Incremental SAT-Based Enumeration of Solutions to the Yang-Baxter Equation

Daimy Van Caudenberg[1][0000−0002−7975−4838], Bart Bogaerts[1,2][0000−0003−3460−4251], and Leandro Vendramin[3][0000−0003−0954−7785]

[1] KU Leuven, Dept. of Computer Science; Leuven.AI, B-3000 Leuven, Belgium
Daimy.VanCaudenberg,Bart.Bogaerts@kuleuven.be
[2] Vrije Universiteit Brussel, Dept. of Computer Science, B-1050 Brussels, Belgium
[3] Vrije Universiteit Brussel, Dept. of Mathematics and Data Science, B-1050 Brussels, Belgium
Leandro.Vendramin@vub.be

Abstract. We tackle the problem of enumerating set-theoretic solutions to the Yang-Baxter equation. This equation originates from statistical and quantum mechanics, but also has applications in knot theory, cryptography, quantum computation and group theory. Non-degenerate, involutive solutions have been enumerated for sets up to size 10 using constraint programming with partial static symmetry breaking [1]; for general non-involutive solutions, a similar approach was used to enumerate solutions for sets up to size 8. In this paper, we use and extend the SAT Modulo Symmetries framework (SMS), to expand the boundaries for which solutions are known. The SMS framework relies on a *minimality check*; we present two solutions to this, one that stays close to the original one designed for enumerating graphs and a new incremental, SAT-based approach. With our new method, we can reproduce previously known results much faster and also report on results for sizes that have remained out of reach so far.

Keywords: satisfiability · symmetries · Yang-Baxter equation · enumeration

1 Introduction

The Yang-Baxter Equation The Yang–Baxter equation (YBE) is a fundamental equation in mathematical physics, with origins in the study of statistical mechanics. Its significance, however, has quickly extended to various areas of pure mathematics, including representation theory and low-dimensional topology. More recently, a discrete version of the equation has become central in algebra and combinatorics. The set-theoretic solutions to this discrete version of the equation were first highlighted by Drinfeld [12], who emphasized the importance of discovering new solutions and the inherent difficulty of this task. He suggested that set-theoretical solutions, in particular, merited further investigation. The study and construction of such solutions began with a series

of pioneering papers [16, 14, 23, 30]. These works apply tools from ring theory, group theory and homological algebra to analyze non-degenerate solutions, as these solutions naturally have groups acting on them. Moreover, these papers demonstrated that these solutions offer a particularly rich source of examples, with numerous applications in algebra [7, 18, 26] and connections with other topics, such as Hopf–Galois structures [8].

Constructing finite solutions of small size is particularly intriguing for several reasons. Firstly, it presents a highly challenging problem at the intersection of computational techniques and algebra. Secondly, solutions provide, for example, concrete colouring invariants of knots [24] and intriguing examples of algebraic structures to study [9]. Thirdly, examples of solutions allow for experimentation, which can reveal patterns that provide deeper insights into the structure of combinatorial solutions. Lastly, the ubiquity of the Yang–Baxter equation suggests that the methods used to construct its solutions could be adapted to other contexts, even those not directly related to the equation itself. For example, these ideas can be applied to the problem of constructing L-algebras [27], something that has direct applications in algebraic logic.

SAT Modulo Symmetries As usual in mathematics, one is not interested in generating *all* solutions, but only all non-isomorphic solutions. One prominent technique for isomorph-free enumeration is SAT Modulo Symmetries (SMS) [21]. Given a formula in conjunctive normal form (CNF) and a possibly empty set of symmetries, the goal of SMS is to find all satisfying assignments that are lexicographically minimal among their symmetric counterparts (here, symmetries of the CNF correspond to isomorphisms of the original problem). To do this, SMS uses a minimality check that takes place *during* the solving phase. In other words, given a (possibly) partial assignment, it is verified whether that assignment can still be extended to a complete, lexicographically minimal assignment.

When the underlying problem is known, the minimality check can take into account that not all complete assignments are considered, but only the assignments that model the formula. Hence, the minimality check can be optimized using domain-specific knowledge. This has been used for enumerating graphs [21], matroids [20] and so-called *Kochen-Specker vector systems* [19].

Our contributions Inspired by the success of SMS, we now extend it to enumerate non-isomorphic solutions to the Yang-Baxter equation. First, we discuss how to encode the YBE in CNF. The most challenging aspect is then to develop a domain-specific minimality check, taking into account the specific problem structure. We can then use this minimality check to learn clauses that are added *during* the solving phase to eliminate parts of the search tree that contain no lexicographically minimal solutions.

We describe and test two implementations of this minimality check. The first is a backtracking approach that stays close to the original SMS work, also making use of clever representations of sets of symmetries. The second one is a new

approach based on incremental SAT solving where the (non-)minimality problem *itself* is encoded as a CNF and a SAT oracle is repeatedly used to search for symmetries that guarantee that the solution at hand is non-minimal. Both approaches result in quite significant speed-ups compared to the previous approach [1]; experiments show that the incremental, SAT-based minimality check outperforms the backtracking approach. Moreover, with this new approach, we can report on results for sizes that have remained out of reach so far.

2 Preliminaries

Yang-Baxter Equation and Cycle Sets The *Yang-Baxter equation* in general is concerned with vector spaces and mappings between them. In this paper, we focus on set-theoretic solutions to this equation, as introduced by Drinfeld [13], and particularly on involutive non-degenerate solutions. These objects are known to be equivalent to *non-degenerate cycle sets* [25], which form a much simpler class of combinatorial objects. We omit the definition of the Yang-Baxter equation in full generality and immediately define cycle sets; we refer the reader for example to Akgün, Mereb and Vendramin [1] for more details.

Definition 1. *A cycle set $(X, \cdot)$ is a pair consisting of a non-empty set X and a binary operation $\cdot$ on X that fulfills the following relations:*

1. *the map $\phi_x : X \to X : y \mapsto x \cdot y$ is bijective for all $x \in X$ and*
2. *for all $x, y, z \in X$:*

$$(x \cdot y) \cdot (x \cdot z) = (y \cdot x) \cdot (y \cdot z). \qquad \textit{(the cycloid equation)}$$

It is called non-degenerate *if the map $X \to X : x \mapsto x \cdot x$ is bijective.*

The rest of this paper is concerned with computer-aided enumeration of all non-degenerate cycle sets of a given (finite) size.

Boolean Satisfiability (SAT) A *(Boolean) variable* takes values in $\{\mathbf{t}, \mathbf{f}\}$; a *literal* is a variable x or its negation $\overline{x}$. A *clause* is a disjunction of literals and a *formula* (in conjunctive normal form) is a conjunction of clauses. A *(partial) assignment* is a consistent set of literals (i.e., a set of literals that does not contain a literal and its negation). An assignment is *complete* (for formula F) if it contains either x or $\overline{x}$ for each variable x occurring in F. An assignment α *satisfies* a formula F if it contains at least one literal from each clause in F. The *SAT problem* consists of deciding for a given formula F whether an assignment exists that satisfies it.

Symmetry in SAT Symmetry has a long history in SAT, with various tools being used to exploit them either before search (e.g., [2, 11, 3]) or during the search (e.g., [28, 4, 10, 21]). The *SAT Modulo Symmetries (SMS)* framework [21] stands out in this list by going a step further than just deciding whether a formula

is satisfiable; instead, the goal is to enumerate assignments that satisfy it. It was designed with use cases in mathematics in mind and in particular it was first used to enumerate graphs that have certain interesting properties. In such use cases, one is often not interested in *all* graphs that satisfy these properties, but only in generating non-isomorphic graphs. The core idea underlying SMS is that we can *(1)* encode as a propositional formula what it means to be a graph that satisfies these interesting properties (or more generally, a suitable mathematical structure) and *(2)* force a SAT solver, during the search, to generate *canonical representations* of each of the classes of isomorphic solutions. The second point is achieved by designing a procedure that is aware of the isomorphisms of the problem at hand, known as the *minimality check*. This takes the state of the SAT solver (a partial interpretation) as input and checks whether it can still be extended to a complete assignment that represents a lexicographically minimal graph (among all graphs isomorphic to it). If not, it forces the solver to abort the current branch of the search tree by analyzing *why* this is no longer possible and learning a new clause from it that is then added to the solver's working formula. In general, this minimality check is incomplete but guaranteed to be complete when run on complete assignments. This minimality check needs to be designed for each application, taking into account the (encoding of) the mathematical problem at hand as well as the structure of the set of isomorphisms. Seress [29] gives a detailed introduction on group theory, for a lighter intoduction we refer readers to Gent et al. [17].

3 SAT Modulo Symmetries for the Yang-Baxter Equation

We now explain how we use the SMS framework to enumerate cycle sets of a given size. To do this we first discuss how to construct a propositional formula that encodes the properties of cycle sets. Next, we discuss the isomorphisms of the problem, and how the minimality check is adapted to deal with these isomorphisms.

3.1 SAT Encoding

A cycle set of size n can be viewed as an $n \times n$ matrix taking at each entry a number between 1 and n and satisfying some structural properties. This matrix represents the binary operation of the cycle set: at position i, j, the value is $i \cdot j$. In our algorithms, we will often take the view of a cycle set being such an integer-valued matrix, while the actual SAT encoding, obviously, talks about lower-level variables. The properties our matrix $C \in X^{n \times n}$ should satisfy are:

1. for all $x, y, z \in X$ with $y \neq z$, $C_{x,y} \neq C_{x,z}$ (ϕ_x is bijective for all $x \in X$),
2. for all $x, y, z \in X$, $C_{C_{x,y},C_{x,z}} = C_{C_{y,x},C_{y,z}}$ (the cycloid equation holds), and
3. for all $x, y \in X$ with $x \neq y$, $C_{x,x} \neq C_{y,y}$ (the cycle set is non-degenerate).

To encode these properties in CNF, we will make use of the one-hot encoding, i.e., for each $i, j, k \in X$ we introduce a Boolean variable $v_{i,j,k}$ which is true if

and only if $C_{i,j} = k$. We then add clauses stating that for each cell, exactly one of its indicator variables must hold;

$$\texttt{ExactlyOne}(\{v_{i,j,k} \mid k \in X\}), \qquad \text{(for each } i, j \in X) \tag{1}$$

where `ExactlyOne` refers to a set of clauses that may use auxiliary variables and that enforce that exactly one of its input variables holds. These clauses are constructed using either a binary encoding or the commander encoding [22]. We then encode the properties in clauses over these variables, ensuring that each number should occur exactly once on each row and the diagonal. Last, we also ensure that the cycloid equation holds. To do this, for each $i, j, k, b \in X$, we introduce a new variable $y_{i,j,k,b}$ that is true precisely when $C_{C_{i,j},C_{i,k}} = b$ and that is also true precisely when $C_{C_{j,i},C_{j,k}} = b$. The complete encoding can be found in Appendix 1.

Fixing the Diagonal The problem defined in the previous section can be solved as is, however, the search space can be significantly reduced using a trick introduced by Akgün, Mereb and Vendramin [1]. In short, if the diagonals of two cycle sets are conjugates, we know that the cycle sets are isomorphic. If the goal is to only enumerate *non-isomorphic* cycle sets, we can safely partition the problem by enumerating the solutions for one diagonal per conjugacy class. Given a cycle set $(X, \cdot)$, its diagonal can be expressed as a permutation of the elements in $X = \{1, \ldots, n\}$. In other words, the set of possible diagonals is given by the symmetric group over X, i.e., $\mathcal{S}_n$. It is known that $\mathcal{S}_n$ has $p(n)$ conjugacy classes, where $p(n)$ is the number of integer partitions of n. Hence, the search space can be drastically reduced by partitioning the problem into $p(n)$ problems with the diagonal fixed to a representative of its conjugacy class.

Given a fixed diagonal (i.e., values for all matrix cells $C_{i,i}$), we fix these variables in our encoding by only introducing variables $v_{i,j,k}$ whenever if $j \neq i$ and $k \neq C_{i,i}$ and simplifying the rest of the theory accordingly. Fixing the diagonal does not only allow us to simplify the encoding, later we will show that it also allows us to optimize the minimality check.

3.2 Isomorphisms and Symmetries

Two cycle sets $(X, \cdot)$ and $(X, \times)$ are isomorphic if and only if there exists an isomorphism $\pi : X \to X$ such that $\pi(x \cdot y) = \pi(x) \times \pi(y)$. For the matrices C and C' corresponding to $(X, \cdot)$ and $(X, \times)$ respectively, this means that $C' = \pi(C)$ where $\pi(C)_{i,j} = \pi^{-1}(C_{\pi(i),\pi(j)})$. When enumerating all cycle sets of a fixed size, we are only interested in enumerating non-isomorphic solutions; we choose to enumerate cycle sets that are lexicographically minimal (among all isomorphic solutions). In other words, we are only interested in those cycle sets $(X, \cdot)$, whose associated matrices are lexicographically smaller than or equal to those of their isomorphic variants. Concretely, given a cycle set $(X, \cdot)$, with associated matrix C, we need the following to hold:

$$\forall \pi \in \mathcal{I} : C \preceq \pi(C),$$

where $\preceq$ is the lexicographical ordering given by $C \preceq C'$ if $C = C'$ or $C_c < C'_c$ for the first[4] cell c where C and C' differ, and $\mathcal{I}$ is the set of isomorphisms of the problem. However, it is important to note that these isomorphisms may not correspond to symmetries of the propositional formula. Indeed, for our specific use case, the commander encoding of the `ExactlyOne` constraint introduces auxiliary variables that break some of the symmetries. As a consequence, symmetries cannot be detected from the CNF encoding but should really be determined by the problem at hand.

In the general case, all isomorphisms $\pi \in \mathcal{S}_n$ with $n = |X|$ need to be considered. However, when working with a fixed diagonal, not all $\pi \in \mathcal{S}_n$ are isomorphisms of the problem, but only the permutations that fix the diagonal. Specifically, the isomorphisms of a problem with fixed diagonal T consist of the set $C_{\mathcal{S}_n}(T)$, i.e., the centralizer of T in the symmetric group. These are exactly the elements $\pi \in \mathcal{S}_n$ for which it holds that $\pi \circ T = T \circ \pi$.

3.3 Minimality Check

We now discuss the core of the SAT Modulo Symmetries framework, namely the minimality check, the goal of which is deciding whether the current assignment (which can be thought of as a partially constructed matrix) can still be extended to a lexicographically minimal matrix (among its symmetric images).

Partial Cycle Sets A *partial cycle set* is a matrix $P \in (2^X)^{n \times n}$ with $n = |X|$, where each cell $c \in X \times X$ of the matrix represents a non-empty *domain* $P_c \subseteq X$ of values that are still possible. Given a (partial) assignment α, we can extract a partial cycle set P^α as follows:

$$\forall c \in X \times X : P^\alpha_c = \{x \in X \mid \overline{v}_{c,x} \notin \alpha\},$$

where if $c = \langle i, j \rangle$, we denote $v_{i,j,x}$ as $v_{c,x}$. In other words, the partial cycle set will contain only values that can still be true according to the assignment.

Given a partial cycle set P and cell c, we say that the entry P_c is *defined* if it equals a singleton $\{x\}$, and write $P_c = x$. In all other cases, we say that P_c is *undefined*. A complete cycle set C is a partial cycle set with no undefined values and for which the cycle set constraints discussed in Section 3.1 hold. In this case, we associate C with an actual cycle set. The set $\mathcal{P}_n$ denotes all partial cycle sets, similarly, the set of all complete cycle sets is denoted $\mathcal{C}_n$. A partial cycle set $P \in \mathcal{P}_n$ can be extended to another partial (or complete) cycle set $P' \in \mathcal{P}_n$, if for all cells c it holds that $P'_c \subseteq P_c$. Given a partial cycle set $P \in \mathcal{P}_n$, we denote the set of complete cycle sets that P can be extended to by $\mathcal{X}(P)$.

Example 1. Given a partial cycle set

$$P = \begin{bmatrix} \{2\} & \{1\} & \{3\} \\ \{2\} & \{1\} & \{3\} \\ \{1,2\} & \{1,2\} & \{3\} \end{bmatrix}, \text{ we have that } \mathcal{X}(P) = \left\{ \begin{bmatrix} 2\ 1\ 3 \\ 2\ 1\ 3 \\ 1\ 2\ 3 \end{bmatrix}, \begin{bmatrix} 2\ 1\ 3 \\ 2\ 1\ 3 \\ 2\ 1\ 3 \end{bmatrix} \right\}.$$

[4] Given a fixed order of the cells.

The goal of the minimality check is, given a partial cycle set, to determine whether it is possible to extend it to a lexicographically minimal complete cycle set. One way to achieve this would be to loop over all (exponentially many) such extensions and check for minimality. This would, however, clearly not be feasible. Instead, inspired by and building on the work of the original SMS paper [21], we take a different approach, and we will search for a symmetry π (in our case, this is a permutation of X that respects the fixed diagonal) that guarantees that for every complete extension C of P,

$$\pi(C) \prec C.$$

We will call such a permutation a *witness of non-minimality*, and the goal of the minimality check is to find such *witnesses*.

Let us describe a sufficient condition for finding such witnesses that forms the basis of our algorithms. Given $P \in \mathcal{P}_n$ and a permutation π, we can apply π to P as follows; for each cell $c = \langle i, j \rangle$, we define the image of c under π as $\pi(c) = \langle \pi(i), \pi(j) \rangle$. Given a non-empty domain $S \subseteq X$, we define $\pi(S) = \{\pi(x) \mid x \in S\}$. Using this, we define $\pi(P)$ as the partial cycle set where $\pi(P)_c = \pi^{-1}(P_{\pi(c)})$ for each cell c. Note that for complete cycle sets, this coincides with the definition of $\pi(C)$ given above. We also define a lexicographical order $\trianglelefteq$ over partial cycle sets, with the specific characteristic that if $P \triangleleft P'$ then for all extensions $C \in \mathcal{X}(P)$ and $C' \in \mathcal{X}(P')$ it holds that $C \prec C'$. Hence, if the minimality check can find a permutation π such that $\pi(P) \triangleleft P$, we have that $\pi(C) \prec C$ for all extensions $C \in \mathcal{X}(P)$, i.e., that π is a witness of non-minimality. In order to define $\triangleleft$, we note that $P \triangleleft P'$ should guarantee that

- there is a cell c, the value of which in P is guaranteed to be strictly smaller than that of P'_c, and,
- for all cells $c' < c$, the value of c' in P is guaranteed to be at most the value it takes in P'.

Since we want to compare the value of cells where the value is potentially not determined yet, we extend the order on values to domains as follows.

Definition 2. *Let S and S' be two non-empty subsets of X, we define*

- $S \trianglelefteq S'$ *if and only if* $\max S \leq \min S'$ *and*
- $S \triangleleft S'$ *if and only if* $\max S < \min S'$.

Definition 3. *Let P and P' be two partial cycle sets and let c be a cell. We say that P* is below *P'* up to cell *c (and denote this $P \trianglelefteq_c P'$) if for all cells $c' \leq c$ it holds that $P_{c'} \trianglelefteq P'_{c'}$.*

Definition 4. *We say P is* strictly smaller *than P' (and denote this $P \triangleleft P'$) if there is a cell c such that*

- $P \trianglelefteq_{\mathrm{pred(c)}} P'$, *where* $\mathrm{pred}(c)$ *is the cell immediately preceding c in the lexicographic ordering, and*
- $P_c \triangleleft P'_c$.

We say that P is at most *P' (and denote this $P \trianglelefteq P'$) if either $P \triangleleft P'$ or if for all cells c, $P_c \trianglelefteq P'_c$.*

These definitions guarantee strong properties of complete extensions of P and P'. Proofs of our results can be found in Appendix 2.

Proposition 1. *Let P and P' be two partial cycle sets. The following properties hold.*

1. *If $P \triangleleft P'$, then for all $C \in \mathcal{X}(P)$ and $C' \in \mathcal{X}(P')$, it holds that $C \prec C'$.*
2. *If $P \trianglelefteq P'$, then for all $C \in \mathcal{X}(P)$ and $C' \in \mathcal{X}(P')$, it holds that $C \preceq C'$.*
3. *If $P \trianglelefteq P'$ and $P' \trianglelefteq P$, then $P = P'$ and both cycle sets are complete.*

This now gives rise to a sufficient condition for being a witness of non-minimality (which allows us to conclude that there are no lexicographically minimal extensions of P) or that some values of a cell are impossible (which would allow us to propagate extra information during search).

Theorem 1. *Given a partial cycle set P and permutation π, we have that:*

- *if $\pi(P) \triangleleft P$, then π is a witness of non-minimality,*
- *If $\pi(P) \trianglelefteq_c P$ then there is an extension P' of P such that*
  1. *P' and $\pi(P')$ are fully defined on all $c' \leq c$, and*
  2. *all $\preceq$-minimal extensions $C \in \mathcal{X}(P)$ are also extensions of P'.*

Hence, when given a partial cycle set P, not only permutations π for which $\pi(P) \triangleleft P$ are useful, but also those for which $\pi(P) \trianglelefteq_c P$ (if P is not complete yet).

3.4 Clause Learning

In this section, we explain which clauses are constructed after the minimality check has found a permutation π that can be used to refine or exclude the current assignment α. Let us first focus on the case where π is a witness of non-minimality, i.e., we will describe the clause that justifies backtracking.

In principle, we could learn a clause that simply eliminates the current partial assignment by $\bigvee_{\ell \in \alpha} \overline{\ell}$, i.e., stating that at least one literal needs to be different. However, we want to learn a clause that captures the essence of why π is a witness of minimality so that it excludes as many assignments as possible. Explaining why π is a witness of non-minimality boils down to:

- explaining why $\pi(P)_{c'} \trianglelefteq P_{c'}$ for each $c' < c$, and
- explaining why $\pi(P)_c \triangleleft P_c$.

Hence, the clause will state that (at least) one of these conditions has to change. For $\pi(P)_{c'} \trianglelefteq P_{c'}$ to change, either $\pi(P)_{c'}$ should be able to take a value larger than $\min P_{c'}$ or $P_{c'}$ should be able to take a value below $\max \pi(P)_{c'}$. The explanation for $\pi(P)_c \triangleleft P_c$ is similar. As such, the clause that we learn is:

$$\bigvee_{x \geq \min P_c} v_{\pi(c),\pi(x)} \vee \bigvee_{x \leq \max \pi(P)_c} v_{c,x} \vee \bigvee_{c' < c} \left(\bigvee_{x > \min P_{c'}} v_{\pi(c'),\pi(x)} \vee \bigvee_{x < \max \pi(P)_{c'}} v_{c',x} \right) \tag{2}$$

Propagation For propagation, everything is completely analogous to the case where we find a witness of non-minimality. The reason is that literal ℓ can be propagated in α using Theorem 1 if and only if π would be a witness of non-minimality in $\alpha \cup \{\overline{\ell}\}$. There is one important design decision to make here; often there are multiple literals that can be propagated. We can add a clause for every such literal, or we can choose one of them and only propagate that. The propagation of a single literal often causes more propagations at the same row of the cycle set, resulting in further eliminations of values from the same cell. If our minimality check detects multiple potential propagations, they will always be over the same cell. In this case, one could prefer to avoid adding propagation constraints for things that can already be caught by the unit propagation phase that follows. So, while adding all clauses is feasible, we have chosen to only propagate one literal at a time and let the solver continue.

Optimization The clause that is learned in this way can be optimized further by taking the problem structure into account in some ways. In particular, we know that for several sets of variables an `ExactlyOne` constraint holds (e.g., all variables $v_{c,\cdot}$ for any cell c, or all variables $v_{i,\cdot,x}$ for a fixed i and x). If for such a set S of variables (for which we know exactly one will be true in every satisfying assignment) and for some variable $s \in S$, our clause C contains $\bigvee_{s' \in S, s' \neq s} s'$, then this can be replaced by simply stating $\overline{s}$. Indeed, since exactly one of the variables holds, stating that the last one is false is equivalent to stating that one of the others is true. Similarly, if the clause already contains $\overline{s}$, all literals s' with $s' \in S$ can safely be removed from the clause.

For example, assume we are searching for cycle sets of size 4 and have fixed a diagonal that has $P_{\langle 1,1\rangle} = 1$. Suppose the clause (2) is $v_{1,2,2} \vee v_{1,2,3}$, then this expresses that the cell $\langle 1, 2\rangle$ should contain a 2 or a 3. Now this is the same as saying that it cannot contain the value four: $\overline{v}_{1,2,4}$. This simple trick can be used to get shorter clauses with better propagation properties.

4 Implementating the Minimality Check

Given a partial cycle set $P \in \mathcal{P}_n$, the goal of the minimality check is to find an isomorphism π of the problem that guarantees that $\pi(P) \trianglelefteq (P)$ so that Theorem 1 guarantees that we can either discard this partial cycle set or refine it. Inspired by the original SMS papers, the minimality check can perform a backtracking search (over the space of permutations!) with a strong pruning mechanism. It iteratively refines a so-called *partial permutation*, until either a useful permutation (satisfying one of the conditions of Theorem 1) is found, or until it is certain that no such permutations exist.[5]

On the other hand, the minimality check can also be viewed as a combinatorial search problem. More specifically, given the current assumptions, i.e., the

[5] We will discuss that it is sometimes useful not to do a complete check here, as long as completeness is guaranteed on complete assignments.

current (partial) cycle set, we want to decide whether there exists an isomorphism that is a witness of non-minimality. If no such permutation exists, we know that the current cycle set is lexicographically minimal; otherwise, we can use the satisfying assignment to extract the witness.

4.1 Backtracking Approach

We first describe the backtracking approach based on the original SMS paper [21]. A *partial permutation* is a function $\pi : X \to 2^X \setminus \emptyset$ that maps every element of X to a non-empty set of values. Intuitively, the value $\pi(x)$ represents the possible images of x under any completion of π. A partial permutation is called *complete* if for all $x \in X$, $\pi(x)$ is a singleton and if $\pi(x) \neq \pi(y)$ for all $y \in X$ where $y \neq x$. In this case, we identify π with the actual permutation. Given two partial permutations π and π', we say that π' *extends* π if and only if for all $x \in X$ it holds that $\pi'(x) \subseteq \pi(x)$.

Our algorithm will recursively refine the permutation using the given partial cycle set until it holds that for all complete extensions π' of π, $\pi'(P) \trianglelefteq P$ (where the inequality will be guaranteed to be strict if P is complete). To do this, we recursively assign a value y to $\pi(x)$, starting with $x = 1$, and refine the obtained partial permutation using available (domain-specific) knowledge. To refine a partial permutation π after making a new assignment, we need to take the following properties into account:

- the image of P under π should be smaller than or equal to P, and
- the partial permutation π can be extended to an isomorphism of the problem.

As mentioned earlier, the algorithm first assigns a value $x_1 \in \pi(1)$ to $\pi'(1)$. Because the completion of π' needs to be well-defined, we refine π' such that x_1 is no longer an option for other elements in $X \setminus \{1\}$. To ensure that $\pi'(P) \trianglelefteq P$, we need to enforce that $\pi'(P)_{\langle 1,1\rangle} \trianglelefteq P_{\langle 1,1\rangle}$, or equivalently, $\max \pi'(P)_{\langle 1,1\rangle} \leq \min P_{\langle 1,1\rangle}$. Hence, we can refine π' such that

$$\max \pi'(P)_{\langle 1,1\rangle} = \max \pi'^{-1}(P_{\langle \pi'(1),\pi'(1)\rangle}) \leq \min P_{\langle 1,1\rangle}.$$

Some care is needed here: even though we know the possible values in the cell $P_{\langle \pi'(1),\pi'(1)\rangle}$, since we are still constructing π', we do not know exactly what the values in $\pi'^{-1}(P_{\langle \pi'(1),\pi'(1)\rangle})$ will be. This will on the one hand cause extra propagation: if $x \in P_{\langle \pi'(1),\pi'(1)\rangle}$, we will propagate that $\pi^{-1}(x) \neq y$ for any $y > \min P_{\langle 1,1\rangle}$. But after doing this there might still be multiple options left for this cell. In that case, we make a further choice on refining π so that for each $x \in P_{\langle \pi'(1),\pi'(1)\rangle}$, $\pi^{-1}(x)$ is fixed (and again, bijectivity is enforced).

Last, we need to ensure that the extended permutation π' can still be extended to an isomorphism of the problem. If the diagonal is not fixed, all permutations $\pi \in \mathcal{S}_n$ are isomorphisms and hence no extra measures need to be taken. If however, a diagonal T is fixed, only the permutations $\pi \in C_{\mathcal{S}_n}(T)$ are isomorphisms of the current problem being solved. Note that a permutation π fixes the diagonal if it maps each cycle of the diagonal onto a same-length cycle,

in the same order. As a consequence, as soon as $\pi'(x)$ is fixed, for each y in the same cycle as x, $\pi'(y)$ will be fixed accordingly. For example, consider the diagonal (231)(564). In case we fix $\pi'(1) = 6$, we will need to map $\pi'(3) = 5$ and $\pi'(2) = 4$ in order to fix this diagonal.

As soon as this is done, if we obtained $\pi'(P)_{\langle 1,1\rangle} \lhd P_{\langle 1,1\rangle}$, we have found a witness of non-minimality. Otherwise, if $\pi'(P)_{\langle 1,1\rangle} \trianglelefteq P_{\langle 1,1\rangle}$ and either $\pi'(P)_{\langle 1,1\rangle}$ or $P_{\langle 1,1\rangle}$ is not complete (i.e., still allows multiple values), we can choose to either learn clauses that eliminate all but one remaining values in the cell, or continue the search for a witness of non-minimality. When continuing the search, we repeat the same process for the next cell $\langle 1, 2\rangle$, etcetera. As soon as π is fully defined, it can be verified without further choices against the remaining cells of P. Whenever this process gets stuck, we backtrack and make a different assignment to $\pi(x)$ for the last assigned x.

For the checks of partial cycle sets, the search can be aborted early, as a result, the check is postponed and repeated when more information is available. Specifically, users can limit the number of nodes visited in the search tree (in a depth-first fashion). Next, users can also limit the number of partial cycle sets that are checked by setting a frequency at which the minimality check should be executed. For complete cycle sets, a full search is necessary to guarantee complete symmetry breaking.

Representing Partial Permutations We give a short description of the internal representation of partial permutations. The internal representation used in this tool was inspired by the one used in SMS. We represent partial permutations as an ordered partition of X, i.e., $\pi = [X_1, X_2, \ldots, X_m]$, where $X_1 \cup \ldots \cup X_m = X$ and $X_1, X_2, \ldots, X_m$ are pairwise disjoint. The ordered partition represents the partial permutation π where $\pi^{-1}(x_1) < \pi^{-1}(x_2)$ for all $x_1 \in X_i, x_2 \in X_j$ with $i < j$. In other words, given $x_1 < x_2$, π maps x_2 to the same set as x_1 or to a set that appears after that set in the ordered partition. Given the ordered partition $\pi = [\{6, 5, 4\}, \{3\}, \{2, 1\}]$, we have that $\pi(1) = \pi(2) = \pi(3) = \{4, 5, 6\}$, $\pi(4) = 3$ and $\pi(5) = \pi(6) = \{1, 2\}$.

Note that extracting a fully defined permutation π' from an ordered partition π can be done quite easily. We represent the ordered partition π using two vectors; a vector of integers V, representing the content of the partitions and a vector which signals the start of a partition, P. In this case, the vector V already represents a valid permutation π' if we define $\pi'(x) = y$ if and only if $V[x] = y$.

This representation becomes even more useful when considering fixed diagonals. In this case, we only need to consider complete permutations π that leave the diagonal invariant. Hence, we can use the initial ordered partition to represent only those permutations.

4.2 Incremental SAT-Based Approach

The minimality check itself can also be viewed as a combinatorial search problem. In this paper, we have chosen to encode the minimality check using a propositional formula, whose satisfiability is determined using a second SAT-solver.

Specifically, we encode what it means for an isomorphism of the problem to be a witness of non-minimality. In order to reason about a specific (partial) cycle set, we can use so-called *assumptions* to fix the truth values for variables representing the cycle set. As such, we can use the same SAT-solver to reason about many same-sized cycle sets, while keeping track of previously learned clauses. In the future, this incremental minimality check can be extended to also detect whether the current partial cycle set could be refined. This technique is similar to the co-certificate learning added to SAT Modulo Symmetries [19], however co-certificate learning is concerned with properties that are orthogonal to the minimality check. To the best of our knowledge, the minimality check itself has never been performed using a SAT oracle.

Minimality Check Encoding Before describing how this incremental minimality check works, we first give a high-level overview of the used encoding, a more detailed description can be found in Appendix 3. The problem that we encode in propositional logic verifies whether there exists a witness of non-minimality (i.e., a permutation π for which Definition 4 holds) for a given cycle set $(X, \cdot)$ represented by matrix M. We start by describing an encoding for the minimality check of complete cycle sets before adapting it in order to also deal with partial cycle sets.

First, we introduce Boolean variables that represent the matrix M and variables that represent its image $\pi(M)$, also referred to as M' here. These matrices are encoded using the same one-hot encoding described in Section 3.1. The first cycle set, represented by matrix M, is encoded using the variables $w_{i,j,k}$ (for all $i, j, k \in X$), where $w_{i,j,k}$ is true if $M_{i,j} = k$. Similarly, we encode its image represented by the matrix M', using the variables $w'_{i,j,k}$ for all $i, j, k \in X$. In order to represent the permutation π, we introduce the variables $p_{i,j}$ for all $i, j \in X$ where $p_{i,j}$ is true if $\pi(i) = j$.

Next, we add clauses to ensure that the permutation is indeed a well-defined isomorphism of the problem and that M' equals $\pi(M)$. Furthermore, we ensure that π is a witness of non-minimality (i.e., that $M' = \pi(M) < M$). To do this, we add static symmetry breaking constraints inspired by the compact encoding for lex-leader constraints [11]. First, we introduce an order $\leq_{w'}$ over the variables used to encode the matrix M'. This order should ensure that an assignment is lexicographically minimal if $M' = \pi(M) \leq M$, assuming that $\mathbf{f} < \mathbf{t}$. We encode the static symmetry breaking constraints using the auxiliary variables $n_{c,i}$ which are true if $w'_{c',i'} \Leftrightarrow w_{c',i'}$ for all $w'_{c',i'} \leq_{w'} w'_{c,i}$.

Because we are only considering complete cycle sets we can add redundant `ExactlyOne` constraints over the matrix variables (similar to those added in the encoding of the original problem). These constraints ensure that each matrix cell is assigned exactly one value, and that rows in the matrices contain unique values. For the original matrix M, this will not have any impact since all variables are fixed using assumptions, however, in context of the image of M, i.e., M', these constraints will enhance propagation.

In order to reason about partially defined cycle sets, some minor adaptations to this encoding are needed. First, the variables representing the matrices get a slightly different meaning; when $w^{(\prime)}_{c,k}$ is true it no longer holds that $M^{(\prime)}_c = k$, but instead we have that $k \in M^{(\prime)}_c$. As a result, it is no longer correct to include the redundant `ExactlyOne` constraints over these variables. Next, to decide whether π is a witness of non-minimality, we now use the order defined in Definition 4, and ensure that $M \lhd M'$. To do this, we add variables $l_{c,k}$ and $g_{c,k}$ for all cells c and $k \in X$, which are true respectively if $M'_c < k$ and $M_c > k$. These variables allow us to reason about the maximum and minimum possible values of a matrix cell. Next, we introduce the variables $n'_{c,k}$ for all cells c and $k \in X$, which similar to the $n_{c,k}$-variables indicate that π can still be a witness of non-minimality at this point. Similar to the encoding described in Section 3.1, both the partial and complete encoding can be optimized when a diagonal is fixed.

Note that the partial encoding can also be used to perform the minimality check for complete cycle sets. However, preliminary experiments showed that using separate complete and partial minimality checks results in a faster enumeration time, hence this is the setup used in our experiments below.

The Incremental Minimality Check During the search, the enumerating SAT solver will make calls to the minimality check to verify whether the current (partial) assignment is lexicographically minimal. The incremental minimality check consists of one (or two, if a separate encoding is used for complete cycle sets) incremental SAT solvers that repeatedly try to find satisfying assignments for the given formula. To ensure that the solvers reason about the current cycle set, we use assumptions to ensure that the w-variables defined in the previous section get the correct truth values. If the SAT solver can find a satisfying assignment, there exists a witness of non-minimality for the assumed cycle set. The minimality check will extract this permutation from the satisfying assignment to create a breaking clause that excludes the current assignment (and all of its extensions). This clause can then in turn be learned by the enumerating SAT solver. If, on the other hand, the formula is unsatisfiable given the current assumptions, the current solution is minimal. Note that if the current assignment is complete, this implies that we have found a lexicographically minimal solution which can be added to the list of non-isomorphic solutions.

Similar to the backtracking approach, one can opt to not run partial minimality checks until completion. In this context, one could decide to limit the number of conflicts or decisions or to only add a selection of the symmetry breaking constraints. Once again, the frequency of the minimality check for partial cycle sets can also be adapted. For complete cycle sets, no such limits can be imposed, in order to guarantee a fully non-isomorphic enumeration.

5 Experimental Evaluation

Our enumeration tool, `YBE-SMS`, is implemented on top of the state-of-the-art SAT-solver `CaDiCal` [5] (version 1.9.4), extended with a so-called *user-propagator*

implemented using the `IPASIR-UP`-API [15][6]. This API allows users to closely interact with the solver, for example by tracking variable assignments, choosing variables to branch on and most importantly by adding new clauses to the solver *during* the search process, which allows users to implement custom propagators. In our case, the custom propagator will ensure that current assignments remain lexicographically minimal by performing the minimality check. If the minimality check fails, a clause is constructed in order to make the solver backtrack or propagate. The minimality check can either be performed using the backtracking approach or by using a second (incremental) instance of the `CaDiCal` solver. Each time when the minimality is performed, the current partial assignment is passed to that second instance by means of assumptions. All experiments were performed on a machine with an AMD(R) Genoa-X CPU running Rocky Linux 8.9 with Linux kernel 4.18.0.

5.1 Generating Non-Degenerate Involutive Solutions of Size 10

In the first set of experiments, we compare the new `YBE-SMS` implementations against the implementation of Akgün, Mereb and Vendramin [1] (referred to as AMV22). To do this, we have identified configurations that find a suitable balance between the frequency of performing this check and the limitations we impose on it; these details can be found in Appendix 4. In Fig. 1, we see that both new approaches are significantly faster than AMV22 for all sizes of the problem up to 10 (which is the highest cardinality solved by AMV22). For example, enumerating all solutions of size 10 using the AMV approach took more than 8 days. The same enumeration with the backtracking approach took only 12 hours and with the incremental approach all solutions of size 10 were enumerated in less than two hours. For the backtracking approach, this speed-up seems to diminish as the problem grows but for the incremental approach, the opposite appears to be true. This new incremental approach is approximately 106 times faster than the AMV22 approach at enumerating all solutions of size 10, the backtracking approach only reaches a speedup of 16.1 for the enumeration of size 10 compared to AMV22. As a result, with this new incremental approach, we were able to enumerate all solutions of size 11 which is a novel result, of value to people studying the YBE. A more detailed view of the results in Fig. 1 is given in Table 3 which can be found in Appendix 5.

5.2 Comparing the SMS Approaches

Previously, no database containing the solutions for the YBE over a set X with cardinality $|X| = 11$ was known. Using the backtracking approach discussed here, we were able to enumerate all solutions for all diagonals of this problem except the diagonal that equals the identity. However, with the incremental approach, we were able to improve on these results, and as such we have constructed a complete database of solutions over a set X with cardinality $|X| = 11$.

[6] The reproduction artifact is available on `https://zenodo.org/records/14604450`.

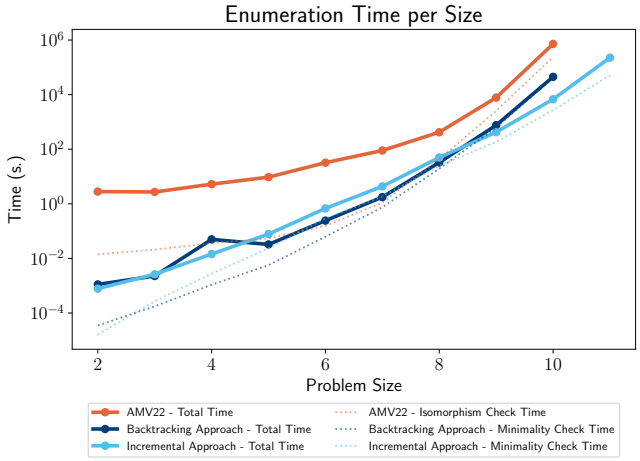

Fig. 1. Comparing the runtimes of the implementation of AMV22 and our approaches building on SAT Modulo Symmetries.

In order to understand why the backtracking approach was not able to enumerate all solutions, we study the time usage of the minimality check in more detail. We break down the time usage of the minimality check into four categories:

- checking partial cycle sets where a clause was added (i.e., a witness or refinement was found),
- checking partial cycle sets where no clause was added,
- checking fully defined cycle sets where a clause was added (i.e., a witness was found) and,
- checking fully defined cycle sets where no clause was added (i.e., the cycle set is lexicographically minimal).

In Table 1, we observe that as size increases, the backtracking approach spends a larger amount of time verifying whether lexicographically minimal cycle sets are indeed minimal. To understand why this is the case, note that in these cases all possible symmetries of the problem need to be considered. In the very worst case, when enumerating cycle sets of size n with the diagonal fixed to the identity, where all rows equal the identity as well; all $n!$ permutations need to be considered. This explains why the speed-up diminishes as the problem grows and why we were unable to expand the results beyond size 10. In Table 2, we observe that the incremental approach does not have these issues and as a result, it was able to enumerate all solutions for size 11. Note that with this approach, the identity diagonal is no longer the bottleneck. Instead, for size 11, we see that enumerating the diagonal (12)(34) took longer than enumerating the identity diagonal. There are several reasons why this might be the case, one possible reason is that there are so many solutions to exclude (i.e., $\pm 11\,800\,000$

			Backtracking Approach							
	Diag.	# Sols.	Time (% of total time)	Number of partial checks	Partial check, Non-minimal (% of Time)	Partial check, No conclusion (% of Time)	Number of complete checks	Complete check, Non-minimal (% of Time)	Complete check, Minimal (% of Time)	Mininimality Check (% of Time)
8	**id**	2 041	**15.45s. (47.33)**	6147	0.78	7.20	2815	5.66	76.25	**89.90**
	(12)	4 988	2.87s. (8.78)	2953	1.65	9.26	5322	1.51	45.39	57.81
	(12)(34)	7 030	2.48s. (7.59)	3538	1.76	9.65	7375	0.93	27.82	40.17
9	**id**	15 534	**514.63s. (67.67)**	43957	0.09	1.87	18413	2.78	92.41	**97.15**
	(12)	41 732	68.82s. (9.05)	19328	0.35	4.54	43008	1.13	75.90	81.92
	(12)(34)	61 438	37.69s. (4.96)	23178	0.58	9.62	62538	0.46	46.78	57.44
10	**id**	150 957	**35 396.79s. (79.02)**	371265	0.01	0.28	163126	1.13	98.03	**99.45**
	(12)	474 153	3 998.35s. (8.93)	184837	0.04	1.18	480790	0.58	92.08	93.88
	(12)(34)	807 084	1 380.82s. (3.08)	251264	0.12	4.57	814658	0.31	63.98	68.97
11	**id**	1 876 002								
	(12)	6 563 873	364 026.91s.	2175972	0.00	0.21	6593100	0.19	96.61	97.02
	(12)(34)	11 807 217	102 863.75s.	3131880	0.01	1.27	11839127	0.11	67.30	68.72

Table 1. A breakdown of the time-usage of the backtracking approach for the diagonals id, (12) and (12)(34) of sizes 8,9,10 and 11. Note that we were unable to enumerate the identity diagonal for size 11 using this approach.

			Incremental Approach							
	Diag.	# Sols.	Time (% of total time)	Number of partial checks	Partial check, Non-minimal (% of Time)	Partial check, No conclusion (% of Time)	Number of complete checks	Complete check, Non-minimal (% of Time)	Complete check, Minimal (% of Time)	Mininimality Check (% of Time)
8	**id**	2 041	**14.53s. (29.27)**	1097	0.58	2.67	4582	15.05	40.44	**58.73**
	(12)	4 988	4.08s. (8.21)	221	0.27	2.03	6518	11.70	43.23	57.22
	(12)(34)	7 030	4.52s. (9.10)	250	0.35	1.54	8596	9.17	43.04	54.09
9	**id**	15 534	**135.86s. (32.25)**	7898	0.53	4.01	25164	10.95	42.71	**58.19**
	(12)	41 732	37.68s. (8.95)	1639	0.17	2.17	48124	8.21	41.75	52.31
	(12)(34)	61 438	42.00s. (9.97)	1612	0.20	1.84	67463	5.79	40.22	48.05
10	**id**	150 957	**1 073.65s. (15.80)**	47416	0.34	5.15	189756	7.92	50.25	**63.66**
	(12)	474 153	605.32s. (8.91)	15056	0.10	2.78	504172	4.03	40.61	47.52
	(12)(34)	807 084	817.65s. (12.03)	15174	0.06	1.86	837097	2.39	36.38	40.70
11	**id**	1 876 002	16 153.55s. (7.14)	435489	0.23	5.10	2041121	4.04	43.28	**52.65**
	(12)	6 563 873	17 578.69s. (7.76)	137194	0.03	1.31	6692410	1.03	26.40	28.77
	(12)(34)	11 807 217	**39 061.15s. (17.25)**	170158	0.01	0.71	11917937	0.33	17.71	18.76

Table 2. A breakdown of the time-usage of the incremental approach for the diagonals id, (12) and (12)(34) of sizes 8,9,10 and 11.

compared to $\pm 1\,000\,000$ original clauses) that the SAT-solver is severely slowed down by the solution-excluding constraints. We hope that further experiments will give us more insights into why this is the case.

6 Conclusions and Future Work

In this paper, we showed how to apply the SAT Modulo Symmetries framework to the generation of non-degenerate, involutive, set-theoretic solutions to the Yang-Baxter equation. Our methods outperform the state-of-the-art on large instances by two orders of magnitude, and we have extended known results to include non-degenerate cycle sets of size 11.

Databases of solutions up to size 10 have inspired a vast body of research in math research. For instance, a recent survey [31] contains several mathematical conjectures that were inspired by mining the database of solutions up to size 10 for interesting properties (see for instance Problems 57 and 61). The list of solutions for size 11 will be useful for similar purposes and provides a substantial number of decomposable or multipermutation solutions (both of which are very important in the combinatorial theory of the Yang-Baxter equation). This increased data gives us the opportunity to better understand how solutions can be constructed from smaller components.

One important question that might remain is why one should trust our implementation, except for the fact that up to size 10 our results coincide with what is known so far. In combinatorial optimization, *proof logging*, which is the idea that solvers should not just output a solution (or in this case, a set of solutions), but also a *machine-checkable proof* that their answer is indeed correct is gaining popularity. The SAT solver `CaDiCaL` that underlies SMS supports proof logging. However, this is a very weak guarantee as it only allows checking that the SAT solver did not make any mistakes, and does not provide any guarantees whatsoever on the encoding or on the correctness of the custom propagator. The most promising approach to achieve proof logging for SMS appears to be using the VeriPB proof system, which was recently used to certify static symmetry breaking [6]. However, there are many challenges on the road ahead to achieve true trustworthy isomorphism-free generation. First of all, the isomorphisms are symmetries of the original problem, but not necessarily of the encoding used. Secondly, while VeriPB could be used to certify that symmetry breaking does not remove all solutions, it cannot be used (unless the set of symmetries it is allowed to use as a witness is somehow forced to be precisely the set of symmetries of the original problem) to certify that *at least one* representative of each isomorphic class is preserved. Finally, it cannot certify that the performed symmetry breaking was *complete*, i.e., that no duplicate solutions were enumerated. Providing true proof logging for isomorphism-free generation appears to be a major challenge.

In the future, the methods we used can be extended to the construction of other combinatorial structures similar to those considered here. This includes racks and quandles, which are used in topology to construct invariants of knots; arbitrary solutions (e.g., non-involutive or with relaxed degeneracy conditions); and objects that appear in algebraic logic, especially L-algebras.

Acknowledgments. This work was partially supported by Fonds Wetenschappelijk Onderzoek – Vlaanderen (projects G070521N and G004124N) and by the project OZR3762 of Vrije Universiteit Brussel. The work was also partially funded by the European Union (ERC, CertiFOX, 101122653). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Akgün, Ö., Mereb, M., Vendramin, L.: Enumeration of set-theoretic solutions to the yang-baxter equation. Math. Comput. **91**(335), 1469–1481 (2022). https://doi.org/10.1090/MCOM/3696
2. Aloul, F.A., Sakallah, K.A., Markov, I.L.: Efficient symmetry breaking for Boolean satisfiability. IEEE Trans. Computers **55**(5), 549–558 (2006). https://doi.org/10.1109/TC.2006.75

3. Anders, M., Brenner, S., Rattan, G.: satsuma: Structure-based symmetry breaking in SAT. In: Chakraborty, S., Jiang, J.R. (eds.) 27th International Conference on Theory and Applications of Satisfiability Testing, SAT 2024, August 21-24, 2024, Pune, India. LIPIcs, vol. 305, pp. 4:1–4:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2024). https://doi.org/10.4230/LIPICS.SAT.2024.4
4. Benhamou, B., Nabhani, T., Ostrowski, R., Saïdi, M.R.: Enhancing clause learning by symmetry in SAT solvers. In: 22nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2010, Arras, France, 27-29 October 2010 - Volume 1. pp. 329–335. IEEE Computer Society (2010). https://doi.org/10.1109/ICTAI.2010.55
5. Biere, A., Fazekas, K., Fleury, M., Heisinger, M.: CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In: Balyo, T., Froleyks, N., Heule, M., Iser, M., Järvisalo, M., Suda, M. (eds.) Proceedings of SAT Competition 2020 – Solver and Benchmark Descriptions. Department of Computer Science Report Series B, vol. B-2020-1, pp. 51–53. University of Helsinki (2020)
6. Bogaerts, B., Gocht, S., McCreesh, C., Nordström, J.: Certified dominance and symmetry breaking for combinatorial optimisation. J. Artif. Intell. Res. **77**, 1539–1589 (2023). https://doi.org/10.1613/jair.1.14296
7. Cedó, F., Jespers, E., Okniński, J.: Braces and the Yang-Baxter equation. Communications in Mathematical Physics **327**(1), 101–116 (2014). https://doi.org/10.1007/s00220-014-1935-y
8. Childs, L.N., Greither, C., Keating, K.P., Koch, A., Kohl, T., Truman, P.J., Underwood, R.G.: Hopf algebras and Galois module theory, Mathematical Surveys and Monographs, vol. 260. Providence, RI: American Mathematical Society (AMS) (2021). https://doi.org/10.1090/surv/260
9. Colazzo, I., Jespers, E., Kubat, Ł., Van Antwerpen, A.: Structure algebras of finite set-theoretic solutions of the Yang–Baxter equation (2023), `https://arxiv.org/abs/2305.06023`
10. Devriendt, J., Bogaerts, B., Bruynooghe, M.: Symmetric explanation learning: Effective dynamic symmetry handling for SAT. In: Gaspers, S., Walsh, T. (eds.) Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10491, pp. 83–100. Springer (2017). https://doi.org/10.1007/978-3-319-66263-3_6
11. Devriendt, J., Bogaerts, B., Bruynooghe, M., Denecker, M.: Improved static symmetry breaking for SAT. In: Creignou, N., Le Berre, D. (eds.) Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9710, pp. 104–122. Springer (2016). https://doi.org/10.1007/978-3-319-40970-2_8
12. Drinfel'd, V.G.: Quantum groups. In: Proceedings of to the International Mathematical Congress in Berkeley (1986). vol. 1, pp. 798–820. Providence, RI: American Mathematical Society (AMS) (1987)
13. Drinfeld, V.G.: On some unsolved problems in quantum group theory. In: Kulish, P.P. (ed.) Quantum Groups. pp. 1–8. Lecture Notes in Mathematics, Springer Berlin Heidelberg, Berlin, Heidelberg (1992)
14. Etingof, P., Schedler, T., Soloviev, A.: Set-theoretical solutions to the quantum Yang-Baxter equation. Duke Mathematical Journal **100**(2), 169–209 (1999). https://doi.org/10.1215/S0012-7094-99-10007-X

15. Fazekas, K., Niemetz, A., Preiner, M., Kirchweger, M., Szeider, S., Biere, A.: IPASIR-UP: user propagators for CDCL. In: Mahajan, M., Slivovsky, F. (eds.) 26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023, July 4-8, 2023, Alghero, Italy. LIPIcs, vol. 271, pp. 8:1–8:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023). https://doi.org/10.4230/LIPICS.SAT.2023.8
16. Gateva-Ivanova, T., Van den Bergh, M.: Semigroups of I-type. Journal of Algebra **206**(1), 97–112 (1998). https://doi.org/10.1006/jabr.1997.7399
17. Gent, I.P., Petrie, K.E., Puget, J.: Symmetry in constraint programming. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming, Foundations of Artificial Intelligence, vol. 2, pp. 329–376. Elsevier (2006). https://doi.org/10.1016/S1574-6526(06)80014-3
18. Guarnieri, L., Vendramin, L.: Skew braces and the Yang-Baxter equation. Mathematics of Computation **86**(307), 2519–2534 (2017). https://doi.org/10.1090/mcom/3161
19. Kirchweger, M., Peitl, T., Szeider, S.: 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. pp. 1944–1953. ijcai.org (2023). https://doi.org/10.24963/IJCAI.2023/216
20. Kirchweger, M., Scheucher, M., Szeider, S.: A SAT attack on rota's basis conjecture. In: Meel, K.S., Strichman, O. (eds.) 25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel. LIPIcs, vol. 236, pp. 4:1–4:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPICS.SAT.2022.4
21. Kirchweger, M., Szeider, S.: SAT modulo symmetries for graph generation. In: Michel, L.D. (ed.) 27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021. LIPIcs, vol. 210, pp. 34:1–34:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPICS.CP.2021.34
22. Klieber, W., Kwon, G.: Efficient CNF encoding for selecting 1 from N objects. In: Fourth Workshop on Constraints in Formal Verification (CFV'07) (2007), `https://api.semanticscholar.org/CorpusID:165159977`
23. Lu, J.H., Yan, M., Zhu, Y.C.: On the set-theoretical Yang-Baxter equation. Duke Mathematical Journal **104**(1), 1–18 (2000). https://doi.org/10.1215/S0012-7094-00-10411-5
24. Nelson, S.: The combinatorial revolution in knot theory. Notices of the American Mathematical Society **58**(11), 1553–1561 (2011)
25. Rump, W.: A decomposition theorem for square-free unitary solutions of the quantum Yang-Baxter equation. Advances in Mathematics **193**(1), 40–55 (2005). https://doi.org/10.1016/j.aim.2004.03.019
26. Rump, W.: Braces, radical rings, and the quatum Yang-Baxter equation. Journal of Algebra **307**(1), 153–170 (2007). https://doi.org/10.1016/j.jalgebra.2006.03.040
27. Rump, W.: L-algebras, self-similarity, and l-groups. Journal of Algebra **320**(6), 2328–2348 (2008). https://doi.org/10.1016/j.jalgebra.2008.05.033
28. Sabharwal, A.: Symchaff: exploiting symmetry in a structure-aware satisfiability solver. Constraints An Int. J. **14**(4), 478–505 (2009). https://doi.org/10.1007/s10601-008-9060-1
29. Seress, A.: Permutation group algorithms. Cambridge tracts in mathematics, Cambridge University Press, New York (2003). https://doi.org/10.1017/CBO9780511546549

30. Soloviev, A.: Non-unitary set-theoretical solutions to the quantum Yang-Baxter equation. Mathematical Research Letters **7**(5-6), 577–596 (2000). https://doi.org/10.4310/MRL.2000.v7.n5.a4
31. Vendramin, L.: Skew braces: a brief survey. In: Geometric methods in physics XL, workshop, Białowieża, Poland, June 20–25, 2023, pp. 153–175. Cham: Birkhäuser (2024). https://doi.org/10.1007/978-3-031-62407-0_12

Appendix 1 Encoding the cycle set properties

First, we encode the following properties:

1. for all $x, y, z \in X$ with $y \neq z$, $C_{x,y} \neq C_{x,z}$ (i.e., the map ϕ_x is bijective for all $x \in X$),
2. for all $x, y, z \in X$, $C_{C_{x,y},C_{x,z}} = C_{C_{y,x},C_{y,z}}$ (i.e., the cycloid equation holds) and
3. for all $x, y \in X$ with $x \neq y$, $C_{x,x} \neq C_{y,y}$ (i.e., the cycle set is non-degenerate).

Properties 1 and 3 can straightforwardly be encoded using clauses that express that each number should occur on each row and on the diagonal:

$$\texttt{ExactlyOne}(\{v_{i,j,k} \mid j \in X\}), \qquad \text{(for each } i, k \in X) \tag{3}$$

$$\texttt{ExactlyOne}(\{v_{i,i,k} \mid i \in X\}). \qquad \text{(for each } k \in X) \tag{4}$$

The cycloid equation, property 2, needs some more care. We achieve this using the clauses

$$\overline{v}_{i,j,x} \vee \overline{v}_{i,k,y} \vee \overline{v}_{x,y,b} \vee y_{i,j,k,b}, \qquad \text{for all } i, j, k, x, y, b \in X \text{ where } i < j \tag{5}$$

$$\overline{v}_{j,i,x} \vee \overline{v}_{j,k,y} \vee \overline{v}_{x,y,b} \vee y_{i,j,k,b}, \qquad \text{for all } i, j, k, x, y, b \in X \text{ where } i < j \tag{6}$$

$$\texttt{ExactlyOne}(\{y_{i,j,k,b} \mid b \in X\}). \qquad \text{for all } i, j, k \in X \text{ where } i < j \tag{7}$$

The first clause expresses that the variable $y_{i,j,k,b}$ must be true whenever $C_{C_{i,j},C_{i,k}} = b$, the second does the same for the right-hand side of the cycloid equation. The totalizer used at the end enforces that this variable can only be true for one b, and as such the two sides of the equation must be the same.

Appendix 2 Proofs

Proposition 2 (Proposition 1, restated). *Let P and P' be two partial cycle sets. The following properties hold.*

1. *If $P \lhd P'$, then for all $C \in \mathcal{X}(P)$ and $C' \in \mathcal{X}(P')$, it holds that $C \prec C'$.*
2. *If $P \trianglelefteq P'$, then for all $C \in \mathcal{X}(P)$ and $C' \in \mathcal{X}(P')$, it holds that $C \preceq C'$.*
3. *If $P \trianglelefteq P'$ and $P' \trianglelefteq P$, then $P = P'$ and both cycle sets are complete.*

Proof. Given two partial cycle sets $P, P' \in \mathcal{P}_n$, for which $P \trianglelefteq P'$, we show that Property 2 holds. If $P \trianglelefteq P'$, for all cells $c \in X \times X$, where $X = \{1, \ldots, n\}$, it holds that $P_c \trianglelefteq P'_c$, or equivalently that $\max P_c \leq \min P'_c$. Hence, for all cells c of all extended cycle sets $C \in \mathcal{X}(P)$ and $C' \in \mathcal{X}(P')$, it holds that C_c is smaller than or equal to C'_c, or equivalently that $C \preceq C'$.

If $P \lhd P'$, there exists a cell c for which $P_c \lhd P'_c$ and for all cells $c' < c$, $P_{c'} \trianglelefteq P'_{c'}$. Hence, for $c' < c$, we have that $\max P_{c'} \leq \min P'_{c'}$, and hence that $C_{c'} \leq C'_{c'}$. However, $\max P_c$ is strictly smaller than $\min P_{c'}$, so C_c will be strictly smaller than C'_c for all extended cycle sets. As a result, we have that $C \prec C'$, and we have shown that Property 1 holds.

Last, we show that Property 3 holds as well. If $P \trianglelefteq P'$ and $P' \trianglelefteq P$, then for all cells c, we have that $\max P_c \leq \min P'_c$ and $\max P'_c \leq \min P_c$. Using that minimum values are smaller than or equal to maximum values gives us that

$$\max P_c \leq \min P'_c \leq \max P'_c \leq \min P_c.$$

However, this implies that $\max P_c \leq \min P_c$, but we know that $\min P_c \leq \max P_c$. So necessarily, $\max P_c = \min P_c$, and hence,

$$\max P_c = \min P_c = \max P'_c = \min P'_c.$$

In other words, all cells in P and P' are fixed such that $P = P'$, which gives us Property 3.

Theorem 2 (Theorem 1, restated). *Given a partial cycle set P and permutation π, we have that:*

- *if $\pi(P) \lhd P$, then π is a witness of non-minimality,*
- *If $\pi(P) \trianglelefteq_c P$ then there is an extension P' of P such that*
  1. *P' and $\pi(P')$ are fully defined on all $c' \leq c$, and*
  2. *all $\preceq$-minimal extensions $C \in \mathcal{X}(P)$ are also extensions of P'.*

Proof. Let $P \in \mathcal{P}_n$ be a partial cycle set, and π a permutation such that $\pi(P) \lhd P$. Using Property 1 from the previous proposition, we obtain that $\pi(C) \prec C$ for all extended cycle sets $C \in \mathcal{X}(P)$. As a result we have shown that π is indeed a witness of non-minimality.

We show that the second case holds as well. Let P be a partial cycle set that is not fully defined and π a permutation such that $\pi(P) \trianglelefteq_c P$. For the first cell $c' \leq c$, where either, or both $P_{c'}$ and $\pi(P)_{c'}$ are not fully defined, the only way to avoid that $\pi(P)_{c'} \lhd P_{c'}$ (in which case this would be a witness of non-minimality, following the first point) is assigning $\pi(P)_{c'}$ its maximal value and $P_{c'}$ its minimal value. We can repeat this for all cells up to c.

Appendix 3 Encoding the minimality check

First, we describe the clauses used to ensure that the permutation π is indeed well-defined and an isomorphism of the given cycle set. In order to ensure that π is well-defined, we add the following clauses:

$$\texttt{ExactlyOne}(\{p_{i,j} \mid j \in X\}), \qquad \text{(for each } i \in X\text{)} \tag{8}$$

$$\texttt{ExactlyOne}(\{p_{i,j} \mid i \in X\}). \qquad \text{(for each } j \in X\text{)}. \tag{9}$$

Next, we ensure that π is indeed an isomorphism of the problem. Similar to the propagation step during the backtracking approach, no extra measures need to be taken if the diagonal equals the identity. In all other cases, we need to ensure that π fixes the diagonal T, or in other words that $\pi \in C_{\mathcal{S}_{|X|}}(T)$. The permutation π fixes the diagonal if it maps each cycle of the diagonal onto a

same-length cycle, in the same order. Hence, if $\pi(i) = j$, the encoding ensures that all values k in the same cycle as i are fixed accordingly. Using $n(k)$ to denote the successor of k in its cycle and C_n to denote the set containing all elements in cycles of length n, we add the following clauses to ensure that the entire cycle is fixed if $\pi(i) = j$; for all $n \in X$ and all $i, j \in C_n$:

$$\neg p_{i,j} \vee p_{n(i),n(j)}. \tag{10}$$

To ensure that M' is the image of M under the given permutation π, we add a set of clauses relating the two matrices and the permutation. We define the following clauses that ensure M' equals $\pi(M)$; for all $i, i', j, j', k, k' \in X$:

$$\neg p_{i,i'} \vee \neg p_{j,j'} \vee \neg p_{k,k'} \vee \neg w_{i',j',k'} \vee w'_{i,j,k}, \tag{11}$$

$$\neg p_{i,i'} \vee \neg p_{j,j'} \vee \neg p_{k,k'} \vee w_{i',j',k'} \vee \neg w'_{i,j,k} \tag{12}$$

This encoding can be simplified using the available domain-specific knowledge. The permutation π is an isomorphism of the problem if it fixes the diagonal, hence, the image of $i \in X$ under π should equal an element $j \in X$ that belongs to a cycle with the same length. As a result, $\pi_{i,j}$ is false if i and j belong to cycles with different lengths, so this variable is no longer introduced and clauses containing it can be simplified (or simply removed).

To ensure that $M' < M$, we use the order $\leq_{w'}$ and auxiliary variables $n_{c,i}$ defined in Section 4.2 to add static symmetry breaking constraints. We use $(c,k)_{++}$ to denote the tuple (c',k'), where $w'_{c,k}$ immediately precedes $w'_{c',k'}$ in the order $\leq_{w'}$, and add the following clauses for all cells c and for each $k \in X$ (except for the last inequality):

$$n_{c,k} \Rightarrow w_{(c,k)_{++}} \vee \neg w'_{(c,k)_{++}} \tag{13}$$

$$n_{c,k} \wedge w'_{(c,k)_{++}} \Rightarrow n_{(c,k)_{++}} \tag{14}$$

$$n_{c,k} \wedge \neg w_{(c,k)_{++}} \Rightarrow n_{(c,k)_{++}}. \tag{15}$$

The first clause forces M' to be smaller than or equal to M, the last two clauses ensure that $n_{c,k}$ is only true if all preceding $w^{(')}$-variables have the same truth value. In order to ensure that M' is strictly smaller than M, we ensure that the last inequality is strict:

$$n_{c_l,2} \Rightarrow w_{c_l,1} \vee \neg w'_{c_l,1} \tag{16}$$

$$\neg n_{c_l,2} \vee \neg w'_{c_l,1} \tag{17}$$

$$\neg n_{c_l,2} \vee w_{c_l,1}, \tag{18}$$

where c_l is the last cell.

In order to reason about partial cycle sets, a different order is used. We no longer verify whether $M' < M$, but whether $M' \lhd M$, or in other words, whether there exists a cell c such that $\max M'_c < \min M_c$ and $M'_{c'} = M_{c'}$ for all previous cells c'. As a result, the symmetry breaking constraints need to be adapted to reason about the maximum and minimum values of the cycle sets.

First, we encode the l- and g-variables by adding the following clauses to the formula for all $i, j, k \in X$:

$$g_{i,j,k} \Leftrightarrow \bigwedge_{l \in X, l \leq k} \neg w_{i,j,k} \tag{19}$$

$$l_{i,j,k} \Leftrightarrow \bigwedge_{l \in X, l \geq k} \neg w'_{i,j,k}. \tag{20}$$

Next, we add the following clauses for all cells c and values $k \in X$:

$$n'_{c,k} \Rightarrow g_{c,k-1} \vee l_{c,k} \tag{21}$$

$$n'_{c,k} \wedge \neg w_{(c,k)_{++}} \Rightarrow n'_{(c,k)_{++}} \tag{22}$$

$$n'_{c,k} \wedge \neg l_{(c,k)_{++}} \Rightarrow n'_{(c,k)_{++}}. \tag{23}$$

In order to ensure that $M' \lhd M$, we again ensure that the last inequality is strict:

$$n'_{c_l,2} \Rightarrow g_{c_l,1} \vee l_{c_l,2} \tag{24}$$

$$\neg n'_{c_l,2} \vee w_{c_l,1} \tag{25}$$

$$\neg n'_{c_l,2} \vee l_{c_l,1}, \tag{26}$$

where c_l is the last cell.

Appendix 4 Configurations

We have used preliminary experiments enumerating sizes 8, 9 and 10 in order to identify good overall parameters for both `YBE-SMS` approaches. For the backtracking approach, this eventually resulted in a situation where almost all the time is spent in unavoidable minimality checks (of complete, minimal solutions), and hence further parameter tweaking cannot result in significant speedups. For the incremental approach, the minimality check is no longer dominating the cost, so further optimization of these parameters (or other ideas for speeding up the search of the main solver) could be beneficial. Both approaches use the following optimizations discussed throughout the paper:

- they partition the problem by fixing the diagonals,
- they optimize the encoding(s) by using the information gained by fixing a diagonal,
- they encode `ExactlyOne` constraints using a binary encoding.

Next, we identified parameters that find a suitable balance between the frequency of and the limitations we impose on the minimality check:

- Backtracking approach:
  - Frequency: $\frac{1}{50}$
  - Limit: 200 visited nodes in the search tree
- Incremental approach:
  - Frequency: $\frac{1}{100}$
  - Limit: 10 conflicts

Appendix 5 Experiments

In Table 3 we give a more detailed breakdown of the results presented in Fig. 1. Here, we compare the new `YBE-SMS` implementations against the implementation of Akgün, Mereb and Vendramin [1] (referred to as AMV22).

		AMV22		**Backtracking Approach**			**Incr. SAT Approach**		
Size	# Sols	Iso Check (s.)	Total (s.)	MinCheck (s.)	Total (s.)	Speedup	MinCheck (s.)	Total (s.)	Speedup
2	2	0.0	2.8	0.0	**0.0**		0.0	**0.0**	
3	5	0.0	2.7	0.0	**0.0**		0.0	**0.0**	
4	23	0.0	5.2	0.0	**0.0**		0.0	**0.0**	
5	88	0.0	9.5	0.0	**0.0**		0.0	**0.0**	
6	595	0.2	32.2	0.1	**0.2**	161.0	0.3	0.7	46.0
7	3 456	1.1	89.8	0.7	**1.8**	49.9	1.9	4.4	20.4
8	34 530	43.1	419.3	19.3	**32.6**	12.9	24.6	49.7	8.4
9	321 931	2 542.3	7 797.7	621.6	760.5	10.2	185.6	**421.2**	18.51
10	4 895 272	237 307.1	720 883.0	41 594.1	44 792.5	16.1	2 706.3	**6796.8**	108.1
11	77 182 093						50 767.2	**226 395.6**	

Table 3. Comparing the runtimes of the implementation of AMV22 and our approaches building on SAT Modulo Symmetries.