SAT Modulo Symmetries for Graph Generation and Enumeration

MARKUS KIRCHWEGER and STEFAN SZEIDER, Algorithms and Complexity Group, TU Wien, Vienna, Austria

We propose a novel SAT-based approach to graph generation. Our approach utilizes the interaction between a CDCL SAT solver and a special symmetry propagator where the SAT solver runs on an encoding of the desired graph property. The symmetry propagator checks partially generated graphs for minimality with respect to a lexicographic ordering during the solving process. This approach has several advantages over a static symmetry breaking: (i) symmetries are detected early in the generation process, (ii) symmetry breaking is seamlessly integrated into the CDCL procedure, and (iii) the propagator performs a complete symmetry breaking without causing a prohibitively large initial encoding. We instantiate our approach by generating extremal graphs with certain restrictions in terms of forbidden subgraphs and diameter. In particular, we could confirm the Murty–Simon Conjecture (1979) on diameter-2-critical graphs for graphs up to 19 vertices and prove the exact number of Ramsey graphs $\mathcal{R}(3,5,n)$ and $\mathcal{R}(4,4,n)$.

CCS Concepts: • **Theory of computation → Constraint and logic programming**; • **Mathematics of computing → Extremal graph theory**;

Additional Key Words and Phrases: Graph generation, dynamic symmetry breaking, SAT solving, Ramsey numbers, extremal graph theory, Murty–Simon Conjecture

ACM Reference format:
Markus Kirchweger and Stefan Szeider. 2024. SAT Modulo Symmetries for Graph Generation and Enumeration. *ACM Trans. Comput. Logic* 25, 3, Article 18 (July 2024), 30 pages.
https://doi.org/10.1145/3670405

1 Introduction

Many challenging problems in Combinatorics can be stated as the question of whether a graph with a particular property exists. Here, the property should be independent of the labeling of the vertices, i.e., an invariant of the graph. A common approach to such problems is to use a tool like Nauty [42] to generate all connected graphs up to isomorphism with a given number n of vertices, known as isomorph-free generation, and to check each of them for the desired property. However, up to isomorphism, already for $n = 11$ there are over a billion connected graphs, so this method quickly approaches its limit (see Table 1).

As demonstrated by Codish et al. [11], constraint-based graph generation offers a compelling alternative approach. The graph property is expressed in terms of constraints, and further constraints

The authors acknowledge the support from the Austrian Science Fund, project 10.55776/P36688, and the Vienna Science and Technology Fund (WWTF), project ICT19-065.
Authors' Contact Information: Markus Kirchweger, Algorithms and Complexity Group, TU Wien, Vienna, Austria; e-mail: mk@ac.tuwien.ac.at; Stefan Szeider (Corresponding author), Algorithms and Complexity Group, TU Wien, Vienna, Austria; e-mail: sz@ac.tuwien.ac.at.

ACM 1557-945X/2024/7-ART18
https://doi.org/10.1145/3670405

Table 1. Number of Connected Graphs and Number of Connected Graphs Up to Isomorphism for a Given Order [1, 2]

Order	Number of connected graphs	Number of connected graphs up to isomorphism
1	1	1
2	1	1
3	4	2
4	38	6
5	728	21
6	26,704	112
7	1,866,256	853
8	251,548,592	11,117
9	66,296,291,072	261,080
10	34,496,488,594,816	11,716,571
11	35,641,657,548,953,344	1,006,700,565
12	73,354,596,206,766,622,208	164,059,830,476

expressing a *static symmetry break* are added. The latter constraints are based on a lexicographic ordering of solution graphs and exclude some graphs that are not minimal for this ordering. This approach has the advantage that the graph property is taken into account already during the generation process. However, a polynomial-size, complete symmetry breaking is not known, i.e., breaking all symmetries statically with current methods would result in a prohibitively large encoding size. Therefore, for static symmetry breaking, one needs to confine to a partial check, e.g., that swapping two vertices doesn't yield a lexicographically smaller graph.

We propose the novel approach ***SAT Modulo Symmetries*** **(SMS)** to constraint-based graph generation and enumeration. SMS utilizes the interaction between a **Conflict-Driven Clause Learning (CDCL)**[1] SAT solver and a special propagator excluding lexicographically non-minimal graphs during the search. Thus, in contrast to static symmetry breaking, the minimality check is not added to the encoding but is carried out dynamically by the symmetry propagator.

The symmetry propagator is called from within a CDCL SAT solver already when only a few of the graph's edges are determined. For this purpose, we introduce *partially defined graphs*, the corresponding lexicographic ordering, and the algorithm MinCheck that checks their minimality. If the symmetry propagator detects that the current partially defined graph is not minimal, clauses are learned and added to the solver's collection of clauses. On partially defined graphs, the minimality check is not guaranteed to be complete, but eventually, when all edges are determined, it performs a complete minimality check. Consequently, we can guarantee that all generated graphs are minimal and unique up to isomorphism. MinCheck also detects, in some cases, whether the current partially defined graph implies under minimality the existence or non-existence of further edges. When the minimality check reveals that the current partially defined graph isn't minimal, this conflict is analyzed, and a suitable clause is added to the solver; also when an implied edge is detected, a unit clause is added.

We implemented a prototype version of SMS and tested it on three prominent problems from Extremal Graph Theory [6]. The first asks for graphs with a prescribed minimum *girth* (length of a shortest cycle) and the largest number of edges. The second asks for graphs whose *diameter* (largest distance between any two vertices) is 2 but increases when any edge is deleted. The third

[1]CDCL is the predominantly leading algorithmic paradigm for SAT solvers [41].

problem asks for the smallest number n of vertices, given x and y, such that each graph with n vertices contains either an independent set of size x or a clique of size y. These values for given x, y are called *Ramsey numbers*.

All of them are fundamental problems that have been studied for many decades [6, 17, 40, 48]. We could verify some extremal numbers for the girth problem, confirm the Murty–Simon Conjecture [7] on diameter-2-critical graphs with up to 19 vertices, improving upon the known bound of 11, and we were able to generate, up to isomorphism, all graphs without an independent set of size x and a clique of size y for $(x, y) \in \{(3, 5), (4, 4)\}$.

Moreover, we are able to generate proofs. The proof is split into two parts, one for verifying the reasoning of the CDCL solver, the second for verifying the correctness of the added symmetry breaking clauses.

Our experimental results show that SMS exhibits an encouraging performance, particularly on unsatisfiable instances. We developed the testing algorithm MinCheck from scratch, since existing methods based on various other forms of graph canonization cannot handle partially defined graphs [42]. The check often takes a notable amount of the solving time; here, we see ample room for improvement. However, the time which SMS spends on the SAT solving itself is significantly reduced in comparison to a static symmetry breaking.

This article is an extended version of an earlier conference paper [35]. Here, we give additional experiments, improve the presentation by including more examples, and restructure the theoretical part.

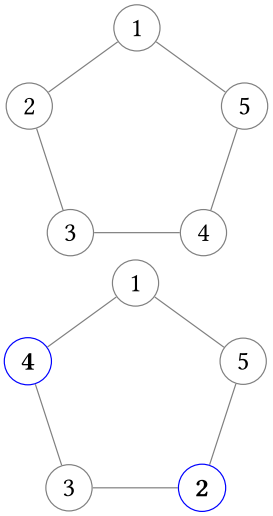
2 Preliminaries

2.1 Graphs

All considered graphs are undirected and simple (i.e., without parallel edges or self-loops). A *graph* G consists of a set $V(G)$ of vertices and a set $E(G)$ of edges; we denote the edge between vertices $u, v \in V(G)$ by uv or equivalently vu. The *order* of a graph G is the size of $V(G)$. We write $G - e$ for the graph obtained from G by deleting the edge e and $G - v$ for the graph obtained from G by deleting the vertex v. $\mathcal{G}_n$ denotes the class of all graphs G with $V(G) = \{1, \ldots, n\}$. A_G denotes the *adjacency matrix* of a graph $G \in \mathcal{G}_n$ where the element at row v and column u, denoted $A_G(v, u)$, is 1 if $vu \in E(G)$ and 0 otherwise. $A_G[v]$ denotes the vth row of A_G. $\mathcal{S}_n$ denotes the set of all permutations of $\{1, \ldots, n\}$. We use the cyclic notation for permutations in $\mathcal{S}_n$.

Graphs $G_1, G_2 \in \mathcal{G}_n$ are *isomorphic* if there is a permutation $\pi \in \mathcal{S}_n$ such that for all $1 \leq u < v \leq n$, we have $uv \in E(G_1)$ if and only if $\pi(u)\pi(v) \in E(G_2)$. $\pi(G)$ denotes the graph obtained from $G \in \mathcal{G}_n$ by the permutation $\pi \in \mathcal{S}_n$, where $E(\pi(G)) = \{\pi(u)\pi(v) : uv \in E(G)\}$. A *canonical form* of a graph $G \in \mathcal{G}_n$ is a graph $\mathrm{Can}(G) \in \mathcal{G}_n$ isomorphic to G, such that every graph $H \in \mathcal{G}_n$ isomorphic to G has the same canonical form, i.e., $\mathrm{Can}(G) = \mathrm{Can}(H)$. A graph is *canonical* if its canonical form is the graph itself. The total order $\preceq$ of $\mathcal{G}_n$ is defined by $G \preceq H$ if and only if $A_G[1]A_G[2] \ldots A_G[n]$ is *lexicographically smaller or equal* to $A_H[1]A_H[2] \ldots A_H[n]$ for $G, H \in \mathcal{G}_n$. G is *lexicographically smaller* than H (in symbols $G \prec H$) if $G \preceq H$ and $G \neq H$. $G \in \mathcal{G}_n$ is *lexicographically minimal* or $\preceq$*-minimal* if $G \preceq \pi(G)$ for every $\pi \in \mathcal{S}_n$. Figure 1 shows two different isomorphic copies of a five cycle, where the labeling on the top leads to a lexicographically smaller graph than the graph on the bottom.

We will also consider the lexicographic ordering of vertex pairs from $\{1, \ldots, n\}$, where $(v_1, v_2) \prec (u_1, u_2)$ if and only if either (i) $v_1 < u_1$ or (ii) $v_1 = u_1$ and $v_2 < u_2$. We will say that a vertex pair is *more important* than another if the vertex pair is smaller by this order. Similarly, we will say that an entry $A(v_1, v_2)$ of the adjacency matrix A is more important than $A(u_1, u_2)$ if $(v_1, v_2) \prec (u_1, u_2)$.



$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & \mathbf{0} & 0 & \mathbf{1} & 1 \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} \\ 0 & \mathbf{1} & 0 & \mathbf{1} & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ 1 & \mathbf{1} & 0 & \mathbf{0} & 0 \end{pmatrix}$$

Fig. 1. Two ways to label C_5 and their corresponding adjacency matrices.

Observation 1. *Let $G, H \in \mathcal{G}_n$. Then $G \prec H$ if and only if there are $1 \leq i, j \leq n$ such that $A_G(i, j) = 0$, $A_H(i, j) = 1$, and for all more important vertex pairs (i', j'), the values of the adjacency matrices are equal, i.e., $A_G(i', j') = A_H(i', j')$.*

2.2 Formulas and Satisfiability

A *literal* is a propositional variable or negated propositional variable. A *clause* is a disjunction of literals. A formula in **Conjunctive Normal Form (CNF)** is a conjunction of clauses. An *assignment* is a function $f : X \to \{\text{true}, \text{false}\}$ defined on a set X of propositional variables. For a variable $x \notin X$, we say that x is *undefined*. A negated variable is undefined if the variable itself is undefined. Assignments extend to literals in an obvious way. A *model* of a CNF formula F is an assignment f defined on the variables of F such that each clause of F contains a literal that is set to true by f. Given an assignment f, a clause is a *unit clause* if exactly one literal in the clause is undefined and all others are assigned to false.

2.3 Static Symmetry Breaking for Graph Generation

We outline a static symmetry breaking approach, which is essentially the "improved lexicographic break" proposed by Codish et al. [11]. The SAT solver runs on a CNF formula $F \wedge M$, where F encodes the properties of the graph sought for and M encodes a minimality property. A complete symmetry breaking would most likely require a prohibitively large encoding size. Therefore, M just ensures that swapping any two vertices does not lead to a lexicographically smaller graph (if it would, the current graph is not lexicographically minimal).

Permuting two vertices i, j results in swapping the entries $A_G(i, k)$ with $A_G(j, k)$, and $A_G(k, i)$ with $A_G(k, j)$, for $k \in \{1, \ldots, n\} \setminus i, j$, in the adjacency matrix. The remaining entries in the matrix will remain unchanged as the diagonal elements are all zero and the graph is undirected, meaning that $A_G(i, j) = A_G(j, i)$. So we have to check whether swapping the entries does not lead to a $\preceq$-smaller graph. Following [11], we compare the rows $A[i]$ and $A[j]$ by setting $A[i] \prec_{i,j} A[j]$ if and only if $A[i]$ is lexicographically smaller than $A[j]$ while ignoring the ith and jth elements in both rows.

Proposition 1 ([11]). *If $G \in \mathcal{G}_n$ is canonical, then $A_G[i] \preceq_{i,j} A_G[j]$ for all $1 \leq i < j \leq n$.*

We use the compact encoding of lex-leader constraints given by Devriendt et al. [15, Theorem 1] to encode M based on Proposition 1. The encoding is given by the following clauses using variables $y_{i,j,k}$ for $i, j \in \{1, \ldots, n\}, k \in \{0, \ldots, n\} \setminus \{i, j\}$ and a predecessor function $p : \{1, \ldots, n\} \setminus \{i, j\} \to \{0, \ldots, n-1\} \setminus \{i, j\}$ with $p(k) = \max(\{0, 1, \ldots, k-1\} \setminus \{i, j\})$

- $y_{i,j,k}$ for all $1 \leq i < j \leq n$, and $k = 0$.
- $\neg y_{i,j,p(k)} \vee \neg e_{i,k} \vee e_{j,k}$ for all $1 \leq i < j \leq n$, and $k \in \{1, \ldots, n\} \setminus \{i, j\}$.
- $\neg y_{i,j,p(k)} \vee \neg e_{i,k} \vee y_{i,j,k}$ for all $1 \leq i < j \leq n$, and $k \in \{1, \ldots, n-1\} \setminus \{i, j\}$.
- $\neg y_{i,j,p(k)} \vee e_{j,k} \vee y_{i,j,k}$ for all $1 \leq i < j \leq n$, and $k \in \{1, \ldots, n-1\} \setminus \{i, j\}$.

Informally, if the truth values of $e_{i,k'}$ and $e_{j,k'}$ coincide for all $k' \in \{1, \ldots, k\} \setminus \{i, j\}$, then the variable $y_{i,j,k}$ is true. If $y_{i,j,k}$, then the second case ensures $A_G(i, k) \preceq A_G(j, k)$.

Instead of this direct SAT encoding, Codish et al. [11] used the solver BEE to encode the property stated in Proposition 1. BEE [45] compiles finite domain constraints to SAT while additionally applying transformations to simplify the encoding and optimize it.

Recently, another incomplete static symmetry breaking for graph generation was introduced using higher dimensional graph invariants and combining different invariants [30].

3 Related Work

The goal of symmetry breaking in the context of combinatorial search is to restrict the search space by avoiding isomorphic solutions. In a broader sense, the goal is pruning the search space by adding clauses, which are not necessarily implied by the formula but preserve satisfiability. Depending on whether *symmetry breaking clauses* are added before search or during search, we distinguish between *static* and *dynamic* symmetry breaking.

Our method SMS falls into the category of dynamic symmetry breaking techniques. Dynamic symmetry breaking in the broader sense, has a long history, see, e.g., [5, 16, 18, 24, 46]. More recently, this has also been combined with nogood (or clause) learning [9, 14, 50], using the fact that if a new clause/nogood is learned, and the symmetries of the problem are known, then one can propagate and learn further clauses/nogoods. Metin et al. [44] explored another way of dynamically utilizing symmetries in the context of SAT by considering the lexicographic order on the assignments themselves; the symmetries are computed by external tools or provided by the user before the SAT solving and are then taken into account by the solver.

All these dynamic symmetry breaking techniques are not well suited for graph generation, because the number of permutations for a given order n is quickly increasing, and one has to restrict the symmetry breaking to a restricted set of permutations,[2] otherwise these techniques render themselves useless. This leads to an *incomplete* symmetry breaking, i.e., several isomorphic graphs are kept in the search space.

For graph generation, there exist complete static symmetry breaking constraints [10, 27–29]. These constraints are only practical for small objects, such as graphs with up to roughly 12 vertices.

Another approach is incomplete static symmetry breaking [11, 30] as discussed in Section 2.3, which scales better but is not as effective at filtering out symmetric copies.

4 Dynamic Symmetry Breaking in SMS

In this section, we present our approach for dramatically reducing the search space for finding all graphs in $\mathcal{G}_n$ modulo isomorphism that satisfy a given property. We will use CNF formulas that contain all the propositional variables $e_{u,v}$, for $1 \leq u < v \leq n$, which are true if and only if the edge uv is present in the implicitly represented graph, plus some auxiliary variables needed for

[2] For example, one can choose a generator of the symmetry group.

encoding the graph property. From a model of the formula, we can extract a graph satisfying the given property.

Our aim is to decide during the CDCL SAT solver's run whether the current partial assignment can be extended to a model, such that the graph represented by the model is $\preceq$-minimal. Performing this test is computationally hard in the worst case. For example, computing the lexicographically smallest isomorphic graph is known to be NP hard [4].

Since during solving, we don't know the final graph yet, we introduce the concept of *partially defined graphs*. A partially defined graph is a graph G where $E(G)$ is split into two disjoint sets $D(G)$ and $U(G)$. $D(G)$ contains the *defined* edges, $U(G)$ contains the *undefined* edges. A (fully defined) graph is a partially defined graph G with $U(G) = \emptyset$. Similarly to $\mathcal{G}_n$, let $\mathcal{P}_n$ denote the class of all partially defined graphs G with $V(G) = \{1, \ldots, n\}$. For two partially defined graphs $G_1, G_2 \in \mathcal{P}_n$, we write $G_1 \sqsubseteq G_2$ if $D(G_1) \subseteq D(G_2)$ and $U(G_2) \subseteq U(G_1)$. We write $G_1 \sqsubset G_2$ if $G_1 \sqsubseteq G_2$ and $G_1 \neq G_2$. The relation $\sqsubseteq$ is a partial order on P_n, having as its maximal elements the fully defined graphs in $\mathcal{P}_n$ and having one minimal element $G_\star$, the partially defined graph with all vertex pairs being undefined edges.

Analogously to the adjacency matrix of a fully defined graph, we define the adjacency matrix A_G of a partially defined graph $G \in \mathcal{P}_n$ as follows:

$$A_G(i,j) = \begin{cases} 1 & \text{if } ij \in D(G), \\ \star & \text{if } ij \in U(G), \\ 0 & \text{otherwise.} \end{cases}$$

From an assignment $f : X \to \{\text{true}, \text{false}\}$, defined on some set X of variables that may contain some of the edge variables, we can extract the partially defined graph G with $V(G) = \{1, \ldots, n\}$, $D(G) = \{\, ij : e_{i,j} \in X, f(e_{i,j}) = \text{true} \,\}$ and $U(G) = \{\, ij : 1 \leq i < j \leq n, e_{i,j} \notin X \,\}$.

Example 1. *Let the number of vertices be $n = 10$ and the assignment $f : X \to \{true, false\}$ with*

$$X = \{e_{1,8}, e_{1,9}, e_{1,10}, e_{2,4}, e_{2,5}, e_{2,6}, e_{2,9}, e_{2,10}, e_{3,5}, e_{1,2}, e_{1,3}, e_{1,4}, e_{1,5}, e_{2,3}, e_{2,7}, e_{2,8}, e_{4,5}, e_{5,6}\}$$

and

$$f(x) = \begin{cases} true & \text{if } x \in \{e_{1,8}, e_{1,9}, e_{1,10}, e_{2,4}, e_{2,5}, e_{2,6}, e_{2,9}, e_{2,10}, e_{3,5}\}, \\ false & \text{if } x \in \{e_{1,2}, e_{1,3}, e_{1,4}, e_{1,5}, e_{2,3}, e_{2,7}, e_{2,8}, e_{4,5}, e_{5,6}\}. \end{cases}$$

Then, the adjacency matrix of the corresponding partially defined graph G_{ex} is

$$A_{G_{ex}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & \star & \star & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & \star & 1 & \star & \star & \star & \star & \star \\ 0 & 1 & \star & 0 & 0 & \star & \star & \star & \star & \star \\ 0 & 1 & 1 & 0 & 0 & 0 & \star & \star & \star & \star \\ \star & 1 & \star & \star & 0 & 0 & \star & \star & \star & \star \\ \star & 0 & \star & \star & \star & \star & 0 & \star & \star & \star \\ 1 & 0 & \star & \star & \star & \star & \star & 0 & \star & \star \\ 1 & 1 & \star & \star & \star & \star & \star & \star & 0 & \star \\ 1 & 1 & \star & \star & \star & \star & \star & \star & \star & 0 \end{pmatrix}.$$

We use the assignment f and the graph G_{ex} as running example throughout this section to illustrate various concepts and algorithms.

A partially defined graph $G \in \mathcal{P}_n$ can be *extended* to a graph $H \in \mathcal{G}_n$ if $D(G) \subseteq E(H) \subseteq D(G) \cup U(G)$. We write $\mathcal{X}(G)$ for the set of all graphs to which G can be extended. Note that $G_1 \sqsubseteq G_2$ implies $\mathcal{X}(G_2) \subseteq \mathcal{X}(G_1)$. In the context of this section, we consider lexicographically

minimal graphs as canonical. Thus, a graph $G \in \mathcal{G}_n$ is *non-canonical* if there exists a permutation $\pi \in \mathcal{S}_n$ such that $\pi(G) \prec G$. A partially defined graph $G \in \mathcal{P}_n$ is *canonical* if $\mathcal{X}(G)$ contains a *canonical* graph.

A permutation $\pi \in \mathcal{S}_n$ is a ***Certificate* for the Non-Canonicity (nc-certificate)** of $G \in \mathcal{P}_n$ if $\pi(H) \prec H$ for all $H \in \mathcal{X}(G)$; observe that in that case G cannot be canonical. We use nc-certificate if the graph is clear from the context.

A partially defined graph $G \in \mathcal{P}_n$ is *certified non-canonical* if there is an nc-certificate of G in $\mathcal{S}_n$.

We want to highlight the difference between a partially defined graph G being non-canonical and being certified non-canonical. The former states that for every $H \in \mathcal{X}(G)$, there is a permutation $\pi \in \mathcal{S}_n$ such that $\pi(H) \prec H$, while the later states that there is a permutation $\pi \in \mathcal{S}_n$ such that $\pi(H) \prec H$ for every $H \in \mathcal{X}(G)$. Informally, the quantifiers are swapped.

Let $G \in \mathcal{P}_n$. A partially defined graph $G' \in \mathcal{P}_n$ is a *G-obstruction* if

- G' is non-canonical,
- $G' \sqsubseteq G$, and
- no $G'' \sqsubset G'$ is certified non-canonical.

Observe that excluding G' can potentially prune the search more than excluding G.

Below, we will present an algorithm MinCheck, which takes a partially defined graph G as input and eventually computes a G-obstruction. The algorithm MinCheck proceeds by iteratively refining ordered partitions to find relevant permutations. In the worst case, it still requires considering all the $n!$ permutations of the vertices. However, we use a more sophisticated approach that exploits the partially defined graph's structure which often allows us to consider only a small fraction of the permutations.

We will show in Theorem 1, *given a partially defined graph $G \in \mathcal{P}_n$ as input, MinCheck(G) returns a G-obstruction or nil; if G is certified non-canonical then MinCheck(G) returns a G-obstruction.*

Before we state the algorithm MinCheck and prove Theorem 1, we explain how MinCheck is integrated into the CDCL algorithm. Whenever the CDCL algorithm assigns an edge variable a truth value, we check whether the current partially defined graph G is non-canonical. If MinCheck detects that this is the case, we add a clause. The clause is constructed from the obstruction G' blocking all partially defined graphs $G'' \in \mathcal{P}_n$ with $G' \sqsubseteq G''$

$$\bigvee_{i<j, A_{G'}(i,j)=1} \neg e_{i,j} \vee \bigvee_{i<j, A_{G'}(i,j)=0} e_{i,j}.$$

To state the MinCheck algorithm and establish its properties, we need some additional concepts and terminology. Let $G \in \mathcal{P}_n$, and $\pi \in \mathcal{S}_n$.

- A vertex pair (i, j) is (G, π)-*greater* if $A_G(i, j) = 1$ and $A_{\pi(G)}(i, j) = 0$.
- A vertex pair (i, j) is (G, π)-*greater-equal* if and only if $A_H(i, j) \geq A_{\pi(H)}(i, j)$ for every $H \in \mathcal{X}(G)$.
- A vertex pair (i, j) is π-stable if $(i, j) \in \{(\pi(i), \pi(j)), (\pi(j), \pi(i))\}$.

Observe that if (i, j) is π-stable, then $\{i, j\} = \{\pi(i), \pi(j)\}$, so $ij \in E(G)$ if and only if $ij \in E(\pi(G))$.

Observation 2. *Let $G \in \mathcal{P}_n$, and $\pi \in \mathcal{S}_n$. A vertex pair (i, j) is (G, π)-greater-equal if and only if (i, j) is π-stable, $A_G(i, j) = 1$, or $A_{\pi(G)}(i, j) = 0$.*

Example 2. *Let us again consider the partially defined graph G_{ex} and the permutation $\pi = (1)(25)(34)(6)(7)(8)(9)(10)$. Then, the adjacency matrices of G_{ex} and $\pi(G_{ex})$ are as follows:*

$A_{G_{ex}} =$

	1	2	3	4	5	6	7	8	9	10
1	0	[0]	[0]	[0]	[0]	[★]	[★]	[1]	[1]	[1]
2	0	0	[0]	⟨1⟩	[1]	⟨1⟩	0	0	⟨1⟩	⟨1⟩
3	0	0	0	[★]	⟨1⟩	★	★	★	★	★
4	0	1	★	0	[0]	★	★	★	★	★
5	0	1	1	0	0	0	[★]	[★]	★	★
6	★	1	★	★	0	0	[★]	[★]	[★]	[★]
7	★	0	★	★	★	★	0	[★]	[★]	[★]
8	1	0	★	★	★	★	★	0	[★]	[★]
9	1	1	★	★	★	★	★	★	0	[★]
10	1	1	★	★	★	★	★	★	★	0

$A_{\pi(G_{ex})} =$

	1	5	4	3	2	6	7	8	9	10
1	0	[0]	[0]	[0]	[0]	[★]	[★]	[1]	[1]	[1]
5	0	0	[0]	⟨1⟩	[1]	⟨0⟩	★	★	⟨★⟩	⟨★⟩
4	0	0	0	[★]	⟨1⟩	★	★	★	★	★
3	0	1	★	0	[0]	★	★	★	★	★
2	0	1	1	0	0	1	[0]	[0]	1	1
6	★	1	★	★	0	0	[★]	[★]	[★]	[★]
7	★	0	★	★	★	★	0	[★]	[★]	[★]
8	1	0	★	★	★	★	★	0	[★]	[★]
9	1	1	★	★	★	★	★	★	0	[★]
10	1	1	★	★	★	★	★	★	★	0

We highlight (G_{ex}, π)-greater-equal and (G_{ex}, π)-greater vertex pairs as follows, omitting marking the lower triangular matrix for symmetric reasons:

- *the vertex pairs marked with a gray square are π-stable,*
- *the vertex pair marked with a gray pentagon is (G_{ex}, π)-greater,*
- *a pair (i, j) is marked with a white pentagon if $A_{G_{ex}}(i, j) = 1$ and it's neither π-stable nor (G_{ex}, π)-greater,*
- *a pair (i, j) is marked with a white square if $A_{\pi(G_{ex})}(i, j) = 0$ and it's neither π-stable nor (G_{ex}, π)-greater.*

Next we will introduce indicator pairs, which is a key concept for checking whether a partially defined graph has an nc-certificate. For $G \in \mathcal{P}_n$ and $\pi \in \mathcal{S}_n$, the vertex pair (i, j) is a (G, π)-*indicator pair* if $(A_G(i, j), A_{\pi(G)}(i, j)) \in \{(1, 0), (\star, 0), (1, \star)\}$ and every more important vertex pair (i', j') for $i' < j'$ is (G, π)-greater-equal. A (G, π)-indicator pair is a *strong* (G, π)*-indicator pair* if it is a (G, π)-indicator pair and (i, j) is (G, π)-greater. An indicator pair which isn't a strong (G, π)-indicator pair is called a *weak* (G, π)*-indicator pair.*

Example 3. *Let us again consider the graph G_{ex} and the permutation π from the previous example. The vertex pair $(2, 6)$, highlighted in red, is a (G_{ex}, π)-indicator pair, since all more important vertex pairs than (i, j) are (G_{ex}, π)-greater-equal. More precisely, for all vertex pairs $(i', j') \in$*

$\{(1,2),(1,3),(1,4),(1,5),(2,3)\}$, *we have* $A_{\pi(G_{ex})}(i',j') = 0$, *all vertex pairs in* $\{(1,6),(1,7),(1,8),(1,9),(1,10),(2,5)\}$ *are* π*-stable, and for the vertex pair* $(2,4)$, *we have* $A_{G_{ex}}(2,4) = 1$.

Next, we state the relation between nc-certificates and indicator pairs, allowing us to focus on indicator pairs instead of nc-certificates.

Proposition 2. *Let* $G \in \mathcal{P}_n$ *and* $\pi \in \mathcal{S}_n$. *If a strong* (G,π)*-indicator pair exists, then the permutation* π *is an nc-certificate of* G.

Proof. Let $G \in \mathcal{P}_n$ and $\pi \in \mathcal{S}_n$.

Let (i,j) be a strong (G,π)-indicator pair. It follows that (i,j) is (G,π)-greater. For the sake of a contradiction, assume that π is not an nc-certificate of G. By definition of an nc-certificate, there is a fully defined graph $H \in \mathcal{X}(G)$ such that $\pi(H) \prec H$ does not hold, so $H \preceq \pi(H)$.

First, we show for all more important vertex pairs (i',j') than (i,j) (including $i' > j'$) that $A_H(i',j') = A_{\pi(H)}(i',j')$. Assume that there is a vertex pair $(i',j') \prec (i,j)$ such that $A_H(i',j') \neq A_{\pi(H)}(i',j')$. Let (i',j') be the minimal vertex pair with this property. We distinguish five cases and show that $A_H(i',j') = A_{\pi(H)}(i',j')$ must hold for each case assuming $H \preceq \pi(H)$.

(1) If $i' > j'$ then $(j',i') \prec (i',j')$, hence $A_H(i',j') = A_H(j',i') = A_{\pi(H)}(j',i') = A_{\pi(H)}(i',j')$ holds.
(2) If the pair (i',j') is π-stable and $(i',j') = (\pi(i'),\pi(j'))$, then we have $A_{\pi(H)}(i',j') = A_H(\pi^{-1}(i'),\pi^{-1}(j')) = A_H(i',j')$ holds.
(3) If the pair (i',j') is π-stable and $(i',j') = (\pi(j'),\pi(i'))$, then we have $A_{\pi(H)}(i',j') = A_H(\pi^{-1}(i'),\pi^{-1}(j')) = A_H(j',i') = A_H(i',j')$ holds.
(4) If $A_G(i',j') = 1$, then $A_H(i',j') = 1$. By minimality of the vertex pair (i',j'), $A_{\pi(H)}(i'',j'') = A_H(i'',j'')$ holds for all more important vertex pairs (i'',j'') than (i',j'). Hence, $A_{\pi(H)}(i',j') = 1$ must hold, otherwise $\pi(H) \prec H$ by Observation 1, so $A_H(i',j') = A_{\pi(H)}(i',j')$.
(5) If $A_{\pi(G)}(i',j') = 0$, then $A_{\pi(H)}(i',j') = 0$. Again, by minimality of the vertex pair (i',j'), $A_{\pi(H)}(i'',j'') = A_H(i'',j'')$ holds for all vertex pairs $(i'',j'') \prec (i',j')$. Hence, $A_H(i',j') = 0$ must hold by Observation 1, so $A_H(i',j') = A_{\pi(H)}(i',j')$.

We have shown that $A_{\pi(H)}(i',j') = A_H(i',j')$ for all vertex pairs $(i',j') \prec (i,j)$. Additionally, we have $A_H(i,j) = 1$, and $A_{\pi(H)}(i,j) = 0$, because (i,j) being a strong (G,π)-indicator pair. Therefore, by Observation 1, we have $\pi(H) \prec H$ in contradiction to our assumption $H \preceq \pi(H)$. □

Lemma 1. *Let* $G \in \mathcal{P}_n, \pi \in \mathcal{S}_n$ *and* (i,j) *a weak* (G,π)*-indicator pair. If* $H \in \mathcal{X}(G)$ *is canonical, then* $A_H(i,j) = A_{\pi(H)}(i,j)$.

Proof. Let $G \in \mathcal{P}_n$ and $\pi \in \mathcal{S}_n$. Let $H \in \mathcal{X}(G)$ be a canonical graph and (i,j) a weak (G,π)-indicator pair. For the sake of contradiction, assume that $A_H(i,j) \neq A_{\pi(H)}(i,j)$. Then $A_H(i,j) > A_{\pi(H)}(i,j)$, so (i,j) is a strong (H,π)-indicator pair. This is a contradiction with H being canonical according to Proposition 2. □

Proposition 3. *Let* $G \in \mathcal{P}_n$ *and* $\pi \in \mathcal{S}_n$. *If* π *is an nc-certificate of* G, *then there is a* (G,π)*-indicator pair.*

Proof. Let $G \in \mathcal{P}_n, \pi \in \mathcal{S}_n$, and G be without a (G,π)-indicator pair. We construct a graph $H \in \mathcal{X}(G)$ such that $H \preceq \pi(H)$, i.e., we show that π isn't an nc-certificate of G.

We distinguish between two cases:

- For all vertex pairs (i, j) it holds that (i) (i, j) is (G, π)-equal or (ii) $A_G(i, j) = A_{\pi(G)}(i, j)$ and $A_G(i, j) \neq \star$. In this case, we have $H = \pi(H)$ for arbitrary $H \in \mathcal{X}(G)$, so $H \preceq \pi(H)$ holds.
- Otherwise, there is a vertex pair (i, j) which isn't (G, π)-equal nor $A_G(i, j) = A_{\pi(G)}(i, j)$ and $A_G(i, j) \neq \star$. Let (i, j) be the smallest one with respect to the order $\preceq$. Because G doesn't have a (G, π)-indicator pair, we have $A_G(i, j) \neq 1$ and $A_{\pi(G)}(i, j) \neq 0$. We extend G to a fully defined graph $H \in \mathcal{X}(G)$ as follows. We add the non-edge ij and the defined edge $\pi^{-1}(i)\pi^{-1}(j)$ if not already present. All other undefined edges can be either added as defined edge or non-edge. By construction, we have $H \preceq \pi(H)$.

In either case, we showed that π isn't an nc-certificate. Since π was arbitrary, the proposition holds. □

4.1 Minimality Check

On a high level, the algorithm MinCheck proceeds as follows: First it searches for indicator pairs. If it is a strong indicator pair, then the current partially defined graph is non-canonical by Proposition 2. If we find a weak indicator pair, then we add a non-edge or defined edge based on Lemma 1 resulting in a new graph which is canonical if and only if the original one is canonical. We proceed with the new graph and start from the beginning. Since the number of undefined edges decreases in each iteration, the algorithm terminates. The most technical part is finding indicator pairs. Constructing the obstruction is relatively straightforward.

Finding Indicator Pairs

The procedure GetIndicatorPair is based on the concept of an *ordered partition*[3] of $V = \{1, \ldots, n\}$, which is a list $P = [V_1, V_2, \ldots, V_k]$ such that $V_1 \cup \cdots \cup V_k = V$ and V_i, V_j are disjoint for $1 \leq i < j \leq n$. The sets $V_1, \ldots, V_n$ are equivalence classes. We associate with the ordered partition $P = [V_1, V_2, \ldots V_n]$ a set of permutations

$$\text{Perm}(P) = \{\pi \in \mathcal{S}_n : \text{for all } u \in V_i \text{ and } v \in V_j, i < j \text{ implies } \pi(u) < \pi(v)\}.$$

For example, $\text{Perm}([\{2, 3\}, \{1\}]) = \{(213), (13)(2)\}$.

Let f be the mapping that indicates to which equivalence class a vertex belongs to, i.e., $f(v) = i$ if and only if $v \in V_i$. For $i \in \{1, \ldots, k\}$, let $l_i = \sum_{j<i} |V_j| + 1$ and $u_i = \sum_{j \leq i} |V_j|$. Using f, l_i, u_i, we get an equivalent definition of $\text{Perm}(P)$ which is better suited for describing upcoming procedures

$$\text{Perm}(P) = \{\pi \in \mathcal{S}_n : l_{f(v)} \leq \pi(v) \leq u_{f(v)} \text{ for all } v \in V\},$$

i.e., the ordered partition gives a range for each vertex to which it can potentially be mapped.

Next, we describe the algorithm GetIndicatorPair. The input for the initial call of GetIndicatorPair is a partially defined graph $G \in \mathcal{P}_n$. The idea is to start with the ordered partition $P = [V]$ and refine it until we have found a (G, π)-indicator pair for some permutation π represented by the ordered partition or can conclude that no such indicator pair exists. Therefore, we recursively assign a vertex v to some $r \in V$, starting with $r = 1$ and adapting the ordered partition correspondingly by splitting a set V_i if necessary, i.e., replacing V_i by multiple equivalence classes.

We define GetIndicatorPair in a recursive way, with the following input: a partially defined graph $G \in \mathcal{P}_n$, an ordered partition $P = [V_1, \ldots, V_k]$, and a row $r \in \{1, \ldots, n\}$. At the initial call, the partition P is $[\{1, \ldots, n\}]$ and $r = 1$. For all $j < r$ we have $|V_j| = 1$, in other words, all the vertices

[3] Codish et al. [11] also use the term ordered partition, but our definition is more general. We don't require that for a later set in the partition all elements must be greater than in all previous sets from the partition.

Procedure 1: GetIndicatorPair

Input: A partially defined graph G, an ordered partition $P = [V_1, \dots, V_k]$, a row $r \in V$.
Output: A (G, π)-indicator pair if it exists, and nil otherwise.
GetIndicatorPair(G, P, r):

We branch over all possible choices $v \in V_r$.
First, we split V_r into $\{v\}$ and $V_r \setminus \{v\}$, so $\pi(v) = r$ for all $\pi \in \mathrm{Perm}(P_v)$.
For each V_i, $i \in \{r, \dots, k\}$, we apply the following steps, where $V_i^0 := \{ u \in V_i : A_G(v,u) = 0 \}$, $V_i^\star := \{ u \in V_i : A_G(v,u) = \star \}$, and $V_i^1 := \{ u \in V_i : A_G(v,u) = 1 \}$. (Some special care is needed for $i = r$ since it was already split, so we use $V_r \setminus \{v\}$ instead of V_r):

(1) We split V_i into the sets V_i^0 and $V_i^\star \cup V_i^1$; this ensures that the vertices not adjacent to v are mapped to the smallest vertices possible without violating the previous ordered partition.
Again, let $l_i = \sum_{j<i} |V_j| + 1$ and $u_i = \sum_{j \le i} |V_j|$.
If the set $J = \{ u : A_G(r,u) \neq 0, l_i \le u < l_i + |V_i^0| \} \neq \emptyset$, then we put $j = \min J$. In that case (r, j) is a (G, π)-indicator pair for any permutation represent by the ordered partition and we return the indicator pair. If J is empty then all vertex pairs (r, u) with $l_i \le u < l_i + |V_i^0|$ are (G, π)-greater-equal, for all $\pi \in \mathrm{Perm}(P_v)$.
(2) Now we iterate over the remaining indexes $p \in [l_i + |V_i^0|, \dots, u_i]$. We distinguish three cases:
  (a) $A_G(r,p) = \star$: All vertices from V_i^0 are mapped to smaller vertices than p, so the only way to guarantee that the pair is (G, π)-greater-equal for all $\pi \in \mathrm{Perm}(P_v)$ is to ensure that (r, p) is π-stable:
    —If $v = r$ and $p \in V_i^\star$, we split $V_i^\star \cup V_i^1$ into $\{p\}$ and $(V_i^\star \cup V_i^1) \setminus \{p\}$ and remove p from $V_i^\star$.
    —If $v = p$ and $r \in V_i^\star$, we split $V_i^\star \cup V_i^1$ into $\{r\}$ and $(V_i^\star \cup V_i^1) \setminus \{r\}$ and remove r from $V_i^\star$.
    In all other cases, we backtrack.
  (b) If $A_G(r,p) = 0$ we backtrack, since $A_{\pi(G)}(r,p) \neq 0$ for every $\pi \in \mathrm{Perm}(P_v)$.
  (c) If $A_G(r,p) = 1$ then we again distinguish two cases:
    —$V_i^\star \neq \emptyset$ then we split $V_i^\star \cup V_i^1$ into $V_i^\star$ and V_i^1 resulting in (r,p) being a weak (G, π)-indicator pair for all $\pi \in \mathrm{Perm}(P_v)$.
    —Otherwise, (r,p) is still (G, π)-greater-equal for all $\pi \in \mathrm{Perm}(P_v)$.
If $r = n$ we backtrack, otherwise we call GetIndicatorPair$(G, P_v, r+1)$.
If all choices $v \in V_r$ have been exhausted unsuccessfully, we return nil.

which are mapped to the first $r-1$ vertices are already fixed. Furthermore, we will see that all pairs up to $(r-1, n)$ are (G, π)-greater-equal for all $\pi \in \mathrm{Perm}(P)$.

Now we choose a vertex $v \in V_r$: we refine the ordered partition P such that $\pi(r) = v$ for all permutations represented by the refined ordered partition. Next we refine all sets V_i with $i \ge r$ (in the order they occur in the list) such that also all pairs in the current row (i.e., all pairs (r, j) with $j > r$) are (G, π)-greater-equal for all permutations π represented by the ordered partition. The resulting ordered partition is given by P_v. If we find an (G, π)-indicator pair during this process, we return it immediately. If every vertex pair in the current row is (G, π)-greater-equal for every permutation in $\mathrm{Perm}(P_v)$, we call GetIndicatorPair$(G, P_v, r+1)$ with the refined ordered partition P_v; otherwise we backtrack. If every choice of $v \in V_r$ returned no indicator pair, then we return nil.

GetIndicatorPair is given in detail by Procedure 1.

Proposition 4. *Given $G \in \mathcal{P}_n$ as input, GetIndicatorPair $(G, \{1, \dots, n\}, 1)$ returns a (G, π)-indicator pair if G is certified non-canonical.*

Proof. For proving the proposition, we mention some invariants for GetIndicatorPair which will allow us to argue that the invariants are also satisfied at recursive calls. The invariants are as follows:

P1. The vertices which are mapped to the first $r-1$ vertices are already determined, i.e., $|V_i| = 1$ for every $i < r$.
P2. For every permutation $\pi \in \mathrm{Perm}(P)$, every vertex pair $(i', j') \preceq (r-1, n)$ is (G, π)-greater-equal.
P3. For every $\pi \in \mathrm{Perm}(P)$, no vertex pair $(i, j) \preceq (r-1, n)$ is a (G, π)-indicator pair.

Invariant P2 helps us to show that we indeed find indicator pairs, so the returned permutations are nc-certificates by Proposition 2. We need Invariant P2 to show that we don't omit any indicator pair. More precisely, if there is a (G, π)-indicator pair, then it should have been found at a more shallow recursion depth.

Let $G \in \mathcal{P}_n$, $P = [V_1, \dots, V_k]$, $r \in \{1, \dots, n\}$ and assume the Invariants P1–P3 are satisfied. We show by induction on the recursion level that GetIndicatorPair(G, P, r) will return a (G, π)-indicator pair of G with $\pi(u) \in V_u$ for $u < r$ and a (G, π)-indicator pair if and only if such an indicator pair exists; otherwise the procedure returns nil. From this the proposition follows, since $\mathcal{S}_n = \mathrm{Perm}([\{1, \dots, n\}])$ and all invariants are satisfied for GetIndicatorPair$(G, [\{1, \dots, n\}], 1)$.

So, let $G \in \mathcal{P}_n$, $r \in \{1, \dots, n\}$, and $P = [V_1, \dots, V_k]$ an ordered partition such that Invariants P1–P3 hold. Due to Invariant P3, we only have to check the presence of (G, π)-indicator pairs (i, j) satisfying $(r-1, n) \prec (i, j)$ for $\pi \in \mathrm{Perm}(P)$. Additionally, we are only interested in permutations, where $\pi(v) = r$ for $v \in V_r$, hence, we don't have to consider any vertices not in V_r for branching.

By splitting V_r into $\{v\}$ and $V_r \setminus \{v\}$ for $v \in V_r$, Invariant P1 is satisfied for recursive calls. It remains to show that the splitting of the equivalence classes is correct. We do this for each $v \in V_r$ separately. We say that

—Invariant P2 holds up to (i, j) if for every permutation $\pi \in \mathrm{Perm}(P_v)$, every vertex pair $(i', j') \preceq (i, j)$ is (G, π)-greater-equal,
—Invariant P3 holds up to (i, j) if for every $\pi \in \mathrm{Perm}(P_v)$, no vertex pair $(i', j') \preceq (i, j)$ is a (G, π)-indicator pair.

We show that Invariants P2 and P3 hold up to (r, u_i) after refining the equivalence class V_i. Additionally, we show that partitions are refined in a way, such that only permutations are excluded, which have been shown to not have an indicator pair. More precisely, we exclude a permutation π, if there is a vertex pair which isn't (G, π)-greater-equal and more important than any (G, π)-greater vertex pair. Observe that in this case π cannot have a (G, π)-indicator pair.

In each step of the procedure for splitting an equivalence class V_i, we ensure that the vertex pairs are (G, π)-greater-equal for all permutations represented by the refined partition. So, Invariant P2 holds up to (r, u_i). Let us have a closer look at the two steps for refining the equivalence class.

(1) If the set $J = \{ u : A_G(r, u) \neq 0, l_i \leq u < l_i + |V_i^0| \} \neq \emptyset$, then we found an indicator pair and return; otherwise, if any vertex in $V_i^\star \cup V_i^1$ is mapped to a vertex w in $\{l_i, \dots, l_i + |V_i^0|\}$, then (r, w) is neither (G, π)-greater-equal nor (G, π)-greater, but more important than any (G, π)-greater vertex pair. Hence, splitting V_i into V_i^0 and $V_i^\star \cup V_i^1$ removes only irrelevant permutations from $\mathrm{Perm}(P_v)$ and Invariant P3 holds up to the vertex pair (r, u_i) by splitting V_i.
(2) We focus on the case $A_G(r, p) = \star$, the other two cases are already clarified in the procedure itself.

	{1}	{2,3,4,5}				{6}	{7}	{8,9,10}		
Select vertex 1:	*0*	*0*	*0*	*0*	*0*	⋆	⋆	*1*	*1*	*1*

	{1}	{2}	{3}	{4,5}		{6}	{7}	{8}	{9,10}	
Select vertex 1:	*0*	*0*	*0*	*0*	*0*	⋆	⋆	*1*	*1*	*1*
Select vertex 2:	*0*	*0*	*0*	*1*	*1*	*1*	*0*	*0*	*1*	*1*

⋮

	{1}	{5}	{4}	{2,3}		{6}	{7}	{8,9,10}		
Select vertex 1:	*0*	*0*	*0*	*0*	*0*	⋆	⋆	*1*	*1*	*1*
Select vertex 5:	*0*	*0*	*0*	*1*	*1*	***0***				

Fig. 2. Example for the minimality check.

For any permutation represented by the ordered partition, we have $A_G(r,p) = \star$ and $A_\pi(G)(r,p) \in \{\star, 1\}$. The only way to ensure that (r,p) is (G,π)-greater-equal is by refining the ordered partition such that (r,p) is π-stable for every permutation represented by the refined partition. This can only be ensured if the branching vertex v is in $\{p,r\}$, one vertex of $\{p,r\}$ is in $V_i^\star$.

Since $u_k = n$, all invariants hold for recursive calls. To sum up, if we find an indicator pair in the current row, we return it and we only backtrack if we can ensure that there is no indicator pair with a permutation represented by the current ordered partition. □

Example 4. *Figure 2 shows how the initial ordered partition* $[V]$ *is refined, given the partially defined graph* G_{ex}*. Starting with the first row, we select vertex* 1 *to be mapped to itself, resulting in the ordered partition* $[\{1\},\{2,3,4,5\},\{6\},\{7\},\{8,9,10\}]$*, because the vertices in* $\{2,3,4,5\}$ *are not adjacent to* 1 *(therefore, it is best to place them on the left side). It is still undefined whether the vertices* 6 *and* 7 *are adjacent to* 1*, so they have to be in their own equivalence class; the remaining vertices are adjacent to* 1*. We can see that for the first row* $r = 1$ *all vertex pairs* (r,v) *are* (G_{ex},π)*-greater-equal for each* $\pi \in \mathrm{Perm}([\{1\},\{2,3,4,5\},\{6\},\{7\},\{8,9,10\}])$*, especially* $(1,6)$ *and* $(1,7)$ *are* π*-stable. The selection for later rows is indicated at the right-hand side. We select vertex* 2 *for row* 2*. We add* $\{2\}$ *as a new equivalence class and refine the other equivalence classes. For example,* $\{8,9,10\}$ *is split into* $\{8\}$ *and* $\{9,10\}$*, since* 8 *is not adjacent to* 2*, but* 9 *and* 10 *both are.*

If we select vertex 5 *for the second row, we get a different ordered partition. We split* $\{2,3,4\}$ *into* $\{4\}$ *and* $\{2,3\}$ *because, again,* 4 *is the only vertex adjacent in this equivalence class to* 5*. The set* $\{6\}$ *only consists of a single vertex which is not adjacent to* 5*, while in the original matrix on this position there is an edge. We found an indicator pair highlighted with a box. More precisely, for any* $\pi \in \mathrm{Perm}([\{1\},\{5\},\{4\},\{2,3\},\{6\},\{7\},\{8,9,10\}])$*,* $(2,6)$ *is a* (G_{ex},π)*-indicator pair and hence* π *is an nc-certificate of* G_{ex} *due to Proposition 2.*

Given Procedure 1 for finding indicator pairs, a procedure for checking whether a partially defined graph is canonical is given by Procedure 2.

Proposition 5. *Let* $G \in \mathcal{P}_n$*. If* IsMinimal(G) *returns false, then* G *is non-canonical. If* G *is certified non-canonical, then* IsMinimal(G) *returns false.*

Procedure 2: IsMinimal

Input: A partially defined graph G.
Output: G is non-canonical if false is returned.
IsMinimal(G):

```
p := GetIndicatorPair G, V, 1
if p = nil then
    return unknown
else if p is a weak (G, π)-indicator pair then
    i, j := p
    if A_G(i, j) = ⋆ then
        Add non-edge ij to G
    else
        Add defined edge π^{-1}(i)π^{-1}(j) to G
    return IsMinimal(G)
else                                          ▷ p is a strong indicator pair
    return false
```

Procedure 3: GetObstruction

Input: A partially defined non-canonical graph G.
Output: A G-obstruction.
GetObstruction(G):

```
for all defined edges and non-edges e of G do
    G' := G except e is undefined
    if IsMinimal G' = false then
        G := G'
return G
```

Proof. The lemma follows from the correctness of GetIndicatorPair established in Proposition 4 and from Lemma 1 and Proposition 3. □

Constructing Obstructions

Finally, we present a procedure GetObstruction(G) (Procedure 3) to extract a G-obstruction G'. The procedure removes defined edges and non-edges as long as the resulting graph is shown to be non-canonical.

Proposition 6. *Given a non-canonical partially defined graph $G \in \mathcal{P}_n$, GetObstruction(G) returns a G-obstruction.*

Proof. Since IsMinimal guaranties to correctly classify certified non-canonical partially defined graphs, we ensure that every $G'' \sqsubseteq G' =$GetObstruction(G) isn't certified non-canonical and G' is non-canonical, hence G' is a G-obstruction. □

Recall that a clause blocking the obstruction G' is given by

$$\bigvee_{i<j, A_{G'}(i,j)=1} \neg e_{i,j} \vee \bigvee_{i<j, A_{G'}(i,j)=0} e_{i,j}.$$

Since G' is non-canonical, this clause must be satisfied by every assignment which represents a canonical graph.

Procedure 4: ExtractSmall

Input: A partially defined graph G, a nc-certificate π and a (G, π)-indicator pair (i, j).
Output: A partially defined non-canonical graph $G' \sqsubseteq G$.
ExtractSmall$(G, \pi, (i, j))$:

We start with a partially defined graph $G' \in \mathcal{P}_n$ where all vertex pairs are undefined.
We iterate over all vertex pairs $(i', j') \prec (i, j)$ with $i' < j'$:
—If (i', j') is π-stable, we ignore the vertex pair and continue with the next vertex pair.
—If $A_G(i', j') = 1$ we set $A_{G'}(i', j') = A_{G'}(j', i') = 1$ and continue with the next vertex pair.
—If $A_{\pi(G)}(i', j') = 0$ we set $A_{\pi(G')}(i', j') = A_{\pi(G')}(j', i') = A_{G'}(\pi^{-1}(i), \pi^{-1}(j))$
$= A_{G'}(\pi^{-1}(j), \pi^{-1}(i)) = 0$.
At the end, we set $A_{G'}(i, j) = A_{G'}(j, i) = 1$ and
$A_{\pi(G')}(i, j) = A_{G'}(\pi^{-1}(i), \pi^{-1}(j)) = A_{G'}(\pi^{-1}(j), \pi^{-1}(i)) = 0$. We return G' given by its adjacency matrix.

If IsMinimal concludes that the partially defined graph is not canonical without a recursive call, i.e., we have a certificate π and a strong (G, π)-indicator pair, then we can start the procedure GetObstruction with a partially defined graph with potentially more undefined edges constructed by the procedure ExtractSmall. The resulting graph G' has the property that $G' \sqsubseteq G$, (i, j) is a strong (G'', π)-indicator pair, but for any $G'' \sqsubset G'$, the vertex pair (i, j) isn't a strong (G'', π)-indicator pair.

Example 5. *For constructing a symmetry breaking clause given G_{ex}, let us pick a concrete permutation π from the ordered partition $[\{1\}, \{5\}, \{4\}, \{2, 3\}, \{6\}, \{7\}, \{8, 9, 10\}]$, e.g., $\pi = (1)(25)(34)(6)(7)(8)(9)(10)$. ExtractSmall$(G_{ex}, \pi, (i, j))$ results in a partially defined graph G' with $A_{G'}(i', j') = 1$ for $(i', j') \in \{(2, 4), (2, 6)\}$ and $A_{G'}(i', j') = 0$ for $(i', j') \in \{(1, 5), (1, 4), (1, 3), (1, 2), (5, 6)\}$, all other vertex pairs are undefined edges. Removing any defined edge or non-edge results in graph which isn't certified non-canonical.*

So, the resulting clause is $\neg e_{2,4} \vee e_{1,5} \vee e_{1,4} \vee e_{1,3} \vee e_{1,2} \vee \neg e_{2,6} \vee e_{5,6}$.

An intuitive way for getting the clause is by iterating over the more important vertex pairs (i', j') than the indicator pair (i, j) like in ExtractSmall, and proceeding as follows (i) if (i', j') is π-stable then ignore it, (ii) if $A_G(i', j') = 1$ add the literal $\neg e_{i',j'}$, and (iii) if $A_{\pi(G)}(i', j') = 0$ add the literal $e_{\pi^{-1}(i'),\pi^{-1}(j')}$. At the end, we add $\neg e_{2,6} \vee e_{5,6}$ resulting in the same clause as given above.

We see that the clause is conflicting with the current partially defined graph. The given clause forbids a certain partially defined graph $G'' \sqsubseteq G_{ex}$. We highlight the edges and non-edges of G''. For symmetric reasons, we omit marking the lower triangular matrix

$$\begin{pmatrix}
0 & \mathit{0} & \mathit{0} & \mathit{0} & \mathit{0} & \star & \star & 1 & 1 & 1 \\
0 & 0 & \mathit{0} & \mathit{1} & 1 & \mathit{1} & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & \star & 1 & \star & \star & \star & \star & \star \\
0 & 1 & \star & 0 & 0 & \star & \star & \star & \star & \star \\
0 & 1 & 1 & 0 & 0 & \mathit{0} & \star & \star & \star & \star \\
\star & 1 & \star & \star & 0 & 0 & \star & \star & \star & \star \\
\star & 0 & \star & \star & \star & \star & 0 & \star & \star & \star \\
1 & 0 & \star & \star & \star & \star & \star & 0 & \star & \star \\
1 & 1 & \star & \star & \star & \star & \star & \star & 0 & \star \\
1 & 1 & \star & \star & \star & \star & \star & \star & \star & 0
\end{pmatrix}.$$

Bringing everything together, the algorithm MinCheck is given by Algorithm 5.

Algorithm 5: MinCheck

Input: A partially defined graph $G \in \mathcal{P}_n$.
Output: A G-obstruction if G is certified non-canonical.

```
1: if IsMinimalG = false then
2:     return GetObstructionG
3: else
4:     return nil
```

Theorem 1. *Given $G \in \mathcal{P}_n$, as input, MinCheck(G) returns a G-obstruction or nil. If G is certified non-canonical then MinCheck(G) returns a G-obstruction.*

Proof. The correctness of Theorem 1 follows from Proposition 5 and Proposition 6. □

Corollary 1. *Given a fully defined graph $G \in \mathcal{G}_n$ as input, MinCheck(G) returns a G-obstruction if and only if G is non-canonical.*

Propagating Edges

With some minor modifications of the algorithms, we can propagate the presence of a defined edge/non-edge given a partially defined graph $G \in \mathcal{P}_n$ assuming the resulting fully defined graph $H \in \mathcal{X}(G)$ to be canonical.

We can use the procedure ExtractSmall as is with weak indicator pairs for getting a partially defined graph G' with an nc-certificate. More precisely, the weak indicator pair of G is a strong indicator pair of G'. The graph G' is not necessarily an obstruction of G but still a non-canonical partially defined graph. So, the clause forbidding all partially defined graphs G'' with $G' \sqsubseteq G''$ still must be satisfied by every canonical graph. Given the current variable assignment, all literals of the clause are assigned false or it is a unit clause. In the latter case, a literal can be propagated by boolean constraint propagation.

This means that for each symmetry breaking clause, we have exactly one permutation used for constructing it. As we will see later, this also has the benefit that these eases validating the symmetry breaking clauses in a practical setting.

Restricting nc-Certificates

In many applications, it is tempting to fix a certain (induced) subgraph by assigning specific edge variables due to some knowledge of the structure of the searched for graphs. For example, if it is known that all graphs, satisfying the specified property, contain a clique of size k, one can add clauses to the encoding ensuring that the edge variables $e_{u,v}$ for $1 \leq u \leq v \leq k$ are true in each model. As a result, SMS is not applicable, since the encoded property is not preserved by every permutation in $\mathcal{S}_n$. We still can use the SMS framework if we restrict the set of all permutations $\mathcal{S}_n$ to permutations preserving the substructure. For example, if we encode that the first k vertices form a clique, then applying any permutations in $\mathrm{Perm}([\{1, \ldots, k\}, \{k+1, \ldots, n\}])$ to the graph still results in a clique of the first k vertices.

SMS can be adapted to search for nc-certificates s solely described by an ordered partition P, by calling GetIndicatorPair with the ordered partition P (i.e., GetIndicatorPair$(G, P, 1)$) instead of the ordered partition $[\{1, \ldots, n\}]$ representing $\mathcal{S}_n$. Then, the procedure only returns nc-certificates in $\mathrm{Perm}(P)$. Given an nc-certificate $\pi \in \mathrm{Perm}(P)$, we apply GetObstruction(G, π) to get a (G, π)-obstruction and construct a clause blocking this partially defined graph.

5 Prototype Implementation and Experimental Setup

In this section, we will describe our experimental setup and some implementation details.

As the underlying solver, we use CaDiCaL with the new IPASIR-UP interface [20] which supports custom propagators. We used this interface to integrate our implementation of MinCheck written in C++ and add symmetry-breaking clauses during search. The most important callback functions in SMS for CaDiCaL are "cb_has_external_clause," which indicates whether a symmetry breaking clause can be added, and "cb_add_external_clause_lit" allowing to add a clause literal by literal to the underlying solver. For more details, when "cb_has_external_clause" is called, we refer to the IPASIR-UP paper.

In previous work, we used Clingo [22, 23], an answer set programming solver containing a CDCL SAT solver. The source code and scripts for reproducing the results are available as part of the SMS package at https://github.com/markirch/sat-modulo-symmetries with documentation at https://sat-modulo-symmetries.readthedocs.io/.

For the sake of simplicity, we generate our CNF encodings in Python and feed the encoding to the solver.

The parameter *frequency* allows us to balance the time spent on the minimality check and the time spent by the SAT solver itself. If *frequency* has the value $1/q$, then MinCheck is called only every qth time when "cb_has_external_clause" is called assuming a new edge variable has been assigned since the last call of MinCheck. The parameter *cutoff* is used to abort MinCheck if too much time is spend in the algorithm. If the cutoff has the value c, then we return nil, if after c calls of GetIndicatorPair, including recursive calls, no certificate was found.

Using the parameters frequency and cutoff may result in an incomplete symmetry breaking. To circumvent this problem, one can either call MinCheck for fully defined graphs without a cutoff or use tools like Nauty [42] to filter isomorphic copies in a post-processing phase. In our experiments, we call MinCheck for fully defined graphs with a cutoff, if specified, and filter isomorphic copies with Nauty.

Each of our experiments runs on a cluster of machines with different processors,[4] under Ubuntu 18.04. All tests are executed with a single thread.

6 Extremal Graphs with Required Girth

A prominent research topic in Extremal Graph Theory [6] is the study of extremal graphs (i.e., graphs with the largest possible number of edges) on n vertices that exclude a given family $\mathcal{F}$ of graphs as subgraphs. $\mathrm{EX}(n, \mathcal{F})$ denotes the class of extremal graphs with that property, and $\mathrm{ex}(n, \mathcal{F})$ denotes the number of edges of the graphs in $\mathrm{EX}(n, \mathcal{F})$. The special case, where $\mathcal{F} = C_k$, the family of cycles up to length k, has received much attention; for convenience, we write $f_k(n) = \mathrm{ex}(n, C_k)$. The *girth* of a graph G is the length of a shortest cycle in G (or ∞ if G is acyclic). Hence, $\mathrm{EX}(n, C_k)$ contains precisely the edge-maximal graphs of girth $> k$. The base case of $k = 3$ has been settled over a century ago by Mantel's Theorem [40]: $f_3(n) = \mathrm{ex}(n, C_3) = \lfloor n^2/4 \rfloor$, where $\mathrm{EX}(n, \mathcal{F})$ contains precisely the complete bipartite graph $K_{\lceil n/2 \rceil, \lfloor n/2 \rfloor}$. For the general case $k > 3$, however, no closed formula is known, and researchers have tried to compute $f_k(n)$ for small values of k [3, 11, 21, 51, 52], or at least provide lower and upper bounds.

We describe a SAT encoding that produces for given integers n, m, k a propositional CNF formula $F(n, m, k)$. The formula is satisfiable if and only if there is a graph $G \in \mathrm{EX}(n, C_k)$ with m edges. We will then evaluate the formula with our SMS-solver and report the experimental results.

[4] Intel Xeon E5540, E5649, E5-2630 v2, E5-2640 v4@ at most 2.60 GHz, AMD EPYC 7402@2.80 GHz.

6.1 Encoding

We state a useful result before we present the encoding for $F(n, m, k)$, where δ_G and Δ_G denote the minimum and maximum degree of a graph G, respectively.

Lemma 2 ([21]). *If G is a graph of girth ≥ 5 with n vertices and m edges, then $n \geq 1 + \Delta_G \cdot \delta_G$, $\delta_G \geq m - f_4(n-1)$, and $\Delta_G \cdot n \geq 2m$.*

Proof. By taking a maximum degree vertex r and the simple observation that for each distinct $u_1, u_2 \in N(r)$ the vertex r is the only common neighbor, we conclude that G contains a rooted tree T as subgraph with root r, where the Δ_G children of r have at least $\delta_G - 1$ children, resulting in $1 + \Delta_G + (\Delta_G \cdot \delta_G - 1) = 1 + \Delta_G \cdot \delta_G$ vertices.

The inequality $\delta_G \geq m - f_4(n-1)$ follows from the fact that a graph resulting from deleting a vertex with the lowest degree results in a graph with girth ≥ 5.

The last inequality follows from the Handshaking Lemma [53]. □

In particular, Lemma 2 applies to all graphs in $\mathrm{EX}(n, C_k)$ for $k \geq 4$. We also use the obvious inequalities $\delta_G(n-1) + \Delta_G \leq 2m$ and $\Delta_G(n-1) + \delta_G \geq 2m$, which follow from the Handshaking Lemma and the presence of one maximum/minimum degree vertex, to discard some cases.

According to Lemma 2, we can compute for each pair n, m the set $I_{n,m}$ of possible intervals $[a, b]$ such that for each graph G with n vertices and m edges, we have $a \leq \delta_G \leq \Delta_G \leq b$ for some $[a, b] \in I$. We add to $F(n, m, k)$ suitable cardinality constraints that ensure that vertex degrees belong to one of the intervals.

For higher girths, we generalize Lemma 2 as follows:

Lemma 3. *If G is a graph of girth $k \geq 7$ with n vertices and m edges, then $n \geq 1 + \Delta_G + \Delta_G \cdot (\delta_G - 1) + \Delta_G \cdot (\delta_G - 1)^2$, $\delta_G \geq m - f_{k-1}(n-1)$, and $\Delta_G \cdot n \geq 2m$.*

Proof. We take the rooted tree T mentioned in the proof of Lemma 2 as starting point. Two leaves of the tree T cannot have a common neighbor in $V(G) \setminus V(T)$. Otherwise, there would exist a cycle in G of length ≤ 6. Hence, there is a subtree with $1 + \Delta_G + \Delta_G \cdot (\delta_G - 1) + \Delta_G \cdot (\delta_G - 1)^2$ vertices.

The proof of the other two claims is analog to the proof of Lemma 2. □

To guarantee that the resulting graph has girth $> k$, we use two methods: a basic one and a compact one for higher girths. The *basic method* explicitly forbids that any subset of up to k vertices forms a cycle. The set of all possible cycles of length k can be described with some basic symmetry breaking as follows:

$$C_k = \{\, (v_1, \ldots, v_k) \in \{1, \ldots, n\}^k : i \neq j \rightarrow v_i \neq v_j, v_1 = \min\{v_1, \ldots, v_k\}, v_2 < v_k \,\}.$$

Taking v_1 as the minimum fixes a particular rotation of the cycle, requiring $v_2 < v_k$ fixes an orientation of the cycle. Now we add for each element of C_k the constraint that one edge of the corresponding cycle must be absent

$$\bigwedge_{(v_1,\ldots,v_k)\in C_k} (\neg e_{v_1,v_2} \vee \neg e_{v_2,v_3} \vee \cdots \vee \neg e_{v_{k-1},v_k} \vee \neg e_{v_k,v_1}).$$

For $k \leq 4$ this is a workable solution, but the *improved method* scales better for larger k. It is based on the following observation where $dist_G(u, v)$ denotes the number of edges of a shortest path between vertices u and v in graph G.

Observation 3. *A shortest cycle in a graph G containing the edge $uv \in E(G)$ has length $dist_{G-uv}(u, v) + 1$.*

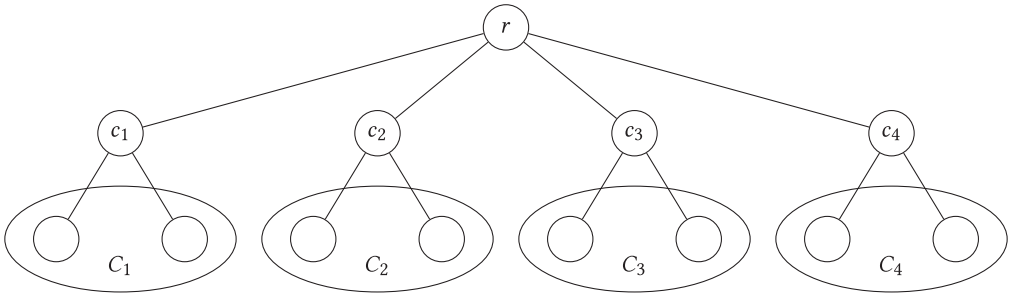


Fig. 3. Example of a star with $\Delta = 4$ and $\delta = 3$.

Hence, we can enforce that for every edge uv, $dist_{G-uv}(u,v) + 1 \geq g$ for a required girth g, equivalently $dist_{G-uv}(u,v) \geq g-1$. We proceed similar to breadth-first search. We start at vertex i and mark all vertices adjacent to i in $G - ij$. In the next step, we additionally mark all vertices which are adjacent to already marked vertices. This will be repeated $g-2$ times. If at the end the vertex j is marked, the girth is smaller than desired.

For the concrete encoding, we introduce propositional variables $reached_{i,j,k,s}$ for representing that $dist_{G-ij}(i,k) \leq s$. Consequently

$$\begin{aligned} reached_{i,j,k,1} &= e_{i,k} \text{ for } k \in \{1,\dots,n\} \setminus \{i,j\} \text{and} \\ reached_{i,j,k,s} &= \bigvee_{l \in V(G) \setminus \{k\}} (e_{l,k} \wedge reached_{i,j,l,s-1}) \text{ for } s \in \{2,\dots,g-2\}, k \in \{1,\dots,n\} \setminus \{i\}. \end{aligned}$$

If at any point vertex j is reached, the girth restriction is violated. Hence, we can use the following encoding:

$$girth = \bigwedge_{i<j} \bigwedge_{s=2}^{g-2} (\neg reached_{i,j,j,s} \vee \neg e_{i,j}).$$

We further improve this encoding. If we start checking for a vertex v whether $dist_{G-uv} + 1 \geq g$ for all $vu \in E$, and we conclude yes, then v cannot be on a cycle which is shorter than the girth g. So for all subsequent vertices $v' > v$, we only consider the graph $G - v$. This yields the following final encoding for the variables $reached_{i,j,k,s}$:

$$\begin{aligned} reached_{i,j,k,1} &= e_{i,k} \text{ for } k \in \{i+1,\dots,n\} \setminus \{j\} \text{ and} \\ reached_{i,j,k,s} &= \bigvee_{l \in V(G) \setminus \{k\}} (e_{l,k} \wedge reached_{i,j,l,s-1}) \text{ for } s \in \{2,\dots,g-2\}, k \in \{i+1,\dots,n\}. \end{aligned}$$

Codish et al. [11] improved their encoding by fixing the rooted tree mentioned in Lemma 2 as subgraph. These subtrees are also known as *stars*. By fixing a subgraph, not all permutations are applicable for symmetry breaking. As solution, they only test swapping vertex pairs which are not in the star and pairs of leaf nodes from the star with the same parent.

Our framework allows breaking all symmetries coming from permuting vertices not in the star and leaf nodes from the star with the same parent. This is achieved by starting the minimality check with a different ordered partition P which ensures that each $\pi \in \mathrm{Perm}(P)$ preserves the star. More precisely, let Δ be the maximum degree of the graph sought for, r the root of the star, $c_1,\dots,c_\Delta$ the children of r, C_i the set of children of c_i, and R the set of vertices not in the star (see Figure 3), then

$$P = [R, C_1, \dots, C_\Delta, \{c_1\}, \dots, \{c_\Delta\}, \{r\}],$$

assuming that $n = r$, $c_i = n - (\Delta + 1) + i$, $v_1 < v_2$ for all $v_1 \in C_i, v_2 \in C_j$ if $i < j$, and $R = \{1,\dots,|R|\}$. For example, if $\Delta = 4$, $\delta = 3$ and $n = 16$, then $R = \{1,2,3\}, C_1 = \{4,5\}, C_2 = \{6,7\}, C_3 = \{8,9\}, C_4 = \{10,11\}, c_1 = 12, c_2 = 13, c_3 = 14, c_4 = 15$, and $r = 16$.

In general, we can consider all permutations for symmetry breaking which preserve the fixed subgraph, i.e., all permutations which, if restricted to the vertices of the star, are automorphisms. This allows us to further generalize the previous idea by swapping two subtrees of the star, i.e., starting with the ordered partition

$$P_{i,j} = [R, C_1, \ldots, C_j, \ldots, C_i, \ldots, C_\Delta, \{1\}, \ldots, \{c_j\}, \ldots, \{c_i\} \ldots, \{c_\Delta\}, \{r\}]$$

instead of

$$P_{i,j} = [R, C_1, \ldots, C_i, \ldots, C_j, \ldots, C_\Delta, \{c_1\}, \ldots, \{c_i\}, \ldots, \{c_j\} \ldots, \{c_\Delta\}, \{r\}]$$

for $i < j$ and $i, j \in \{1, \ldots, \Delta\}$.

To sum up, instead of calling the minimality check once, we call it several times with different initial ordered partitions. Note that this still results in an incomplete symmetry breaking. In future work, we plan to extend SMS to allow fixing arbitrary subgraphs while still producing only one graph per isomorphism class.

6.2 Results

We compute $f_k(n)$ for $k \in \{4, 5, 6\}$, and thereby verify known results [3, 11]. For fixed k and n, we run SMS on the formulas $F(n, f_k(n), k)$ and $F(n, f_k(n) + 1, k)$; we performed separate runs for all the intervals in $I_{n,m}$ where m denotes the number of edges. The first formula must be satisfiable for at least one interval in $I_{n,f_k(n)}$ while the second must be unsatisfiable for every interval in $I_{n,f_k(n)+1}$. In some cases, we don't need to compute $F(n, f_k(n) + 1, k)$, since already the bounds from Lemma 2 or Lemma 3 show the non-existence of a graph with $f_k(n) + 1$ edges.

In general, for fixed k and n, we run SMS on $F(n, m, k)$ for different values of m, starting from a lower bound obtained by Lemma 1. As long as $F(n, m, k)$ is satisfiable, we increment m by one and repeat until we arrive at a value for which $F(n, m, k)$ is unsatisfiable. Then, we know that $f_k(n) = m - 1$.

Tables 2 and 3 show our results for $k = 4$ and $n \in \{15, \ldots, 28\}$ without and with using embedded stars. We use the basic method to encode the girth requirement and choose a frequency of $1/20$ and no cutoff. For all tables in the current section, the runtimes are given in seconds; for SMS, we provide in parenthesis the fraction of the total time spent on the minimality check. The columns labeled "sat" give the minimal time over all intervals in $I_{n,m}$ which are satisfiable; the columns labeled "unsat" give the maximum over all intervals. An entry "n.a." indicates that the unsatisfiability check is covered by Lemma 2; "t.o." indicates that the timeout of 4 hours has been reached without producing a result.

We would like to emphasize that the purpose of the experiments is not to identify which algorithm is the fastest but rather to gain insights into the potential of a dynamic symmetry breaking for graph generation. We provide for reference the running times of our encoding of the static symmetry breaking (columns labeled "Static") and the times reported by Codish et al. [11] with their "improved lexicographic break" (given in parentheses) for the same problems. This is not meant as a direct comparison, as the results by Codish et al. [11] have been run on different hardware, but just to give a rough idea on the order of magnitude the two approaches take. It is not completely clear how Codish et al. combined runtimes over all intervals $I_{n,m}$ into one single result. This has no impact on the unsat-times reported by Codish et al., because for those which finished within the given time limit, there is only a single interval in $I_{n,m}$.

We can see that SMS is significantly faster for the unsatisfiable instances. For example, SMS determines the unsatisfiable case for $n = 23$ under 20 seconds without fixing embedded stars, although the static approach reached the timeout of 4 hours. SMS could also establish the unsatisfiability case

Table 2. Results for Computing $f_4(n)$ in Seconds

		sat		unsat	
n	$f_4(n)$	SMS	Static	SMS	Static
15	26	0.28(9%)	0.18 (0.3)	n.a.	n.a.
16	28	0.19(8%)	0.06 (0.61)	1.57(8%)	17.45 (32.2)
17	31	0.97(8%)	0.07 (0.48)	n.a.	n.a.
18	34	0.85(9%)	0.18 (0.6)	n.a.	n.a.
19	38	0.30(13%)	0.11 (0.57)	n.a.	n.a.
20	41	4.75(10%)	2.11 (161.41)	n.a.	n.a.
21	44	1.14(12%)	0.23 (151.84)	1.51(15%)	280.88 (1,019.41)
22	47	3.54(26%)	2.99 (16.69)	6.35(28%)	8,848.13 (t.o.)
23	50	23.54(36%)	5.69 (367.83)	19.59(34%)	t.o. (t.o.)
24	54	3.16(13%)	22.30 (80.74)	n.a.	n.a.
25	57	433.47(31%)	4.09 (655.66)	1,161.00(34%)	t.o. (t.o.)
26	61	69.27(27%)	26.05 (168.9)	n.a.	n.a.
27	65	4,143.65(34%)	403.25 (193.44)	n.a.	n.a.
28	68	328.30(10%)	96.50 (t.o.)	t.o.	t.o. (t.o.)

Table 3. Results for Computing $f_4(n)$ Fixing a Subtree

		sat		unsat	
n	$f_4(n)$	SMS + star	Static + star	SMS + star	Static + star
15	26	0.05(1%)	0.06 (0.11)	n.a.	n.a.
16	28	0.05(2%)	0.06 (0.25)	0.47(3%)	5.63 (3.63)
17	31	0.09(2%)	0.06 (0.19)	n.a.	n.a.
18	34	0.16(5%)	0.08 (0.27)	n.a.	n.a.
19	38	0.10(1%)	0.08 (0.21)	n.a.	n.a.
20	41	1.62(5%)	4.70 (8.51)	n.a.	n.a.
21	44	0.16(3%)	0.16 (15.57)	0.23(3%)	0.78 (0.56)
22	47	0.39(3%)	0.29 (23.72)	0.56(3%)	6.37 (24.31)
23	50	0.62(3%)	0.34 (6.94)	3.98(4%)	265.51 (2,269.73)
24	54	3.95(5%)	2.28 (12.47)	n.a.	n.a.
25	57	2.54(4%)	4.66 (59.7)	215.51(5%)	t.o. (t.o.)
26	61	17.51(6%)	5.85 (24.41)	n.a.	n.a.
27	65	120.78(7%)	16.13 (23.62)	n.a.	n.a.
28	68	68.73(6%)	70.07 (52.81)	t.o.	t.o. (t.o.)

for $n = 25$. We see that SMS, in many cases, uses a significant fraction of the time for the minimality check. Therefore, a speedup for the check would have a substantial impact on the runtime. As expected, the time spend in the minimality check decreases when fixing embedded stars as fewer permutations must be considered by the minimality check.

Next, we report on results for computing $f_k(v)$ for $k \in \{5, 6\}$. For these cases, we used the girth constraints based on edge removal from Section 6.1. In these experiments, we see that far less time is spent on the minimality check than in the previous experiments, although there we had a frequency of 1/20. Most likely, the reasons are the increased number of clauses and the additionally

Table 4. Results for Computing $f_5(v)$ and $f_6(v)$ in Seconds

n	$f_5(n)$	sat	unsat	$f_6(n)$	sat	unsat
15	22	1.31(7%)	5.96(6%)	18	0.44(8%)	2.13(5%)
16	24	0.28(6%)	1.19(5%)	20	0.63(6%)	n.a.
17	26	3.05(9%)	1.26(6%)	22	1.59(5%)	n.a.
18	29	0.79(7%)	n.a.	23	2.36(5%)	14.62(9%)
19	31	9.52(10%)	6.67(7%)	25	2.93(5%)	n.a.
20	34	5.86(9%)	n.a.	27	3.39(5%)	n.a.
21	36	4.39(11%)	29.30(12%)	29	7.24(6%)	n.a.
22	39	10.83(16%)	n.a.	31	26.57(9%)	n.a.
23	42	109.99(18%)	n.a.	33	6.87(6%)	n.a.
24	45	47.88(13%)	n.a.	36	3.79(4%)	n.a.
25	48	23.33(10%)	n.a.	37	22.21(8%)	23,291.21(18%)
26	52	3.29(7%)	n.a.	39	3.99(4%)	n.a.
27	53	43.61(10%)	t.o.	41	79.56(14%)	n.a.
28	56	7.28(26%)	25.35(17%)	43	1,226.54(13%)	n.a.

Table 5. Results for Different Frequencies

frequency	$n = 27$ (sat)	$n = 25$ (unsat)
1/1	1,220.22(84%)	5,756.80(90%)
1/2	1,789.70(83%)	3,447.59(84%)
1/5	9,846.08(62%)	1,699.04(70%)
1/10	t.o.	901.56(54%)
1/20	1,869.03(43%)	1,203.74(40%)
1/50	2,513.94(18%)	1,221.33(19%)
1/100	841.28(9%)	594.95(10%)
1/200	797.20(4%)	871.52(5%)
1/500	379.97(1%)	1,694.70(2%)
1/1,000	423.31(0%)	2,303.42(1%)

created variables for the girth-constraints, because the minimality check is only called when a variable $e_{i,j}$ is assigned. The results are shown in Table 4.

Table 5 shows the influence of the parameter frequency on SMS's performance. For this analysis, we took the unsatisfiable case for $f_4(25)$ with the degree interval $[4, 6]$ and the satisfiable case for $f_4(27)$ with the degree interval $[4, 5]$.

For the satisfiable instance, it seems that lower frequencies perform better peaking with 1/500 while for the unsatisfiable instance, it performed best with a frequency of 1/100. For very low frequencies, both instances do not perform well. The reason seems to be the high fraction of the time spent in MinCheck and possibly the increased number of added symmetry breaking clauses.

7 Diameter-2-Critical Graphs

The *diameter* of a graph G is the largest distance over all vertex pairs 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. The study of extremal properties of graphs with prescribed diameter has been initiated by Erdős and Rényi in the early 1960s [17] and has been the subject of intensive research. An important topic in the field is the characterization of diameter-d-critical graphs [7, 8, 26, 39]. An intriguing open problem is whether the Murty–Simon Conjecture [7] holds, which states that if G is a diameter-2-critical graph with n vertices and m edges, then $n \leq \lfloor n^2/4 \rfloor$, with equality precisely for the complete bipartite graph $K_{\lceil n/2 \rceil, \lfloor n/2 \rfloor}$ (i.e., similar to Mantel's Theorem mentioned above). Using the list of non-isomorphic graphs generated with Nauty [42], Radosavljević and Živković [47] computed all diameter-2-critical graphs with up to 10 vertices. Also, Dailly et al. [13] reported on a "computer search" for graphs with up to 11 vertices, focusing on graphs with a certain number of edges. With SMS, we were able to extend these results to graphs with 13 vertices. The basis for this computation is a SAT encoding that produces for given integers n and m a propositional CNF formula $D_2(n, m)$ which is satisfiable if and only if there is a diameter-2-critical graph G with n vertices and m edges. As above, one can construct G from the satisfying assignment.

7.1 Encoding

Equivalently to the above definition, a graph with at least three vertices is diameter-2-critical if and only if (i) its diameter is at most 2 and (ii) when any edge is deleted, the diameter is larger than 2. We observe that property (i) holds for graphs with diameter one, i.e., complete graphs. However, for graphs with order of at least 3, deleting any edge would still result in a diameter of at most 2, which violates property (ii).

Our encoding of $D_2(n, m)$ handles both properties separately. We use variables $c_{i,j,k}$ to indicate whether k is a common neighbor of i and j, i.e., $c_{i,j,k} \leftrightarrow (e_{i,k} \wedge e_{j,k})$. To encode property (i), we use

$$\bigwedge_{1 \leq i < j \leq n} \Big(e_{i,j} \vee \bigvee_{\substack{1 \leq k \leq n, \\ k \notin \{i,j\}}} c_{i,j,k} \Big).$$

For property (ii), we use the following observation:

Observation 4. *A graph G with diameter 2 is diameter-2-critical if and only if for every edge $ij \in E(G)$ at least one of the following cases holds:*

- *The vertices i, j have no common neighbor.*
- *There is a vertex $k \notin \{i, j\}$ such that $ik \notin E(G)$ and the vertex j is the only common neighbor of i and k.*
- *Similarly, there is a vertex $k \notin \{i, j\}$ such that $jk \notin E(G)$ and the vertex i is the only common neighbor of j and k.*

Proof. ($\Leftarrow$) In the first case, $dist_{G-ij}(i, j) > 2$ in the later cases either $dist_{G-ij}(i, k) > 2$ or $dist_{G-ij}(j, k) > 2$.

($\Rightarrow$) Let G be a diameter-2-critical graph. By definition, for every edge $ij \in E(G)$, there exists two vertices $k_1, k_2 \in V(G)$ such that $dist_{G-ij}(k_1, k_2) > 2$ and $dist_G(k_1, k_2) \leq 2$. This means, that each path between k_1 and k_2 with length ≤ 2 contains the edge ij and there is a path between k_1 and k_2 with length ≤ 2 containing the edge ij. Let $ij \in E(G)$ and $k_1, k_2 \in V(G)$ such that $dist_{G-ij}(k_1, k_2) > 2$ and $dist_G(k_1, k_2) \leq 2$. We distinguish the following cases:

- $k_1 = k_2$: this cannot be the case, otherwise $dist_{G-ij}(k_1, k_2) = 0$, in conflict with $dist_{G-ij}(k_1, k_2) > 2$.
- $k_1 \notin \{i, j\}, k_2 \notin \{i, j\}$, and $k_1 \neq k_2$: this is not possible, because each path in G containing the vertices k_1, k_2 and the edge ij would have length > 2, in conflict with the existence of a path between k_1 and k_2 with length ≤ 2 containing the edge ij.

Table 6. Results for Generating All Diameter-2-Critical Graphs with $n \leq 13$

		SMS		Static	
n	#-graphs	#-sol	time	#-sol	time
3	1	1	0.00(23%)	1	0.00
4	2	2	0.00(22%)	2	0.00
5	3	3	0.00(23%)	4	0.00
6	5	5	0.00(32%)	11	0.00
7	10	10	0.01(37%)	32	0.01
8	30	30	0.05(47%)	163	0.04
9	103	103	0.17(39%)	1,024	0.30
10	519	519	0.73(26%)	9,836	3.58
11	**3,746**	3,748	4.48(18%)	135,010	77.00
12	**40,866**	40,876	47.71(14%)	t.o.	t.o.
13	**688,120**	688,143	1,184.47(8%)	t.o.	t.o.

- $\{k_1, k_2\} = \{i, j\}$: in this case, the vertices i, j cannot have a common neighbor, otherwise $dist_{G-ij}(k_1, k_2) = dist_{G-ij}(i, j) = 2$.
- $k_1 = i$, and $k_2 \notin \{i, j\}$: the vertices i, k_2 are not adjacent and have only j as common neighbor, otherwise $dist_{G-ij}(k_1, k_2) = dist_{G-ij}(i, k_2) \leq 2$.
- $k_1 = j$, and $k_2 \notin \{i, j\}$: analog to the previous case, the vertices j, k_2 are not adjacent and have only i as common neighbor.
- Because the distance of two vertices is symmetric, we omit the following two cases: (i) $k_2 = i$, and $k_1 \notin \{i, j\}$, and (ii) $k_2 = j$, and $k_1 \notin \{i, j\}$.

The case distinction is exhaustive. Thus, we showed that if a graph G is diameter-2-critical and $ij \in E(G)$, then (i) the vertices i, j have no common neighbor, (ii) there is a vertex $k \notin \{i, j\}$ such that $ik \notin E(G)$ and j is the only common neighbor of i and k, or (iii) there is a vertex $k \notin \{i, j\}$ such that $jk \notin E(G)$ and i is the only common neighbor of j and k. □

The case distinction from Observation 4 can be encoded as follows:

- $F_1 := \bigwedge_{k \in V \setminus \{i,j\}} \neg c_{i,j,k}$
- $F_2 := \bigvee_{k \in V \setminus \{i,j\}} \left(\neg e_{i,k} \wedge \bigwedge_{l \in V \setminus \{i,j,k\}} \neg c_{i,k,l}\right)$
- $F_3 := \bigvee_{k \in V \setminus \{i,j\}} \left(\neg e_{j,k} \wedge \bigwedge_{l \in V \setminus \{i,j,k\}} \neg c_{j,k,l}\right)$

Hence, we can encode property (ii) as follows:

$$\bigwedge_{1 \leq i < j \leq n} e_{i,j} \rightarrow (F_1 \vee F_2 \vee F_3).$$

7.2 Results

We use the encoding $D_2(n, m)$ to enumerate all canonical, diameter-2-critical graphs in $\mathcal{G}_n$. Therefore, we run SMS repeatedly; each time we find a new graph, we explicitly exclude it from the search space until no further graph is found. We us a frequency of 1/20 and a cutoff of 20,000.

Table 6 shows the results of this computation. Column "#-sol" gives the number of solutions found; column "#-graphs" gives the number of solutions up to isomorphism computed produced by SMS and filtered with Nauty; column "time" gives the runtime in seconds; as above, the percentage of the runtime that has been spent for the minimality check is given in parenthesis. Column "Static"

Table 7. Confirming the Murty–Simon Conjecture for $n \in \{14, \ldots, 19\}$

n	total time	max-time	#-seq
14	16 minutes	9 seconds	406
15	2.2 hours	20 seconds	1,729
16	7.9 hours	35 seconds	3,480
17	19.9 hours	74 seconds	5,620
18	3.4 days	132 seconds	12,974
19	23.7 days	312 seconds	50,054

Column "n" denotes the number of vertices, column "#-seq" the number of degree sequences, "max-time" the maximal runtime of a single degree sequence, and "total-time" the accumulated runtime over all sequences.

gives the number of solutions found with the static symmetry breaking method, without filtering isomorphic solutions. The static version could not find all solutions for $n = 12$ within an hour.

The number of solutions for $n \in \{11, 12, 13\}$, stated in boldface, was unknown, as this goes beyond a generate-and-test approach. One can see from the table, even with a cutoff parameter of 20,000, SMS produces only a few isomorphic copies, which can be easily filtered. Checking the computed graphs, we could confirm the Murty–Simon Conjecture for graphs with up to 13 vertices.

If we know the degree of the vertices in advance, we can create an initial ordered partition for the minimality check, such that only vertices with the same degree can be permuted. So, we can use SMS for every possible degree sequence. Trivially, all cases where a vertex has degree 1 can be excluded. Additionally, we can use the following theorem by Fan [19] to discard further sequences in advance ($d_G(v)$ denotes the degree of vertex v in G):

Theorem 2 ([19]). *If G is a diameter-2-critical with n vertices and m edges, then $\sum_{v \in V(G)} d_G(v)^2 \leq \frac{4}{15}n^3$. If $n \leq 24$, then $m \leq \lfloor n^2/4 \rfloor$.*

Consequently, since $\sum_{v \in V(G)} d_G(v) = 2m$, Theorem 2 limits the degree sequences, i.e., we can list all of them and separately search for graphs with SMS satisfying a specific degree sequence. Further, we can restrict the maximal degree using the following theorem:

Theorem 3 ([25]). *If G is a diameter-2-critical with n vertices and maximum degree $\geq 0.7 \cdot n$, then G has fewer than $\lfloor n^2/4 \rfloor$ edges.*

Parallelizing over all relevant combinations of vertex degrees resulting from Theorems 1 and 2, we could confirm the conjecture for $n \in \{14, \ldots, 19\}$. We give some details on the computation in Table 7.

Corollary 2. *The Murty–Simon Conjecture holds for graphs with up to 19 vertices.*

8 Ramsey Numbers and Sets of Ramsey Graphs

Our last showcase example are Ramsey numbers and sets of Ramsey graphs. Given two positive integers x and y, $\mathcal{R}(x, y, n)$ denotes the set of all graphs up to isomorphism with n vertices not containing an independent set of size x nor a clique of size y. The Ramsey number $R(x, y)$ is the smallest integer such that $\mathcal{R}(x, y, R(x, y)) = \emptyset$. The problem of finding lower and upper bounds for Ramsey numbers is a well-known research topic started by Ramsey in 1930 [48, 49]. Despite the great interest and effort in this research area, exact Ramsey numbers are only known for small values of x and y. Table 8 summarizes these results. For symmetric reasons, i.e., $R(x, y) = R(y, x)$, we omit the lower half.

Table 8. Exact Values and Bounds for Ramsey Numbers $R(x, y)$

	$x = 3$	$x = 4$	$x = 5$	$x = 6$	$x = 7$	$x = 8$
$y = 3$	6	9	14	18	23	28
$y = 4$		18	25	36–41	49–61	59–84
$y = 5$			43–48	58–87	80–143	101–216
$y = 6$				102–165	115–298	134–495

Although we could not improve these bounds, to the best of our knowledge, we are the first to be able to produce proofs for sets $\mathcal{R}(x, y, n)$ being complete. This can be achieved by a verification process described by Kirchweger et al. [33]. First, we generate the additional symmetry breaking clauses needed and verify them for correctness. To efficiently verify the correctness of a symmetry breaking clause, we additionally store the nc-certificate used for deriving the clause. Each clause blocks a partially defined graph G. Given the nc-certificate π, we check whether G has a π-indicator pair. This is possible in polynomial time by iterating over all vertex pairs in lexicographic order, and verifying that all more important vertex pairs than the most important (G, π)-greater vertex pair are (G, π)-greater-equal.

Next, we extend our original formula with the additional clauses and produce a **deletion, reverse asymmetric tautology (DRAT)** proof [12] by a second solver supporting proof production. The DRAT proof itself can be verified by a separate proof checker. To verify that the set of (canonical) graphs is complete, we add clauses ensuring that at least one edge/non-edge is different for each graph. If the set is complete, SMS reports unsat.

It remains to verify that the list of graphs doesn't contain isomorphic copies. Since we generate lexicographically minimal graphs, isomorphic copies would be identical. However, for this argument one needs to trust that the minimality check is indeed complete for fully defined graphs. To provide a certificate for minimality, we transform the adjacency matrix of a graph $G \in \mathcal{G}_n$ into a CNF-formula M_G, such that M_G is unsatisfiable if and only if G is canonical. Consequently, we can use DRAT proofs for certifying that G is canonical. Let us informally sketch the construction of M_G based on Proposition 2. We use propositional variables $p_{v,u}$ representing a permutation and $q_{i,j}$ for indicating whether (i, j) is an indicator pair with respect to the permutation represented by the variables $p_{v,u}$ and the graph G. Encoding that there is an indicator pair and the variables $p_{v,u}$ indeed represent a permutation is straightforward.

8.1 Encoding and Results

We encode the problem by ensuring that every subset of vertices of size x contains an edge, and for every subset of vertices of size y, there is a non-edge. Hence, our encoding $F(x, y, n)$ is as follows:

$$F(x, y, n) = \bigwedge_{S \subseteq \{1,\dots,n\}, |S|=x} \Big(\bigvee_{i,j \in S, i<j} e_{i,j} \Big) \wedge \bigwedge_{S \subseteq \{1,\dots,n\}, |S|=y} \Big(\bigvee_{i,j \in S, i<j} \neg e_{i,j} \Big).$$

Using this encoding, we are able to enumerate the sets $\mathcal{R}(3, 5, n)$ and $\mathcal{R}(4, 4, n)$. The results are summarized in Table 9. The results themselves were already computed by other intricate tools that do not provide any kind of certification. The graphs were previously used for computing the Ramsey number $R(4, 5)$ by "gluing" together graphs in $\mathcal{R}(3, 5, n_1)$ and $\mathcal{R}(4, 4, n_2)$, i.e., by fixing some induced subgraphs [43]. More precisely, for every graph $G \in \mathcal{R}(4, 5, n)$, one can pick a vertex v. The neighborhood of the vertex must form a graph in $\mathcal{R}(4, 4, n_1)$ and all non-adjacent vertices form a graph in $\mathcal{R}(3, 5, n_2)$, where n_1 is the size of the neighborhood of v and $n_2 = n - n_1 - 1$.

Table 9. Results for Generating All $(4, 4, n)$ Graphs and $(3, 5, n)$ Graphs in Seconds

	$\mathcal{R}(3,5,n)$		$\mathcal{R}(4,4,n)$	
n	#-sol	time	#-sol	time
1	1	0.00	1	0.00
2	2	0.00	2	0.00
3	3	0.00	4	0.00
4	7	0.00	9	0.00
5	13	0.00	24	0.00
6	32	0.00	84	0.01
7	71	0.01	362	0.03
8	179	0.03	2,079	0.16
9	290	0.05	14,701	1.25
10	313	0.05	103,706	11.99
11	105	0.05	546,356	80.92
12	12	0.03	1,449,166	531.44
13	1	0.02	1,184,231	227.95
14			130,816	28.70
15			640	0.66
16			2	0.15
17			1	0.14
total	1,029	0.24	3,432,184	883.41

9 Conclusion

We presented SMS, a novel approach for SAT-based graph generation that utilizes dynamic symmetry breaking. A key ingredient of SMS is the concept of partially defined graphs and an algorithm that checks the lexicographical minimality of such graphs. We evaluated a prototype implementation of SMS on three showcase problems from Extremal Combinatorics, related to the girth, diameter, and forbidden cliques and independent sets, respectively. We compared SMS with static symmetry breaking, using the same encoding for the graph property and the same underlying SAT solver for both approaches. We think that this double strategy might be of independent interest, as it supports comparing the very same SAT-encoding on both methods. The experiments show encouraging results for SMS, in particular on unsatisfiable instances. As a side effect of our experiments on diameter-2-critical graphs, we could compute some values that haven't been known before and confirm the Murty–Simon Conjecture for graphs with up to 19 vertices.

Since the release of the preliminary version of SMS in 2021 [36], some extensions and applications have come to the fore. SMS has been extended to matroids, contributing to advancements in Rota's Basis Conjecture [33]. Additionally, it has been adapted for hypergraphs to tackle the Erdős–Faber–Lovász Conjecture [32], extended to encompass planar and directed graphs [34], and utilized to compute smallest universal graphs [55], smallest unsatisfiable (k, s)-SAT instances [54], and the exact rainbow cycle numbers for small values [37]. Also, it was the first method to show that the size of a Kochen-Specker vector system is at least 24 [31]. Notably, this result was independently matched using a SAT + CAS approach [38], in which a SAT solver interacts with a computer algebra system.

In the future, we plan to use SMS to make progress on many other conjectures related to combinatorial structures and try to find better constraint-based approaches for graph generation and enumeration.

References

[1] 2023a. The On-Line Encyclopedia of Integer Sequences. Number of Connected Graphs with n Nodes. Retrieved from http://oeis.org/A001349

[2] 2023b. The On-Line Encyclopedia of Integer Sequences. Number of Connected Labeled Graphs with n Nodes. Retrieved from http://oeis.org/A001187

[3] E. Abajo and A. Diánez. 2015. Exact Value of $ex(n; \{C_3, \ldots, C_s\})$ for $n \leq \lfloor \frac{25(s-1)}{8} \rfloor$. *Discrete Math.* 185, 1–7. DOI: https://doi.org/10.1016/j.dam.2014.11.021

[4] László Babai and Eugene M. Luks. 1983. Canonical Labeling of Graphs. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*. ACM, New York, NY, 171–183. DOI: https://doi.org/10.1145/800061.808746

[5] Rolf Backofen and Sebastian Will. 2002. Excluding Symmetries in Constraint-Based Search. *Constraints* 7, 3–4 (2002), 333–349. DOI: https://doi.org/10.1023/A:1020533821509

[6] Béla Bollobás. 1978. *Extremal Graph Theory*. Academic Press.

[7] Louis Caccetta and Roland Häggkvist. 1979. On Diameter Critical Graphs. *Discrete Math.* 28, 3 (1979), 223–229. DOI: https://doi.org/10.1016/0012-365X(79)90129-8

[8] Ya-Chen Chen and Zoltán Füredi. 2005. Minimum Vertex-Diameter-2-Critical Graphs. *J. Graph Theory* 50, 4 (2005), 293–315. DOI: https://doi.org/10.1002/jgt.20111

[9] Geoffrey Chu, Maria Garcia de la Banda, Christopher Mears, and Peter J. Stuckey. 2014. Symmetries, Almost Symmetries, and Lazy Clause Generation. *Constraints* 19, 4 (2014), 434–462. DOI: https://doi.org/10.1007/s10601-014-9163-9

[10] Michael Codish, Graeme Gange, Avraham Itzhakov, and Peter J. Stuckey. 2016. Breaking Symmetries in Graphs: The Nauty Way. In *Proceedings of the Principles and Practice of Constraint Programming—22nd International Conference, CP'16*, Lecture Notes in Computer Science, Vol. 9892. Springer Verlag, 157–172. DOI: https://doi.org/10.1007/978-3-319-44953-1_11

[11] Michael Codish, Alice Miller, Patrick Prosser, and Peter J. Stuckey. 2019. Constraints for Symmetry Breaking in Graph Representation. *Constraints* 24, 1 (2019), 1–24. DOI: https://doi.org/10.1007/s10601-018-9294-5

[12] Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt, Matt Kaufmann, and Peter Schneider-Kamp. 2017. Efficient Certified RAT Verification. In *Automated Deduction—CADE 26*. Leonardo de Moura (Ed.). Springer International Publishing, Cham, 220–236.

[13] Antoine Dailly, Florent Foucaud, and Adriana Hansberg. 2019. Strengthening the Murty-Simon Conjecture On Diameter 2 Critical Graphs. *Discrete Math.* 342, 11 (2019), 3142–3159. DOI: https://doi.org/10.1016/j.disc.2019.06.023

[14] Jo Devriendt, Bart Bogaerts, and Maurice Bruynooghe. 2017. Symmetric Explanation Learning: Effective Dynamic Symmetry Handling for SAT. In *Proceedings of the Theory and Applications of Satisfiability Testing—SAT 2017 — 20th International Conference*, Lecture Notes in Computer Science, Vol. 10491. Springer Verlag, 83–100. DOI: https://doi.org/10.1007/978-3-319-66263-3_6

[15] Jo Devriendt, Bart Bogaerts, Maurice Bruynooghe, and Marc Denecker. 2016. Improved Static Symmetry Breaking for SAT. In *Proceedings of the Theory and Applications of Satisfiability Testing—SAT 2016—19th International Conference*, Lecture Notes in Computer Science, Vol. 9710. Springer Verlag, 104–122. DOI: https://doi.org/10.1007/978-3-319-40970-2_8

[16] Jo Devriendt, Bart Bogaerts, Broes De Cat, Marc Denecker, and Christopher Mears. 2012. Symmetry Propagation: Improved Dynamic Symmetry Breaking in SAT. In *Proceedings of the IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI12*. IEEE Computer Society, 49–56. DOI: https://doi.org/10.1109/ICTAI.2012.16

[17] P. Erdős and A. Rényi. 1962. On a problem in the theory of graphs. *Publ. Math. Inst. Hung. Acad. Sci.* 7A, 623–641.

[18] Torsten Fahle, Stefan Schamberger, and Meinolf Sellmann. 2001. Symmetry Breaking. In *Proceedings of the Principles and Practice of Constraint Programming—CP'01*, Lecture Notes in Computer Science, Vol. 2239. Springer Verlag, 93–107. DOI: https://doi.org/10.1007/3-540-45578-7_7

[19] Genghua Fan. 1987. On Diameter 2-Critical Graphs. *Discret. Math.* 67, 3 (1987), 235–240. DOI: https://doi.org/10.1016/0012-365X(87)90174-9

[20] Katalin Fazekas, Aina Niemetz, Mathias Preiner, Markus Kirchweger, Stefan Szeider, and Armin Biere. 2023. IPASIR-UP: User Propagators for CDCL. In *Proceedings of the 26th International Conference on Theory and Applications of Satisfiability Testing, SAT'23*, LIPIcs, Vol. 271. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 8:1–8:13. DOI: https://doi.org/10.4230/LIPICS.SAT.2023.8

[21] David K. Garnick, Y. H. Harris Kwong, and Felix Lazebnik. 1993. Extremal Graphs Without Three-Cycles or Four-Cycles. *J. Graph Theory* 17, 5 (1993), 633–645. DOI: https://doi.org/10.1002/jgt.3190170511

[22] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Philipp Wanko. 2016. Theory Solving Made Easy with Clingo 5. In *Proceedings of the Technical Communications of the 32nd International Conference on Logic Programming (ICLP'16)*, OASICS, Vol. 52. Dagstuhl, 2:1–2:15. DOI: https://doi.org/10.4230/OASIcs.ICLP.2016.2

[23] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. 2014. Clingo = ASP + Control: Preliminary Report. http://arXiv.org/abs/1405.3694arXiv:1405.3694.

[24] Ian P. Gent, Karen E. Petrie, and Jean-François Puget. 2006. Symmetry in Constraint Programming. In *Handbook of Constraint Programming*. Foundations of Artificial Intelligence, Vol. 2. Elsevier, 329–376.

[25] Teresa Haynes, Michael Henning, Lucas Merwe, and Anders Yeo. 2014. A Maximum Degree Theorem for Diameter-2-Critical Graphs. *Open Math.* 12, 12 (2014), 1882–1889. DOI: https://doi.org/doi:10.2478/s11533-014-0449-3

[26] Teresa W. Haynes, Michael A. Henning, Lucas C. van der Merwe, and Anders Yeo. 2015. Progress on the Murty-Simon Conjecture on Diameter-2 Critical Graphs: A Survey. *J. Comb. Optim.* 30, 3 (2015), 579–595. DOI: https://doi.org/10.1007/s10878-013-9651-7

[27] Marijn J. H. Heule. 2019. Optimal Symmetry Breaking for Graph Problems. *Math. Comput. Sci.* 13, 4 (2019), 533–548. DOI: https://doi.org/10.1007/S11786-019-00397-5

[28] Avraham Itzhakov and Michael Codish. 2016. Breaking Symmetries in Graph Search with Canonizing Sets. *Constraints* 21, 3 (2016), 357–374. DOI: https://doi.org/10.1007/S10601-016-9244-Z

[29] Avraham Itzhakov and Michael Codish. 2020. Incremental Symmetry Breaking Constraints for Graph Search Problems. In *Proceedings of the 34rth AAAI Conference on Artificial Intelligence, AAAI'20, the 32nd Innovative Applications of Artificial Intelligence Conference, IAAI'20, the 10th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI'20*. AAAI Press, 1536–1543. DOI: https://doi.org/10.1609/AAAI.V34I02.5513

[30] Avraham Itzhakov and Michael Codish. 2023. Breaking Symmetries with High Dimensional Graph Invariants and Their Combination. In *Proceedings of the Integration of Constraint Programming, Artificial Intelligence, and Operations Research—20th International Conference, CPAIOR'23,* Lecture Notes in Computer Science, Vol. 13884. Springer Verlag, 133–149. DOI: https://doi.org/10.1007/978-3-031-33271-5_10

[31] Markus Kirchweger, Tomás Peitl, and Stefan Szeider. 2023a. Co-Certificate Learning with SAT Modulo Symmetries. In *Proceedings of the 32nd International Joint Conference on Artificial Intelligence, IJCAI'23.* 1944–1953. DOI: https://doi.org/10.24963/ijcai.2023/216

[32] Markus Kirchweger, Tomás Peitl, and Stefan Szeider. 2023b. A SAT Solver's Opinion on the Erdős-Faber-Lovász Conjecture. In *Proceedings of the 26th International Conference on Theory and Applications of Satisfiability Testing, SAT'23,* LIPIcs, Vol. 271. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 13:1–13:17. DOI: https://doi.org/10.4230/LIPIcs.SAT.2023.13

[33] Markus Kirchweger, Manfred Scheucher, and Stefan Szeider. 2022. A SAT Attack on Rota's Basis Conjecture. In *Proceedings of the Theory and Applications of Satisfiability Testing—SAT 2022—25th International Conference,* Proceedings. 1–18. DOI: https://doi.org/10.4230/LIPIcs.SAT.2022.4

[34] Markus Kirchweger, Manfred Scheucher, and Stefan Szeider. 2023c. SAT-Based Generation of Planar Graphs. In *Proceedings of the 26th International Conference on Theory and Applications of Satisfiability Testing, SAT'23,* LIPIcs, Vol. 271. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 14:1–14:18. DOI: https://doi.org/10.4230/LIPIcs.SAT.2023.14

[35] Markus Kirchweger and Stefan Szeider. 2021a. *SAT Modulo Symmetries for Graph Generation.* DOI: https://doi.org/10.5281/zenodo.5170575

[36] Markus Kirchweger and Stefan Szeider. 2021b. SAT Modulo Symmetries for Graph Generation. In *Proceedings of the 27th International Conference on Principles and Practice of Constraint Programming (CP'21),* LIPIcs. Dagstuhl, 39:1–39:17. DOI: https://doi.org/10.4230/LIPIcs.CP.2021.34

[37] Markus Kirchweger and Stefan Szeider. 2024. Computing Small Rainbow Cycle Numbers with SAT Modulo Symmetries. To appear.

[38] Zhengyu Li, Curtis Bright, and Vijay Ganesh. 2023. A SAT Solver and Computer Algebra Attack on the Minimum Kochen-Specker Problem. CoRR. DOI: https://doi.org/10.48550/arXiv.2306.13319

[39] Po-Shen Loh and Jie Ma. 2016. Diameter Critical Graphs. *J. Combin. Theory Ser. B* 117, 34–58. DOI: https://doi.org/10.1016/j.jctb.2015.11.004

[40] W. Mantel. 1907. Problem 28. *Wiskundige Opgaven* 10, 60–61.

[41] João Marques-Silva and Sharad Malik. 2018. Propositional SAT Solving. In *Handbook of Model Checking*. Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith and Roderick Bloem (Eds.), Springer, 247–275. DOI: https://doi.org/10.1007/978-3-319-10575-8_9

[42] Brendan D. McKay and Adolfo Piperno. 2014. Practical Graph Isomorphism, II. *J. Symbolic Comput.* 60, 94–112. DOI: https://doi.org/10.1016/j.jsc.2013.09.003

[43] Brendan D. McKay and Stanisław P. Radziszowski. 1995. R(4,5)=25. *J. Graph Theory* 19, 3 (May 1995), 309–322. DOI: https://doi.org/10.1002/jgt.3190190304

[44] Hakan Metin, Souheib Baarir, Maximilien Colange, and Fabrice Kordon. 2018. CDCLSym: Introducing Effective Symmetry Breaking in SAT Solving. In *Proceedings of the Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS'18,* Lecture Notes in Computer Science, Vol. 10805. Springer, 99–114. DOI: https://doi.org/10.1007/978-3-319-89960-2_6

[45] Amit Metodi and Michael Codish. 2012. Compiling Finite Domain Constraints to SAT with BEE. *Theory Pract. Log. Program.* 12, 4–5 (2012), 465–483. DOI: https://doi.org/10.1017/S1471068412000130
[46] Jean-Francois Puget. 2003. Symmetry Breaking Using Stabilizers. In *Proceedings of the Principles and Practice of Constraint Programming—CP'23*, Lecture Notes in Computer Science, Vol. 2833. Francesca Rossi (Ed.). Springer Verlag, 585–599. DOI: https://doi.org/10.1007/978-3-540-45193-8_40
[47] Jovan Radosavljević and Miodrag Živković. 2020. The List of Diameter-2-Critical Graphs with at Most 10 Nodes. *IPSI Trans. Adv. Res.* 16, 1 (Jan. 2020), 1–5. Retrieved from http://ipsitransactions.org/journals/papers/tar/2020jan/p9.pdf.
[48] Stanislaw Radziszowski. 2021. Small Ramsey Numbers. *Electron. J. Comb.* 1, 1–29. DOI: https://doi.org/10.37236/21
[49] F. P. Ramsey. 1930. On a Problem of Formal Logic. *Proc. Lond. Math. Soc.* s2–30, 1 (1930), 264–286. DOI: https://doi.org/10.1112/plms/s2-30.1.264
[50] Bas Schaafsma, Marijn Heule, and Hans van Maaren. 2009. Dynamic Symmetry Breaking by Simulating Zykov Contraction. In *Proceedings of the Theory and Applications of Satisfiability Testing—SAT'09*, Lecture Notes in Computer Science, Vol. 5584. Springer Verlag, 223–236. DOI: https://doi.org/10.1007/978-3-642-02777-2_22
[51] Jianmin Tang, Yuqing Lin, Camino Balbuena, and Mirka Miller. 2009. Calculating the Extremal Number $ex(v; \{C_3, C_4, \ldots, C_n\})$. *Discr. Appl. Math.* 157, 9 (2009), 2198–2206. DOI: https://doi.org/10.1016/j.dam.2007.10.029
[52] P. Wang, G. W. Dueck, and S. MacMillan. 2001. Using Simulated Annealing to Construct Extremal Graphs. *Discr. Math.* 235, 1–3 (2001), 125–135. DOI: https://doi.org/10.1016/S0012-365X(00)00265-XCombinatorics (Prague, 1998).
[53] Douglas B. West. 2021. *Combinatorial Mathematics*. Cambridge University Press. 220 pages.
[54] Tianwei Zhang, Tomáš Peitl, and Stefan Szeider. 2024. Small Unsatisfiable k-CNFs with Bounded Literal Occurrence. In *Proceedings of the 27th International Conference on Theory and Applications of Satisfiability Testing (SAT'24)*, LIPIcs. Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik.
[55] Tianwei Zhang and Stefan Szeider. 2023. Searching for Smallest Universal Graphs and Tournaments with SAT. In *Proceedings of the 29th International Conference on Principles and Practice of Constraint Programming, CP'23*, LIPIcs, Vol. 280. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 39:1–39:20. DOI: https://doi.org/10.4230/LIPICS.CP.2023.39

Received 8 November 2023; revised 14 February 2024; accepted 5 May 2024