Computing Small Rainbow Cycle Numbers with SAT Modulo Symmetries

Markus Kirchweger
Algorithms and Complexity Group, TU Wien, Austria

Stefan Szeider
Algorithms and Complexity Group, TU Wien, Austria

Abstract

Envy-freeness up to any good (EFX) is a key concept in Computational Social Choice for the fair division of indivisible goods, where no agent envies another's allocation after removing any single item. A deeper understanding of EFX allocations is facilitated by exploring the rainbow cycle number ($R_f(d)$), the largest number of independent sets in a certain class of directed graphs. Upper bounds on $R_f(d)$ provide guarantees to the feasibility of EFX allocations (Chaudhury et al., EC 2021).

In this work, we precisely compute the numbers $R_f(d)$ for small values of d, employing the SAT modulo Symmetries framework (Kirchweger and Szeider, CP 2021). SAT modulo Symmetries is tailored specifically for the constraint-based isomorph-free generation of combinatorial structures. We provide an efficient encoding for the rainbow cycle number, comparing eager and lazy approaches. To cope with the huge search space, we extend the encoding with invariant pruning, a new method that significantly speeds up computation.

2012 ACM Subject Classification Mathematics of computing → Extremal graph theory; Software and its engineering → Constraint and logic languages; Hardware → Theorem proving and SAT solving; Mathematics of computing → Graph enumeration

Keywords and phrases EFX, rainbow cycle number, SAT modulo Symmetries, combinatorial search

Digital Object Identifier 10.4230/LIPIcs.CP.2024.37

Category Short Paper

Supplementary Material
Software (Source Code): `https://github.com/markirch/sat-modulo-symmetries/`
Software (Documentation): `https://sat-modulo-symmetries.readthedocs.io/`

Funding The authors acknowledge the support from the Austrian Science Fund (FWF), project 10.55776/P36688, and the Vienna Science and Technology Fund (WWTF), project ICT19-065.

1 Introduction

The quest for a fair division of indivisible goods, a fundamental challenge in economics, computer science, and social choice theory, hinges on developing allocation protocols that balance fairness and efficiency. Envy-freeness is among the most celebrated concepts in this domain, a criterion that ensures no agent prefers another's allocation over their own. However, the indivisibility of goods often precludes the possibility of genuinely envy-free allocations, prompting researchers to explore various relaxations of this ideal. Envy-freeness up to any good (EFX) is a particularly prominent and well-studied relaxation, which requires that no agent should envy another's allocation after the hypothetical removal of any single good from the latter's allocation [2, 4, 5, 6, 7, 8].

Despite its conceptual appeal, the existence of EFX allocations remains unresolved in general settings, marking it as one of the most intriguing open problems in discrete fair division. This has led to a vibrant area of study that seeks to establish the existence of such allocations and explores the combinatorial structures underlying these problems. A key development in this research trajectory is the exploration of the *rainbow cycle number* $R_f(d)$. This combinatorial constant, which will be defined more formally in Section 2, is the largest integer k such that a directed k-partite graph exists in which every vertex of each block has an incoming edge from any other block, each block has size at most d and the whole graph contains no *rainbow cycle*, i.e., a cycle that runs through every block at most once. Bounds on $R_f(d)$ directly influence the feasibility and construction of approximate EFX allocations with minimal discarded goods. Variants $R_i(d)$ and $R_p(d)$ of the rainbow cycle number (see Section 2 for definitions) have also been studied [3].

Determining $R_f(d)$ and its variants is a natural question of extremal graph theory, in particular to zero-sum combinatorics, independently of its application to EFX-allocations [1, 7]. The first upper bound $R_f(d) \in O(d^4)$ was shown by Chaudhury et al. [7]. This bound was later improved to $d^{2+o(1)}$ [3], and $O(d \cdot \log(d))$ [1, 17].

It is known that for all d, $R_i(d) \leq R_p(d) \leq R_f(d)$, $R_i(d) \geq d$, and it has been conjectured that $R_f(d) = d$; however, this conjecture has only been established so far for $d \leq 3$ in the general setting, and for $d \leq 6$ for $R_i(d)$ [23]. Computing $R_f(d)$ for $d > 3$ is challenging since the number of combinatorial objects to consider explodes.

In this paper, we develop a constraint-based approach within the *SAT modulo Symmetries (SMS)* framework to obtain the upper bounds of $R_f(d)$, which allows us to show $R_f(4) = 4$ and $R_p(5) = 5$. SMS is a recently proposed framework for graph generation modulo isomorphisms under constraints [21]. SMS combines a conflict-driven (CDCL) SAT solver with a custom propagator (since recently via the IPASIR-UP interface [12]) to determine whether a partial assignment encoding a graph can be extended to a canonical, fully defined graph. By employing dynamic symmetry breaking, SMS efficiently searches the vast combinatorial space, avoiding the exploration of isomorphic copies.

We provide an efficient propositional encoding for the rainbow cycle numbers, comparing eager and lazy approaches to ensure the absence of rainbow cycles. The eager approach adds constraints to the encoding at the beginning, while the lazy approach uses a custom propagator to check for the presence of rainbow cycles during the search.

To cope with the huge search space, we introduce the novel technique of *invariant pruning*, which significantly speeds up the computation by strengthening the encoding with graph invariants not known a priori. Invariant pruning exploits the fact that one can observe a significant speedup if a graph invariant is explicitly stated in the encoding and proceeds along a complete decision tree on values for the invariant under consideration (in our concrete setting, it is the maximum degree).

We also employ further symmetry-breaking techniques within the SMS framework to avoid exploring isomorphic copies of graphs in the search space. The applicable permutations for symmetry breaking are described, and the set of all such permutations is represented using ordered partitions. When invariant pruning is applied, the permutations used for symmetry breaking are further restricted to maintain the invariant properties.

Experimental results demonstrate the effectiveness of our approach, with invariant pruning providing a speedup of almost two orders of magnitude in some cases. The time spent in the propagator to ensure the absence of rainbow cycles is only a small fraction of the total computation time. The results also indicate that restricting the functions to permutations significantly decreases the search space compared to the general case.

2 Preliminaries

For a positive integer n, we write $[n] = \{1, 2, \ldots, n\}$, $\mathcal{F}_n$ for the set of all functions from $[n]$ to $[n]$, and $\mathcal{S}_n$ for the set containing all permutations over $[n]$. We assume familiarity with fundamental notions of propositional logic [22].

Graphs. We review basic notions from graph theory that are relevant to our discussion. All considered graphs are *directed* and simple (i.e., without parallel edges or self-loops, but pairs of edges in the opposite direction are allowed). A *graph* G consists of a set $V(G)$ of vertices and a set $E(G)$ of edges; we denote the edge from $u \in V(G)$ to $v \in V(G)$ by (u, v). The *order* of a graph G is the number of its vertices, $|V(G)|$. We write $\mathcal{G}_n$ to denote the class of all graphs with $V(G) = [n]$. The *adjacency matrix* of a graph $G \in \mathcal{G}_n$, denoted by A_G, is the $n \times n$ matrix where the element $A_G(v, u)$ at row v and column u is 1 if $(v, u) \in E(G)$ and 0 otherwise. We write $N^-_G(v)$ for the *in-neighborhood* of a vertex v in G, i.e., $N^-_G(v) = \{\, u \in V(G) \mid (u, v) \in E(G)\,\}$. The indegree is given by $d^-_G(v) = |N^-_G(v)|$.

Fixed point cycles and rainbow cycles. A function $f \in \mathcal{F}_d$ has a *fixed point* if $f(x) = x$ for some $x \in [d]$. A d*-labeled graph* is a pair (G, f) such that $G \in \mathcal{G}_n$ and $f_e \in \mathcal{F}_d$ for each $e \in E(G)$; f is an *edge labeling* of G. A *fixed point cycle* of a d-labeled graph (G, f) is a simple cycle $(v_1, v_2, \ldots, v_k)$, i.e., $(v_1, v_2), (v_2, v_3), \ldots, (v_k, v_1) \in E(G)$ and $v_i \neq v_j$ for $i \neq j$, such that $f_{(v_k,v_1)} \circ \cdots \circ f_{(v_2,v_3)} \circ f_{(v_1,v_2)}$ has a fixed point.

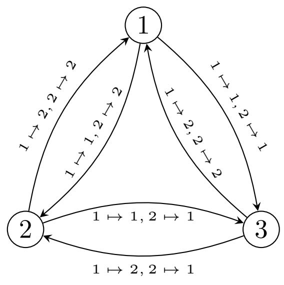
A graph $G \in \mathcal{G}_n$ is ℓ*-partite* if there is a partition $\mathcal{B} = \{B_1, \ldots, B_\ell\}$ of $V(G)$ such that no edge of G has both its ends in the same $B \in \mathcal{B}$. We call the sets in $\mathcal{B}$ *blocks*. Let $G \in \mathcal{G}_n$ be an ℓ-partite graph and $\mathcal{B}$ the corresponding partition of $V(G)$. A *rainbow cycle* is a directed cycle that contains at most one vertex of each block in $\mathcal{B}$.

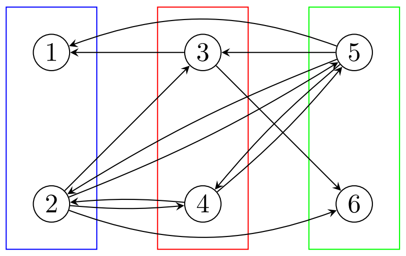
Next, we describe the connection between d-labeled graphs with fixed point cycles and ℓ-partite graphs with rainbow cycles allowing us to focus on the latter. Each d-labeled graph (G, f) with $G \in \mathcal{G}_\ell$ can be transformed into an ℓ-partite graph $H \in \mathcal{G}_{\ell \cdot d}$ such that (G, f) has a fixed point cycle if and only if H has a rainbow cycle. The idea is to introduce for each vertex in $V(G)$ a block with d vertices representing the set $[d]$. The edges in $E(H)$ indicate the mapping, i.e., if $f_{(i,j)}(x) = y$ then there is an edge from the vertex associated with y in the j-th block to the vertex associated with x in the i-th block. More formally

$$E(H) = \{\, ((j-1) \cdot d + f_{(i,j)}(x)\ ,\ (i-1) \cdot d + x) \mid (i, j) \in E(G), x \in [d]\,\}$$

with the partition $\mathcal{B}_{d,\ell} := \{\, \{d \cdot (i-1) + 1, \ldots, d \cdot i\} \mid i \in [\ell]\,\}$.

▶ **Example 1.** We give an example for $d = 2$ and $\ell = 3$. There, we have $\mathcal{B}_{2,3} = \{\{1, 2\}, \{3, 4\}, \{5, 6\}\}$.


1
2
3
$1 \mapsto 2, 2 \mapsto 2$
$1 \mapsto 1, 2 \mapsto 2$
$1 \mapsto 2, 2 \mapsto 2$
$1 \mapsto 1, 2 \mapsto 1$
$1 \mapsto 1, 2 \mapsto 1$
$1 \mapsto 2, 2 \mapsto 1$



On the left side, we see a d-labeled graph. The graph itself is the complete directed graph on 3 vertices. On the right side, we see the corresponding 3-partite graph. The rectangles indicate the blocks. The cycle $(2, 5)$ forms a rainbow cycle because at most one vertex of each block is in the cycle. This also indicates that $f_{(3,1)} \circ f_{(1,3)}$ has 2 as a fixed point.

The number $R_f(d)$ is the largest integer, such that the edges of the complete directed graph of order $R_f(d)$ can be labeled with $f_e \in \mathcal{F}_d$ without a fixed point cycle. Equivalently, there exists an $R_f(d)$-partite graph such that each block has size d, each vertex has exactly one incoming edge from each other block, and there is no rainbow cycle [3]. $R_f(d)$ is also known as a *rainbow cycle number*. $R_p(d)$ and $R_i(d)$ are defined similarly, except for $R_p(d)$, the edge labelings are restricted to permutations and for $R_i(d)$, the edge labelings are restricted to functions of the form $x \mapsto x + k \mod d$ for $k \in \mathbb{N}$. These numbers are related as follows: $R_i(d) \leq R_p(d) \leq R_f(d)$ [3]. Further, it is known that $R_i(d) \geq d$ [7], given by the edge labeling f:

$$f_{(i,j)}(x) = \begin{cases} x & \text{if } i < j, \\ x + 1 \mod d & \textit{otherwise}. \end{cases}$$

EFX. Let $[n]$ be the set of agents, M the set of indivisible goods, and $v_i : 2^M \to \mathbb{R}_{\geq 0}$ an evaluation function for each $i \in [n]$. An *allocation* $A = (A_1, \dots, A_n)$ assigns the goods to the agents, i.e., $A_i \subseteq 2^M$ for $i \in [n]$ such that $A_i \cap A_j = \emptyset$ for $i = j$ and $\bigcup_{i\in[n]} A_i = M$. An allocation A is envy-free if $v_i(A_i) \geq v_i(A_j)$ for all $i, j \in [n]$, in other words, no agent prefers the goods of another agent. An *EFX* (envy-freeness up to any good) allocation is an allocation such that $v_i(A_i) \geq v_i(A_j \setminus \{g\})$ for all $i, j \in [n], g \in A_j$. An α-EFX allocation for some $\alpha \in (0, 1]$ is an allocation A such that $v_i(A_i) \geq \alpha \cdot v_i(A_j \setminus \{g\})$ for all $i, j \in [n], g \in A_j$. In a partial-EFX allocation, not all goods need to be assigned, i.e., $\bigcup_{i\in[n]} A_i \subseteq M$.

The relation between the rainbow cycle number and partial α-EFX allocations is given by the following theorem:

▶ **Theorem 2** ([7]). *Let $\varepsilon \in (0, 1/2]$ and let $g(y)$ be the smallest integer d such that $d \cdot R_f(d) \geq y$. Then, there is always a partial $(1-\varepsilon)$-EFX allocation with at most $4n/(\varepsilon \cdot g(2n\varepsilon))$ many unallocated items.*

SAT modulo Symmetries. SAT Modulo Symmetries (SMS) [21] is a recently proposed framework for graph generation modulo isomorphisms under constraints. The original work is restricted to undirected graphs, but later work extends it to directed graphs [20]. SMS combines a conflict-driven (CDCL) SAT solver with a custom propagator to determine whether a partial assignment (which encodes a graph) can be extended to a *canonical* fully defined graph; if not, the solver immediately backtracks. A canonical graph is a distinguished member of its isomorphism class – in SMS, this is typically the graph with the lexicographically smallest adjacency matrix[1]. SMS thus employs what is known as dynamic symmetry breaking [11, 24]. For undirected graphs, prior work has addressed static symmetry breaking [9, 10, 15, 16].

An important concept in the context of SMS are 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

[1] We refer to the original work [20] for the precise canonical form.

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 $V(G) = [n]$. During solving, the presence or absence of some edges is not known, hence a partially defined graph is a suitable way to represent the current solver state. Based on that, the SMS framework additionally enables adding custom propagators to refute partially defined graphs during search.

3 Encoding

In this section, we describe propositional encodings $F_{d,\ell}$ which are satisfiable if and only if $R_f(d) \geq \ell$ for $d, \ell \in \mathbb{N}$. In the satisfiable case, we can extract a directed graph from a model of the formula. Using these formulas, we compute the exact number $R_f(d)$ using several SAT calls. In this work, we are only interested in validating the conjecture $R_f(d) = d$ for small d. For that, showing unsatisfiability of $F_{d,d+1}$ is sufficient.

Given $d, \ell \in \mathbb{N}$, the number of vertices in the searched-for directed graph is $n_{d,\ell} := d \cdot \ell$. The variables $e_{i,j}$ for $i, j \in [n_{d,\ell}], i \neq j$, denote whether the edge (i,j) is present.
First, we ensure that the resulting graph is ℓ-partite with respect to the partition $\mathcal{B}_{d,\ell}$:

$$\mathit{partition}_{d,\ell} = \bigwedge_{B \in \mathcal{B}_{d,\ell}} \bigwedge_{\substack{i,j \in B,\\ i<j}} \neg e_{i,j}.$$

Each vertex in each block has exactly one incoming edge from each other block. In our encoding, the direction of edges is reversed, as it is faster in our experiments in combination with SMS. We encode that each vertex in each block has exactly one outgoing edge to each other block as follows:

$$\mathit{function}_{d,\ell} := \bigwedge_{B_1 \in \mathcal{B}_{d,\ell}} \bigwedge_{i \in B_1} \bigwedge_{\substack{B_2 \in \mathcal{B}_{d,\ell},\\ B_1 \neq B_2}} \Big(\bigvee_{j \in B_2} e_{i,j} \wedge \bigwedge_{\substack{j_1, j_2 \in B_2,\\ j_1 < j_2}} (\neg e_{i,j_1} \wedge \neg e_{i,j_2}) \Big).$$

To ensure that no rainbow cycle is present, we distinguish between an eager and a lazy approach. In the first case, we add an encoding to the formula $F_{d,\ell}$ at the beginning. We also refer to this as the static approach. The lazy approach uses a propagator to ensure the absence of a rainbow cycle during search.

3.1 Using a propagator to ensure acyclicity

Given a partially defined graph $G \in \mathcal{P}_{d \cdot \ell}$ and the partition $\mathcal{B}_{d,\ell}$, we want to decide whether a rainbow cycle is present. We use a second SAT solver to check the presence of a rainbow cycle. In the satisfiable case, we use the model to compute a rainbow cycle, which is then excluded by a single clause, ensuring that at least one of the directed edges in the cycle is not present. We design the encoding dependent on the edge variables and use assumptions to fix the graph we want to test. We consider all undefined edges to be absent, because this doesn't introduce additional cycles.

We use variables s_i for $i \in [n_{d,\ell}]$ to indicate whether vertex i is part of the selected cycle. Only one vertex of each block can be selected:

$$\bigwedge_{B \in \mathcal{B}_{d,\ell}} \bigwedge_{i,j \in B, i<j} (\neg s_i \vee \neg s_j).$$

Additionally, we ensure that each selected vertex has exactly one incoming and one outgoing edge to another selected vertex. This is done by using additional variables $se_{i,j}$, which indicate whether the directed edge (i,j) is present and both vertices are selected:

$$se_{i,j} \leftrightarrow (e_{i,j} \wedge s_i \wedge s_j).$$

With these additional variables, we ensure that the indegree and outdegree of the induced subgraph given by the selected vertices is exactly 1. Selecting a minimal rainbow cycle results in a model of the formula. Note that the encoding allows multiple disjoint cycles to be selected, but in this case, we only select one cycle from the model.

We use the IPASIR-UP [12] interface to integrate this propagator.

3.2 Static encoding

We present an encoding exponential in size to ensure that no rainbow cycle is present. One possibility is enumerating all potential cycles and, for each, add a clause to ensure that at least one edge is not present. We use a more compact version using additional variables $p_{u,v,\mathcal{B}'}$ for $B_u, B_v \in \mathcal{B}_{d,\ell}, u \in B_u, v \in B_v, B_u \neq B_v$, and $\mathcal{B}' \subseteq \mathcal{B}_{d,\ell} \setminus \{B_u, B_v\}$. The variables indicate whether there is a colorful path from u to v, i.e., a path only using at most one vertex from each block, only containing vertices in $B_u \cup B_v \cup \bigcup_{B \in \mathcal{B}'} B$.

We have the following constraints using B_v as shorthand for indicating the block containing vertex v:

- $p_{u,v,\emptyset} \leftrightarrow e_{u,v}$: this is the base case.
- $(e_{u,v} \wedge p_{v,w,\mathcal{B}'}) \rightarrow p_{u,w,\mathcal{B}' \cup \{B_v\}}$ if $B_u \notin \mathcal{B}'$: this ensures that if there is a directed edge from u to v and a colorful path from v to w not using the block containing u, then there is a colorful path from u to w additionally using block B_v.
- $p_{u,v,\mathcal{B}'} \rightarrow p_{u,v,\mathcal{B}' \cup \{B\}}$ for $B \in \mathcal{B}' \setminus \{B_u, B_v\}$: adding additional blocks preserves rainbow paths.
- $\neg e_{u,v} \vee \neg p_{v,u,\mathcal{B}' \setminus \{B_u, B_v\}}$: this guaranties that no rainbow cycle is present.

For very small values of d and ℓ, the static approach is still valid, although the number of variables and clauses is exponential.

3.3 Invariant pruning

We introduce the new technique of *invariant pruning* in the context of SMS. A graph *invariant* is a property of a graph invariant under graph isomorphisms. Examples of invariants are the maximum indegree and the maximum outdegree. The goal is to strengthen the formula by computing invariants that are unknown beforehand.

We explain invariant pruning for the rainbow cycle problem regarding the invariant maximum indegree. Let Δ be an upper bound on the maximum indegree, for example, the number of vertices. Instead of solving the formula directly, we fix the first vertex, i.e., the vertex with label 1, to have exactly indegree Δ and all other vertices indegree $\leq \Delta$ in addition to the standard formula $F_{d,\ell}$. If this formula is unsatisfiable, then the maximum indegree is at most $\Delta - 1$, since for any vertex v, there is permutation mapping the vertex v to the first vertex and preserving the partition.

Now, either one solves $F_{d,\ell}$ with the restriction that the maximum indegree is at most $\Delta - 1$ or again tries to refute the case of a vertex having indegree exactly $\Delta - 1$ by fixing the first to have indegree $\Delta - 1$.

In our experiments, we continue pruning the invariant until we reach a certain value at which solving the formula becomes fast. As we see in the experiments in Section 4, this gives a tremendous speedup.

It is intriguing that explicitly demonstrating the impossibility of certain invariants using a SAT solver and then solving the formula proves faster than solely running the solver on the original formula. This phenomenon might stem from the solver finding it easier to identify a "bad" property when it is already associated with a vertex and does not appear arbitrarily.

Table 1 Results for computing $R_f(d)$ and $R_p(d)$ in seconds. For "prop", we additionally provide the fraction of time spent in the cycle-propagator.

	$R_f(d)$		$R_p(d)$	
d	static	prop	static	prop
3	0.03	0.07 (23%)	0.01	0.02 (13%)
4	1603.10	1374.23 (22%)	0.23	1.30 (18%)
5	t.o.	t.o.	1160.23	1236.24 (14%)

3.4 Symmetry breaking

An important part of solving graph search problems with constraint-based methods is symmetry breaking to avoid isomorphic copies in the search space. The vertices can be permuted arbitrarily except the partition $\mathcal{B}_{d,\ell}$ must be preserved. The applicable permutations can be described as follows:

$$\{\, \pi \in \mathcal{S}_{n_{d,\ell}} \mid \forall B \in \mathcal{B}_{d,\ell}\ \exists B' \in \mathcal{B}_{d,\ell} : \pi(B) = B' \,\}.$$

In other words, the image of each block maps to another block.

We break all these symmetries using SMS. Let $\mathcal{O} = [B_1, \ldots, B_m]$ be an ordered partition, $l_i = \sum_{j<i} B_i$ and $u_i = \sum_{j\leq i} B_i$. We associate with an ordered partition $\mathcal{O}$ the set of permutations

$$\mathrm{Perm}(\mathcal{O}) = \{\, \pi \in \mathcal{S}_{n_{d,\ell}} \mid l_i \leq \pi(i) \leq u_i \text{ for all } i \in [n_{d,\ell}] \,\}.$$

An ordered partition describes a range to which each vertex can be mapped. For example $\mathrm{Perm}([\{2,3\},\{1\}]) = \{\{2 \mapsto 1, 3 \mapsto 2, 1 \mapsto 3\}, \{2 \mapsto 2, 3 \mapsto 1, 1 \mapsto 3\}\}$.

SMS allows breaking symmetries given by a set of ordered partitions. The set of all applicable permutations for $d, \ell \in \mathbb{N}$ can be represented as follows:

$$\{\, [\{(\pi(1)-1)\cdot d + 1, \ldots, \pi(1)\cdot d\}, \ldots, \{(\pi(\ell)-1)\cdot d + 1, \ldots, \pi(\ell)\cdot d\}] \mid \pi \in \mathcal{S}_\ell \,\}.$$

The permutation $\pi \in \mathcal{S}_\ell$ describes how the blocks are swapped; the vertices within the block can be permuted arbitrarily.

Note that the presented symmetry breaking is not necessarily compatible with invariant pruning and needs to be adapted slightly. For the example of fixing the indegree of the first vertex, we must further restrict the permutations applicable for symmetry breaking. The first vertex is restricted to map to itself, which also fixes the first block.

4 Experimental Results

In this section, we present the results of our computations. We perform several experiments on a cluster of machines with Intel Xeon E5-2640 v4 processors at 2.40GHz, running Ubuntu 18.04 on Linux 4.15. The source code is available on GitHub[2] and the documentation on Read the Docs[3]. All experiments are executed with a single thread.

[2] `https://github.com/markirch/sat-modulo-symmetries`
[3] `https://sat-modulo-symmetries.readthedocs.io/applications/#computing-small-rainbow-cycle-numbers`

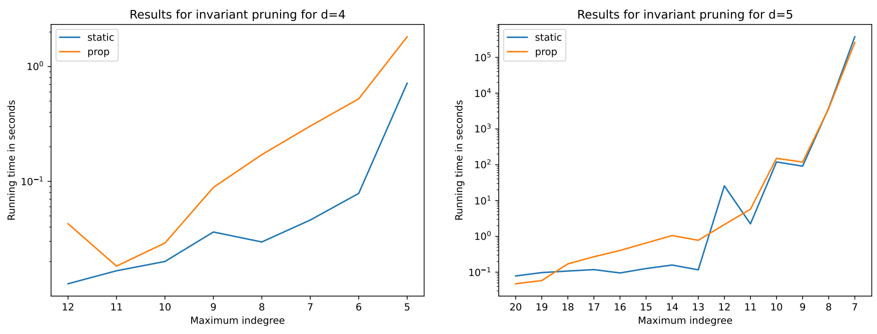


Figure 1 Results for computing $R_f(d)$ using invariant pruning.

Table 1 summarizes the duration in seconds for computing $R_f(d)$ and $R_p(d)$, i.e., showing unsatisfiability for $F_{d,d+1}$. “prop” refers to the version using an external propagator for ensuring the absence of a rainbow cycle (Section 3.1) and “static” to the static encoding (Section 3.2). The results $R_f(4) = R_p(4) = 4$, and $R_p(5) = 5$ were not known before.

As a sanity check, we also enumerate all graphs up to isomorphism encoded by $F_{3,3}$. It results in 64 graphs, two of which encode permutations. For $F_{4,4}$ we stop after enumerating more than 3 million graphs. This also means that there are many extremal graphs. To highlight the importance of symmetry breaking, we run the SAT encoding for $R_p(4)$ without SMS, which doesn’t terminate within a day, whilst with SMS, the computation terminates in less than 3 seconds.

We see that the time spent in the propagator is not a bottleneck of the computation. For both $R_f(d)$ and $R_p(d)$, the static and propagator version perform relatively similar for the hardest solved case. The result also shows that restricting the functions only to permutations decreases the search drastically.

Next, we provide results computing $R_f(d)$ using invariant pruning. As invariant, we use the maximum indegree. Note that the outdegree is already known to be $\ell - 1$, i.e., exactly d for the case $\ell = d + 1$. The results are summarized in Figure 1. The figure gives for given fixed maximum indegree the time in seconds to refute this case. We use a logarithmic y-axis. We see that the cases with high indegree are refuted relatively quickly.

For $d = 4$, we can determine within 0.95 seconds that the maximum indegree is strictly smaller than 5 using the static approach and 2.99 seconds using the propagator. The case where the maximum indegree is ≤ 4 is solved in 36.17 and 31.25 seconds, respectively. This means we have a speedup of almost two orders of magnitude using invariant pruning. For $d = 5$, we can restrict the indegree to ≤ 6 within 5 days, but refuting the remaining cases is not feasible yet.

We test a second invariant with invariant pruning, namely the maximum indegree of a vertex with respect to only one block, i.e.,

$$\max_{v \in V(G)} \max_{B \in \mathcal{B}_{d,\ell} \setminus \{B_v\}} |\{\, u \mid (u,v) \in E(G) \,\}|.$$

We call this the *block degree.* If the block degree is determined to be 1, then this is equivalent to the edge labelings being permutations. For $d = 5$, using the previous results that the maximum indegree is ≤ 6, we compute that the block degree is ≤ 2 within 5 days.

Following previous work [18, 19], we verify the reasoning of the solver using DRAT proofs [14] for all our experiments and check the correctness of the symmetry-breaking clauses with a separate script. This is done by first letting the solver run, including propagators, and storing all additionally produced clauses. Next, we feed the original formula, enhanced with the additional clauses, to a SAT solver to produce a DRAT proof, which can then be checked by a DRAT proof checker [25]. Note that it is also possible to use a recently introduced incremental proof format [13] to produce proofs including clauses from propagators without the necessity of running a SAT solver twice.

5 Conclusion

We have investigated the computation of exact rainbow numbers using a constraint-based approach with SAT modulo Symmetries, focusing on the efficacy of invariant pruning to expedite the search process. While our study has primarily examined two invariants, further exploration could uncover additional invariants that might prove beneficial in efficiently handling the case $d = 5$.

A point of improvement for invariant pruning is reusing learned clauses from different invariants by using assumptions for asserting a certain invariant and using one solver incrementally.

An interesting avenue for future research is to evaluate the impact of invariant pruning on various graph generation and enumeration problems. Additionally, we see a potential for the applicability of invariant pruning techniques to solve other highly symmetric formulas not directly tied to graphs.

References

1 Hannaneh Akrami, Bhaskar Ray Chaudhury, Jugal Garg, Kurt Mehlhorn, and Ruta Mehta. EFX allocations: Simplifications and improvements. *CoRR*, abs/2205.07638, 2022. `doi:10.48550/arXiv.2205.07638`.

2 Georgios Amanatidis, Evangelos Markakis, and Apostolos Ntokos. Multiple birds with one stone: Beating 1/2 for EFX and GMMS via envy cycle elimination. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 1790–1797. AAAI Press, 2020. `doi:10.1609/AAAI.V34I02.5545`.

3 Benjamin Aram Berendsohn, Simona Boyadzhiyska, and László Kozma. Fixed-point cycles and approximate EFX allocations. In *47th International Symposium on Mathematical Foundations of Computer Science, MFCS 2022, August 22-26, 2022, Vienna, Austria*, volume 241 of *LIPIcs*, pages 17:1–17:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. `doi:10.4230/LIPICS.MFCS.2022.17`.

4 Ben Berger, Avi Cohen, Michal Feldman, and Amos Fiat. Almost full EFX exists for four agents. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 4826–4833. AAAI Press, 2022. `doi:10.1609/AAAI.V36I5.20410`.

5 Ioannis Caragiannis, David Kurokawa, Hervé Moulin, Ariel D. Procaccia, Nisarg Shah, and Junxing Wang. The unreasonable fairness of maximum Nash welfare. *ACM Trans. Economics and Comput.*, 7(3):12:1–12:32, 2019. `doi:10.1145/3355902`.

6 Bhaskar Ray Chaudhury, Jugal Garg, and Kurt Mehlhorn. EFX exists for three agents. *J. ACM*, 71(1):4:1–4:27, 2024. `doi:10.1145/3616009`.

7 Bhaskar Ray Chaudhury, Jugal Garg, Kurt Mehlhorn, Ruta Mehta, and Pranabendu Misra. Improving EFX guarantees through rainbow cycle number. In *EC '21: The 22nd ACM Conference on Economics and Computation, Budapest, Hungary, July 18-23, 2021*, pages 310–311. ACM, 2021. `doi:10.1145/3465456.3467605`.

8 Bhaskar Ray Chaudhury, Telikepalli Kavitha, Kurt Mehlhorn, and Alkmini Sgouritsa. A little charity guarantees almost envy-freeness. *SIAM J. Comput.*, 50(4):1336–1358, 2021. doi:10.1137/20M1359134.

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

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

11 Jo Devriendt, Bart Bogaerts, Broes De Cat, Marc Denecker, and Christopher Mears. Symmetry propagation: Improved dynamic symmetry breaking in SAT. In *IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012, Athens, Greece, November 7-9, 2012*, pages 49–56. IEEE Computer Society, 2012. doi:10.1109/ICTAI.2012.16.

12 Katalin Fazekas, Aina Niemetz, Mathias Preiner, Markus Kirchweger, Stefan Szeider, and Armin Biere. IPASIR-UP: user propagators for CDCL. In Meena Mahajan and Friedrich Slivovsky, editors, *26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023, July 4-8, 2023, Alghero, Italy*, volume 271 of *LIPIcs*, pages 8:1–8:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.SAT.2023.8.

13 Katalin Fazekas, Florian Pollitt, Mathias Fleury, and Armin Biere. Certifying incremental SAT solving. In *LPAR 2024: Proceedings of 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning, Port Louis, Mauritius, May 26-31, 2024*, volume 100 of *EPiC Series in Computing*, pages 321–340. EasyChair, 2024. doi:10.29007/PDCC.

14 Marijn J. H. Heule. The DRAT format and DRAT-trim checker. *CoRR*, abs/1610.06229, 2016. URL: http://arxiv.org/abs/1610.06229.

15 Marijn J. H. Heule. Optimal symmetry breaking for graph problems. *Math. Comput. Sci.*, 13(4):533–548, 2019. doi:10.1007/S11786-019-00397-5.

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

17 Shayan Chashm Jahan, Masoud Seddighin, Seyed Mohammad Seyed Javadi, and Mohammad Sharifi. Rainbow cycle number and EFX allocations: (almost) closing the gap. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 2572–2580. ijcai.org, 2023. doi:10.24963/IJCAI.2023/286.

18 Markus Kirchweger, Tomáš Peitl, and Stefan Szeider. Co-certificate learning with SAT modulo symmetries. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 1944–1953. ijcai.org, 2023. doi:10.24963/IJCAI.2023/216.

19 Markus Kirchweger, Manfred Scheucher, and Stefan Szeider. A SAT attack on Rota's Basis Conjecture. In *Theory and Applications of Satisfiability Testing - SAT 2022 - 25th International Conference, Haifa, Israel, August 2-5, 2022, Proceedings*, 2022. doi:10.4230/LIPIcs.SAT.2022.4.

20 Markus Kirchweger, Manfred Scheucher, and Stefan Szeider. SAT-based generation of planar graphs. In Meena Mahajan and Friedrich Slivovsky, editors, *The 26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023), July 04-08, 2023, Alghero, Italy*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.SAT.2023.14.

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

22 Hans Kleine Büning and Theodor Lettman. *Propositional logic: deduction and algorithms*. Cambridge University Press, Cambridge, 1999.

23 Tamás Mészáros and Raphael Steiner. Zero sum cycles in complete digraphs. *Eur. J. Comb.*, 98:103399, 2021. doi:10.1016/J.EJC.2021.103399.

24 Hakan Metin, Souheib Baarir, Maximilien Colange, and Fabrice Kordon. CDCLSym: introducing effective symmetry breaking in SAT solving. In *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018*, volume 10805 of *Lecture Notes in Computer Science*, pages 99–114. Springer, 2018. doi:10.1007/978-3-319-89960-2_6.

25 Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *Theory and Applications of Satisfiability Testing – SAT 2014*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer Verlag, 2014. doi:10.1007/978-3-319-09284-3_31.