Symmetries for Cube-and-conquer in Finite Model Finding

João Araújo
Universidade Nova de Lisboa, Lisbon, Portugal

Choiwah Chow
Universidade Aberta, Lisbon, Portugal

Mikoláš Janota
Czech Technical University in Prague, Czechia

Abstract

The cube-and-conquer paradigm enables massive parallelization of SAT solvers, which has proven to be crucial in solving highly combinatorial problems. In this paper, we apply the paradigm in the context of finite model finding, where we show that isomorphic cubes can be discarded since they lead to isomorphic models. However, we are faced with the complication that a well-known technique, the Least Number Heuristic (LNH), already exists in finite model finders to effectively prune (some) isomorphic models from the search. Therefore, it needs to be shown that isomorphic cubes still can be discarded when the LNH is used. The presented ideas are incorporated into the finite model finder Mace4, where we demonstrate significant improvements in model enumeration.

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

Keywords and phrases finite model enumeration, cube-and-conquer, symmetry-breaking, parallel algorithm, least number heuristic

Digital Object Identifier 10.4230/LIPIcs.CP.2023.1

Funding *João Araújo*: Fundação para a Ciência e a Tecnologia, through the projects UIDB/00297-/2020 (CMA), PTDC/MAT-PUR/31174/2017, UIDB/04621/2020 and UIDP/04621/2020.
Mikoláš Janota: The results were supported by the Ministry of Education, Youth and Sports within the dedicated program ERC CZ under the project *POSTMAN* no. LL1902. This article is part of the *RICAIP* project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857306.

1 Introduction

An important tool that working algebraists need in their research is libraries of the algebras they are interested in. These libraries allow them to get intuitions, test or refute hypotheses and conjectures, and gain insights into the properties of the algebras (see examples on p. 2891 of [30]). Many libraries of algebraic models of small orders, such as the smallsemi package [14] for semigroups and the loops package [36] for quasigroups, are available in the GAP [16] system. A lot more such libraries are needed, but they often take an inordinate amount of time and computing resources to generate.

First-order logic (FOL) has been the most popular language to define algebras. There are two major resource-intensive steps in generating non-isomorphic models from FOL [27]. The first step is to generate models according to the laws specified by the FOL formula. This step often generates a huge number of isomorphic models. For example, given the first-order formula for semigroups, which is $(x * y) * z = x * (y * z)$, Mace4 [35] generates 1,021,120,198 models of order 7, out of which only 1,627,672 ($\approx 0.16\%$) [44] are pairwise non-isomorphic. The second step is to eliminate the isomorphic models generated in the first step. In this paper, we propose a novel efficient and scalable parallel algorithm that not only speeds up the first step but also generates fewer isomorphic models. Suppressing the generation of isomorphic models in the first step reduces the workloads of both the first and the second steps. Not only does it make the whole process much faster, but the required computing resources (disk space, etc.) are also reduced.

While modern-day general-purpose computers are predominantly multi-core, harnessing parallelism for combinatorial search is surprisingly difficult. Consequently, there are few parallel algorithms in constraints programming in general, and in finite model enumeration in particular. Indeed, in satisfiability modulo theories (SMT), even negative results are concluded for cube-and-conquer [23]. A recent literature review concludes that "there is little overall guidance that can be given on how best to exploit multi-core computers to speed up constraint solving" [18]. We aim to help close this gap by devising new parallel algorithms for finite model enumeration.

This paper advances finite model enumerators toward the following two objectives:

1. Mathematicians can use the tool to quickly generate all models (up to isomorphism) of the classes of algebras of their interests on their multi-core computers.
2. The tool can also take advantage of massively parallel computing architectures to pre-generate models (up to isomorphism) of the classes of algebras of general interest.

We find inspiration in the well-established cube-and-conquer approach introduced for SAT [20]. In SAT this means splitting the search space by mutually exclusive conjunctions of propositional literals (cubes). In the context of finite model finding, the structure is richer—a decision of the solver corresponds to inserting a point into the graph of one of the considered functions, e.g., $f(0, 1) = 3$. We comment on cube-and-conquer in more detail in Section 6.

We show that a cube can be excluded from the search if it is isomorphic to an existing one. Effectively, this is breaking symmetries in the search space. However, the task is nontrivial because finite model finders already contain a technique, called the least number heuristic (LNH), to exclude some isomorphic models. The LNH[1] enables the solver to consider only certain values from the co-domain for a given decision point. Therefore, we show that isomorphic cubes can be pruned in the presence of the LNH. Like so, we can take advantage of the two powerful and complementary techniques and ultimately suppress the generation of a large number of isomorphic models.

This paper's contributions are the following:

1. Devise a low runtime overhead parallel algorithm based on the cube-and-conquer approach for finite model enumeration. This scalable algorithm divides finite model enumeration into many independent non-overlapping search jobs to make full use of the available resources.
2. We show that isomorphic cubes can be discarded without losing isomorphic models even in the presence of the well established symmetry breaking technique already present in finite model finders—the least number heuristic (LNH).
3. We extend the model finder Mace4 with the proposed techniques and evaluate it on a large number of problems, where significant speed-up is observed.

[1] Despite the technique being called a heuristic, it does not sacrifice the completeness of the solver.

2 Preliminaries

Familiarity with the general notions of abstract algebra such as groups, semigroups, and quasigroups is assumed, and so is general knowledge about functions and isomorphisms. A good reference is Chapters 2 and 5 of [9].

In this paper, the domain of the search space is denoted by the set $D = \{0, \dots, n-1\}$, where $n \geq 2$, That is, we exclude the trivial case of searching on domains of size 1.

Let π denote an arbitrary permutation on D, π_{id} denote the identity permutation, and $\pi_{(a,b)}$ denote the permutation cycle (a, b). For example, $\pi_{(0,1)}$ is the permutation cycle $(0, 1)$.

2.1 Finite Model Enumeration

For a signature Σ and a FOL formula $\mathcal{F}$ on Σ, a traditional finite model finder first expands the FOL formula to its ground representation by its domain elements in D, then searches for models by backtracking to exhaustively explore the search space [49]. The domain elements in D are seen as special constants not appearing in the original $\mathcal{F}$, c.f. [40].

Following the terminology of [49], a *value assignment (VA) clause* is a term $f(a_1, \dots, a_k) = v$, where f is a k-ary function symbol in Σ and $a_j, v \in D$. We refer to the term $f(a_1, \dots, a_k)$ as the *cell term* (or simply *cell*) since conceptually the search fills the operation table of f.

To search for finite models in $\mathcal{F}$, the finite model finder employs a *cell selection* strategy to pick cell terms successively, without duplicates, to assign values from D to form VA clauses. If a newly formed VA clause causes any failure in the axioms in $\mathcal{F}$, then a new value will be tried for that cell term. If no value can be assigned to that cell term without failing the axioms in $\mathcal{F}$, then the model finder backtracks to the previous cell term to try to assign another value to it. When all cell terms in $\mathcal{F}$ are assigned values without violating the axioms in $\mathcal{F}$, a model, as represented by its VA clauses, is found. After a model is found, the process can continue with backtracking to find more models.

A set of models can be partitioned into equivalence classes by isomorphisms. Intuitively, a model can be transformed into any other model in the same equivalence class by renaming its domain elements. Two models are said to be isomorphic to each other if an isomorphism exists from one model to the other.

The search space can be organized as a search tree in which nodes are VA clauses and edges join successive nodes with cell terms in the search order. The root node is an empty VA clause. The cell term in each node is selected by the cell selection strategy. A *search path* in a search tree is a path from the root to a node in the search tree. It can be represented by a sequence of VA clauses $\langle t_0 = v_0; t_1 = v_1; \cdots \rangle$, where t_i is the cell term in the i^{th} position of the sequence and $v_i \in D$, and $t_i \neq t_j$ when $i \neq j$. Furthermore, a search path will be terminated at the first VA clause that results in a violation of any axiom of $\mathcal{F}$.

If the length of a search path is the same as the total number of cell terms in $\mathcal{F}$, then it is a complete search path and its VA clauses represent a model. Otherwise, it is an incomplete search path representing *partial assignments* of cell values in $\mathcal{F}$.

The backtracking algorithm in its simplest form is to try every possible value assignment for every cell. For example, to search an FOL formula $\mathcal{F}$ with just one binary operation, there are n^{n^2} possible combinations (n^2 cells with n possible values each). Even the very small domain size of 4 gives over 4 billion combinations of cell values. However, in practice, the number of viable combinations to check is much smaller than the theoretically maximum number because of the constraints imposed by $\mathcal{F}$. Furthermore, a finite model finder may infer new VA clauses from existing ones by *propagation*.

▶ **Example 1.** Suppose the FOL formula contains only the equation $f(x, y) = f(y, x)$, that is, the operation f is commutative. After the assignment $f(0, 1) = 0$, the finite model finder can infer $f(1, 0) = 0$. This is referred to as positive propagation.

On the other hand, if the FOL formula contains the inequality $f(x, y) \neq f(y, x)$, then after the assignment $f(0, 1) = 0$, the finite model finder can exclude 0 from the list of possible values for the cell $f(1, 0)$. This is referred to as negative propagation. ◀

2.2 Least Number Heuristic

The least number heuristic (LNH) [4, 50, 51] is a very effective symmetry-breaking algorithm widely implemented in model finders/enumerators such as Mace4. The main idea of the LNH is that all domain elements that have not yet appeared in any VA clauses and the current cell term in the search are indistinguishable to each other and therefore only one of them, say, the smallest one, needs to be considered in a cell value assignment.

To ease discussions of the LNH, we introduce the notation $\mathrm{Vals}(P)$ to denote the set of all domain elements appearing in P, where P can be a search path, a VA clause, or a cell term.

▶ **Example 2.** For the cell term $f(1, 1)$: $\mathrm{Vals}(f(1, 1)) = \{1\}$. For the VA clause $f(1, 1) = 0$: $\mathrm{Vals}(f(1, 1) = 0) = \{0, 1\}$. For the partial search path $S = \langle f(0, 0) = 0; f(1, 1) = 0\rangle$: $\mathrm{Vals}(S) = \{0, 1\}$. ◀

The LNH can now be stated precisely: In adding a VA clause, $t = v$, to extend a search path S, the possible choices of v allowed under the LNH are $\mathrm{Vals}(S) \cup \mathrm{Vals}(t) \cup \{s\}$ where s is the smallest domain element in $D \setminus (\mathrm{Vals}(S) \cup \mathrm{Vals}(t))$, and they are D if $\mathrm{Vals}(S) \cup \mathrm{Vals}(t) = D$. Strictly speaking, it is not necessary to set s to be the smallest domain element not seen so far, it could as well be the biggest one, for example. But the rule to set s must be unambiguous - only one value is consistently picked by the rule each time. In this paper, we always set s to be the the smallest domain element not seen so far.

Furthermore, we say a search path is *LNH-compliant* if it respects the LNH restrictions on the choices of values assigned to its VA clauses.

▶ **Example 3.** Suppose the domain size, $|D|$, is 4. Then the complete search path $\langle f(1) = 0; f(0) = 3; f(3) = 1; f(2) = 1\rangle$ is not LNH-compliant.

For the first VA clause in the search path, $S = \emptyset$ and $t = f(1)$. So, $\mathrm{Vals}(S) \cup \mathrm{Vals}(t) = \emptyset \cup \{1\} = \{1\}$, and therefore $D \setminus (\mathrm{Vals}(S) \cup \mathrm{Vals}(t)) = \{0, 2, 3\}$. Thus, $s = \min(\{0, 2, 3\}) = 0$. The LNH limits the choices of the value for $f(1)$ to $\mathrm{Vals}(S) \cup \mathrm{Vals}(t) \cup \{s\} = \{0, 1\}$. So the first VA clause $f(1) = 0$ is LNH-compliant. However, for the second VA clause in the search path, $S = \{f(1) = 0\}$ and $t = f(0)$. So, $\mathrm{Vals}(S) \cup \mathrm{Vals}(t) = \{0, 1\} \cup \{0\} = \{0, 1\}$, and therefore $D \setminus (\mathrm{Vals}(S) \cup \mathrm{Vals}(t)) = \{2, 3\}$. Thus, $s = \min(\{2, 3\}) = 2$. The LNH limits the choices of the value for $f(0)$ to $\mathrm{Vals}(S) \cup \mathrm{Vals}(t) \cup \{s\} = \{0, 1, 2\}$, so $f(0) = 3$ is not allowed under the LNH. Therefore, the whole search path is not LNH-compliant. ◀

The LNH does not impose any restrictions on the order of the cell terms in the search path[2]. It speeds up the search by limiting the choices of the values for the cell terms. Therefore, its effectiveness decreases with the increase in the length of the search path as more domain elements are used when more VA clauses are added to the search path.

[2] In practice, a number called the *maximal designated number* (*mdn*) is often used to partition the domain into 2 subsets so that $\{0, \ldots, \mathrm{mdn}\}$ are domain elements already seen, and $\{mdn + 1, \ldots, n - 1\}$ are domain elements not seen so far [49]. In this case, cell selection strategies that keep the *mdn* small are preferred because the search tree will be kept narrower.

▶ **Example 4.** The *concentric cell selection strategy* is a simple cell selection strategy to minimize the growth of choices of values in the finite model search with the LNH. This strategy picks the cell $f(a_0, \ldots, a_{k-1})$ with the least $r = \max(a_0, \ldots, a_{k-1})$ from all available cells. Any fixed tie-breaker can be used in case of a tie. For example, one of the possible orders of the cells by this cell selection strategy for a binary operation is $f(a_0, a_1) < f(b_0, b_1)$ if $a_0 = a_1 \vee a_0 + a_1 < b_0 + b_1 \vee (a_0 + a_1 = b_0 + b_1 \wedge a_0 < b_0)$. This gives the sequence $f(0,0)$, $f(1,1)$, $f(0,1)$, $f(1,0)$, $f(2,2)$, $f(0,2)$, $f(2,0)$, $f(2,1)$, $f(1,2)$, $f(3,3)$ .... ◀

2.3 Cube

A *cube* is a prefix of a search path, and as such, it can be specified by a sequence of VA clauses. Permutations and isomorphisms can be applied to a cube by applying them to its VA clauses. Specifically, if π is a permutation on D and B is a cube, then $\pi(B) \coloneqq \{f(\pi(a_1), \ldots, \pi(a_k)) = \pi(v) \mid f(a_1, \ldots, a_k) = v$ is a VA clause in $B\}$. Observe that $\pi_{id}(B)$ is the (unordered) set of all individual VA clauses in the cube B.

Note that predicates in an FOL formula can be implemented as functions with two values, T (true) and F (false), which do not affect the LNH because they are not domain elements. For convenience, we consider $I(T) = T$ and $I(F) = F$ for any isomorphism I so that the same terminology is used for both relations and functions.

Cubes are said to be isomorphic if their VA clauses are isomorphic. In particular, two cubes B_0 and B_1 are isomorphic if there is a permutation π on D such that $\pi(B_0) = \pi_{id}(B_1)$.

▶ **Example 5.** If $B_0 = \langle f(0) = 0; g(0,0) = 0; f(1) = 0; g(1,1) = 0\rangle$ and $B_1 = \langle f(0) = 1; g(0,0) = 1; f(1) = 1; g(1,1) = 1\rangle$, then B_0 and B_1 are isomorphic because $\pi_{(0,1)}(B_0) = \{f(1) = 1, g(1,1) = 1, f(0) = 1, g(0,0) = 1\} = \pi_{id}(B_1)$. ◀

3 Isomorphic Cubes Redundancy

The main objective of this section is to show that isomorphic cubes can be removed from the search. More formally, if cubes B_0 and B_1 are isomorphic, then it is sufficient to explore assignments extending B_0 and ignore *all* assignments extending B_1. We need to prove that any model lost by discarding B_1 must necessarily be isomorphic to some model obtained from extending B_0 under the LNH. This statement is intuitive, but the proof requires some care as effectively, we are dealing with a combination of two symmetry-breaking techniques: LNH and isomorphic cube pruning, under an arbitrary search strategy.

As a motivational example, consider the cube $\langle f(0,0) = 0\rangle$, which states that f is idempotent in 0. But because 0 does not appear in the original FOL formula, intuitively, the constant 0 cannot play a special role in the formula. Consequently, this cube searches *all* interpretations of f that have at least one idempotent. For instance, the cube $\langle f(1,1) = 1\rangle$ will search the same interpretations, up to isomorphism. Now, we need to show this property formally and that it holds when the solver searches with the LNH restriction.

The key idea of the proof is that given a model B_1 with VA clauses A, any cube that is isomorphic to a subset of A can be gradually extended to be a model isomorphic to B_1. Each extension step of the cube must uphold the following properties: (1) The cube is isomorphic to some subset of A. (2) The cube is LNH-compliant. The extension step is illustrated in Figure 1. We are given a cube B_0 that is isomorphic to an $A_0 \subseteq A$. When the finite model finder decides on some empty cell t, we need to show that it is possible to find a value according to the LNH such that the extended cube is isomorphic to some subset of A containing A_0.

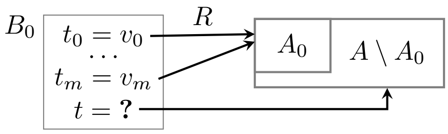


Figure 1 Extension of a cube according to the VA clauses A

▶ **Notation 1.** *For a mapping R from D to D and a value $d \in D$ we write $\mathcal{E}_R^d$ for a mapping that maps d to $R(d)$ if $d \in \mathsf{dom}(R)$ and otherwise maps d to $\min(D \setminus \mathsf{rng}(R))$. We further write $\mathcal{E}_R^{d_1,\dots,d_k}$ for successive extensions by $d_1, \dots, d_k$, i.e. $\mathcal{E}_R^{d_1,d_2} = \mathcal{E}_{\mathcal{E}_R^{d_1}}^{d_2}$ etc.* ◀

▶ **Example 6.** Suppose $D = \{1, 2, 3\}$ and $R : \{1\} \to \{2\}$ s.t. $R(1) = 2$ and $R^{-1}(2) = 1$ (so, R is a bijection). Then $\mathcal{E}_R^2$ s.t. $R(2) = \mathcal{E}_R^2(2) = 1$ is a valid extension of R because $\min(D \setminus \{2\}) = 1$. Furthermore, $\mathcal{E}_R^{2,1}$ s.t. $\mathcal{E}_R^{2,1}(1) = 2$ is a valid (but trivial) extension of $\mathcal{E}_R^2$. ◀

▶ **Lemma 7.** *If R is a bijection between some $D_0, D_1 \subseteq D$ and $d \in D$ then $\mathcal{E}_R^d$ is well-defined and also a bijection.*

Proof. If $d \in D_0$, then $\mathcal{E}_R^d = R$ and there is nothing to proof. If $d \in D \setminus D_0$, then by definition, $\mathcal{E}_R^d = R \cup \{(d, p)\}$ for some $p \in D \setminus D_1$. Since R is a bijection from D_0 to D_1, $d \notin \mathsf{dom}(R)$, and $p \notin \mathsf{rng}(R)$, so $\mathcal{E}_R^d$ is well-defined, one-one, and onto. That is, it is a bijection. ◀

▶ **Notation 2.** *$B \oplus \langle t = u \rangle$ is the new cube formed by extending the cube B with the VA clause $t = u$.* ◀

The following lemma is the core of our proof. We have a cube B isomorphic to some partial assignment A_0 and now we need to prove that for *any* model A completing A_0 and *any* search strategy, we are able to extend B while observing the LNH. Then, the lemma is used to prove that isomorphic cubes can be discarded by induction on cube length (Theorem 9).

▶ **Lemma 8.** *Let B be an LNH-compliant cube and A a model s.t. B is isomorphic to some $A_0 \subseteq A$. Then for any cell term t not appearing in B, there exists a value u and a VA clause $t' = u' \in A \setminus A_0$, s.t. $B \oplus \langle t = u \rangle$ is LNH-compliant and isomorphic to $A_0 \cup \{t' = u'\}$.*

Proof. Let R be an isomorphism mapping B to A_0 and let t be a cell term $f(a_1, \dots, a_k)$. Define R_1 as $\mathcal{E}_R^{a_1,\dots,a_k}$, and let t' denote the cell term $f(R_1(a_1), \dots, R_1(a_k))$, i.e., map the cell that the solver searches on into a cell in the prescribed model A.

Since A is a model, there must exist a value $u' \in D$ with $(t' = u') \in A$, i.e. u' can be found by a lookup of t' in A. Since t is not a cell term in B and R_1 is a bijection, so t' is not a cell term in A_0 and must therefore be in $A \setminus A_0$. Thus, $t' = u'$ is a VA clause in $A \setminus A_0$.

To obtain u (a value for cell t), define R_2 as $\mathcal{E}_{R_1^{-1}}^{u'}$, i.e. map u' back into the search by extending the inverse. Then, set $u = R_2(u')$. By Lemma 7, R_2 is bijection and it is therefore an isomorphism from $A_0 \cup \{t' = u'\}$ to $B \oplus \langle t = u \rangle$. Finally, by definition of R_2, u either already appears in B or otherwise is the smallest domain element not in B. Therefore, the extension of the cube B by the VA clause $t = u$ is LNH-compliant. ◀

▶ **Theorem 9.** *Suppose we are searching under the LNH with any cell selection strategy on a signature* Σ *and a FOL formula,* $\mathcal{F}$*, on* Σ*. If* B_0 *and* B_1*, of length* $l \geq 0$*, are isomorphic cubes, and if* M_1 *is a model obtained by completing (not necessarily under the LNH) the search path in* B_1*, then* B_0 *can be extended by a search path* S *under the said LNH and cell selection strategy to a model* M_0 *which is isomorphic to* M_1*.*

Proof. We will use mathematical induction on the length of the extension, m, on S to prove the theorem. Let A denote the VA clauses of M_1, and A_0 denote the VA clauses of B_1.

Base case is trivial as B_0 and B_1 are given as isomorphic when $m = 0$.

Induction step: Suppose the search path S is extended m times, where $m > 1$, so that S_m is LNH-compliant and isomorphic to a subset $A_m \subseteq A$. Then by Lemma 8, S_m can be extended by one VA clause with the cell term t_{m+1}, chosen by the said cell selection strategy, to S_{m+1} which is LNH-compliant and isomorphic to $A_{m+1} \subseteq A$.

Note that a model finder may do propagations after a cell value assignment. That is, some cell terms can be assigned values inferred from existing VA clauses. Propagations can be viewed as part of the cell selection strategy and be handled the same way as regular cell value assignments.

We can therefore conclude by mathematical induction that S can be extended to a complete search path when all cell terms in $\mathcal{F}$ are filled with values such that S represents the model M_0, is LNH-compliant, and is isomorphic to $A_s \subseteq A$. Since M_0 and M_1 are of the same size, so A_s and A must necessarily be of the same size and hence must be equal. Therefore, M_0 is isomorphic to M_1. ◀

Theorem 9 shows that isomorphic cubes always extend to isomorphic models. So, one of the isomorphic cubes may be discarded without losing any non-isomorphic model.

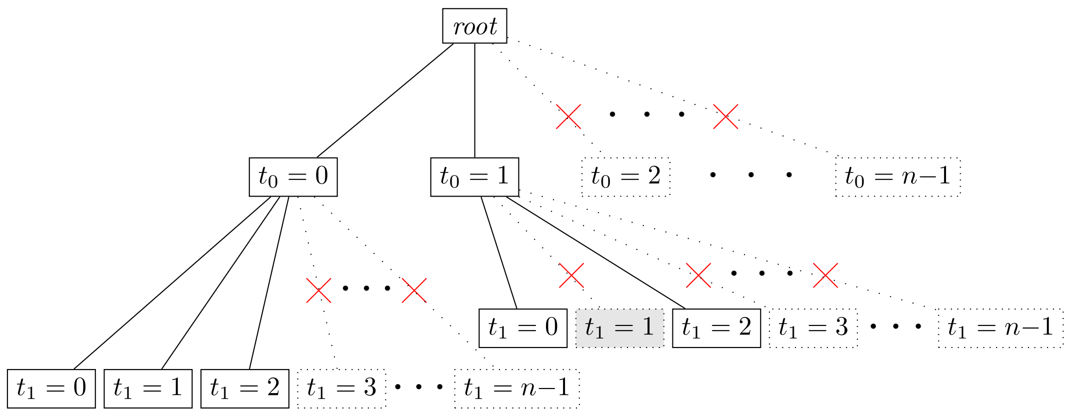
▶ **Corollary 10.** *On searching under the LNH with any cell selection strategy on a signature* Σ *and an FOL formula* $\mathcal{F}$ *on* Σ*, if* M_1 *is a model in* $\mathcal{F}$*, then there is a complete search path* S *under the said LNH that results in a model* M_0 *which is isomorphic to* M_1*.*

Corollary 10 proves the completeness of the LNH in that every model in any search is isomorphic to some model found by searching under the LNH. An alternative proof of the corollary is given in [50].

4 Searching with Cubes

Cubes can be constructed to partition the search space into non-overlapping subtrees that can be processed in parallel. It is not necessary to search all the subtrees that originate from the collection of cubes that span the entire search space because isomorphic cubes in the same collection can be eliminated without losing non-isomorphic models. For example, suppose we want to search for models of order 3 or more on a function $f : D^2 \to D$ under the LNH with a cell selection strategy that selects $f(0,0)$ then $f(1,1)$ as the first 2 cell terms in the search process. There are at most 6 cubes of length 2 (listed below) under the said LNH and cell selection strategy, so together they must span the whole search space in the sense that every search path that starts with the cell terms $f(0,0)$ then $f(1,1)$ in the search tree must include one of the 6 cubes in it.

1. $\langle f(0,0) = 0; f(1,1) = 0\rangle$.
2. $\langle f(0,0) = 0; f(1,1) = 1\rangle$.
3. $\langle f(0,0) = 0; f(1,1) = 2\rangle$.
4. $\langle f(0,0) = 1; f(1,1) = 0\rangle$.



Note: t_0 denotes $f(0,0)$ and t_1 denotes $f(1,1)$. A dotted line with a cross is a branch pruned by the LNH, except for the branch ending on the VA clause $t_1 = 1$ (the shaded node), which is pruned by the isomorphic cubes removal algorithm.

Figure 2 Partial Search Tree Showing Cubes of Length 2

5. $\langle f(0,0) = 1; f(1,1) = 1 \rangle$.

6. $\langle f(0,0) = 1; f(1,1) = 2 \rangle$.

Since $\pi_{(0,1)}(\text{Cube } 1) = \{f(1,1) = 1, f(0,0) = 1\} = \pi_{id}(\text{Cube } 5)$, so Cubes 1 and 5 are isomorphic and one of them can thus be discarded without losing non-isomorphic models per Theorem 9. This example demonstrates the importance of keeping the LNH in the search — it cuts the search space from potentially n^2 cubes down to 6. Theorem 9 allows us to further cut the number of cubes down to 5 (see Figure 2 for illustration). More isomorphic cubes can be removed with longer cubes (see Table 2).

The procedure of removing isomorphic cubes starts with generating a set of short cubes (typically of length 2 for a binary operation) that spans the entire search space. The model finder takes short cubes as inputs and runs with them as if they are generated by itself to generate longer cubes of predefined length l. Specifically, the model finder runs as usual, except that it emits the cubes of length l when the depth of the search tree reaches l. After outputting the cube, the model finder backtracks as if it has reached the bottom of the search tree, and runs on a new branch as usual until all cubes of length l are generated. Some models may be generated in this process due to propagation, and they are kept as part of the final outputs. Next, the cubes are compared for isomorphism and only one of any pair of isomorphic cubes is kept. This new set of non-isomorphic cubes of length l will be used as inputs to the model finder in the next round of generation of longer cubes. The process is repeated until the desired length of cubes is reached.

For searching models defined by one operation of arity k, we use the sequence of lengths l: $k, 2^k, 3^k, 4^k, \ldots$. This is to match the concentric cell selection strategy (see Example 4 for its definition) of the finite model finder such as Mace4. We will discuss the best cube length to use in Section 5.3.

Finally, non-isomorphic cubes of the target length can then be processed independently in parallel and their output models collected separately.

4.1 Invariants

To speed up the isomorphic cubes removal process, the same invariant-based algorithm described in [2] to remove isomorphic models can be applied to cubes. Invariants, such as number of distinct domain elements, are properties that must be identical for cubes to be isomorphic. For example, the cubes $A = \langle f(2,2) = 2; f(2,3) = 4\rangle$ and $B = \langle f(3,3) = 2; f(1,2) = 2\rangle$ cannot be isomorphic because A contains an idempotent 2 but B does not. Powerful and inexpensive invariants for binary operations include:

1. Number of y such that $x = (x * y) * x$.
2. Number of y such that $y = x * z$ for all $z \in D$.
3. Number of y such that $y = z * y$ for all $z \in D$.
4. Number of idempotents x (i.e. $x * x = x$) for all $x \in D$.
5. Number of y such that $y * y = x$ for each $x \in D$.

First, invariant vectors (i.e. ordered lists of invariants) for cubes are calculated and used as hash keys to group cubes having the same invariant vectors into hash buckets. Then, cubes within the same bucket are tested for isomorphism. There is no need to test for isomorphism across buckets because isomorphic cubes must have the same invariant vectors. This saves tremendous amounts of testing time. Furthermore, buckets can be processed independently and in parallel to further speed up the process.

4.2 Work Stealing

In the basic form of this cube-based parallel algorithm, cubes are statically generated before the model enumeration process begins. It has the advantage of low runtime overheads as no synchronization among running finite model finders is needed. The preprocessing time for generating the cubes is also small for short to medium-length cubes. The disadvantage is that the workload may be uneven among the parallel processes. Some jobs may take a long time to finish when free workers sitting idle after finishing their jobs.

This problem can be solved with *work stealing* algorithms (also used in SAT [26]) in which a busy finite model searcher releases cubes that are not currently being worked on. For example, suppose a running model searcher is working on a cube $B_0 = \langle f(0,0) = 0\rangle$, and its cell selection strategy picks the cell $f(1,1)$ to assign value next. Under the LNH, $f(1,1)$ may be assigned a value from $\{0,1,2\}$. If the model searcher is requested to spin out some work for other free workers, then it generates three cubes, $B_0 = \langle f(0,0) = 1; f(1,1) = 0\rangle$, $B_1 = \langle f(0,0) = 1; f(1,1) = 1\rangle$, and $B_2 = \langle f(0,0) = 1; f(1,1) = 2\rangle$. It continues to work on the cube B_0 and releases B_1 and B_2 to other free workers.

5 Experimental Results

We integrate the cube-based algorithms into the finite model enumerator Mace4, which supports searching on FOL with the LNH and many cell selection strategies [35]. Parallelization is controlled outside Mace4. Only minor changes are made to Mace4 to

1. Accept cubes as inputs and continue searching for longer cubes or models from them.
2. Periodically check for signal for work stealing to spin off cubes for other workers.

The model searching logic in Mace4 remains intact. The concentric cell selection strategy (see Example 4 for its definition) is used in the experiments. A separate program removes isomorphic cubes by separating the cubes with equal invariants then check for isomorphisms (two cubes are isomorphic if one can be transformed to the other by a permutation).

We run the experiments on an Intel® Xeon®Silver 4110 CPU 2.0 GHz ×32 computer, with 64 GB of random access memory (RAM), using 30 parallel processes unless otherwise stated. All times reported are wall clock times.

We pick many disparate and challenging problems from the MarcieX database [3], which contains a collection of 158 most popular algebras. We also draw an example of semigroup subvariety from [1]. The definitions of the algebras used in the experiments in this section are listed in Table 1, in which all clauses are implicitly universally quantified.

Table 1 Definitions of Algebras Used in Experiments

Algebra	FOL Definition
Semigroups	$x * (y * z) = (x * y) * z.$
Loops	$x * y = x * z \to y = z.\;\; y * x = z * x \to y = z.\;\; x * 0 = x.\;\; 0 * x = x.$
var$\{N_2^1 \cap [x^2 = y^2]\}$	$x * (y * z) = (x * y) * z.\;\; (x * x) * x = x * x.\;\; x * y = y * x.\;\; x * x = y * y.$
Tarski Algebras	$(x * y) * y = (y * x) * x.\;\; x * (y * z) = y * (x * z).\;\; (x * y) * x = x.$
Quasi-ordered Set	$x < y \land y < z \to x < z.\;\; x < x.$
Involutive Lattices	$(x * y) * z = x * (y * z).\;\; x * y = y * x.\;\; (x + y) + z = x + (y + z).$
	$x + y = y + x.\;\; (x * y) + x = x.\;\; (x + y) * x = x.$
	$-(x + y) = -x * -y.\;\; - - x = x.$

In the tables showing experimental results in this section, the rows with cube length 0 show the results of running Mace4 in a single thread without the cube-based algorithms.

Table 2 shows the results of applying Theorem 9 to remove isomorphic cubes for the binary operation of the semigroups of order 7. Observe that the percentage reduction of the number of cubes increases as the cube length increases. The isomorphic cubes removal algorithm is therefore complementary to the LNH because the LNH removes a lot of short cubes but loses its effectiveness as the length of the cubes grows.

Table 2 #Cubes for Semigroups of Order 7

	# Cubes		
Cube Length	w/o Removal of Isomorphic Cubes	w/ Isomorphic Cubes Removed	Reduction (%)
2	6	5	16.7
4	34	28	17.6
9	1,568	888	43.4
16	56,206	12,036	78.2
25	1,028,171	59,056	94.3

We run Mace4 to enumerate semigroups defined by a single binary operation. The results show a speedup of over 100 times when cubes of length 25 are used, with over 96% of the isomorphic models suppressed (see Table 3). The results on semigroups are indicative of the algorithm's usefulness in general to the computational algebraists because algebraic structures related to semigroups are ubiquitous in algebra. Not only are there many well-known semigroup-related algebras, but also many semigroup varieties and subvarieties that are of high research interests [1].

Table 4 shows the results for loops (a quasigroup-related class of algebra) defined by a single non-associative binary operation. Here the reduction in the number of the output isomorphic models is not as pronounced. This is expected because the LNH works very well with the Latin square and removes a high percentage of the isomorphic models [48] before the isomorphic cubes removal takes place. For example, while only 0.16% of semigroups of order 7 generated by the LNH are non-isomorphic, 8.7% (106,228,849 out of 1,216,226,816) of the models generated for the loops of order 8 under the LNH are non-isomorphic. Nevertheless, the parallel algorithm provides 15 times improvement in speed for cube length of 16.

Table 3 Running Cubes on Semigroups of Order 7

Cube Length	#Cubes	#Models (Millions)	Time in min. Gen. Cubes	Total
0		1,021.1		235.2
2	5	717.7	0.0	12.5
4	28	611.1	0.1	9.4
9	888	360.2	0.1	5.2
16	12,036	158.2	0.2	2.8
25	59,056	39.5	0.9	1.7

Table 4 Running Cubes on Loops of Order 8

Cube Length	#Cubes	#Models (Millions)	Time in min. Gen. Cubes	Total
0		1,216		564.0
2	1	1,216	0.0	47.4
4	2	1,216	0.1	47.3
9	18	1,216	0.1	46.2
16	3,583	1,214	0.1	45.3

Table 5 shows the results of running the algorithms on the semigroup subvariety var$\{N_2^1 \cap [x^2 = y^2]\}$ (see p. 40 of [1] for its definition and discussions). With longer cubes, the algorithms speed up the process by 26 times with 30 threads. The results confirm that the proposed algorithms work remarkably well with semigroup-related algebras.

Table 5 Running Cubes on var$\{N_2^1 \cap [x^2 = y^2]\}$ of Order 9

Cube Length	#Cubes	#Models (Millions)	Time in min. Gen. Cubes	Total
0		313.0		72.0
2	1	156.5	0.0	2.9
4	1	156.5	0.1	2.8
9	2	156.5	0.1	2.8
16	5	120.9	0.1	2.3
25	16	55.5	0.2	1.3
36	70	13.0	0.3	0.8
49	331	1.5	1.0	1.1

Table 6 Running Cubes on Tarski Algebras of Order 13

Cube Length	#Cubes	#Models (Millions)	Time in min. Gen. Cubes	Total
0		379.6		1,949.9
2	3	189.8	0.0	70.2
4	1	189.8	0.1	69.9
9	3	183.3	0.1	67.7
16	11	158.8	0.1	58.1
25	55	111.9	0.2	40.1
36	157	62.1	0.2	21.8
49	174	24.9	0.5	8.8
64	171	6.6	1.0	3.7

The Tarski algebras are unlike both the semigroups and the quasigroups in that its multiplication table is not associative and is not a Latin square [3]. It shows the cube-based algorithms perform better and better as the length of the cube increases (see Table 6).

The quasi-ordered set is defined by one binary relation. The isomorphic cubes algorithms work well on relations just as it works well on functions. As shown in Table 7, when cubes of length 36 are used, over 99% of the isomorphic models are suppressed, and the search process is sped up by over 200 times.

Table 7 Running Cubes on Quasi-ordered Set of Order 8

Cube Length	#Cubes	#Models (Millions)	Time in min. Gen. Cubes	Time in min. Total
0		642.8		59.9
2	1	642.8	0.0	4.2
4	3	474.6	0.1	3.2
9	9	209.5	0.1	1.7
16	33	61.3	0.1	0.8
25	139	12.6	0.2	0.3
36	713	2.0	0.3	0.3

Table 8 Running Cubes on Involutive Lattices of Order 13

Cube Length	#Cubes	#Models (Millions)	Time in min. Gen. Cubes	Time in min. Total
0		423.0		4,719.7
3	2	423.0	0.0	432.5
6	3	423.0	0.1	432.8
10	6	263.9	0.1	270.0
21	23	178.6	0.1	180.9
36	108	84.9	0.2	88.3
55	555	46.0	0.3	46.2
78	1,710	19.8	0.5	20.6
105	5,048	8.7	4.9	14.3

As an example to demonstrate the effectiveness of the algorithms on more complex algebras, consider the Involutive Lattice [3], which is defined by two associative binary operations and one unary operation. For Involutive Lattices of order 13, the search tree has a maximum depth of 351. Using cubes of length of 105, we obtain a speedup of 300 times, with almost 98% of the isomorphic cubes suppressed (see Table 8). The results show that the isomorphic cubes algorithms are highly effective for both simple and complex algebras.

The reductions in time and number of models (on top of the LNH) are summarized in Figures 3 and 4. Note that the reduction in total time is over 90% even for short cubes. However, the biggest gain in both reduction in time and in isomorphic models is when longer cubes are used. Reduction in isomorphic models also helps tremendously in the post processing step to extract non-isomorphic models.

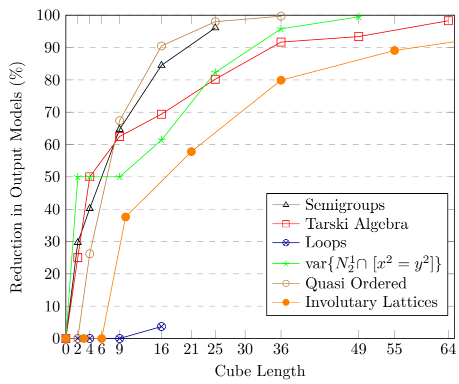


Figure 3 Reduction in Number of Output Models

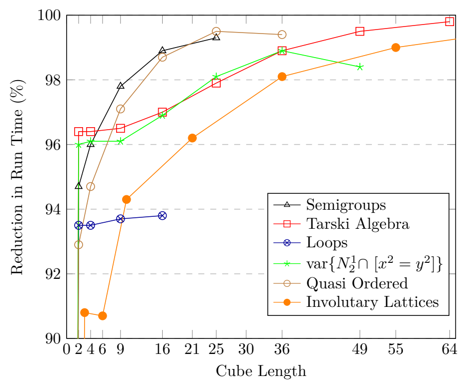


Figure 4 Reduction in Total Time with 30 Parallel Processes

5.1 Speedup of Finite Model Enumeration with Parallelization

As discussed, the cubes algorithms allow low-cost parallelization of the finite model enumeration process. Figure 5 and Table 9 show the performance of the parallel cubes algorithms with

Table 9 Performance w/ Multiprocessing

Algebra	Order	Cube Length	Time in seconds: 1 Process	2 Processes	4 Processes	8 Processes	16 Processes
Semigroups	7	25	6,626	3,397	1,757	940	425
Loops	7	16	202	108	50	35	21
Tarski algebras	13	64	1,766	973	552	273	250
var$\{N_2^1 \cap [x^2 = y^2]\}$	9	49	130	84	80	57	53
Quasi Ordered	8	36	123	77	51	37	25
Involutive Lattices	12	105	1,496	794	480	378	320

1 to 16 parallel processes. Here, the reported times do not include isomorphic mode filtering; they are for Mace4 to generate models only. Note that when many processes compete for limited amount of RAM, swapping could slow down the processes substantially. This helps to explain why larger algebras, such as the Involutive Lattice of order 13, have their curves flattened out much faster than small algebras, such as the Semigroups of order 7. More processes also mean more work-stealing and higher overheads.

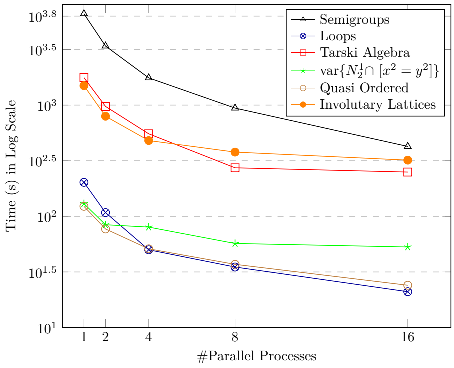


Figure 5 Performance w/ Multiprocessing

5.2 Isomorphic Cubes Removal Speeds up Isomorphic Models Filtering

As pointed out Section 1, reducing the number of Mace4 outputs also reduces the efforts needed to filter out isomorphic models. Table 10 shows, using involutive lattices as an example, the out-sized effect of the reduction of Mace4 outputs on the time to filter out the isomorphic models using the invariant-based isomorphic model filtering algorithm [2], with 30 parallel processes. With the reduction in number of Mace4 models, the isomorphic model filtering process is sped up by 2 orders of magnitude. The improvement in speed is observed to be better with models of higher orders. We would also point out that the isomorphic model filter generates the same non-isomorphic models with or without the cubes algorithms.

5.3 Optimal Cube Length

In general, the search process using longer cubes finishes earlier with fewer isomorphic models. However, we observe that there are three limiting factors on the lengths of the cubes.

Table 10 Running Invariant-based Isomorphic Models Filter on Involutive Lattices

		w/o Cubes		w/ Cubes		
Order	#Non-iso Models	#Mace4 Output	Isomorphic Model Filter Time (s)	Cube Length	#Mace4 Output	Isomorphic Model Filter Time (s)
9	122	72,470	29	78	3,670	1
10	389	575,463	496	105	13,789	4
11	906	4,771,035	28,424	105	97,680	135
12	3,047	43,851,030	N/A	105	971,416	2,802

First, as the length of the cubes gets longer, more and more models are generated as a result of propagations. This reduces the impact of removing isomorphic cubes because they represent a progressively smaller proportion of the isomorphic models. It is observed that when more than $n-2$ symbols out of the n domain elements are used in the cell terms, the number of (isomorphic) models will be substantial and extending the cube length does not bring enough reduction in isomorphic models to justify the increase in processing time.

Second, the isomorphic cubes removal time grows quite fast as the length of the cube grows. When the isomorphic cubes removal process takes more than a few minutes, further lengthening of the cubes will result in prohibitive overheads in the search process.

Lastly, when the final number of cubes is more than tens of thousands, the overheads in processing them becomes so high that the search becomes slower. This factor depends heavily on the number of processors available. More processors mean more parallel processes can be run without slowing down the whole search process.

One heuristic is to run cube generation until the number of cubes reaches some threshold or the runtime exceeds some threshold, then switch to model generation. The thresholds are system-dependent and can further be fine-tuned by experiments with algebras of interest.

6 Related Work

There is extensive research on *paralyzing SAT solving*, where the predominant approaches are search space partitioning and portfolios, c.f. [33]. We find inspiration in the *cube-and-conquer* approach proposed by Heule and colleagues [20–22], where the search space is partitioned by a lookahead solver into (many) cubes and then each subspace is solved by a CDCL SAT solver. In SAT, partitioning by a CDCL solver is nontrivial [32] and that is why the lookahead solver is useful for this task. Nevertheless, the use of the lookahead solver is not seen as an indispensable feature of the cube-and-conquer, as noted by Subercaseaux and Heule [46]. In our approach, we have a tight control over the decisions of the solver and we do not need a separate solver to perform the splitting. Additionally, we invest extra effort into search space splitting by identifying symmetries in the cubes.

The *adaptive prefix-assignment technique* [25] is a symmetry reduction algorithm used in SAT. The prefix is equivalent to a propositional cube, and the algorithm also tries to eliminate isomorphic cubes. In our case, we exploit symmetries specific to FOL—LNH and isomorphism at FOL level, which is absent in their algorithm (and in SAT in general).

Parallel algorithms can be characterized by how the search is done. There are two main search methods: embarrassingly parallel search (EPS) and work stealing search [7, 10, 26, 33, 41, 42]. In the former method, the task is decomposed into many sub-tasks that are queued up to be processed by free worker threads/processes. In the latter method, when a worker completes its task, it asks other workers for more work. The busy workers may split their tasks into smaller sub-tasks and pass some of them to the free workers. The main focus of this method is to keep all the CPUs running until all jobs are done, although for some cases, the work stealing scheme can affect efficiency [10]. The EPS method is a natural choice for the cube-based parallelization scheme because preprocessing can be performed to generate numerous non-isomorphic cubes by splitting the search space. However, a work-stealing procedure is essential in supplementing the EPS to balance uneven workloads [33].

Parallel algorithms can also help select the best strategy in solving a problem with the EPS method [39]. After a problem is decomposed into a large number of sub-tasks, a small number (e.g., 1%) of these sub-tasks are run in parallel using different strategies of the same solver or different solvers. The strategy that gives the best performance on the subset of sub-tasks will be used to run all sub-tasks. The same idea is used in the invariant-based isomorphic models removal algorithm [2]: it randomly generates a large number of invariants, then applies them to a small percentage of models to pick the best performing random invariants to apply to the whole set of models. This idea can be applied to the finite model finders that support multiple cell selection strategies to pick the best function order and cell selection strategy for any specific problem.

Finite model enumeration can be posed as a *constraint programming (CP)* task [27]. Some CP solvers, e.g., Minion [17] and Gecode [37], support parallelization [31]. In CP, the search space can be partitioned by adding constraints to rule in and/or out partitions. Each partition can be processed by a separate worker thread/process. Minion further implements a work stealing search scheme that also partitions the search space dynamically by splitting the existing constraint model after the search has started [15, 29]. However, to effectively add *symmetry-breaking* constraints such as lex-leaders to a CP solver often requires deep knowledge of the solver *and* the problem at hand (e.g., the semigroups in [15]) which may not be available when mathematicians first define and study a new algebraic structure.

Moreover, to use traditional CP solvers for finite model enumerations, mathematicians need to learn a new CP-specific language such as CHR [45] and Savile Row [38]. It is possible to use a translator to translate between languages, but that adds uncertainties to the fidelity and the optimality of the translated specifications. FOL remains one of the most popular languages among mathematicians due to its simple and intuitive syntax. Moreover, a popular automatic theorem prover, Prover9 [34], shares the same input language with Mace4. This adds more than just convenience to the process, as it also reduces the chances of discrepancies between Prover9 and Mace4 on the same problem.

A well-known issue with enumerating models defined with FOL are the isomorphic models included in the outputs. This is an inherent symmetry property of FOL [40]. There is extensive research on *symmetry-breaking* [4,11–13,28,40,43,47]. Although complete symmetry-breaking is known to be computationally challenging [13, 47], many useful algorithms, such as the LNH and the XLNH [4, 5], have emerged in partial symmetry-breaking. The LNH can be considered a symmetry-breaking with interchangeable values in constraint satisfaction problems (CSP) [19]. The XLNH is more restrictive as it only works on unary operations. The LNH is implemented in many systems such as Falcon [50], SEM [51], FMSET [6], and Mace4. The isomorphic cubes algorithm, which removes more cubes as the cube length grows, complements the LNH.

Another important symmetry-breaking strategy is to steer the search engine away from the fruitless exploration of sub-search space by adding symmetry-breaking input clauses [13, 47]. The cube-based parallel algorithms are compatible with algorithms of this kind of strategy as long as they do not break the LNH.

Some finite model finders, such as SEMK [8] and SEMD [24], try to completely suppress isomorphic models in the search process. However, these isomorph-free algorithms are not easy to parallelize as global information is generated and consumed in many steps, requiring high-cost synchronizations between cooperating workers, especially when they run on different computers. The cube-based parallel algorithm, on the other hand, is an EPS method that requires no synchronizations between workers. The static removal of isomorphic cubes done in a preprocessing step is shown to be effective in suppressing isomorphic models even before the actual search begins. The augmented work stealing algorithm is not high in synchronization costs because it does not involve communications between running jobs. The remaining isomorphic models from the cube-based algorithms can be efficiently removed by the invariant-based isomorphic model filtering algorithm as a postprocessing step.

Another algorithm, DSYM [4], exploits local symmetries by finding symmetries (synonym to isomorphisms in their terminology) under invariant partial interpretations (which are invariant cubes) and without parallelism. It also works with the LNH and XLNH. DSYM is a predictive algorithm that works at the *parent* level and predicts which of its immediate children will be isomorphic cubes. It can be seen as a special case of the isomorphic cube algorithm because it removes isomorphic cubes having the same immediate parents, while the isomorphic cube algorithm removes all isomorphic cubes, irrespective of their parents. Nevertheless, for the cases that DSYM covers, it does so right before the cubes are generated, while the isomorphic cube algorithm only detects the symmetries right after the cubes are generated. A disadvantage of DSYM is that it is not clear how it can be effectively parallelized. Furthermore, DSYM only detects symmetries under the same subtree. The isomorphic cubes removal algorithm, on the other hand, detects both global and local symmetries the same way, and hence detects and removes more symmetries than DSYM. Moreover, DSYM uses only two invariants in testing isomorphism between cubes, while we use many invariants that are proven successful in the invariant-based isomorphic model removal algorithm in our isomorphic cubes removal process. Nevertheless, DSYM can be applied to the cube generation process as well as the final model generation process. That is, the isomorphic cube removal algorithm is compatible with DSYM, as with any other symmetry-breaking algorithm that works with the LNH.

7 Conclusions and Future Work

In this paper, we introduce an efficient parallel algorithm together with a novel symmetry-removal mechanism for enumerating finite models. The approach is inspired by the cube-and-conquer paradigm, successfully used in SAT solving, which partitions the search space into cubes and then massively paralyzes. In contrast, our approach applies symmetries specific to finite model finding.

In conclusion, this paper fulfills an important unmet need for an efficient algorithm for enumerating finite algebraic models in computational algebra by enhancing the existing finite model enumeration process with the parallel cubes algorithm and the isomorphic cubes removal algorithm that reduce both the runtime and the number of output isomorphic models. These new algorithms are so scalable that they can be used on a laptop as well as on a cluster of powerful computers, and they require minimal efforts to safely integrate into existing finite model finders. Very importantly, these algorithms can be used as a black-box without requiring the users to have any knowledge about the way they work.

Future research will focus on improving isomorphic cube removal, on best cell selection strategy, and on predicting of optimal cube length.

References

1 João Araújo, João Pedro Araújo, Peter J. Cameron, Edmond W. H. Lee, and Jorge Raminhos. A survey on varieties generated by small semigroups and a companion website, 2019. arXiv:1911.05817.

2 João Araújo, Choiwah Chow, and Mikoláš Janota. Boosting isomorphic model filtering with invariants. *Constraints*, 27(3):360–379, Jul 2022. doi:10.1007/s10601-022-09336-x.

3 João Araújo, David Matos, and João Ramires. MarcieDB: a model and theory database. https://marciedb.pythonanywhere.com, 2022.

4 Gilles Audemard, Belaid Benhamou, and Laurent Henocque. Predicting and detecting symmetries in FOL finite model search. *J. Autom. Reason.*, 36(3):177–212, 2006. doi:10.1007/s10817-006-9040-3.

5 Gilles Audemard and Laurent Henocque. The eXtended least number heuristic. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning, First International Joint Conference, IJCAR*, volume 2083 of *Lecture Notes in Computer Science*, pages 427–442, Berlin, Heidelberg, 2001. Springer. doi:10.1007/3-540-45744-5_35.

6 Belaid Benhamou and Laurent Henocque. A hybrid method for finite model search in equational theories. *Fundam. Informaticae*, 39(1-2):21–38, 1999. doi:10.3233/FI-1999-391202.

7 R.D. Blumofe and C.E. Leiserson. Scheduling multithreaded computations by work stealing. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 356–368, 1994. doi:10.1109/SFCS.1994.365680.

8 Thierry Boy de la Tour and Prakash Countcham. An isomorph-free SEM-like enumeration of models. *Electronic Notes in Theoretical Computer Science*, 125(2):91–113, 2005. Proceedings of the 5th International Workshop on Strategies in Automated Deduction (Strategies 2004). URL: https://www.sciencedirect.com/science/article/pii/S1571066105000976, doi:https://doi.org/10.1016/j.entcs.2005.01.003.

9 Stanley Burris and Hanamantagouda P. Sankappanavar. *A course in universal algebra*, volume 78 of *Graduate texts in mathematics*. Springer, New York, NY, 1981.

10 Geoffrey Chu, Christian Schulte, and Peter J. Stuckey. Confidence-based work stealing in parallel constraint programming. In Ian P. Gent, editor, *Principles and Practice of Constraint Programming - CP*, volume 5732, pages 226–241. Springer, 2009. doi:10.1007/978-3-642-04244-7_20.

11 Koen Claessen and Niklas Sörensson. New techniques that improve MACE-style finite model finding. In *Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications*, 2003.

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

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

14 A. Distler and J. Mitchell. Smallsemi, a library of small semigroups in GAP, Version 0.6.12. https://gap-packages.github.io/smallsemi/, 2019. GAP package.

15 Andreas Distler, Christopher Jefferson, Tom Kelsey, and Lars Kotthoff. The semigroups of order 10. In Michela Milano, editor, *Principles and Practice of Constraint Programming - CP*, volume 7514, pages 883–899. Springer, 2012. doi:10.1007/978-3-642-33558-7_63.

16 The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.11.1*, 2021. URL: https://www.gap-system.org.

17 Ian P. Gent, Christopher Jefferson, and Ian Miguel. Minion: A fast scalable constraint solver. In Gerhard Brewka, Silvia Coradeschi, Anna Perini, and Paolo Traverso, editors, *ECAI, 17th European Conference on Artificial Intelligence, Including Prestigious Applications of*

Intelligent Systems (PAIS), Proceedings, volume 141 of *Frontiers in Artificial Intelligence and Applications*, pages 98–102, Amsterdam, Netherlands, 2006. IOS Press. URL: http://www.booksonline.iospress.nl/Content/View.aspx?piid=1654.

18 Ian P. Gent, Ian Miguel, Peter Nightingale, Ciaran McCreesh, Patrick Prosser, Neil C. A. Moore, and Chris Unsworth. A review of literature on parallel constraint solving. *Theory Pract. Log. Program.*, 18(5-6):725–758, 2018. doi:10.1017/S1471068418000340.

19 Pascal Van Hentenryck, Pierre Flener, Justin Pearson, and Magnus Ågren. Tractable symmetry breaking for CSPs with interchangeable values. In Georg Gottlob and Toby Walsh, editors, *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 277–284. Morgan Kaufmann, 2003. URL: http://ijcai.org/Proceedings/03/Papers/041.pdf.

20 Marijn Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. Cube and conquer: Guiding CDCL SAT solvers by lookaheads. In Kerstin Eder, João Lourenço, and Onn Shehory, editors, *Hardware and Software: Verification and Testing - 7th International Haifa Verification Conference, HVC, Revised Selected Papers*, volume 7261, pages 50–65. Springer, 2011. doi:10.1007/978-3-642-34188-5_8.

21 Marijn J. H. Heule, Oliver Kullmann, and Armin Biere. Cube-and-conquer for satisfiability. In Youssef Hamadi and Lakhdar Sais, editors, *Handbook of Parallel Constraint Reasoning*, pages 31–59. Springer, 2018. doi:10.1007/978-3-319-63516-3_2.

22 Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the Boolean Pythagorean triples problem via cube-and-conquer. In *Theory and Applications of Satisfiability Testing (SAT)*, 2016. doi:10.1007/978-3-319-40970-2_15.

23 Antti E. J. Hyvärinen, Matteo Marescotti, and Natasha Sharygina. Lookahead in partitioning SMT. In *Formal Methods in Computer Aided Design, FMCAD*, pages 271–279. IEEE, 2021. doi:10.34727/2021/isbn.978-3-85448-046-4_37.

24 Xiangxue Jia and Jian Zhang. A powerful technique to eliminate isomorphism in finite model search. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning*, pages 318–331, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

25 Tommi Junttila, Matti Karppa, Petteri Kaski, and Jukka Kohonen. An adaptive prefix-assignment technique for symmetry reduction. *Journal of Symbolic Computation*, 99:21–49, 2020. URL: https://www.sciencedirect.com/science/article/pii/S0747717119300288, doi:https://doi.org/10.1016/j.jsc.2019.03.002.

26 Bernard Jurkowiak, Chu Min Li, and Gil Utard. A parallelization scheme based on work stealing for a class of SAT solvers. *J. Autom. Reason.*, 34(1):73–101, 2005. doi:10.1007/s10817-005-1970-7.

27 Majid Ali Khan. Efficient enumeration of higher order algebraic structures. *IEEE Access*, 8:41309–41324, 2020. doi:10.1109/ACCESS.2020.2976876.

28 Markus Kirchweger and Stefan Szeider. SAT modulo symmetries for graph generation. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP*, volume 210 of *LIPIcs*, pages 34:1–34:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CP.2021.34.

29 Lars Kotthoff and Neil C. A. Moore. Distributed solving through model splitting. *ArXiv*, abs/1008.4328, 2010.

30 Kenneth Kunen. The structure of conjugacy closed loops. *Transactions of the American Mathematical Society*, 352(6):2889–2911, 2000.

31 Arnaud Malapert, Jean-Charles Régin, and Mohamed Rezgui. Embarrassingly parallel search in constraint programming. *J. Artif. Intell. Res.*, 57:421–464, 2016. doi:10.1613/jair.5247.

32 Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Improving search space splitting for parallel SAT solving. In *22nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI*. IEEE Computer Society, 2010. doi:10.1109/ICTAI.2010.56.

33 Ruben Martins, Vasco M. Manquinho, and Inês Lynce. An overview of parallel SAT solving. *Constraints An Int. J.*, 17(3):304–347, 2012. doi:10.1007/s10601-012-9121-3.

34 W. McCune. Prover9 and mace4. `http://www.cs.unm.edu/~mccune/prover9/`, 2005–2010.

35 William McCune. Mace4 reference manual and guide, August 2003. URL: `https://www.cs.unm.edu/~mccune/prover9/mace4.pdf`.

36 Gábor Nagy and Petr Vojtěchovský. LOOPS, computing with quasigroups and loops in GAP, Version 3.4.1. https://gap-packages.github.io/loops/, Nov 2018. Refereed GAP package.

37 Morten Nielsen. Parallel search in gecode. *Technical Report, Gecode*, 2006.

38 Peter Nightingale, Özgür Akgün, Ian P. Gent, Christopher Jefferson, Ian Miguel, and Patrick Spracklen. Automatically improving constraint models in savile row. *Artificial Intelligence*, 251:35–61, 2017. URL: `https://www.sciencedirect.com/science/article/pii/S0004370217300747`, `doi:https://doi.org/10.1016/j.artint.2017.07.001`.

39 Anthony Palmieri, Jean-Charles Régin, and Pierre Schaus. Parallel strategies selection. *CoRR*, abs/1604.06484, 2016. URL: `http://arxiv.org/abs/1604.06484`, `arXiv:1604.06484`.

40 Giles Reger, Martin Riener, and Martin Suda. Symmetry avoidance in MACE-style finite model finding. In Andreas Herzig and Andrei Popescu, editors, *Frontiers of Combining Systems FroCoS*, volume 11715, pages 3–21, Switzerland AG, 2019. Springer. `doi:10.1007/978-3-030-29007-8\_1`.

41 Jean-Charles Régin and Arnaud Malapert. Parallel constraint programming. In Youssef Hamadi and Lakhdar Sais, editors, *Handbook of Parallel Constraint Reasoning*, pages 337–379. Springer, 2018. `doi:10.1007/978-3-319-63516-3\_9`.

42 Jean-Charles Régin, Mohamed Rezgui, and Arnaud Malapert. Embarrassingly parallel search. In Christian Schulte, editor, *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, pages 596–610, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. `doi:10.1007/978-3-642-40627-0\_45`.

43 Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006. URL: `https://www.sciencedirect.com/science/bookseries/15746526/2`.

44 Neil J. A. Sloane and The OEIS Foundation Inc. The on-line encyclopedia of integer sequences, 2020. URL: `http://oeis.org/?language=english`.

45 Jon Sneyers, Peter van Weert, Tom Schrijvers, and Leslie de Koninck. As time goes by: Constraint handling rules: A survey of chr research from 1998 to 2007. *Theory and Practice of Logic Programming*, 10(1):1–47, 2010. `doi:10.1017/S1471068409990123`.

46 Bernardo Subercaseaux and Marijn Heule. Toward optimal radio colorings of hypercubes via SAT-solving. In Ruzica Piskac and Andrei Voronkov, editors, *Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 94 of *EPiC Series in Computing*, pages 386–404. EasyChair, 2023. URL: `https://easychair.org/publications/paper/b8Cn`, `doi:10.29007/qrmp`.

47 Toby Walsh. Symmetry breaking constraints: Recent results. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*. AAAI Press, 2012. URL: `http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/4974`.

48 H. Zhang. Combinatorial designs by SAT solvers. *Handbook of Satisfiability*, pages 533–568, 2009. URL: `https://cir.nii.ac.jp/crid/1571980076163512448`.

49 Hantao Zhang and Jian Zhang. MACE4 and SEM: A comparison of finite model generators. In Maria Paola Bonacina and Mark E. Stickel, editors, *Automated Reasoning and Mathematics - Essays in Memory of William W. McCune*, volume 7788 of *Lecture Notes in Computer Science*, pages 101–130. Springer, 2013. `doi:10.1007/978-3-642-36675-8\_5`.

50 Jian Zhang. Constructing finite algebras with FALCON. *Journal of Automated Reasoning*, 17:1–22, 08 1996. `doi:10.1007/BF00247667`.

51 Jian Zhang and Hantao Zhang. SEM: a system for enumerating models. In *IJCAI*, pages 298–303, 1995. URL: `http://ijcai.org/Proceedings/95-1/Papers/039.pdf`.