	≈ 71.61	11693	≈ 19.96	90876	≈ 6.16			
16	574	≈ 106.25	21028	≈ 28.33					
17	695	≈ 191.28	37509	≈ 43.41					
18	832	≈ 221.25	66427	≈ 49.64					
19	986	≈ 256.27	116888	≈ 56.48					
20	1158	≈ 345.82							

Table 1 illustrates the challenge for computing (partial) symmetry breaks. Complete symmetry breaks quickly become impractical, while the simplest partial break **trans** quickly leads to high redundancy ratios. While involutions can generate very high quality partial symmetry breaks, they also quickly become challenging to compute. The strongest form **inv** hardly scales beyond the complete break; the disjunctive form **disj inv** scales a little further still with low redundancy; and the weakest form **con inv** scales better but generates high redundancy ratios as n grows.

3 Graph Patterns

Graph patterns were introduced in [7] and are similar to the notion of lex-implications introduced in [5]. 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 $\text{edges}(G)$. Formally, graph patterns derive from permutations as stated in the following definition which is adapted from [7].

Definition 5 (graph patterns [7]). *Let π be a permutation, G be an unknown graph of order- n with $\text{edges}(G) = \langle x_1, \dots, x_m \rangle$, $\text{edges}(\pi(G)) = \langle y_1, \dots, y_m \rangle$, and let $1 \leq i \leq m$. The graph pattern, $\text{pat}_i(\pi)$, is the (most general) solution of equations from the right side in Equation (1) for the case of $\text{edges}(\pi(G)) <_{lex}^i \text{edges}(G)$. We will refer to these equations as the “graph pattern equations”. If the equations have no solution, then we denote $\text{pat}_i(\pi) = \perp$.*

The graph pattern $p = \text{pat}_i(\pi)$ represents the set of (non-canonical) graphs G that get smaller at position i with π . Namely, graphs G such that $\pi(G) <^i G$. If p represents G we say that p covers G .

Table 2: Graph patterns for order-5 graphs with corresponding strengths and equations.

$\pi_v - i$	$p = \text{pat}_i(\pi_v)$	$\text{str}(p)$	π_e (prefix)
$[3, 2, 1, 4, 5] - 1$	$[1, A, 0, B, C, D, E, F, G, H]$	8	$[]$
$[3, 1, 2, 4, 5] - 1$	$[1, 0, A, B, C, D, E, F, G, H]$	8	$[]$
$[1, 2, 4, 3, 5] - 2$	$[A, 1, B, 0, C, D, E, F, G, H]$	8	$[1]$
$[1, 2, 3, 5, 4] - 4$	$[A, B, C, 1, D, E, 0, F, G, H]$	8	$[1, 2, 3]$
$[3, 5, 4, 2, 1] - 2$	$[A, 1, B, C, D, 0, E, F, A, G]$	7	$[9]$
$[2, 3, 1, 4, 5] - 2$	$[0, 1, 0, A, B, C, D, E, F, G]$	7	$[3]$
$[2, 1, 3, 4, 5] - 4$	$[A, B, B, 1, 0, C, D, E, F, G]$	7	$[1, 3, 2]$
$[1, 2, 3, 5, 4] - 5$	$[A, B, C, D, 1, E, D, 0, F, G]$	7	$[1, 2, 3, 7]$
$[4, 3, 1, 2, 5] - 3$	$[A, 0, 1, 0, B, A, C, D, E, F]$	6	$[6, 4]$
$[2, 4, 1, 3, 5] - 3$	$[A, A, 1, 0, A, B, C, D, E, F]$	6	$[5, 1]$
$[3, 1, 2, 4, 5] - 4$	$[A, A, A, 1, B, 0, C, D, E, F]$	6	$[2, 3, 1]$
$[2, 1, 4, 3, 5] - 7$	$[A, B, C, C, B, D, 1, 0, E, F]$	6	$[1, 5, 4, 3, 2, 6]$
$[1, 3, 2, 4, 5] - 8$	$[A, A, B, C, D, D, E, 1, 0, F]$	6	$[2, 1, 3, 4, 6, 5, 7]$
$[1, 2, 4, 3, 5] - 9$	$[A, B, C, B, C, D, E, F, 1, 0]$	6	$[1, 4, 5, 2, 3, 6, 7, 8]$
$[3, 1, 4, 2, 5] - 5$	$[0, 0, A, A, 1, 0, B, C, D, E]$	5	$[2, 6, 4, 3]$
$[2, 1, 4, 3, 5] - 9$	$[A, B, C, C, B, D, E, E, 1, 0]$	5	$[1, 5, 4, 3, 2, 6, 8, 7]$
$[2, 3, 5, 1, 4] - 5$	$[A, 0, A, A, 1, B, C, 0, A, D]$	4	$[3, 8, 9, 1]$
$[3, 5, 1, 2, 4] - 6$	$[0, A, B, B, C, 1, B, C, 0, D]$	4	$[9, 2, 7, 3, 8]$

Example 6. Consider the two graphs, G_1 and G_2 from Example 1. As stated in Example 4, for $\pi = [1, 2, 4, 3]$, $G_2 = \pi(G_1)$ and as demonstrated in Example 3,

$$G_1 <^3 G_2 \Leftrightarrow \{ (x_1=x_1), (x_2=x_4), (x_3=0), (x_5=1) \}.$$

Definition 5 sets $\text{pat}_3(\pi) = [x_1, x_2, 0, x_2, 1, x_6]$ as determined by the (most general) solution of the above graph pattern equations.

Note that $\text{pat}_i(\pi)$ is always a legal graph pattern (represents only non-canonical graphs) because graphs that get smaller under some permutation are trivially non-canonical. Table 2 details, a set of 18 graph patterns for order-5 graphs. These 18 graph patterns form a complete symmetry break for order-5 graphs. Namely, their instances are exactly all of the corresponding non-canonical graphs. The first column specifies a pair consisting of a vertex permutation π_v and an index i . The second column details the corresponding graph pattern $\text{pat}_i(\pi_v)$.

The strength of a graph pattern $p = \text{pat}_i(\pi_v)$, denoted $\text{str}(p)$, is a measure of the number of (non-canonical) graphs that it represents. More precisely, it is the $\log$ of this number, or equivalently, it is the number of (distinct) variables in p . The strongest graph patterns are of strength $\binom{n}{2} - 2$. The third column of Table 2 details the strength of the corresponding graph patterns which is precisely the number of distinct variables in each $p = \text{pat}_i(\pi_v)$. Every instance of p is a non-canonical graph. The rows of Table 2 are partitioned to highlight different patterns of the same strength. It is always the case that $\text{str}(p) \leq \binom{n}{2} - 2$. This is because, by Definition 5, p will always contain at least one “1” and at least one “0”.

The fourth column of Table 2, titled π_e (prefix), specifies the length $i - 1$ prefix of the edge permutation π_e corresponding to the vertex permutation π_v . This prefix determines a set of graph pattern equations. For example, read the prefix $[9, 2, 7, 3, 8]$ from the last row in the table as the set of equations: $\text{eqs}(p) = \{x_1=x_9, x_2=x_2, x_3=x_7, x_4=x_3, x_5=x_8\}$ (without the two

equations assigning the i^{th} edges from G and $\pi(G)$ to 1 and 0). These equations determine which positions in p are equal.

More formally, let $\pi_v - i$ specify a graph pattern p and let π_e be the edge permutation corresponding to π_v . Then the set of graph pattern equations which determine p is $eqs(p) = \{ x_j = x_{\pi_e(j)} \mid 1 \leq j < i \}$.

Question: What weakens a graph pattern $p = \text{pat}_i(\pi)$?

Answer: Each equivalence class in $eqs(p)$ of size k replaces k distinct variables by a single representative in p .

Example 7. Consider the graph pattern $p = [0, 0, A, A, 1, 0, B, C, D, E]$ derived from $\pi_v = [3, 1, 4, 2, 5]$ and $i = 5$ detailed in Table 2. We have $\text{str}(p) = 5$ and $\sigma = [2, 6, 4, 3]$ the length 4 prefix of π_e . Here, the equations derived from σ introduce the equivalence classes: $\{ \{x_1, x_2, x_6\}, \{x_3, x_4\} \}$. The equivalence classes indicate that 3 variables are eliminated from p (2 because of the first equivalence class and 1 because of the second) and so, the strength of p is $10 - 2 - 3 = 5$.

Incrementality: An interesting property of lex-leader canonical graphs, when lexicographic order is defined in terms of the column-wise concatenation of edge variables, is the following: An order- n canonical graph contains a canonical subgraph on the first k vertices for every $1 \leq k \leq n$ [19]. As shown in [16], this property can be exploited when generating symmetry breaking constraints, as the symmetry break for order- n graphs can be extended to one for order- $n + 1$ graphs. The following demonstrates how this applies to graph-patterns.

Example 8. Observe in Table 2, the 9 graph patterns $\text{pat}_i(\pi)$ where $\pi(5) = 5$ and $i \leq 6$. These are extensions of the following 9 graph patterns which are a complete symmetry break for order-4 graphs.

$[3, 2, 1, 4] - 1$	$[1, A, 0, B, C, D]$	$[4, 3, 1, 2] - 3$	$[A, 0, 1, 0, B, A]$
$[3, 1, 2, 4] - 1$	$[1, 0, A, B, C, D]$	$[2, 4, 1, 3] - 3$	$[A, A, 1, 0, A, B]$
$[1, 2, 4, 3] - 2$	$[A, 1, B, 0, C, D]$	$[3, 1, 2, 4] - 4$	$[A, A, A, 1, B, 0]$
$[2, 3, 1, 4] - 2$	$[0, 1, 0, A, B, C]$	$[3, 1, 4, 2] - 5$	$[0, 0, A, A, 1, 0]$
$[2, 1, 3, 4] - 4$	$[A, B, B, 1, 0, C]$		

CEGAR: In [15] and in [5] the authors compute complete symmetry breaks for graphs which are based on permutations and on a refinement of permutations which they term “implications,” respectively. A similar approach is applied in [12] to break the symmetries for finite models. In [8], the authors adapt this approach to compute symmetry breaks consisting of graph patterns. Common to all of these works is an algorithm based on *counter-example guided abstraction refinement (CEGAR)* [4]. In a nutshell, in the context of graph patterns, the CEGAR-based algorithm performs as follows: Let Ψ denote a set of graph patterns, 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: a graph G and a graph pattern $\text{pat}_i(\pi)$ such that G is covered by $\text{pat}_i(\pi)$ but is not covered by any of the graph patterns in Ψ . If such a counter-example is found then $\Psi = \Psi \cup \{\text{pat}_i(\pi)\}$. If no such counter-example is found, then Ψ is a complete symmetry break. The search for counter-examples is implemented using a SAT encoding and incremental SAT solving.

To encode that G is covered by a pattern $\text{pat}_i(\pi)$, denoting $\text{edges}(G) = \langle x_1, \dots, x_m \rangle$ and $\text{edges}(\pi(G)) = \langle y_1, \dots, y_m \rangle$, we introduce clauses to capture that

$$(x_1 = y_1) \wedge \dots \wedge (x_{i-1} = y_{i-1}) \wedge (x_i = 1) \wedge (y_i = 0). \quad (2)$$

Each equation $(x_i = y_i)$ is encoded in the CNF so that the atom $(x_i = y_i)$ can be viewed as a propositional variable. To encode that G is not covered by the pattern $\text{pat}_i(\pi)$, we take the negation of Equation (2) which, given the encodings of the equations $(x_i = y_i)$, is a single CNF clause. In this paper we make extensive use of a CEGAR-based algorithm.

4 Approximating the Strength of Graph Patterns

The aim in this paper is devise a SAT based approach to construct partial symmetry breaks by repeatedly selecting graph patterns, strongest first. However, a SAT encoding to specify the strength of an unknown graph pattern p is complex due to the need to capture the equivalence classes of the equations $\text{eqs}(p)$. This requires an encoding of the transitive closure of the unknown set of equations which is costly. As an alternative, we encode an approximation of $\text{str}(p)$. Underlying the approach is an analysis of a prefix of the edge permutation which governs the specification of these equations. To this end, we analyze the positions j in the domain of π_e which occur before i and reason about how they impact the strength of the pattern. We first demonstrate with an example.

Example 9. Consider again, the graph pattern $p = [0, 0, A, A, 1, 0, B, C, D, E]$ of strength 5 detailed in Table 2 which derives from the vertex permutation $\pi_v = [3, 1, 4, 2, 5]$ with index $i = 5$ (see also Example 7). The length 4 prefix, $[2, 6, 4, 3]$ of the corresponding edge permutation gives rise to 4 equations: equations $x_3 = x_4$ and $x_4 = x_3$ involve a pair of variables prior to $i = 5$ and corresponds to a swap in π_e ; equation $x_1 = x_2$ involves a pair of variables prior to i which is not a swap, and equation $x_2 = x_6$ involves a single variable prior to i .

Let $p = \text{pat}_i(\pi_v)$ be a graph pattern, let σ be the length $i - 1$ prefix of a permutation π_e which corresponds to π_v . We distinguish between four types of values in σ . We say that values j and k in σ are *pre- i -swap* if such that $j < k < i$, $\pi_e(j) = k$ and $\pi_e(k) = j$. The value $j < i$ in σ is *pre- i -lower* if $\pi_e(j) < i$ and $\pi_e(\pi_e(j)) \neq j$. The value $j < i$ in σ is *pre- i -upper* if $\pi_e(j) > i$. All other values $j < i$ in σ are *pre- i -fixed-point*, $\pi(j) = j$.

Each pair of pre- i -swap values $j < k$ in π_e introduces exactly one equivalence (between the variables x_j and x_k) and weakens the value of $\text{str}(p)$ by one. Each pre- i -upper value j in π_e introduces exactly one equivalence (between x_j and a variable with index greater than i) and weakens the value of $\text{str}(p)$ by one. Each pre- i -fixed-point value in π_e introduces zero equivalences and does not influence the value of $\text{str}(p)$. A pre- i -lower value $j < i$ in π_e introduces at most one equivalence, depending on the size of the path or cycle in which j resides.

Table 3 enhances the graph patterns detailed in Table 2 with the number of positions of three types: pre-swap, pre-lower, and pre-upper. All other positions are pre-fixed-point positions. Observe in the table that for all of the rows except that for the pattern $[3, 1, 2, 4, 5] - 4$ we have

$$\text{str}(p) = \binom{n}{2} - 2 - \#\text{pre-swaps} - \#\text{pre-lower} - \#\text{pre-upper}$$

We take the approach that, minimizing the values of $(\#\text{pre-swaps} + \#\text{pre-lower} + \#\text{pre-upper})$ is an approximation of maximizing the value of $\text{str}(p)$.

Definition 10 ($\text{equivs}_n(s_1, s_2, s_3)$). Let $\text{equivs}_n(s_1, s_2, s_3)$ denote the set of order- n graph patterns p , such that the number of pre-swap (pairs), and the numbers of pre-lower and pre-upper (values) in p are at most s_1, s_2 , and s_3 respectively. When clear from the context we omit the subscript n . If pattern p belongs to $\text{equivs}_n(s_1, s_2, s_3)$, we say that p is $\text{equivs}_n(s_1, s_2, s_3)$.

Table 3: Graph patterns for order-5 graphs with corresponding strengths and types of positions (note for swaps we count pairs).

$\pi - i$	$p = \text{pat}_i(\pi)$	$\text{str}(p)$	pre-swaps	pre-lower	pre-upper
$[3, 2, 1, 4, 5] - 1$	$[1, A, 0, B, C, D, E, F, G, H]$	8	0	0	0
$[3, 1, 2, 4, 5] - 1$	$[1, 0, A, B, C, D, E, F, G, H]$	8	0	0	0
$[1, 2, 4, 3, 5] - 2$	$[A, 1, B, 0, C, D, E, F, G, H]$	8	0	0	0
$[1, 2, 3, 5, 4] - 4$	$[A, B, C, 1, D, E, 0, F, G, H]$	8	0	0	0
$[3, 5, 4, 2, 1] - 2$	$[A, 1, B, C, D, 0, E, F, A, G]$	7	0	0	1
$[2, 3, 1, 4, 5] - 2$	$[0, 1, 0, A, B, C, D, E, F, G]$	7	0	0	1
$[2, 1, 3, 4, 5] - 4$	$[A, B, B, 1, 0, C, D, E, F, G]$	7	1	0	0
$[1, 2, 3, 5, 4] - 5$	$[A, B, C, D, 1, E, D, 0, F, G]$	7	0	0	1
$[4, 3, 1, 2, 5] - 3$	$[A, 0, 1, 0, B, A, C, D, E, F]$	6	0	0	2
$[2, 4, 1, 3, 5] - 3$	$[A, A, 1, 0, A, B, C, D, E, F]$	6	0	1	1
$[3, 1, 2, 4, 5] - 4$	$[A, A, A, 1, B, 0, C, D, E, F]$	6	0	3	0
$[2, 1, 4, 3, 5] - 7$	$[A, B, C, C, B, D, 1, 0, E, F]$	6	2	0	0
$[1, 3, 2, 4, 5] - 8$	$[A, A, B, C, D, D, E, 1, 0, F]$	6	2	0	0
$[1, 2, 4, 3, 5] - 9$	$[A, B, C, B, C, D, E, F, 1, 0]$	6	2	0	0
$[3, 1, 4, 2, 5] - 5$	$[0, 0, A, A, 1, 0, B, C, D, E]$	5	1	1	1
$[2, 1, 4, 3, 5] - 9$	$[A, B, C, C, B, D, E, E, 1, 0]$	5	3	0	0
$[2, 3, 5, 1, 4] - 5$	$[A, 0, A, A, 1, B, C, 0, A, D]$	4	0	2	2
$[3, 5, 1, 2, 4] - 6$	$[0, A, B, B, C, 1, B, C, 0, D]$	4	0	1	3

Encoding $\text{equivs}_n(s_1, s_2, s_3)$: Let n , i , and s_1, s_2, s_3 be given integer values such that $1 \leq i, s_1, s_2, s_3 \leq m$ where $m = \binom{n}{2}$. The SAT encoding that specifies that π_v is an order- n permutation such that $\pi_v - i$ corresponds to a graph pattern in $\text{equivs}_n(s_1, s_2, s_3)$ is detailed in Equations (3–5). In our implementation, the constraints from these equations are compiled to CNF using the finite-domain constraint compiler BEE [21].

Step one (*channelling*). First we express in Equation (3) that π_v and π_e are corresponding vertex and edge permutations. Both permutations are represented as lists of integer variables which are constrained to be all different.

$$\begin{array}{ll}
\pi_v = [p_1, \dots, p_n] & \pi_e = [q_1, \dots, q_m] \\
\bigwedge_{i=1}^n \text{integer}(p_i, 1, n) & \bigwedge_{i=1}^m \text{integer}(q_i, 1, m) \\
\text{allDifferent}(p_1, \dots, p_n) & \text{allDifferent}(q_1, \dots, q_m)
\end{array} \tag{3}$$

The integer variables in π_v specify a mapping $[1, \dots, n] \mapsto [p_1, \dots, p_n]$ (the p_i are variables). We constrain π_e to express a corresponding mapping $[1 \dots, m] \mapsto [q_1, \dots, q_m]$. To this end, denote $\text{id}_e = [(1, 2), (1, 3), (2, 3), \dots, (n-1, n)]$. This specifies the edge positions in the column-wise order. So, $q_i = j$ if π_v maps the i^{th} element of id_e to the j^{th} element of id_e as specified by the constraints in Equation (4).

$$\bigwedge_{i=1}^m \bigwedge_{j=1}^m q_i = j \leftrightarrow \text{id}_e[i] = (u_i, v_i) \wedge \text{id}_e[j] = (u_j, v_j) \wedge (p_{u_i} = u_j \wedge p_{v_i} = v_j) \vee (p_{u_i} = v_j \wedge p_{v_i} = u_j) \tag{4}$$

Step two ($\text{equivs}(s_1, s_2, s_3)$). Now we encode the parameters s_1, s_2, s_3 corresponding to the

numbers of pre-swaps, pre-lower and pre-upper positions in the graph pattern corresponding to $\pi_v - i$. Let $\pi_e = [q_1, \dots, q_m]$.

$$\begin{aligned}
\bigwedge_{a < b < i} x_{a,b} &\leftrightarrow q_a = b \wedge q_b = a & \sum_{a < b < i} x_{a,b} &\leq s_1 \\
\bigwedge_{a \leq b \leq i} y_{a,b} &\leftrightarrow q_a = b \text{ xor } q_b = a & \sum_{a < b < i} y_{a,b} &\leq s_2 \\
\bigwedge_{a < i} z_a &\leftrightarrow q_a > i & \sum_{a < i} z_a &\leq s_3
\end{aligned} \tag{5}$$

5 Strength-driven, Redundancy-aware Pattern Generation

We define a series of partial symmetry breaks $\Psi_1^n, \dots, \Psi_k^n$ for order- n graphs defined in terms of sets of graph patterns $\ell_1^n, \dots, \ell_k^n$ where ℓ_1^n is the set of all order- n transpositions and for $j > 1$, $\ell_j^n = \text{equivs}_n(s_1^j, s_2^j, s_3^j)$. The sequence of partial symmetry breaks can be viewed as a sequence of layers: each element is computed “on top” of the previous element.

To compute the symmetry breaks, we start from an empty set of graph patterns, $\Psi_0^n = \emptyset$ and apply in iteration, where j takes values from 1 to k :

$$\Psi_j^n = \text{generate}(\ell_j^n)$$

The procedure $\text{generate}(\ell_j^n)$ is CEGAR-based. It initializes Ψ_j^n with the graph patterns from the previous layer Ψ_{j-1}^n and then, so long as there exists a counter-example from ℓ_j^n to the fact that Ψ_j^n is not a complete symmetry break, it adds the corresponding graph pattern to Ψ_j^n .

We note that when the computation of Ψ_j^n terminates, Ψ_j^n is most likely not a complete symmetry break. However, no counter-example from ℓ_j^n exists. We also note that the set of graph patterns, ℓ_j^n , is not represented explicitly. It is represented as a set of constraints (a propositional formula).

The idea is to design the sequence $\ell_1^n, \dots, \ell_k^n$ so that stronger graph patterns are selected first and also to make layered based computation feasible. Note that we could simply define the sequence $\langle \ell_k^n \rangle$ or the sequence $\langle \ell_1^n, \ell_k^n \rangle$ but this does not work computationally. The specific sequence considered in our experimentation is the following:

$$\begin{aligned}
&\text{trans}, \text{equivs}(3, 2, 2), \text{equivs}(4, 3, 3), \text{equivs}(5, 3, 2), \text{equivs}(6, 3, 3), \text{equivs}(7, 4, 3), \\
&\text{equivs}(8, 4, 4), \text{equivs}(9, 5, 4), \text{equivs}(10, 5, 4), \text{equivs}(11, 6, 5), \text{equivs}(12, 6, 5), \\
&\text{equivs}(13, 6, 6), \text{equivs}(14, 7, 6), \text{equivs}(15, 7, 7), \text{equivs}(16, 8, 7), \text{equivs}(17, 8, 8), \\
&\text{equivs}(18, 9, 9), \text{equivs}(19, 10, 9), \text{equivs}(20, 10, 10).
\end{aligned}$$

However, for values $n > 14$, we only have calculated the symmetry breaks up to $\text{equivs}(17, 8, 8)$. The general approach described above is the basis of our method, but in order to ensure efficient execution we need to make use of additional heuristics and considerations.

Initialization (and incrementality): In fact, because we have chosen to order the edge variables by column, we can do better at the initialization phase in the definition of $\text{generate}(\ell_j^n)$. For $n > 4$, we initialize $\Psi_j^n = \Psi_{j-1}^n \cup \text{extend}(\Psi_{j-1}^{n-1})$ where each graph pattern of order- $n-1$ is extended to one of order- n (by concatenating $n-1$ fresh variables) [16]. See also Example 8.

An inner layer of iteration (and a heuristic): In the implementation, when iterating over layers, when we compute $\text{generate}(\ell_j^n)$, we iterate over all values $1 \leq i \leq \binom{n}{2}$ where i is

the index of the corresponding graph patterns $\text{pat}_i(\pi)$ considered from ℓ_j^n . This simplifies the encoding of “is there a counter-example from ℓ_j^n ”. We encode, for each i : “is there a counter-example of the form $\text{pat}_i(\pi)$ from ℓ_j^n ” for the given i . In practice, especially for larger values of n , the majority of indices $1 \leq i \leq \binom{n}{2}$ do not lead to counter-examples (do not add new graph patterns). In practice, this leads to a large number of expensive SAT calls which are “unsat”. We observe for $n \leq 10$ that there are only a few select values of i for which there exist counter-examples of the form $\text{pat}_i(\pi)$ from ℓ_j^n . Moreover, the set of such indices remains, more or less, constant in ℓ_j^n as n increases. Hence, we introduce a heuristic: First we determine which values of i contribute graph patterns in the computations for smaller values $n \leq 10$. Then, for larger values of $n > 10$ we restrict the search to these specific values of i . It means that we might not be adding all graph patterns that could otherwise be detected at layer ℓ_j^n .

Keeping the symmetry breaks “small” (a heuristic): For the specific layers in our design, the number of graph patterns added at each iteration i for layer ℓ_j^n is usually small (less than hundreds). In some cases it is huge (thousands, tens of thousands). As a heuristic, we restrict the number of graph patterns added at each iteration i for layer ℓ_j^n to 1000. However, we do not take the first 1000 counter-examples. We take the last 1000 counter-examples. These have proven non-redundant given all of the previous ones.

The reduce procedure: An important detail regarding CEGAR-based algorithms is that Ψ may contain redundant elements. For example, if a graph pattern added at some point becomes redundant in view of those 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 [22, 20]). It is important to note that the time to perform the second phase is costly. In our implementation we apply the reduce algorithm between layers and also periodically inside layers (every $n^2 \times 40$ graph patterns derived).

On the selection of the specific layers: The specific values of s_1 , s_2 and s_3 chosen in the layers are the result of trial and error. We did not start from $\text{equivs}(1, 1, 1)$ as this resulted in very few graph patterns. Clearly a more principled investigation of the sequence is warranted.

On the strategy: It is important to note that selecting graph patterns “strongest first” is not guaranteed to make the best choice. Given a set of graph patterns already selected, a strong graph pattern may cover only a small number of additional graphs.

Summary: We call our approach “strength-driven”, as we select patterns with decreasing strength. We call it “redundancy-aware” because the CEGAR-based algorithm ensures that every graph pattern selected is not redundant (at the time it is generated).

Table 4 details the sizes and redundancy ratios for several of the symmetry breaks generated for order-10 graphs. On the left, transpositions which are currently the most widely applied partial symmetry break (with redundancy ratio 15.34). On the right, a complete symmetry break (with redundancy ratio 1.00). Notice that using $\text{equivs}(10, 5, 4)$ we obtain a symmetry break with less than 10% of the graph patterns required for a complete symmetry break with a redundancy ratio of 1.34. This means that when using this symmetry break we will obtain on average 1.34 graphs for every canonical solution.

6 Our Partial Symmetry Breaking Constraints

This section details properties of the partial symmetry breaks constructed: their precision (redundancy ratios), the times required to compute them, and their sizes.

Table 4: Partial and complete symmetry breaks for order-10 graphs.

	trans	equivs(10,5,4)	equivs(14,7,6)	equivs(15,7,7)	equivs(17,8,8)	complete
size	128	2273	3300	5802	7471	25263
ρ	15.34	1.34	1.21	1.14	1.07	1.00

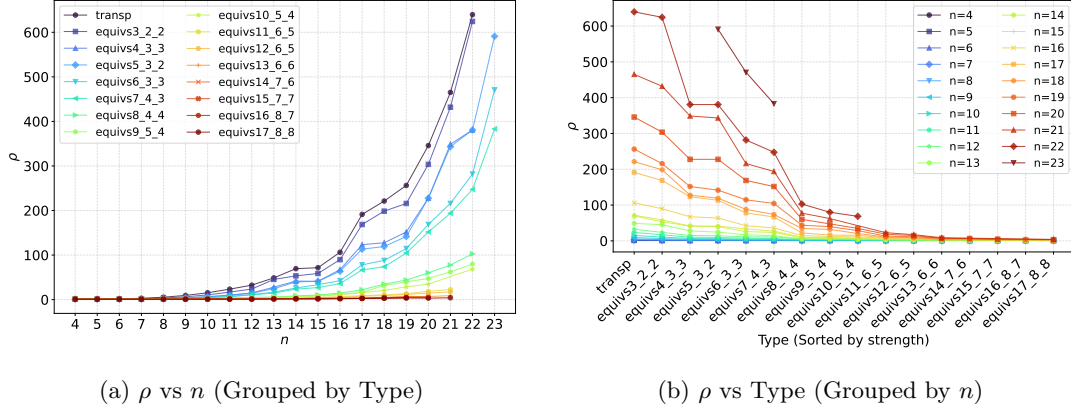
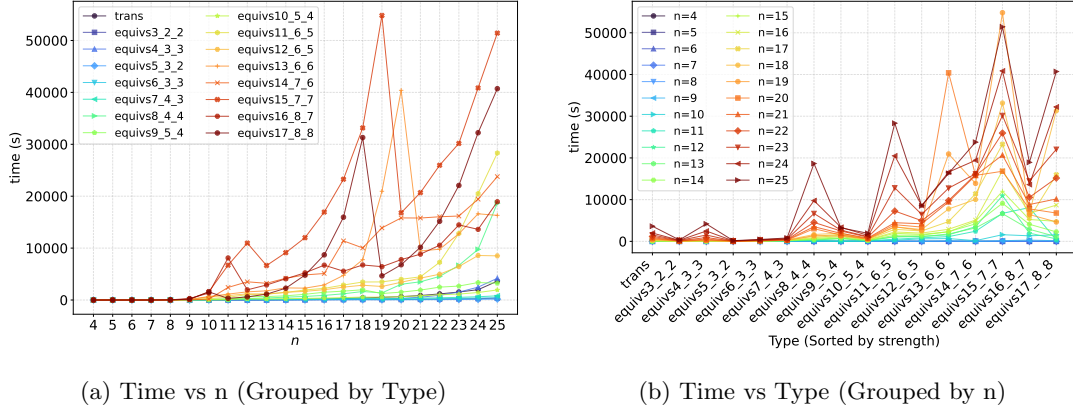
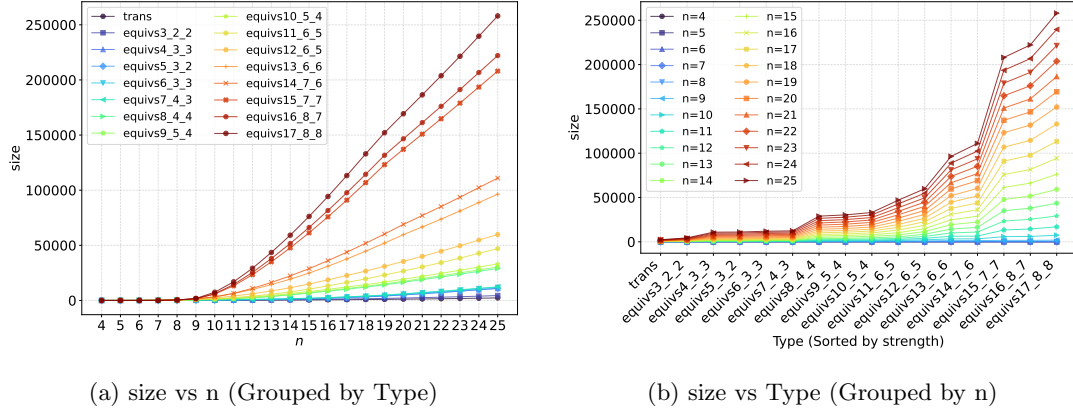
Figure 1: Values of ρ for different symmetry breaks and values of n .

Figure 1 shows redundancy ratios ρ for the different symmetry breaks and different values of n (number of vertices). The redundancy ratio is computed as the ratio between the number of graphs which are solutions of each corresponding symmetry break and the number of non-isomorphic graphs. The numerator is computed by calling a model counter to compute the number of graphs which are solutions of each corresponding symmetry break, and the value of the denominator is found as [23]. When possible with a 4 hour timeout the ratio is computed using an exact model counter **ganak-2.5.2** [25], otherwise it is computed using an approximate model counter **approx-4.2** [27].

Figure 1a shows how the values evolve for each symmetry break for different values of n . Figure 1b shows how the values evolve as the symmetry break gets stronger. In Figure 1a, the upper curve corresponds to the use of transpositions, as in [10, 9]. The lower curve corresponds to our most precise symmetry break. This figure makes it clear that the approach taken to derive symmetry breaks, “strongest-first”, results in symmetry breaks which evolve to give good precision. Missing points in the plots (for instance in the upper plot of Figure 1a) indicate that the corresponding SAT instances were too hard for the approximate model counter, so we cannot compute $\approx \rho$. Similarly, although both plots in Figure 1 detail values up till $n = 23$, symmetry breaks have been computed up to 25. In all cases we can compute the symmetry breaks and apply them.

Figure 2 shows the *incremental* times needed to produce the symmetry breaks for different values of n (number of vertices) for each layer—that is this is the time to complete the layer Ψ_j^n starting from $\Psi_{j-1}^n \cup \text{extend}(\Psi_j^{n-1})$.

Figure 2a shows how the time evolves for each symmetry break for different values of n . While usually the time grows with both n and the layer number j , it is not monotonic in either. This is potentially a result of the differences between subsequent layers not being uniform. Figure 2b shows how the time evolves as the symmetry break gets stronger. Clearly in general while time required grows with layers it is not uniform, again as a result of the fact that the

Figure 2: Time to calculate the sym. break for different symmetry breaks and values of n .Figure 3: The size of the sym. break for different symmetry breaks and values of n .

difference between subsequent layers are not uniform. Many of the peaks in Figure 2b correlate to the number of indices considered by the relevant indices heuristic. Analogously, Figure 3 shows the sizes of the symmetry breaks.

7 Evaluation

We consider two benchmark problems and illustrate the impact of our partial symmetry breaks when solving them. In both cases we compare to the partial symmetry break defined in terms of transpositions and to the use of dynamic symmetry breaking using [18]. For both benchmarks the encoding of the problem instances to SAT is performed using the SMS tool.¹ Then we couple the resulting CNF with other symmetry breaks. All the experiments were performed on a machine with two AMD EPYC 7513 32-Core processors and with 514 GiB RAM with 100 problem instances run in parallel with 4h timeout.

¹Obtained from <https://github.com/markirch/sat-modulo-symmetries> under revision cf890fd, using the tool's options `--no-solve --cnf FILE` to generate a CNF file.

Table 5: Diameter 2 Critical comparison

n	SMS		Static-best			Static Trans.		
	models	time (s)	models	time (s)	ratio	models	time (s)	ratio
4	2	0.00	2	0.00	1.00	2	0.01	1.00
5	3	0.00	3	0.00	1.00	4	0.00	1.33
6	5	0.00	5	0.01	1.00	11	0.01	2.20
7	10	0.00	10	0.01	1.00	32	0.01	3.20
8	30	0.02	30	0.02	1.00	163	0.02	5.43
9	103	0.11	103	0.07	1.00	1018	0.12	9.88
10	520	0.52	533	0.42	1.03	9727	1.62	18.71
11	3,748	2.10	4,051	4.18	1.08	133,316	32.03	35.57
12	40,868	23.34	48,115	41.47	1.18	t.o.	t.o.	t.o.
13	688,130	774.89	934,992	1088.75	1.36	t.o.	t.o.	t.o.

Diameter-2-Critical Graphs: The diameter of a graph G is the largest distance among all pairs of vertices in G , where the distance of two vertices is the length of a shortest path between them. A disconnected graph has diameter ∞ . A graph is diameter- d -critical if its diameter is d , but the deletion of any edge increases the diameter. In [18], the authors report that previous work on this problem provides results for graph of size $n \leq 11$ and that they were able to extend these results to graphs with $n \leq 13$ vertices. We refer the reader to [18] for details on the origin and history of the problem.

Table 5 shows results for the Diameter-2-critical problem. We compare static symmetry breaks with our best symmetry break and using a symmetry break based on transpositions and the computation using the SMS [18] tool. For the static symmetry breaks the models were enumerated by CaDiCaL [2] using a blocking clause mechanism. Times are given in seconds. The two columns titled ratio detail the ratio between the number of models obtained and the actual number of models (found using SMS). For this experiment we have used partial symmetry breaks (static-best) based on *equivs*(20,10,10). Comparing the two static symmetry breaks, “Static-best” is superior both in precision and in time. The dynamic approach using SMS is superior to static symmetry breaking.

Extremal Graphs with Required Girth: An extremal graph theory problem: What is the maximal number of edges in a graph with v vertices and no cycles of length 4 or less (i.e. graphs with girth at least 5). Let $f_4(n)$ denote the maximum number of edges in an order- n graph with girth at least 5. We consider the same encoding as applied in [9] and in [18]. We consider the largest satisfiable instances which contain $f_4(n)$ edges and the smallest unsatisfiable instances which contain $f_4(n) + 1$ edges. In both cases the problem is partitioned depending on values of pairs (δ, Δ) which denote the minimal and maximal degrees of vertices in the solutions (if they exist). These pairs are determined by known constraints related to the problem. For a satisfiable instance with n vertices and m edges, we take the minimal time over the corresponding pairs. For an unsatisfiable instance with n vertices and m edges, we take the maximal time over the corresponding pairs. This is the convention set in [9] and in [18]. We refer the reader to those papers for details on the encoding.

Table 6 shows the time comparison (in seconds) between dynamic symmetry break (SMS), transpositions, and our best symmetry break (*equivs*(17,8,8)). We consider values of $n \leq 25$, corresponding to those for which we have constructed static symmetry breaks. The cells

Table 6: Extremal Graphs Comparison

n	SAT			UNSAT		
	SMS	Static-best	Trans	SMS	Static-best	Trans
15	0.16	0.97	0.13	trivial	trivial	trivial
16	0.20	1.01	0.14	0.52	3.72	10.85
17	0.49	0.93	0.23	trivial	trivial	trivial
18	0.53	2.17	0.55	trivial	trivial	trivial
19	0.14	0.95	0.24	trivial	trivial	trivial
20	1.80	4.26	2.46	trivial	trivial	trivial
21	0.59	2.93	0.85	0.85	7.66	236.90
22	3.44	3.71	3.33	3.00	147.93	2672.83
23	4.49	14.01	0.05	10.79	3718.25	t.o.
24	12.38	30.87	17.54	trivial	trivial	trivial
25	90.34	28.37	5.18	402.31	t.o.	t.o.

marked with `trivial` indicate that an `unsat` result follows trivially from theoretical bounds. The solver Kissat [3] was used to decide the SAT queries. For satisfiable instances, all three of the techniques are similar. For the unsatisfiable instances, the stronger static symmetry break is superior to the use of static symmetry break based on transpositions. The dynamic approach using SMS is superior to static symmetry breaking.

8 Conclusion

Symmetry breaking for graph search problems is challenging: ignoring symmetries leads to vast amounts of redundant work, while removing all symmetries is impossible for large n . Hence partial symmetry breaks are an important approach to practical symmetry breaking for graph search. In this paper we develop a method for creating “small” partial symmetry breaking constraints which nonetheless have low redundancy ratio, and importantly are not too difficult to compute. The method substantially improves on earlier methods for partial symmetry breaking for graph search problems.

The symmetry breaking constraints constructed in this paper are solver independent. They can be applied when solving graph search problems via SAT encodings, constraint solvers, or SMT solvers. In addition, they are problem independent. They can be applied when solving any graph search problem. It is important to note that while dynamic symmetry breaking (e.g. SMS [18]) has its advantages (as indicted also by the benchmarks in Section 7), it requires adapting an existing solver substantially.

Acknowledgements

The research was supported by the European Union under the project ROBOPROX (reg. no. CZ.02.01.01/00/22_008/0004590) and is part of the RICAIP project that has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 857306.

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.
- [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.
- [3] Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL, Gimsatul, IsaSAT and Kissat entering the SAT Competition 2024. In Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2024 – Solver, Benchmark and Proof Checker Descriptions*, volume B-2024-1 of *Department of Computer Science Report Series B*, pages 8–10. University of Helsinki, 2024.
- [4] 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.
- [5] 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.
- [6] 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.
- [7] 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.
- [8] Michael Codish and Mikoláš Janota. Breaking symmetries with involutions. In Maria Garcia de la Banda, editor, *31st International Conference on Principles and Practice of Constraint Programming, CP 2025, Glasgow, Scotland, August 10-15, 2025*, volume 340 of *LIPICs*, pages 8:1–8:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.
- [9] Michael Codish, Alice Miller, Patrick Prosser, and Peter J. Stuckey. Constraints for symmetry breaking in graph representation. *Constraints*, 24(1):1–24, 2019.
- [10] 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.
- [11] 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.
- [12] Marek Dančo, Mikoláš Janota, Michael Codish, and João Jorge Araújo. Complete symmetry breaking for finite models. In *AAAI 2025*, 2025.
- [13] Alan M. Frisch, Chris Jefferson, and Ian Miguel. *Constraints for Breaking More Row and Column Symmetries*, page 318–332. Springer Berlin Heidelberg, 2003.
- [14] Marijn J. H. Heule. Optimal symmetry breaking for graph problems. *Math. Comput. Sci.*, 13(4):533–548, 2019.
- [15] Avraham Itzhakov and Michael Codish. Breaking symmetries in graph search with canonizing sets. *Constraints*, pages 1–18, 2016.

- [16] Avraham Itzhakov and Michael Codish. Incremental symmetry breaking constraints for graph search problems. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 1536–1543. AAAI Press, 2020.
- [17] 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.
- [18] Markus Kirchweger and Stefan Szeider. SAT modulo symmetries for graph generation and enumeration. *ACM Trans. Comput. Log.*, 25(3):1–30, 2024.
- [19] Vladmír Kvasnička and Jiří Pospíchal. Canonical indexing and constructive enumeration of molecular graphs. *J. Chem. Inf. Comput. Sci.*, 30(2):99–105, April 1990.
- [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.
- [21] Amit Metodi, Michael Codish, and Peter J. Stuckey. Boolean equi-propagation for concise and efficient SAT encodings of combinatorial problems. *J. Artif. Intell. Res. (JAIR)*, 46:303–341, 2013.
- [22] Alexander Nöhler, 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.
- [23] OEIS Foundation Inc. The On-Line encyclopedia of integer sequences, A000088: Number of graphs on n unlabeled nodes. <https://oeis.org/A000088>, 2025.
- [24] 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.
- [25] Shubham Sharma, Subhajit Roy, Mate Soos, and Kuldeep S. Meel. GANAK: A scalable probabilistic exact model counter. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.
- [26] Ilya Shlyakhter. Generating effective symmetry-breaking predicates for search problems. *Discrete Applied Mathematics*, 155(12):1539–1548, 2007.
- [27] 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.
- [28] 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.